ST9040 FAMILY
8/16 BIT MCU

DATABOOK

1st EDITION

SGS-THOMSON MICROELECTRONICS

ST9040 FAMILY
8/16 BIT MCU

ST9040 FAMILY
8/16 BIT MCU

DATABOOK

1st EDITION

APRIL 1993

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED.

SGS-THOMSON PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF SGS-THOMSON Microelectronics.
As used herein:

1. Life support devices or systems are those which (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonably expected to result in significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can reasonably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

TABLE OF CONTENTS

GENERAL INDEX

Pages
Number

Pages
Number

Pages
Number

Pages
Number

Pages
Number

GENERAL INDEX

ST9 APPLICATION TAILORED MCU

The ST9 family of 8/16 bit Microcontrollers (MCUs) was designed after the requirements of the most advanced applications in computer, consumer, telecom, industrial and automotive Segments.

Processed with the same proprietary CMOS EPROM and EEPROM technologies that have established SGS-THOMSON as a world leading supplier of non-volatile memories, the ST9 provides high speed computing with reduced power consumption.

Built around a high performance, register based core, the ST9 family offers different program and data memory sizes and a wide range of on-chip peripherals to meet the needs of most systems.

Time to market is minimized with ST9's well defined, socket compatible, evolution path, from application evaluation with EPROMs, to prototyping using OTPs, up to the high volume production using cost effective ROM versions.

All standard ST9 devices include a Serial Peripheral Interface, a Watchdog Timer to ensure system integrity against externally generated malfunctions, bit configurable I/Os, prioritizable Interrupts for real-time data handling, and DMA for fast data transfers with handshake (HSHK).

In addition ST9 family variants include up to three Multi-Function Timers, two Serial Communication Interfaces (SCI), an Analog to Digital converter (A/D) and On-Screen Display and Data-Slicer for TV control.

REGISTER BASED ARCHITECTURE

The Register based architecture provides more efficient data handling and reduced code size compared to an accumulator based MCU. It also provides the capability for fast context switching.

224 of the 256 8-bit Registers in the ST9 Register File are available as accumulators, index registers, or stack pointers and can be cascaded to perform all these functions as 16-bit registers. The remaining registers are dedicated to system and peripheral control.

This architecture is common to all ST9 devices.

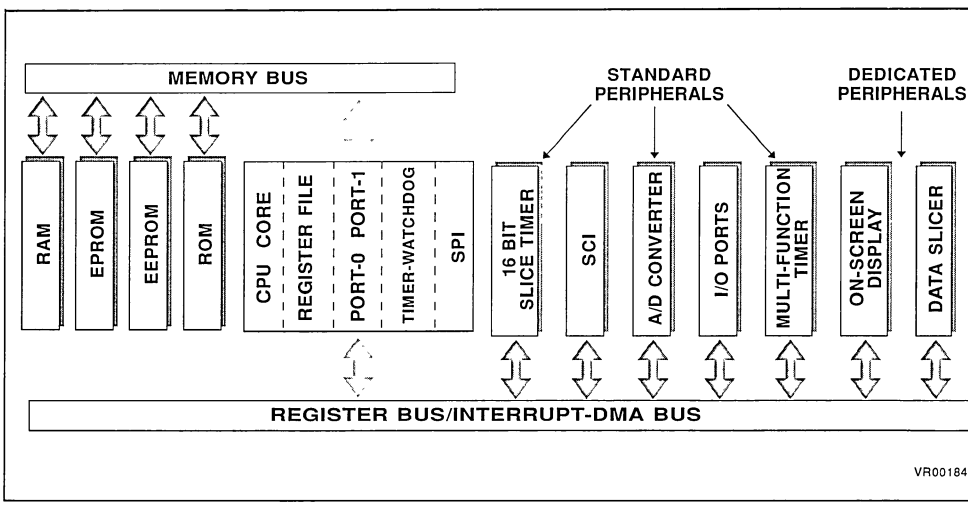
FLEXIBLE I/O

The flexibility of the ST9 I/O pins allow designers to match the MCU to the application, and not the application to the MCU.

Most I/Os can be individually programmed as input (TTL or CMOS thresholds), output (open-drain or push-pull), bidirectional, or as the Alternate Function of a peripheral, such as a Timer or an A/D Converter.

COMPREHENSIVE SCI

Serial communication is easily implemented, using formats and facilities offered by the ST9 Serial Communication Interface.



ST9 Architectural Block Diagram

This peripheral provide full flexibility in character format (5,6,7,8 databits), odd, even or no parity, address bit, 1, 1.5, or 2 stop bits in asynchronous mode, and an integral baud rate generator allowing communication at up to 370k baud in asynchronous mode or 1.5Mbyte/s in synchronous mode.

Industrial, telecom and communication systems users can furthermore benefit from the self-test and address bit wake-up facility offered by the character search mode.

FAST A/D WITH ANALOG WATCHDOG

Up to 8 analog input voltages can be sequentially converted by the Analog to Digital converter including on-chip sample and hold.

The 11µs conversion time, and the possibility to trigger conversions either by the on-chip timers, or by external sources, allows real time processing of analog data.

CPU loading is also reduced by the analog watchdog on two channels, the peripheral interrupts the ST9 when the analog input voltage moves out of a preset threshold window.

UNIVERSAL SPI

A universal Serial Peripheral Interface, providing basic I^2C-bus, Microwire-Bus and S-BUS functionality, allows efficient communication with low-cost external peripherals or serial access memories such as EEPROMs.

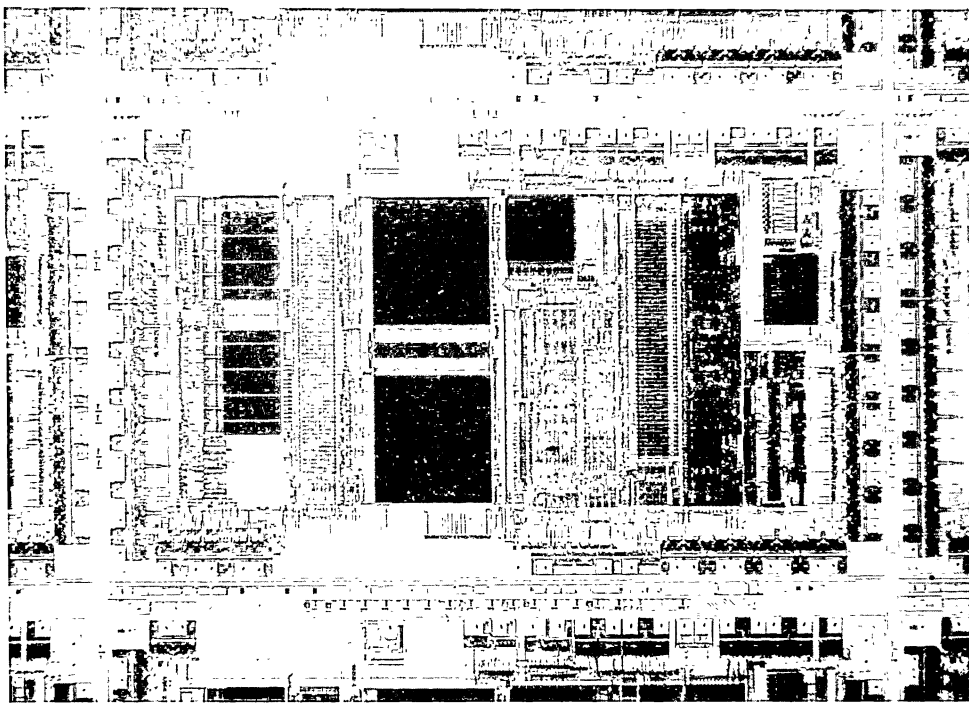
MULTI-FUNCTION TIMERS

The 16 bit up/down counter operating in 13 modes gives the ST9 Multi-function Timer the possibility to cover most application timing requirements.

Two input pins, programmable as external clock, gate or trigger, allow 16 modes of operation, including auto-discrimination of the direction of externally generated signals.

Pulse Width Generation can easily be implemented, using the overflow/underflow signal and the two 16 bit comparison registers, each of them able to independently set, reset, toggle or ignore two output bits.

The Multifunction Timer outputs may also generate interrupts for system scheduling, and trigger DMA transactions of a data byte to or from a data table in memory through an I/O Port with handshake.



ST90E40

ON-SCREEN DISPLAY

Interactive information display for television control is easily implemented with the powerful ST9 On-Screen Display. With up to 34 characters in 15 rows, and colour, italic, underline, flash, tranparent and fringe options, the 128 character set can be adapted for all needs.

DATA-SLICER

Closed Caption Data can be easily extracted from the video signal with the ST9 Data Slicer. When used in conjunction with the ST9 On-Screen Display, a powerful TV controller can be achieved with the minimum of components.

POWERFUL INSTRUCTION SET

The ST9 has 14 addressing modes and instructions (including multiply, divide, table search and block move) to cover all data manipulation needs, bit, byte and word, at the speed required by even the most demanding control application.

Its instruction set was conceived to facilitate the software designer's task, and to improve programming efficiency.

FULL DEVELOPMENT SUPPORT

ST9 Development Tools are designed for application development efficiency.

A high level macro assembler (with IF/THEN, DO/WHILE, SYSTEM/CASE, PROCedure C language constructs in the assembler) is available, as well as an incremental linker able to link up to 16 Mbytes of program and data, a library maintainer for archiving common software routines and Software Simulation of the code execution, allowing off-line code development and timing analysis.

The validated ANSI standard C Compiler generates optimised code for the ST9. In addition a GNU C cross-compiler and Linker allows development support under the Microsoft Windows 3™ environment.

Cost effective emulation is provided either through a software module running on standard PCs, or with the ST9 Starter Kit, offering hardware emulation capability.

Full real time hardware emulation is provided by the ST9-HDS system.

ST9 FAMILY OVERVIEW

All devices have 256 byte Register File with 224 General Purpose Registers (Accumulators/RAM), TWD and SPI Peripherals

DEVICE	ROM x8	EPROM OTP ROM[1] x8	RAM x8	EEPROM x8	MFT	SCI	A/D Inputs	BSS	MAX I/O	HSHK
ST9026	16K		256		1	1			40	1
ST9027	16K		256		1	1			32	1
ST9028	16K		256		1	1			36	1
ST90E26		16K	256		1	1			40	1
ST90E27		16K	256		1	1			32	1
ST90E28		16K	256		1	1			36	1
ST90T26		16K[1]	256		1	1			40	1
ST90T27		16K[1]	256		1	1			32	1
ST90T28		16K[1]	256		1	1			36	1
ST90R26	-	-	256		1	1			32	1
ST9030	8K				2	1	8		56	1
ST90E30		8K			2	1	8		56	1
ST90T30		8K[1]			2	1	8		56	1
ST90R30	-	-			2	1	8		40	1
ST9032	12K	-			2	1	8		56	1
ST9036	16K		256		2	1	8		40	1
ST90E36		16K	256						40	
ST90T36		16K[1]	256						40	
ST9040	16K		256	512	2	1	8		56	1
ST90E40		16K	256	512	2	1	8		56	1
ST90T40		16K[1]	256	512	2	1	8		56	1
ST90R40	-	-	256	512	2	1	8		40	1
ST90R50	-	-			3	2	8	1	56	2
ST90R51	-	-			3	2	8	1	54	2
ST9054	32K		1280		3	2	8	1	72	2
ST90E54		32K	1280		3	2	8	1	72	2
ST90R54			1280		3	2	8	1	72	2
ST9292	24K		384				3[2]		41	
ST92E92		24K	384				3[2]		41	
ST92T92		24K[1]	384				3[2]		41	
ST9293	32K		640				4[2]		41	
ST92E93		32K	640				4[2]		41	
ST92T93		32K[1]	640				4[2]		41	

Keys :

TWD	Timer/Watchdog	SCI	Serial Communications Interface
SPI	Serial Peripheral Interface	A/D	8 bit 8 channel A/D Converter
MFT	Multi-Function Timer	BSS	Bankswitch logic 16M byte address range
I/O	In :TLL/CMOS, Out : OD/PP Alternate Functional Peripheral	HSHK	# Ports with Handshake Capability

Notes :

1 OTP ROM = One Time Programmable
2 6 bit A/D Converter

DEVICE	OTHER FEATURES	PACKAGE (Operating temperature)			PAGE
		DIP	LCC	QFP	
ST9026		P48 (1,6)			9
ST9027		P40 (1,6)			9
ST9028			P44 (1,6)		9
ST90E26		C48W (1)			11
ST90E27		C40W (1)			11
ST90E28			C44W (1)		11
ST90T26		P48 (6)			11
ST90T27		P40 (6)			11
ST90T28			P44 (6)		11
ST90R26		P48 (6)			13
ST9030			P68 (1,6)	P80 (1)	15
ST90E30			C68W (1)	C80W (1)	17
ST90T30			P68 (6)	P80 (1)	17
ST90R30			P68 (6)		19
ST9032			P68 (1,6)	P80 (1)	21
ST9036			P68 (1,6)	P80 (1)	23
ST90E36			C68W (1)	C80W (1)	25
ST90T36			P68 (6)	P80 (1)	25
ST9040			P68 (1,6)	P80 (1)	27
ST90E40			C68W (1)	C80W (1)	29
ST90T40			P68 (6)	P80 (1)	29
ST90R40			P68 (6)		31
ST90R50			P84 (6)		33
ST90R51				P80 (1)	35
ST9054			P84 (1,6)		37
ST90E54			C84W (1)		39
ST90R54			P84 (6)		41
ST9292	} OSD, STM,	PS42 (1)			43
ST92E92	} DSL, PWM	CS42W (1)			45
ST92T92	}	PS42 (1)			45
ST9293	}	PS42 (1)			47
ST92E93	} OSD, STM	CS42W (1)			49
ST92T93	}	PS42 (1)			49

Keys :

OSD	On Screen Display	Pxx	Plastic Package
STM	Slice Timer	PSxx	Plastic Shrink DIP Package
DSL	Data Slicer extracting Closed Caption Data	CxxW	Ceramic Package with window
PWM	Pulse Width Modulation outputs	CSxxW	Ceramic Shrink DIP Package with window

Temperature Ranges :

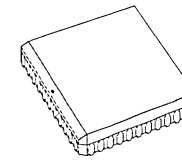
(1)One version available, 0 to +70°C (6)One version available, -40 to+85°C
(1,6)Two versions available, 0 to +70°C and -40 to+85°C

SGS-THOMSON MICROELECTRONICS

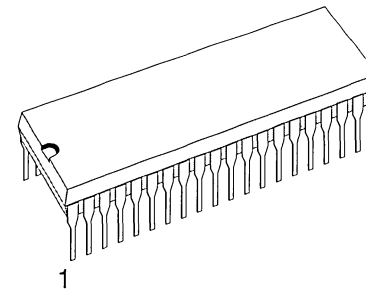
ST9026,ST9027,ST9028

16K ROM HCMOS MCUs WITH RAM

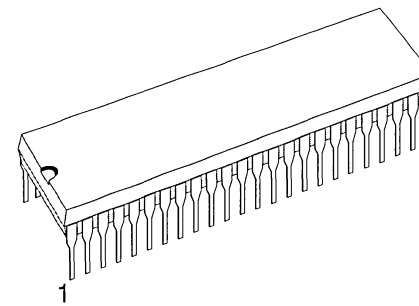
- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 16K bytes of ROM,
  256 bytes of RAM,
  224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 48-pin Dual in Line Plastic package for ST9026
- 40-pin Dual in Line Plastic package for ST9027
- 44-lead Plastic Leaded Chip Carrier package for ST9028
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 40 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- 16-bit Multifunction Timer, with an 8-bit prescaler and 13 operating modes
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed and One Time Programmable EPROM parts available for prototyping and pre-production development phases



PLCC44



PDIP40



PDIP48

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information

GENERAL DESCRIPTION

The ST9026, ST9027 and ST9028 (following mentioned as ST902X) are ROM members of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM parts are fully compatible with their EPROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 16K bytes of on-chip ROM, microcontrollers able to manage external memory, or as parallel processing elements in a system with other processors and peripheral controllers.

The nucleus of the ST902X is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

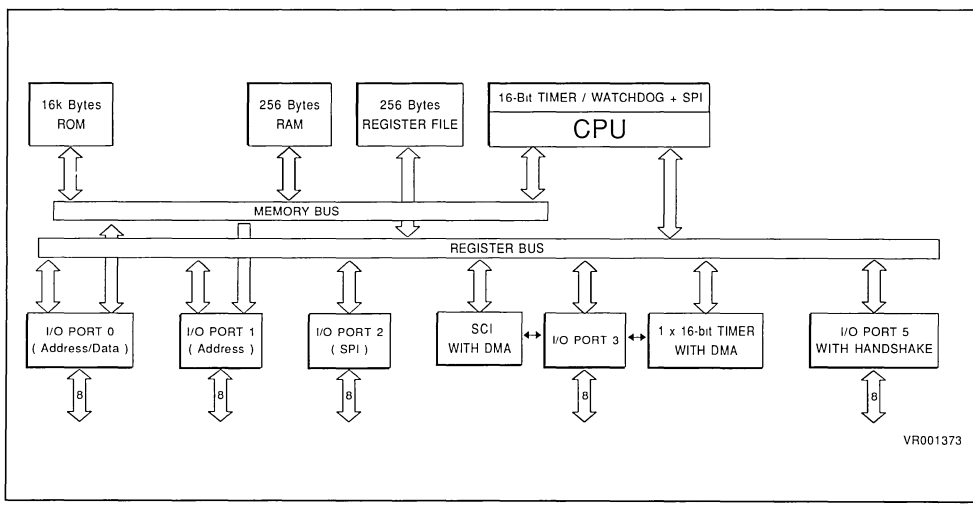
The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST902X with up to 40 I/O lines dedicated to digital Input/Output. These lines are grouped into up to five 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, external interrupts and serial or parallel I/O with or without handshake.

Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

The 16 bit MultiFunction Timer, with an 8 bit Prescaler and 12 operating modes allows simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

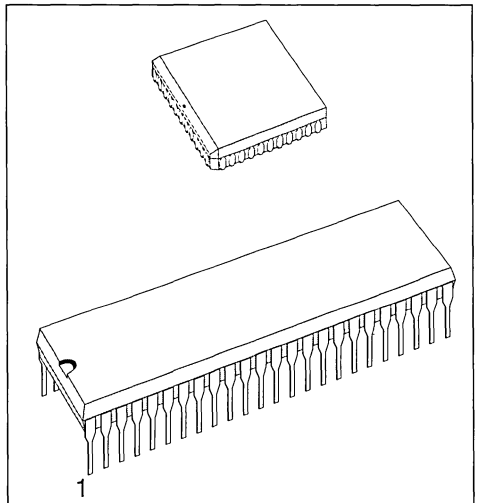
Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST902X Block Diagram

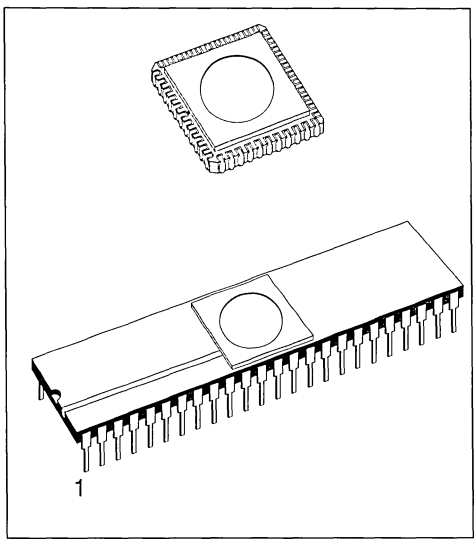


16K EPROM HCMOS MCUs WITH RAM

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 16K bytes of EPROM or OTP ROM
  256 bytes of RAM,
  224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 48-pin Window Dual in Line Ceramic Multilayer package for ST90E26
- 40-pin Window Dual in Line Ceramic Multilayer package for ST90E27
- 44-lead Window Ceramic Leaded Chip Carrier package for ST90E28
- 48-pin Dual in Line Plastic package for ST90T26
- 40-pin Dual in Line Plastic package for ST90T27
- 44-lead Plastic Leaded Chip Carrier package for ST90T28
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 40 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- 16-bit Multifunction Timer, with an 8-bit prescaler and 13 operating modes
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9026/27/28 16K ROM device



(Ordering Information at the end of the Datasheet)



GENERAL DESCRIPTION

The ST90E26, ST90E27 and ST90E28, ST90T26, ST90T27 and ST90T28 (following mentioned as ST90E2X) are EPROM members of the ST9 family of microcontrollers, in windowed ceramic (E) and plastic OTP (T) packages respectively, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The EPROM ST90E2X can be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 16K bytes of on-chip ROM, microcontrollers able to manage external memory, or as parallel processing elements in a system with other processors and peripheral controllers.

The nucleus of the ST90E2X is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

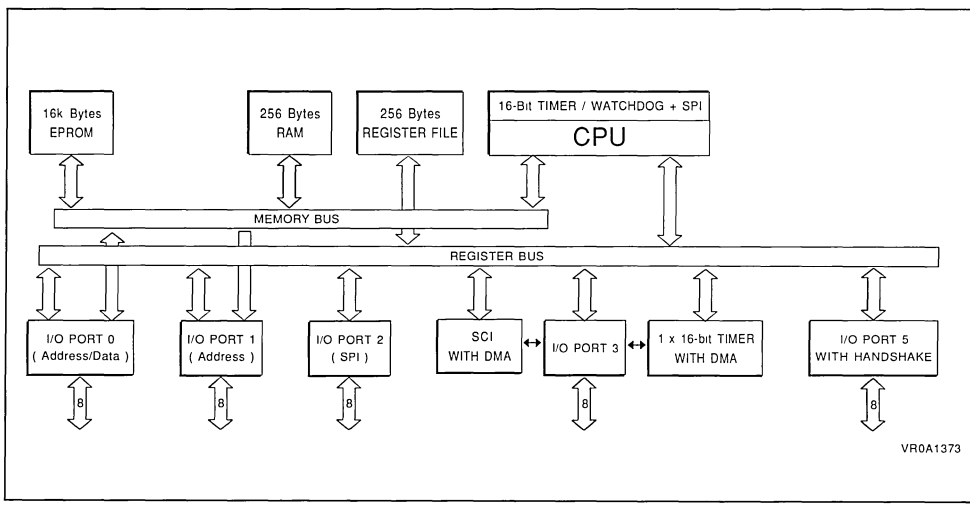
The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90E2X with up to 40 I/O lines dedicated to digital Input/Output. These lines are grouped into up to five 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, external interrupts and serial or parallel I/O with or without handshake.

Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

The 16 bit MultiFunction Timer, with an 8 bit Prescaler and 12 operating modes allows simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90E2X Block Diagram

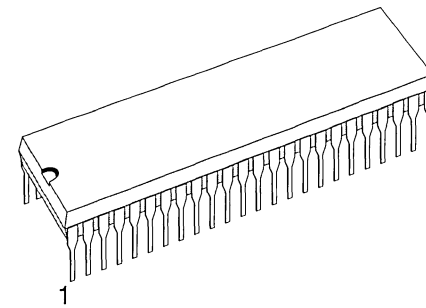


SGS-THOMSON MICROELECTRONICS

ST90R26

ROMLESS HCMOS MCU WITH RAM

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 256 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Romless to allow maximum external memory capability
- 48-lead Plastic Dual in Line package for ST90R26
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 24 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- 16-bit Multifunction Timer, with an 8-bit prescaler and 13 operating modes
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9026 16K ROM device (also available in windowed and One Time Programmable EPROM packages



PDIP48

(Ordering Information at the end of the Datasheet)

January 1993

GENERAL DESCRIPTION

The ST90R26 is a ROMLESS member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROMLESS part may be used for the prototyping and pre-production phases of development, and offers the maximum in program flexibility in production systems.

The ROMLESS ST90R26 can be configured as a microcontroller able to manage external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

The nucleus of the ST90R26 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C-bus and IM-bus Interface, plus memory interface. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

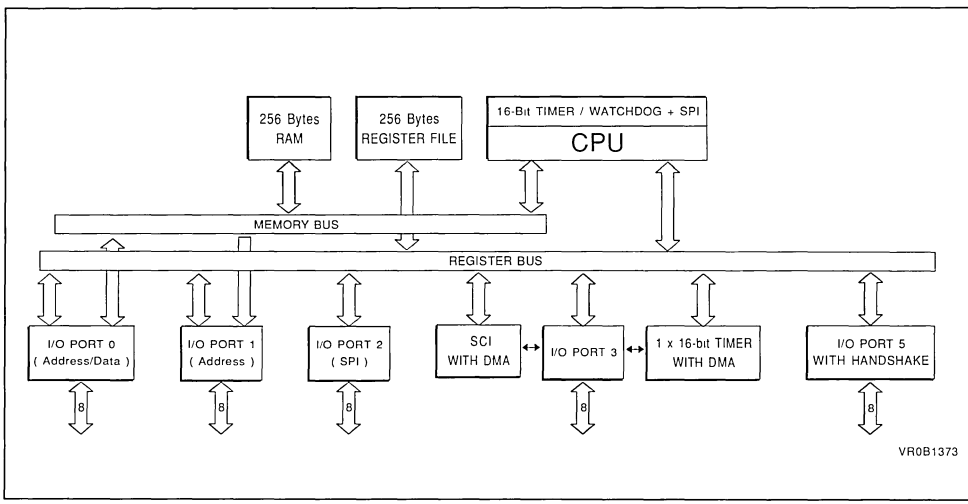
The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90R26 with up to 32 I/O lines dedicated to digital Input/Output. These lines are grouped into up to four 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing and status signals, address lines, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three memory spaces are available: Program Memory (external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

The 16 bit MultiFunction Timer, with an 8 bit Prescaler and 12 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90R26 Block Diagram

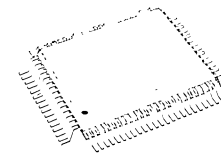


SGS-THOMSON MICROELECTRONICS

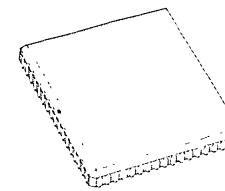
ST9030

8K ROM HCMOS MCU WITH A/D CONVERTER

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 8K bytes of ROM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 80-pin Plastic Quad Flat Pack package for ST9030Q
- 68-lead Plastic Leaded Chip Carrier package for ST9030C
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed and One Time Programmable EPROM parts available for prototyping and pre-production development phases



PQFP80



PLCC68

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information.

GENERAL DESCRIPTION

The ST9030 is ROM member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM part is fully compatible with its EPROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as standalone microcontrollers with 8K bytes of on-chip ROM, microcontrollers able to manage external memory, or as parallel processing elements in a system with other processors and peripheral controllers.

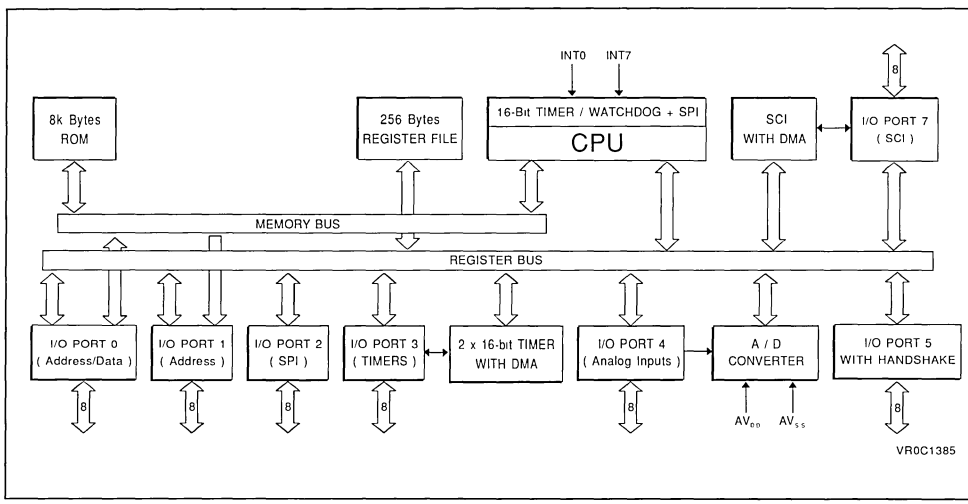
The nucleus of the ST9030 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C-bus and IM BUS Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set. The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9030 with 56 I/O lines dedicated to digital Input/Output. These lines are grouped into seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three memory spaces are available: Program Memory (internal and external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer. In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit 1/2 LSB resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST9030 Block Diagram

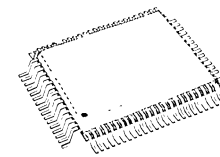


SGS-THOMSON MICROELECTRONICS

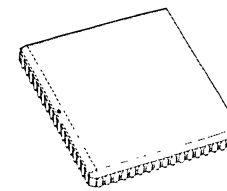
ST90E30
ST90T30

8K EPROM HCMOS MCUs WITH A/D CONVERTER

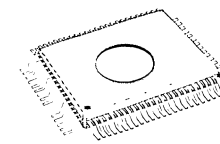
- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 8K bytes of EPROM or OTP ROM 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 80-pin Plastic Quad Flat Pack package for ST90T30Q
- 68-lead Plastic Leaded Chip Carrier package for ST90T30C
- 80-pin Window Ceramic Quad Flat Pack package for ST90E30G
- 68-lead Window Ceramic Leaded Chip Carrier package for ST90E30L
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9030 8K ROM device



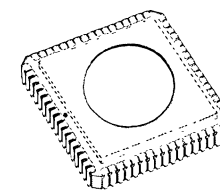
PQFP80



PLCC68



CQFP80W



CLCC68W

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information.

GENERAL DESCRIPTION

The ST90E30 and ST90T30 (following mentioned as ST90E30) are EPROM members of the ST9 family of microcontrollers, in windowed ceramic (E) and plastic OTP (T) packages respectively, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The EPROM ST90E30 may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 8K bytes of on-chip ROM, microcontrollers able to manage external memory, or as parallel processing elements in a system with other processors and peripheral controllers.

The nucleus of the ST90E30 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C Bus and IM BUS Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90E30 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

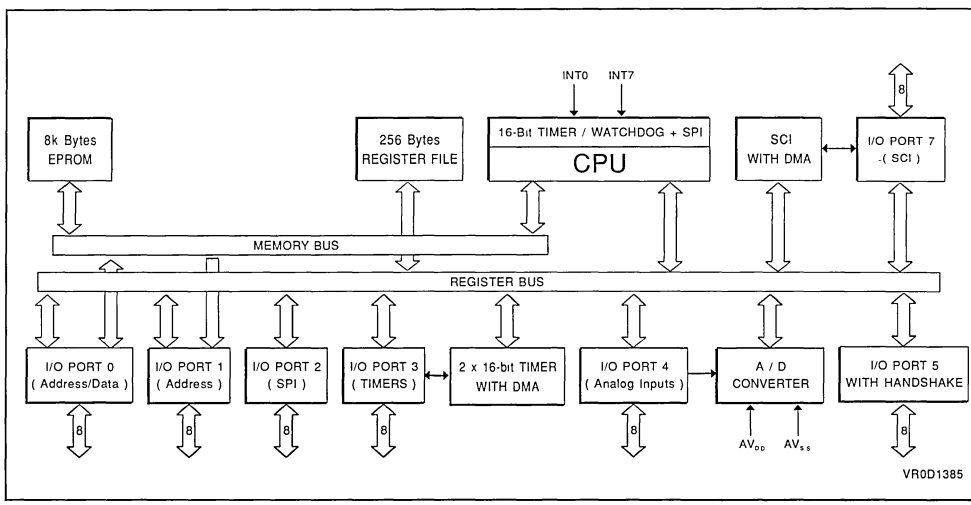
Three memory spaces are available: Program Memory (internal and external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

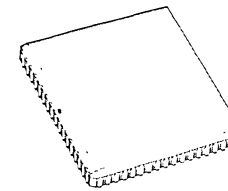
Figure 1. ST90E30 Block Diagram



ROMLESS HCMOS MCU WITH A/D CONVERTER

ADVANCE DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Romless to allow maximum external memory capability
- 68-lead Plastic Leaded Chip Carrier package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 40 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9030 8K ROM device (also available in windowed and One Time Programmable EPROM packages)



PLCC68

(Ordering Information at the end of the Datasheet)

January 1993

GENERAL DESCRIPTION

The ST90R30 is a ROMLESS member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROMLESS part may be used for the prototyping and pre-production phases of development, and offers the maximum in program flexibility in production systems.

The ROMLESS ST90R30 can be configured as a microcontroller able to manage external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

The nucleus of the ST90R30 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C-bus and IM-bus Interface, plus memory interface. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90R30 with up to 40 I/O lines dedicated to digital Input/Output. These lines are grouped into up to five 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing and status signals, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

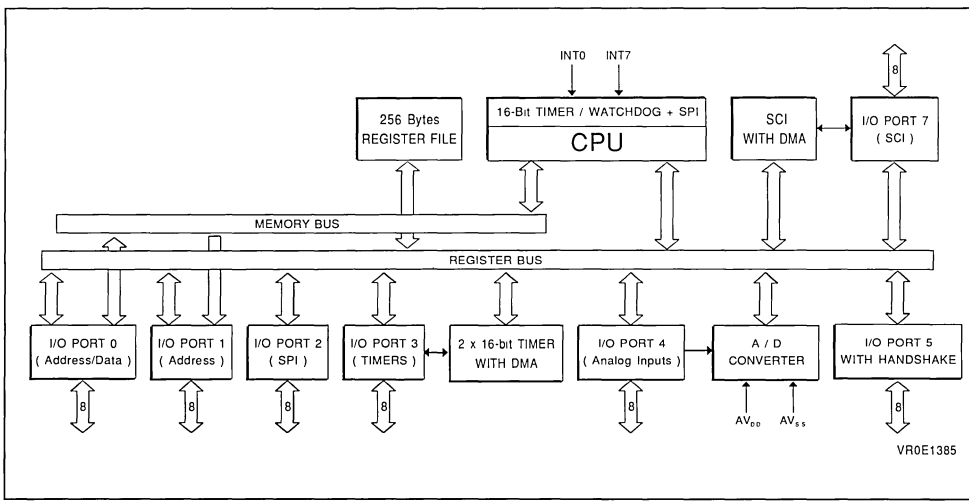
Three memory spaces are available: Program Memory (external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90R30 Block Diagram



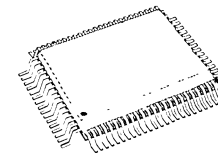
SGS-THOMSON MICROELECTRONICS

ST9032

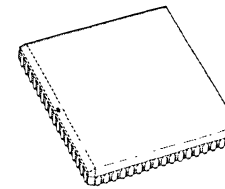
12K ROM HCMOS MCU WITH A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 12K bytes of ROM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 80-pin Plastic Quad Flat Pack package for ST9032Q
- 68-lead Plastic Leaded Chip Carrier package for ST9032C
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed and One Time Programmable EPROM parts available for prototyping and pre-production development phases
- Upward compatible with ST9030



PQFP80



PLCC68

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information.

GENERAL DESCRIPTION

The ST9032 device are ROM members of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM parts are fully compatible with their EPROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as standalone microcontrollers with 12K bytes of on-chip ROM, microcontrollers able to manage external memory, or as parallel processing elements in a system with other processors and peripheral controllers.

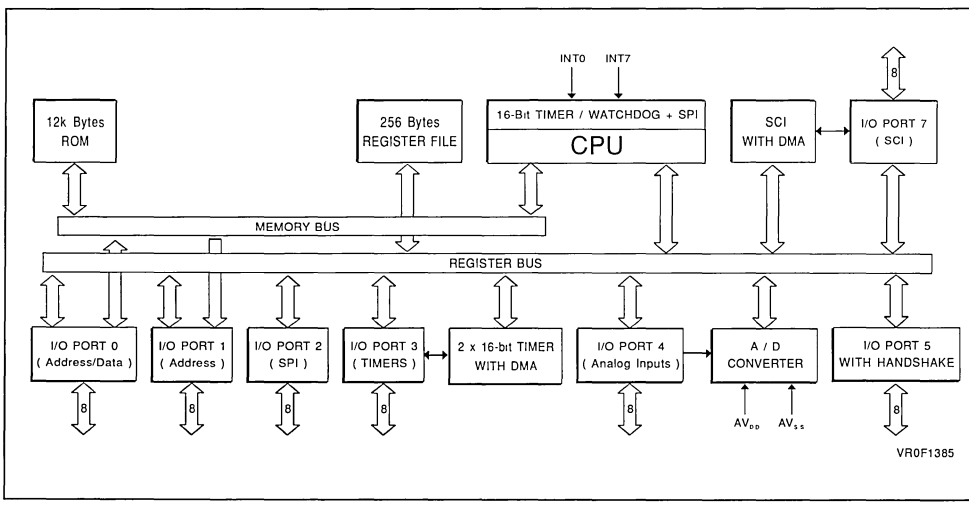
The nucleus of the ST9032 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C Bus and IM BUS Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set. The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9032 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three memory spaces are available: Program Memory (internal and external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer. In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit LSB resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST9032 Block Diagram



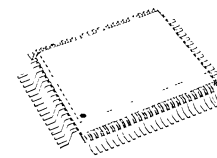
SGS-THOMSON MICROELECTRONICS

ST9036

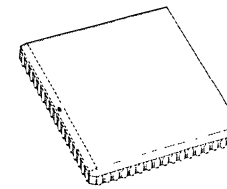
16K ROM HCMOS MCU WITH RAM AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 16K bytes of ROM,
  256 bytes of RAM,
  224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 80-pin Plastic Quad Flat Pack package for ST9036Q
- 68-lead Plastic Leaded Chip Carrier package for ST9036C
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 72 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed and One Time Programmable EPROM parts available for prototyping and pre-production development phases
- Upward compatible with ST9030 and ST9032



PQFP80



PLCC68

(Ordering Information at the end of the Datasheet)

January 1993

GENERAL DESCRIPTION

The ST9036 is a ROM member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM device is fully compatible with the EPROM version (ST90E36), which may be used for the prototyping and pre-production phases of development, and can be configured as: a standalone microcontroller with 16K bytes of on-chip ROM, a microcontroller able to manage external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

The nucleus of the ST9036 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9036 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

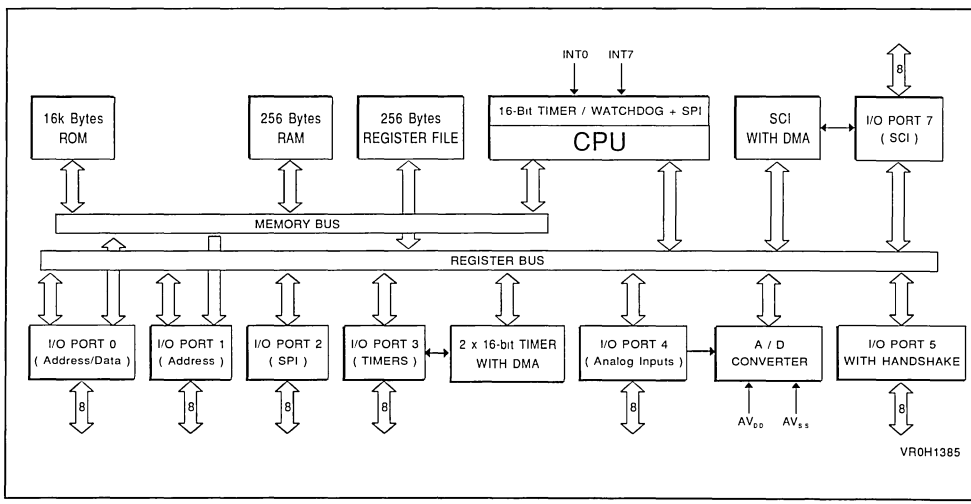
Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375,000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST9036 Block Diagram



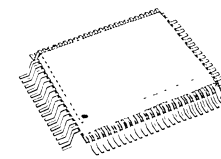
SGS-THOMSON MICROELECTRONICS

ST90E36
ST90T36

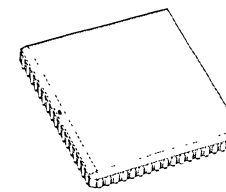
16K EPROM HCMOS WITH RAM AND A/D CONVERTER

PRELIMINARY DATA

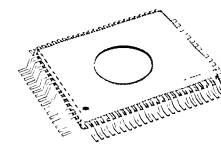
- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 16K bytes of EPROM or OTP ROM, 256 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 80-pin Plastic Quad Flat Pack package for ST90T36Q
- 80-pin Window Ceramic Quad Flat Pack package for ST90E36G
- 68-lead Plastic Leaded Chip Carrier package for ST90T36C
- 68-lead Window Ceramic Leaded Chip Carrier package for ST90E36L
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9036 16K ROM device



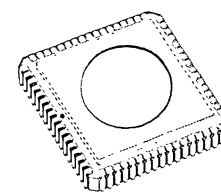
PQFP80



PLCC68



CQFP80W



CLCC68W

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information.

GENERAL DESCRIPTION

The ST90E36 and ST90T36 (following mentioned as ST90E36) are EPROM and OTP members of the ST9 family of microcontrollers, in windowed ceramic (E) and plastic OTP (T) packages respectively, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The EPROM ST90E36 may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 16K bytes of on-chip ROM, microcontrollers able to manage external memory, or as parallel processing elements in a system with other processors and peripheral controllers.

The nucleus of the ST90E36 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C Bus and IM BUS Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90E36 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

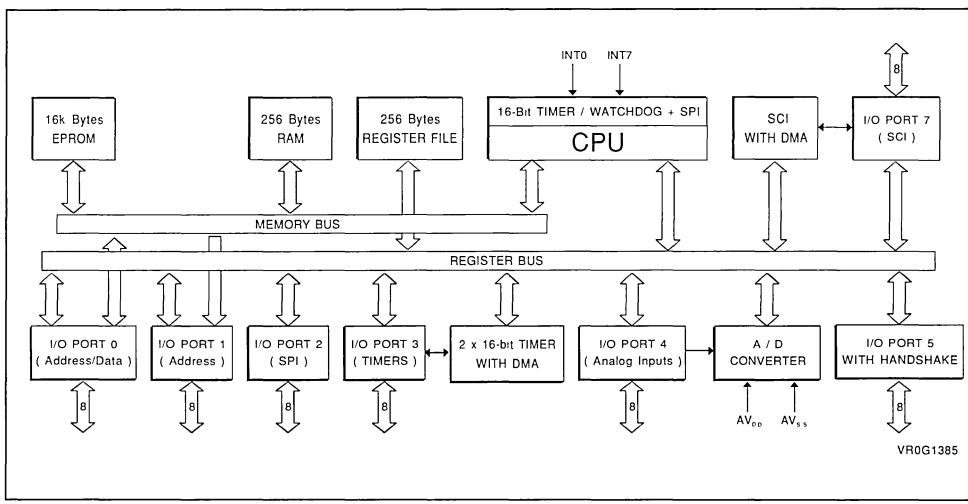
Three memory spaces are available: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90E36 Block Diagram

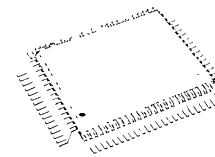


SGS-THOMSON MICROELECTRONICS

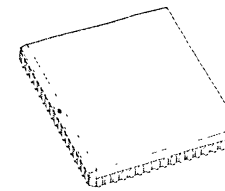
ST9040

16K ROM HCMOS MCU WITH EEPROM RAM AND A/D CONVERTER

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 16K bytes of ROM,
  256 bytes of RAM,
  224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 512 bytes EEPROM
- 80-pin PQFP package for ST9040Q
- 68-lead PLCC package for ST9040C
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed and One Time Programmable EPROM parts available for prototyping and pre-production development phases
- Upward compatible with ST9030, ST9032 and ST9036



PQFP80



PLCC68

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member Please contact SGS-THOMSON for further information

GENERAL DESCRIPTION

The ST9040 is a ROM member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM device is fully compatible with its EPROM version, which may be used for the prototyping and pre-production phases of development, and can be configured as: a standalone microcontroller with 16K bytes of on-chip ROM, a microcontroller able to manage external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

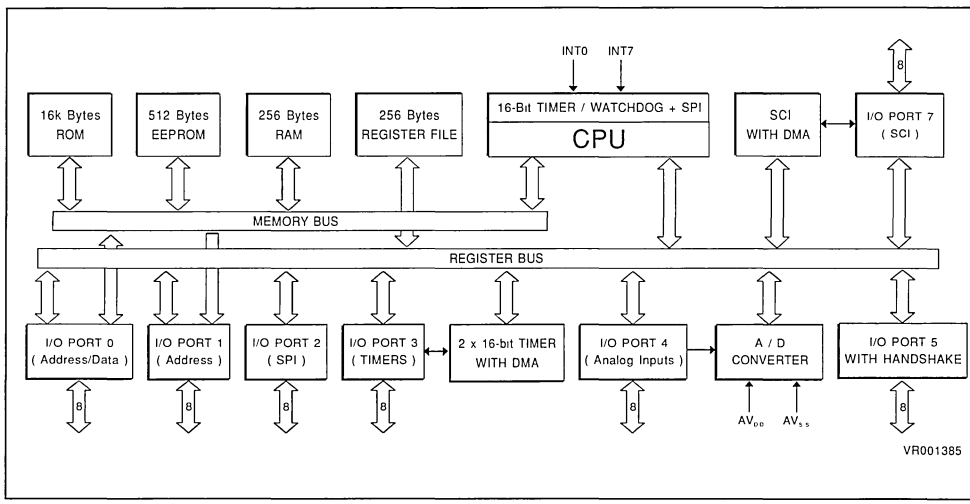
The nucleus of the ST9040 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set. The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9040 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer. In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375,000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST9040 Block Diagram

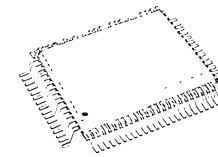


SGS-THOMSON MICROELECTRONICS

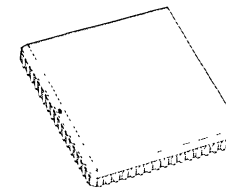
ST90E40
ST90T40

16K EPROM HCMOS MCU WITH EEPROM, RAM AND A/D CONVERTER

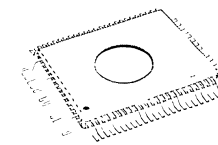
- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 16K bytes of EPROM or OTP ROM, 256 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 512 bytes of EEPROM
- 80-pin Plastic Quad Flat Pack package for ST90T40Q
- 80-pin Window Ceramic Quad Flat Pack package for ST90E40G
- 68-lead Plastic Leaded Chip Carrier package for ST90T40C
- 68-lead Window Ceramic Leaded Chip Carrier package for ST90E40L
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9040 16K ROM device



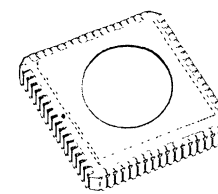
PQFP80



PLCC68



CQFP80W



CLCC68W

(Ordering Information at the end of the Datasheet)

GENERAL DESCRIPTION

The ST90E40 and ST90T40 (following mentioned as ST90E40) are EPROM and OTP members of the ST9 family of microcontrollers, in windowed ceramic (E) and plastic OTP (T) packages respectively, completely developed and produced by SGS-THOMSON Microelectronics using a n-well proprietary HCMOS process.

The EPROM ST90E40 may be used for the prototyping and pre-production phases of development, and can be configured as: a standalone microcontroller with 16K bytes of on-chip ROM, a microcontroller able to manage of external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

The nucleus of the ST90E40 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

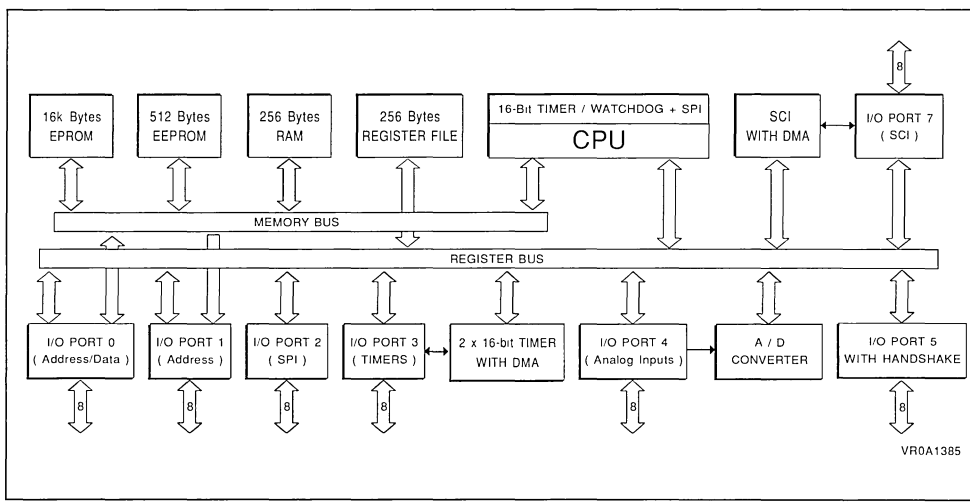
The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90E40 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer. In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375,000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90E40 Block Diagram

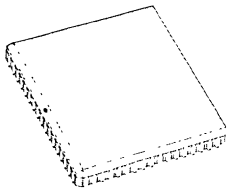


SGS-THOMSON MICROELECTRONICS

ST90R40

ROMLESS HCMOS MCU WITH EEPROM, RAM AND A/D CONVERTER

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 256 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Romless to allow maximum external memory capbility
- 512 bytes EEPROM
- 68-lead Plastic Leaded Chip Carrier package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 40 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9040 16K ROM device (also available in windowed and One Time Programmable EPROM packages)



PLCC68

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member Please contact SGS-THOMSON for further information

GENERAL DESCRIPTION

The ST90R40 is a ROMLESS member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROMLESS part may be used for the prototyping and pre-production phases of development, and offers the maximum in program flexibility in production systems.

The ROMLESS ST90R40 can be configured as a microcontroller able to manage external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

The nucleus of the ST90R40 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C-bus and IM-bus Interface, plus memory iinterface. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

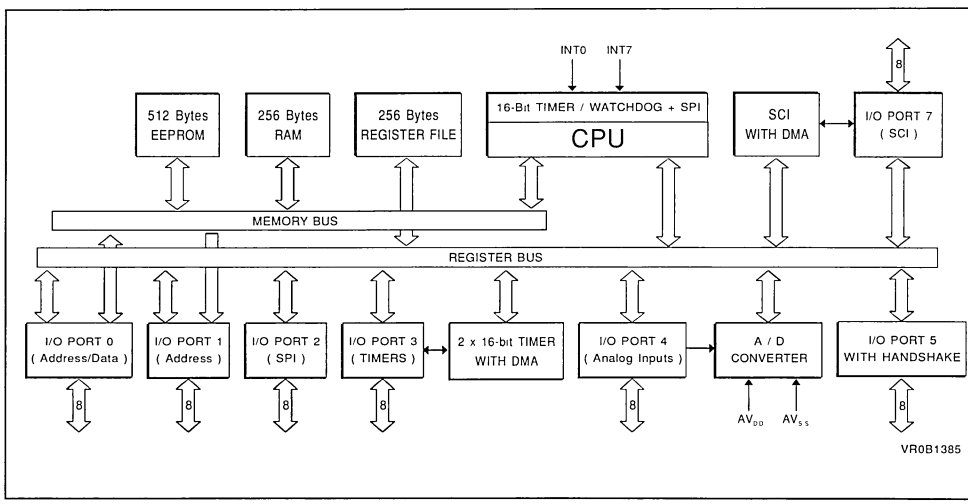
The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90R40 with up to 40 I/O lines dedicated to digital Input/Output. These lines are grouped into up to five 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing and status signals, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three memory spaces are available: Program Memory (external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer. In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375,000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90R40 Block Diagram

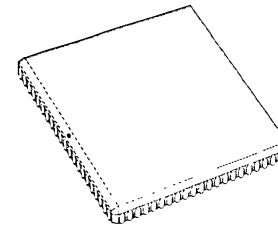


SGS-THOMSON MICROELECTRONICS

ST90R50

ROMLESS HCMOS MCU WITH BANKSWITCH AND A/D CONVERTER

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Romless to allow maximum external memory flexibility in development and production phases
- Bankswitch logic allowing a maximum addressing capability of 8Mbytes for Program and Dataspace (16Mbytes total)
- 84-pin Plastic Leaded Chip Carrier package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Three 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Two Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9054, 32K ROM device (also available in windowed and one time programmable EPROM packages)



PLCC84

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member Please contact SGS-THOMSON for further information

GENERAL DESCRIPTION

The ST90R50 is a ROMLESS member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROMLESS part may be used for the prototyping and pre-production phases of development, and offers the maximum in program flexibility in production systems with its 16M byte addressing space when using the Bankswitch memory expansion.

The nucleus of the ST90R50 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus memory interface. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

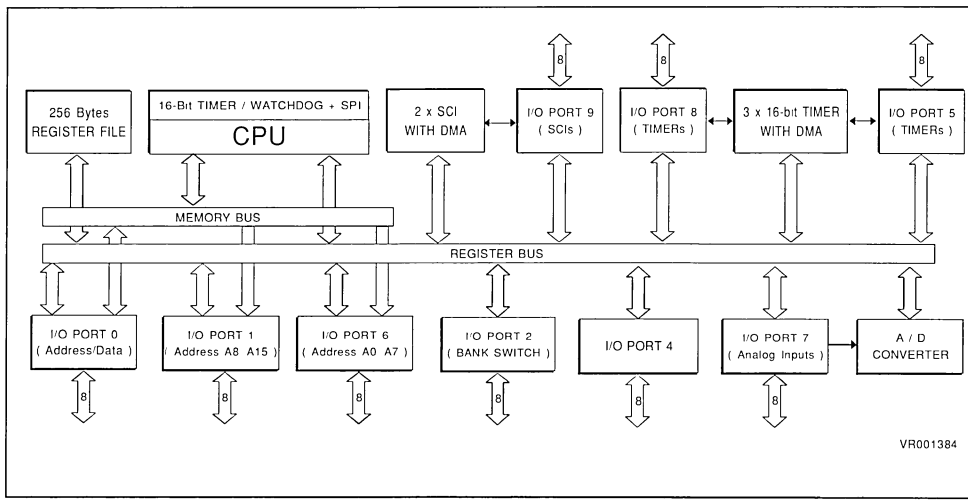
The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90R50 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to nine 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, timer inputs and outputs, external interrupts and serial or parallel I/O with or without handshake.

Three basic memory spaces are available to support this wide range of configurations: Program Memory (external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Three 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

Completing the device are two full duplex Serial Communications Interfaces, each with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90R50 Block Diagram



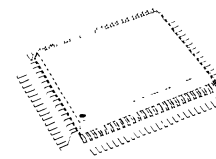
SGS-THOMSON MICROELECTRONICS

ST90R51

ROMLESS HCMOS MCU WITH BANKSWITCH

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Romless to allow maximum external memory capability
- Bankswitch logic allowing a maximum addressing capability of 8Mbytes for Program and Dataspace (16Mbytes total)
- 80-pin Plastic Quad Flat Pack package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 54 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Three 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- Two Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System



PQFP80

(Ordering Information at the end of the Datasheet)

January 1993

GENERAL DESCRIPTION

The ST90R51 is a Romless member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

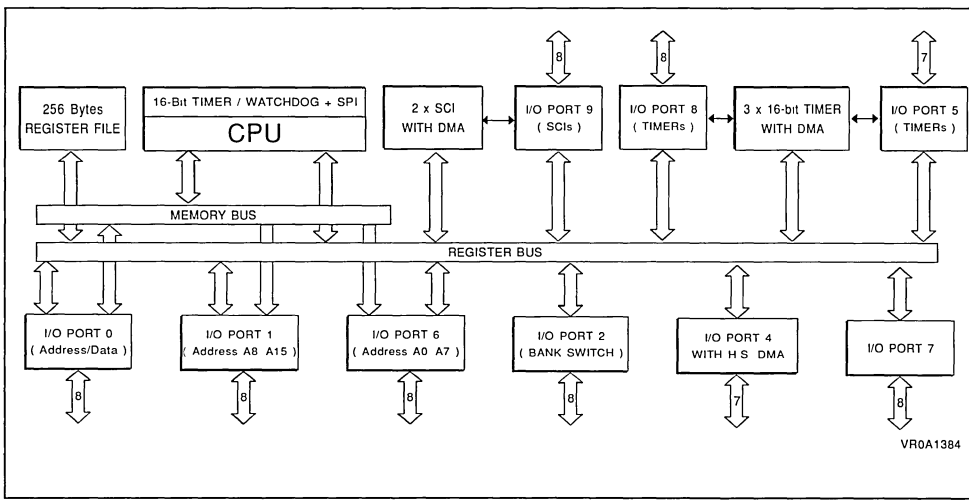
The Romless part may be used for the prototyping and pre-production phases of development, and offers the maximum in program flexibility

The nucleus of the ST90R51 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus memory interface. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set. The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90R51 with up to 54 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.
Three basic memory spaces are available to support this wide range of configurations: Program Memory , Data Memory and the internal Register File, which includes the control and status registers of the on-chip peripherals.

Three 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer. In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device are 2 full duplex Serial Communications Interfaces with an integral 110 to 375,000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90R51 Block Diagram



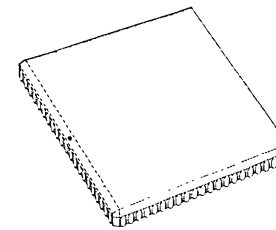
SGS-THOMSON MICROELECTRONICS

ST9054

32K ROM HCMOS MCU WITH BANKSWITCH AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 32K bytes of ROM,
  1280 bytes of RAM,
  224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Bankswitch logic allowing a maximum addressing capability of 8Mbytes for Program and Dataspace (16Mbytes total)
- 84-pin Plastic Leaded Chip Carrier package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 72 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Three 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Two Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed and One Time Programmable EPROM parts available for prototyping and pre-production development phases



PLCC84

(Ordering Information at the end of the Datasheet)

January 1993

GENERAL DESCRIPTION

The ST9054 is a ROM member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM part is fully compatible with its EPROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 32K bytes of on-chip ROM, microcontrollers able to manage external memory (16M byte with the Bankswitch logic), or as parallel processing elements in a system with other processors and peripheral controllers.

The nucleus of the ST9054 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9054 with up to 72 I/O lines dedicated to digital Input/Output. These lines are grouped into up to nine 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, address and data buses for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

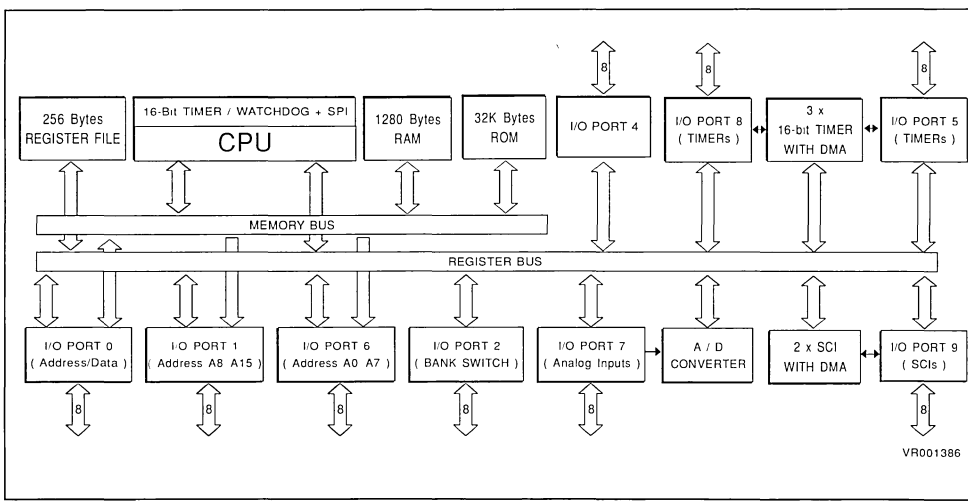
Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Three 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device are two full duplex Serial Communications Interfaces, each with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST9054 Block Diagram



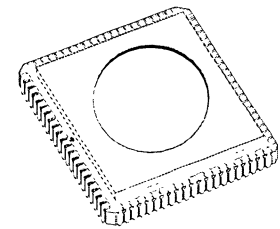
SGS-THOMSON MICROELECTRONICS

ST90E54

32K EPROM HCMOS MCU WITH BANKSWITCH AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 32K bytes of EPROM
  1280 bytes of RAM,
  224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Bankswitch logic allowing a maximum addressing capability of 8Mbytes for Program and Dataspace (16Mbytes total)
- 84-pin Window Ceramic Leaded Chip Carrier package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 72 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Three 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Two Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9054 32K ROM device



CLCC84W

(Ordering Information at the end of the Datasheet)

January 1993

GENERAL DESCRIPTION

The ST90E54 is an EPROM member of the ST9 family of microcontrollers, in windowed ceramic package , completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The EPROM ST90E54 may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 32K bytes of on-chip EPROM, microcontrollers able to manage external memory (16M byte with the Bankswitch logic), or as parallel processing elements in a system with other processors and peripheral controllers.

The nucleus of the ST90E54 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90E54 with up to 72 I/O lines dedicated to digital Input/Output. These lines are grouped into up to nine 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, address and data buses for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

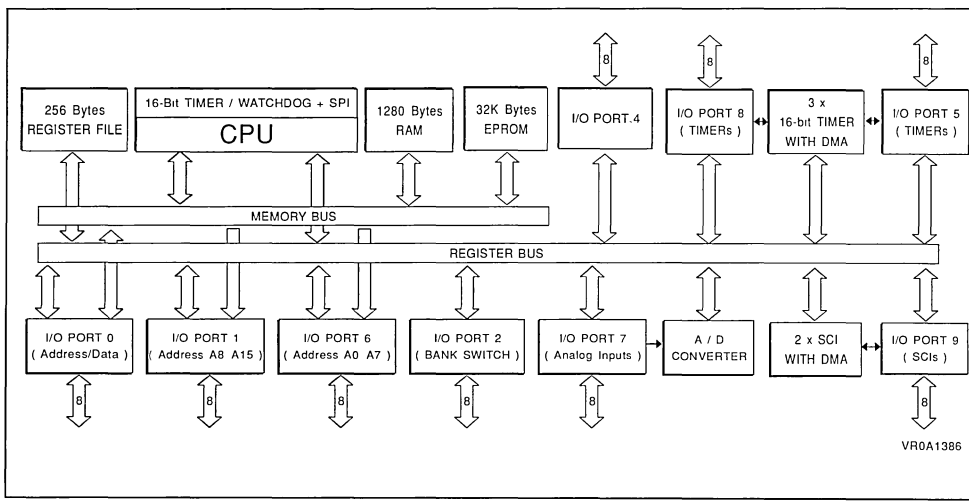
Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Three 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device are two full duplex Serial Communications Interfaces, each with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90E54 Block Diagram



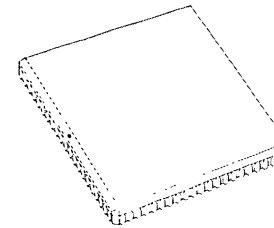
SGS-THOMSON MICROELECTRONICS

ST90R54

ROMLESS HCMOS MCU WITH BANKSWITCH AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 1280 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- Romless to allow maximum external memory flexibility in development and production phases
- Bankswitch logic allowing a maximum addressing capability of 8Mbytes for Program and Dataspace (16Mbytes total)
- 84-pin Plastic Leaded Chip Carrier package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Three 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Two Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9054, 32K ROM device (also available in windowed EPROM packages)



PLCC84

(Ordering Information at the end of the Datasheet)

January 1993

GENERAL DESCRIPTION

The ST90R54 is a ROMLESS member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROMLESS part may be used for the prototyping and pre-production phases of development, and offers the maximum in program flexibility in production systems with its 16M byte addressing space when using the Bankswitch memory expansion.

The nucleus of the ST90R54 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus memory interface. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90R54 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

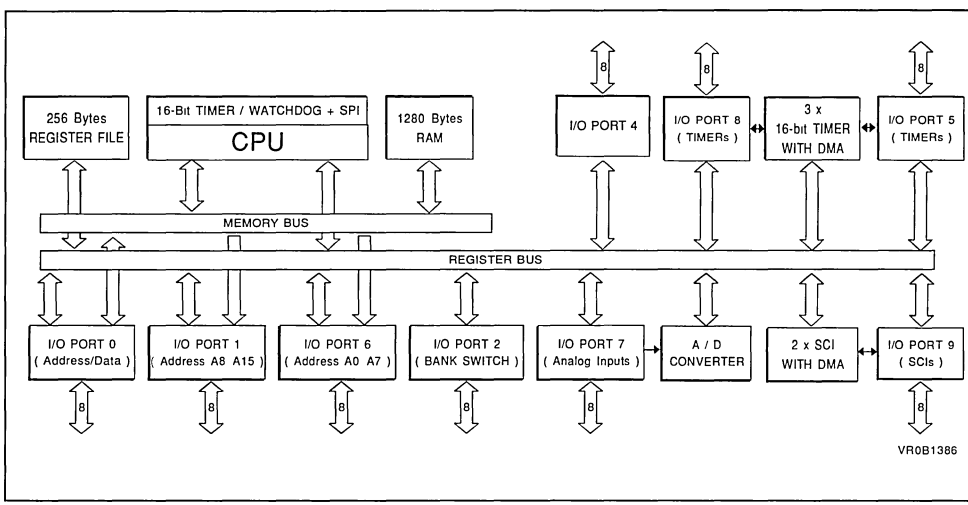
Three basic memory spaces are available to support this wide range of configurations: Program Memory (external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Three 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device are two full duplex Serial Communications Interfaces, each with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 1. ST90R54 Block Diagram



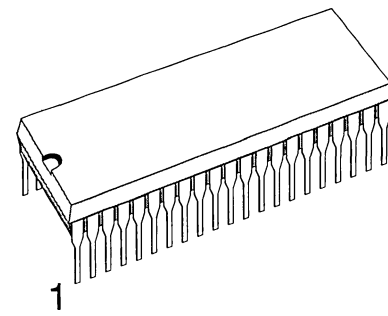
SGS-THOMSON MICROELECTRONICS

ST9292

24K ROM HCMOS MCU WITH ON SCREEN DISPLAY AND CLOSED CAPTION DATA SLICER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 24K bytes of ROM,
  384 bytes of RAM,
  224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 42-lead Shrink DIP package
- Interrupt handler and Serial Peripheral Interface as standard features
- 31 fully programmable I/O pins
- 34 character x15 rows software programmable On Screen Display module with colour, italic, underline, flash, transparent and fringe attribute options
- Digital Data Slicer extracting closed caption data from video
- 8 8-bit PWM D/A outputs with repetition frequency 2 to 32kHz and 12V Open Drain Capability
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- 16-bit programmable Slice Timer with 8-bit prescaler
- 3 channel Analog to Digital Converter, with integral sample and hold, fast 5.75µs conversion time, 6-bit guaranteed resolution
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed EPROM parts available for prototyping and pre-production development phases



PSDIP42

(Ordering Information at the end of the Datasheet)

DEVICE SUMMARY

Device	RAM	ROM
ST9292J4	384 bytes	16Kbytes
ST9292J5	384 bytes	24Kbytes

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information.

GENERAL DESCRIPTION

The ST9292 is a ROM member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM parts are fully compatible with their EPROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 24K bytes of on-chip ROM.

The nucleus of the ST9292 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16-bit Timer/Watchdog with 8-bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9292 with up to 31 I/O lines dedicated to digital Input/Output. These lines are grouped into up to six I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, address and data buses for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O.

Three basic memory spaces are available to support this wide range of configurations: Program Memory, Data Memory and the Register File, which includes the control and status registers of the on-chip peripherals.

The 16-bit Slice Timer with an 8-bit Prescaler.

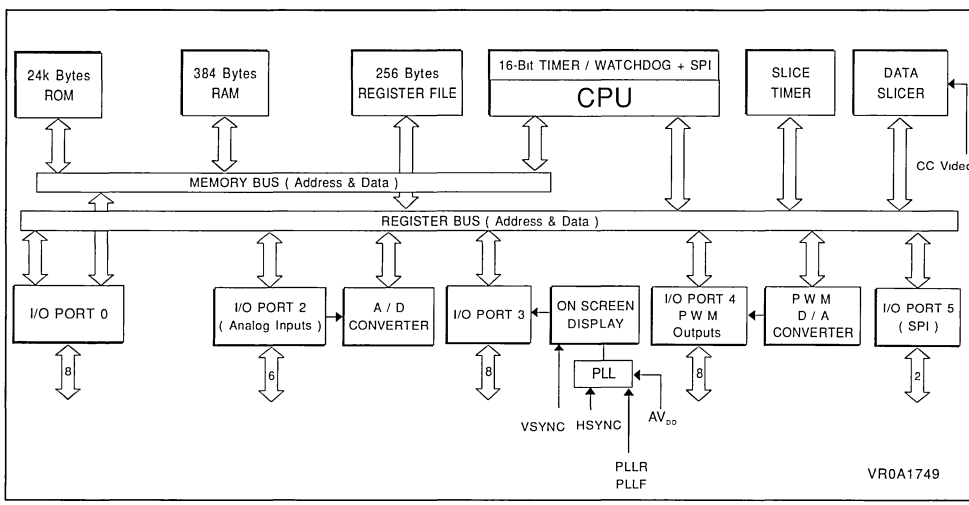
The human interface is provided by the On Screen Display module, this can produce up to 15 lines of up to 34 characters from a ROM defined 128 character set. The 9x13 character can be modified by 4 different pixel sizes, with character rounding, and formed into words with colour and format attributes.

Closed Caption control for the display of information transmitted through the video input is facilitated with the Data Slicer. This module has manual and automatic Slicing levels for both Sync and Data and allows the user to select the video line containing the data relative to the Vertical synchronisation pulse.

Control of TV settings is able to be made with up to eight 8-bit PWM outputs, with a frequency maximum of 23,437Hz at 8-bit resolution (INTCLK = 12MHz). Low resolutions with higher frequency operation can be programmed.

In addition there is a 3 channel Analog to Digital Converter with integral sample and hold, fast 5.75µs conversion time and 6-bit guaranteed precision.

Figure 1. ST9292 Block Diagram



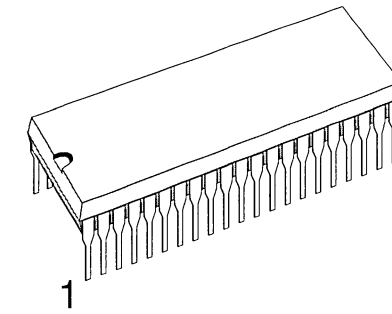
SGS-THOMSON MICROELECTRONICS

ST92E92
ST92T92

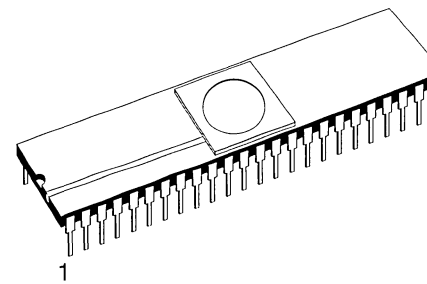
24K EPROM HCMOS MCU WITH ON SCREEN DISPLAY AND CLOSED-CAPTION DATA SLICER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 24K bytes of EPROM or OTP ROM, 384 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 42-lead Plastic Shrink DIP package for ST92T92
- 42-lead Window Ceramic Shrink DIP package for ST92E92
- Interrupt handler and Serial Peripheral Interface as standard features
- 31 fully programmable I/O pins
- 34 character x15 rows software programmable On Screen Display module with colour, italic, underline, Flash, transparent and fringe attribute options
- Digital Data Slicer extracting closed caption data from video
- 8 8-bit PWM D/A outputs with repetition frequency 2 to 32kHz and 12V Open Drain Capability
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- 16-bit programmable Slice Timer with 8-bit prescaler
- 3 channel Analog to Digital Converter, with integral sample and hold, fast 5.75µs conversion time, 6-bit guaranteed resolution
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9292 24K ROM device



PSDIP42



CSDIP42W

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member Please contact SGS-THOMSON for further information

GENERAL DESCRIPTION

The ST92E92 and ST92T92 are EPROM member of the ST9 family of microcontrollers in windowed Ceramic (E) and Plastic OTP (T) packages respectively, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The EPROM parts are fully compatible with their ROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 24K bytes of on-chip EPROM, microcontrollers able to manage up to 64K bytes of external memory.

The nucleus of the ST92E92 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16-bit Timer/Watchdog with 8-bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8-bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST92E92 with up to 31 I/O lines dedicated to digital Input/Output. These lines are grouped into up to five I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O.

Three basic memory spaces are available to support this wide range of configurations: Program Memory, Data Memory and the Register File, which includes the control and status registers of the on-chip peripherals.

The 16-bit Slice Timer with an 8-bit Prescaler.

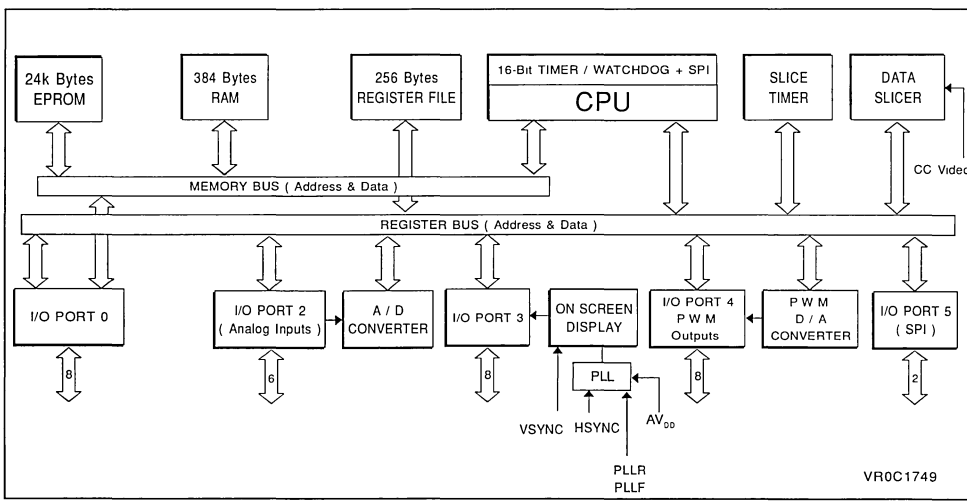
The human interface is provided by the On Screen Display module, this can produce up to 15 lines of up to 34 characters from a ROM defined 128 character set. The 9x13 character can be modified by 4 different pixel sizes, with character rounding, and formed into words with colour and format attributes.

Closed Caption control for the display of information transmitted through the video input is facilitated with the Data Slicer. This module has manual and automatic Slicing levels for both Sync and Data and allows the user to select the video line containing the data relative to the Vertical synchronisation pulse.

Control of TV settings is able to be made with up to eight 8-bit PWM outputs, with a frequency maximum of 23,437Hz at 8-bit resolution (INTCLK = 12MHz). Low resolutions with higher frequency operation can be programmed.

In addition there is a 3 channel Analog to Digital Converter with integral sample and hold, fast 5.75µs conversion time and 6-bit guaranteed resolution.

Figure 1. ST92E92 Block Diagram



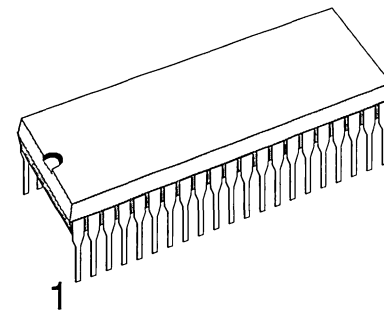
SGS-THOMSON MICROELECTRONICS

ST9293

32K ROM HCMOS MCUs WITH ON SCREEN DISPLAY AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 32K bytes of ROM, 640 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 42-lead Shrink DIP package
- Interrupt handler and Serial Peripheral Interface as standard features
- 31 fully programmable I/O pins
- 34 character x15 rows software programmable On Screen Display module with colour, italic, underline, Flash, transparent and fringe attribute options
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- 16-bit programmable Slice Timer with 8-bit prescaler
- 4 channel Analog to Digital Converter, with integral sample and hold, fast 5.5µs conversion time, 6-bit guaranteed resolution
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed EPROM parts available for prototyping and pre-production development phases



PSDIP42

(Ordering Information at the end of the Datasheet)

DEVICE SUMMARY

Device	RAM	ROM
ST9293J4	640 bytes	16Kbytes
ST9293J5	640 bytes	24Kbytes
ST9293J6	640 bytes	32Kbytes

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information.

GENERAL DESCRIPTION

The ST9293 is a ROM member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM parts are fully compatible with their EPROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 32K bytes of on-chip ROM.

The nucleus of the ST9293 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16-bit Timer/Watchdog with 8-bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8-bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set. The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9293 with up to 31 I/O lines dedicated to digital Input/Output.

These lines are grouped into up to five I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O.

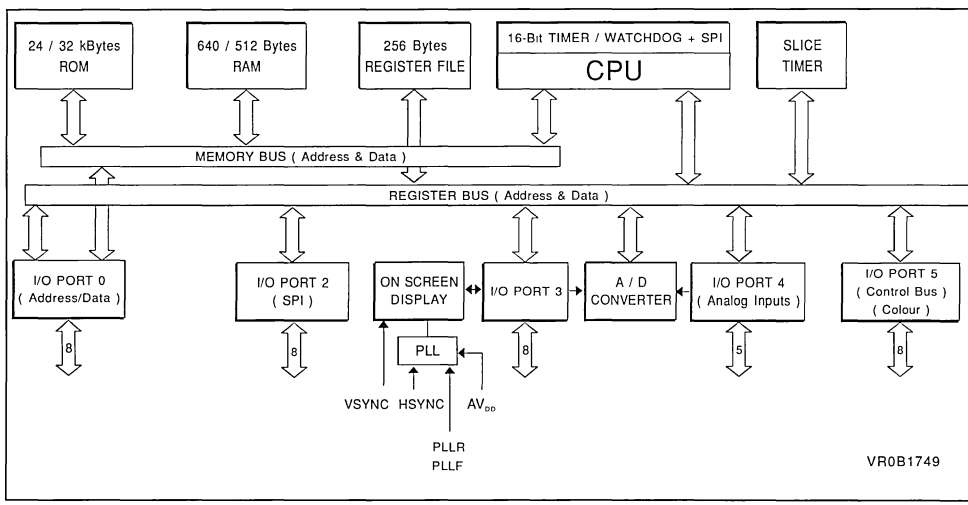
Three basic memory spaces are available to support this wide range of configurations: Program Memory, Data Memory and the Register File, which includes the control and status registers of the on-chip peripherals.

The 16-bit Slice Timer with an 8-bit Prescaler and 6 operating modes allows simple use for waveform generation and measurement, PWM functions and many other system timing functions.

The human interface is provided by the On Screen Display module, this can produce up to 15 lines of of up to 34 characters from a ROM defined 128 character set. The 9x13 character can be modified by 4 different pixel sizes, with character rounding, and formed into words with colour and format attributes.

In addition there is a 4 channel Analog to Digital Converter with integral sample and hold, fast 5.5µs conversion time and 6-bit guaranteed precision.

Figure 1. ST9293 Block Diagram



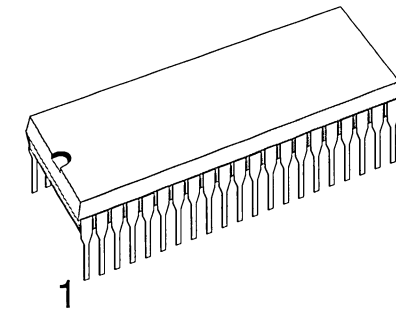
SGS-THOMSON MICROELECTRONICS

ST92E93
ST92T93

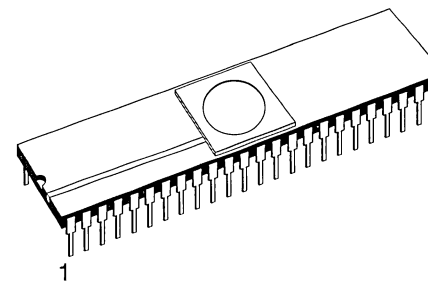
32K EPROM HCMOS MCUs WITH ON SCREEN DISPLAY AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- 32K bytes of EPROM or OTP ROM, 640 bytes of RAM, 224 general purpose registers available as RAM, accumulators or index registers (Register File)
- 42-lead Plastic Shrink DIP package for ST92T93
- 42-lead Window Ceramic Shrink DIP package for ST92E93
- Interrupt handler and Serial Peripheral Interface as standard.features
- 31 fully programmable I/O pins
- 34 character x15 rows software programmable On Screen Display module with colour, italic, underline, Flash, transparent and fringe attribute options
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- 16-bit programmable Slice Timer with 8-bit prescaler
- 4 channel Analog to Digital Converter, with integral sample and hold, fast 5.5µs conversion time, 6-bit guaranteed resolution
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating SystemCompatible with ST9293 32K ROM device



PSDIP42



CSDIP42W

(Ordering Information at the end of the Datasheet)

January 1993

This is short overview of an ST9 Family Member. Please contact SGS-THOMSON for further information.

GENERAL DESCRIPTION

The ST92E93 and ST92T93 are EPROM member of the ST9 family of microcontrollers in windowed Ceramic (E) and Plastic OTP (T) packages respectively, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The EPROM parts are fully compatible with their ROM versions, which may be used for the prototyping and pre-production phases of development, and can be configured as: standalone microcontrollers with 32K bytes of on-chip EPROM.

The nucleus the ST92E93 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16-bit Timer/Watchdog with 8-bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8-bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set. The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST92E93 with up to 31 I/O lines dedicated to digital Input/Output.

These lines are grouped into up to five I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O.

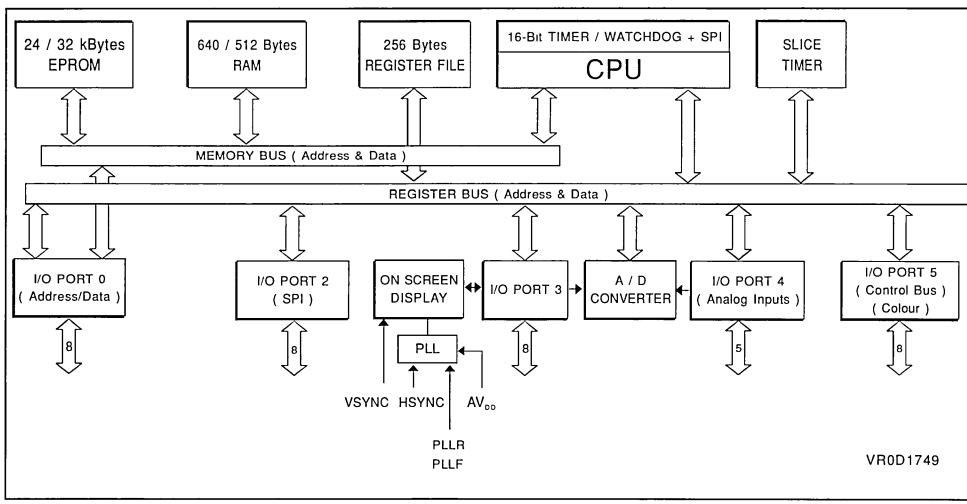
Three basic memory spaces are available to support this wide range of configurations: Program Memory, Data Memory and the Register File, which includes the control and status registers of the on-chip peripherals.

The 16-bit Slice Timer with an 8-bit Prescaler and 6 operating modes allows simple use for waveform generation and measurement, PWM functions and many other system timing functions.

The human interface is provided by the On Screen Display module, this can produce up to 15 lines of of up to 34 characters from a ROM defined 128 character set. The 9x13 character can be modified by 4 different pixel sizes, with character rounding, and formed into words with colour and format attributes.

In addition there is a 4 channel Analog to Digital Converter with integral sample and hold, fast 5.5µs conversion time and 6-bit guaranteed resolution.

Figure 1. ST92E93 Block Diagram



DEVELOPMENTS TOOLS

SGS-THOMSON MICROELECTRONICS

ST9 STARTER KIT

EVALUATION KIT FOR ST9 MCU FAMILY

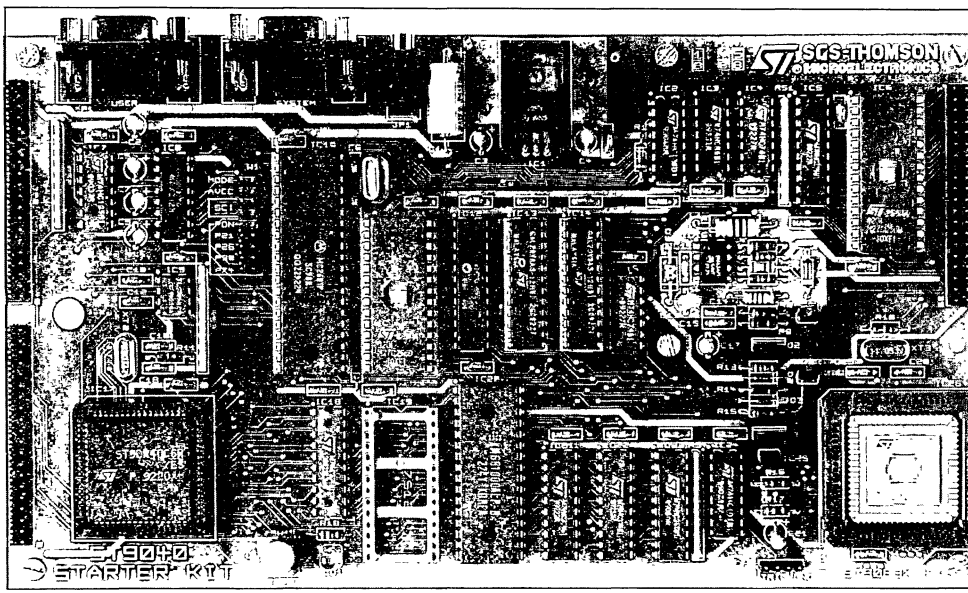
- Full Evaluation Kit for ST9 Family
- Emulation Capability
- Windowed and Command Line interfaces

GENERAL DESCRIPTION

The ST9 Starter Kit includes all the hardware, software and documentation required to evaluate the ST9 family of 8/16-bit MCUs and to develop simple applications. The ST9 Starter Kit includes ROMless (ST90R40) and EPROM-based (ST90E40) microcontrollers, the ST9 family documentation, the ST9 software tools package and an evaluation board for debugging an application and programming the ST90E40.

The board, which measures 225 x 125mm, is based on the ST90R40, which offers all the most important features of the ST9 family, including a built-in DMA controller, a Serial Peripheral Interface (supporting S-bus, I^2C-bus and IM-bus), 256 bytes of internal RAM, 512 bytes of internal EEPROM, 16-bit multifunction timers, A/D converter and a full duplex serial communication interface. On-board memories provide storage for emulated programs and data, the monitor program and breakpoint information.

For maximum flexibility, the board can run in three different modes. In the stand-alone mode, up to 64Kbytes of program space and 64Kbytes of data space are available. In the emulation mode, the board is driven by a monitor program allowing the use of all registers and memories, while single step, software trace and breakpoints are supported on both program and data spaces through debugging software running on a PC host. The third mode is the programming mode, which allows debugged software to be downloaded to an ST90E40 using the ZIF socket provided.



SOFTWARE TOOLS

The software tools include a full macro-assembler which supports modular programming, an incremental linker, an archiver that manages relocatable objects modules, a functional simulator, a windowed debugger which drives the ST9 evaluation board and the EPROM programming software. The fully symbolic debugger allows access to all ST9 resources (memories, registers) while the windowed and menu-driven interface, on-line help and intuitive access to commands make it very easy to use.

The ST9 Starter Kit also includes full documentation on the ST9040 Family, on how to connect and program it and the software tools manuals which describe how to use the ST9 development tools included in the starter kit, as well as a floppy disk containing several application programs for ST9 devices.

Starter Kit Command Line Summary

COMMAND	DESCRIPTION
ARchive	Archive symbols and macros
ASm	On-line Assembler
BAse	Change base of numbers
BYE	Exit from debugger program
CLOSE	Close I/O Channel
CLS	Clear screen
CM	Compare memory
DEfine	Define symbols and macros
DIsasm	On-Line disassembler
DM	Display memory
DO	Execute macro
DR	Display register
DUmp	Save current setup
ENDFOR	End for loop (see FOR)
ENDIF	End conditional block (see IF)
FM	Fill memory with pattern
FOR	Loop command execution
FR	Fill registers with pattern
GO	Execute program
Help	On-Line help
IF	Conditional command execution
JUMP	Go to label
LIstsymbol	List symbols and macros
LOad	Load program from file
LOCATE	Set cursor position at given coords
MAP	Set/Display mapping
MB	Modify memory breakpoints settings
MM	Move memory block
NEXT	Execute program steps

Starter Kit Command Line Summary (Continued)

OPEN	Open I/O channel
PAUSE	Pause for number of seconds
Print	Print strings and values
Quit	Return to Graphical Interface
REset	Reset emulated CPU
SAve	Save memory contents to file
SB	Set/Display memory breakpoints
SEarch	Search for pattern in memory
SET	Set/Reset emulator options
SM	Set memory
SR	Set/Display registers
SYstem	Exit temporarily to operating system
Trace	Display trace
UNdefine	Remove symbols
USE	Execute command file
VER	Print version information
WAtch	Display/Create watch data
WR	Display current working register set
<value>	Evaluate expression
!	Execute single system command
?	Display symbols having a given value
:	Comment line for macros/command files

ST9-GNU TOOLCHAIN

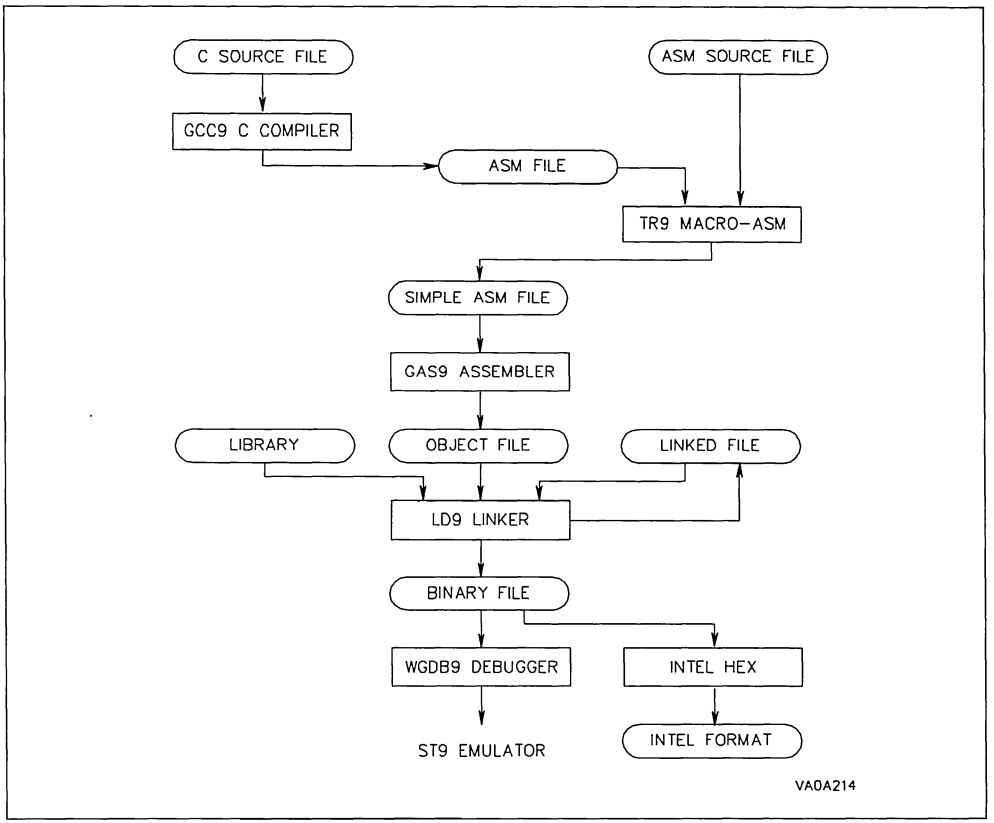
C COMPILER, ASSEMBLER, LINKER AND SOURCE DEBUGGER FOR ST9 MCU FAMILY

CONTENTS

- Optimised C compiler with options for different standards: traditional C, ANSI C, and GNU extensions
- Macro-assembler with powerful pre-processor
- Linker/loader
- Source-level debugger with Microsoft WINDOWS 3™ graphic interface
- Available for SUN 4 (SPARC STATION) under the UNIX system

GENERAL DESCRIPTION

The GNU Toolchain offers the software developer a full set of resources for the development of code for the ST9 microcontroller. This is achieved through the optimised GNU C Compiler, the Macro-assembler, Linker/Loader and Library Archiver. The Assembler is fully compatible at source level with previous version of the ST9 Assembler. Program debugging is made easier with the C Language Source Level Debugger, which runs under MSDOS or Microsoft Windows 3™ .



GNU C COMPILER

- All standard types allowed (char, int, short, long, signed or unsigned, float, and double) with Float types respecting the IEEE 754 standard
- Libraries delivered include string handling, conversion, I/O routines and mathematics
- Direct access to the Register File of the ST9, allowing access to all registers and on-chip peripherals
- Allows inclusion of assembly language instructions, with access to C program symbols
- Options to generate code for one or two memory spaces, one or two stacks and interrupt routines
- Optimisation phase included at final stage

General Description

The GNU C Compiler for the ST9 allows the programmer to write C source code (using traditional C (Kernigan & Richie), ANSI C, or GNU Extensions) and to produce assembly language source code. When used with the Assembler and Linker, it allows the generation of executable object code for all members of the ST9 family.

The generated assembly source file may include interleaved C lines and assembly language lines, and provides information for source-level debugging.

ASSEMBLER

The Assembler pre-processor allows macro substitution, file inclusion and conditional assembly and is fully compatible at source level with the ST9 assembler (AST9) pseudo-instructions and pseudo-macros.

Source level debugging information is generated with the object file by the assembler.

Assembly language programs are fully mixable with C language programs and accept 3 sections (TEXT, DATA, BSS).

LINKER

- Combines object code files issued by the assembler
- Supports incremental linking

General Description

The Linker resolves references to external symbols and searches libraries for necessary modules to produce an output file in a binary format, downloadable by the debugger to the ST9 emulator.

A map file is generated, including all mapping information on sections, files, and symbols and separate files are produced to support ST9 bank switch mechanism

The three sections generated by the C compiler and used by the assembler are accepted.

Options are available for setting the base addresses of sections and stacks and management of two spaces with script files to define memory mapping.

DEBUGGER

- Runs on MS-DOS based PC, with or without WINDOWS 3™
- Connected by serial line to the ST9 hardware emulator
- Offers a WINDOWS 3™ - based graphic interface, supporting all standard features
- Mouse supports access to context sensitive help
- Offers a line mode command interface (able to run on MS-DOS or within a WINDOWS 3™ DOS box) supporting command files
- Includes a window for access to the low-level SDBST9 debugger
- Dumps ST9 memories, system registers, Register File and paged registers

General Description

The ST9 Debugger allows source level debug for C language and assembly language programs, even with optimized C language programs.

The debugger is able to generate trace information, with hardware information interleaved with source lines, and to display the local symbols of the current C procedure and the stack based on the C language source level.

Source lines are displayed, with or without disassembly of memory interleaved with the source lines with symbols under their real types.

Requires 386 class PC or higher with at least 4 Mbytes of memory, under MS-DOS 4.01 or higher.

UTILITIES

- Archiver
- Formatter of INTEL HEX industrial format, allowing download of program to an EPROM programmer
- Binary file deformatter

Note: Windows 3 is a trademark of the Microsoft Corporation.

SGS-THOMSON MICROELECTRONICS

ST9-EPB

EPROM PROGRAMMING BOARD FOR ST9 MCU FAMILY

- Programming tool for EPROM and OTP members of the ST9 Microcontroller Family
- Stand-alone operation mode
- Device EPROM capacity self-identified
- 3 functions performed
- All device packages type supported
- Single Power Supply

GENERAL DESCRIPTION

This board is designed for programming the EPROM versions of the ST9 microcontroller family, including both the ceramic windowed and plastic OTP packages.

The EPROM size of the ST9 microcontroller to be programmed is recognised automatically by the onboard software and three sockets are provided to accept the different existing packages types.

The ST9-EPB board uses a reference EPROM including the customer's code directly generated by the ST9 assembler-linker. The ST9 EPROM device will be programmed from the contents of the reference EPROM. Jumpers allow the selection of different types of reference EPROM (2732, 2764, 27128, 27256).

On board regulation requires only a single power supply of +18 V_{DC} – 0.5A to produce the different voltages necessary for the board functioning.

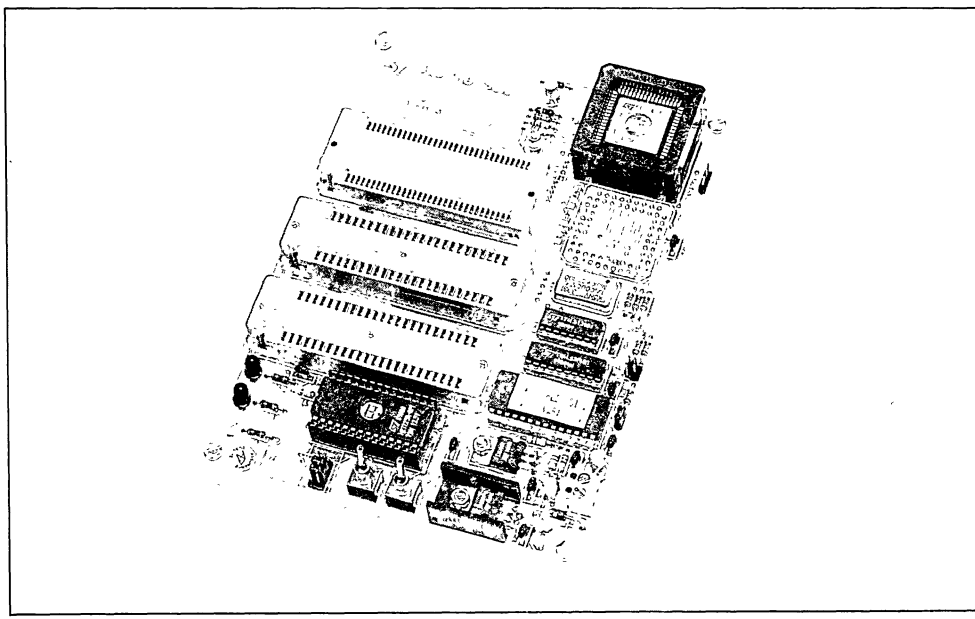
The board can perform 3 operations:

- Verifying the blank state of the microcontroller EPROM.
- Programming microcontroller with the content of the reference EPROM.
- Verifying the microcontroller against the reference EPROM.

The required function is selected by two switches.

During the running procedure the program/verify LED flashes and at the end of operation, the result is displayed on LEDs:

- Green OK LED for succesful operation.
- Red error LED for a programming error.



SGS-THOMSON MICROELECTRONICS

ST9-HDS

HARDWARE DEVELOPMENT SYSTEM FOR ST9 MCU FAMILY

- Designed to communicate with any IBM PC/XT/AT or compatible computer through an RS-232 serial communication link.
- Emulation capability of all present and future members of the ST9 Microcontroller family, ROM and ROMless devices by dedicated option boards
- 128K bytes of system mappable fast static memory, which may be mapped in pages of 512 bytes each
- 4 maskable hardware controlled memory breakpoints and 2 maskable hardware controlled register breakpoints
- Real time trace memory (2048 events)
- Programmable crystal oscillator and external clock option allow an emulated CPU clock frequency ranging from 2 to 24 MHz
- Automatic hardware self test executed every time the emulator is powered on
- 8 User Probes available and included in the trace and breakpoint logic
- The Emulator may be used in Standalone Mode without a Personal Computer control

GENERAL DESCRIPTION

The ST9 Hardware Development System (ST9-HDS) is an intelligent and powerful In Circuit Real Time Emulation System configurable for all current and future members of the ST9 family of microcontrollers. The complete ST9-HDS consists of the emulator, an RS232 Serial Communication Cable providing the interface with an optional Host computer, an ICE Probe and Adapter which may be plugged directly into the user's application, a set of 8 user probes, and a powerful software debugger. The ST9-HDS allows the designer to emulate the system in real time or single step mode. A set of 4 user programmable memory breakpoints which may be logically combined in AND, OR, SEQUENTIAL, or DELAY mode and 2 user programmable register breakpoints allow the user to stop emulation upon very specific conditions, while trace circuitry will collect the latest 2K (by 40 bit) events. Furthermore, a wide range of debug commands provides the user with full control of the Emulator hardware and features several commands for controlling the execution of programs. Memory and registers may be read and written in a number of different formats, while macro commands and conditional block constructs are available for use in automated debugging sessions.

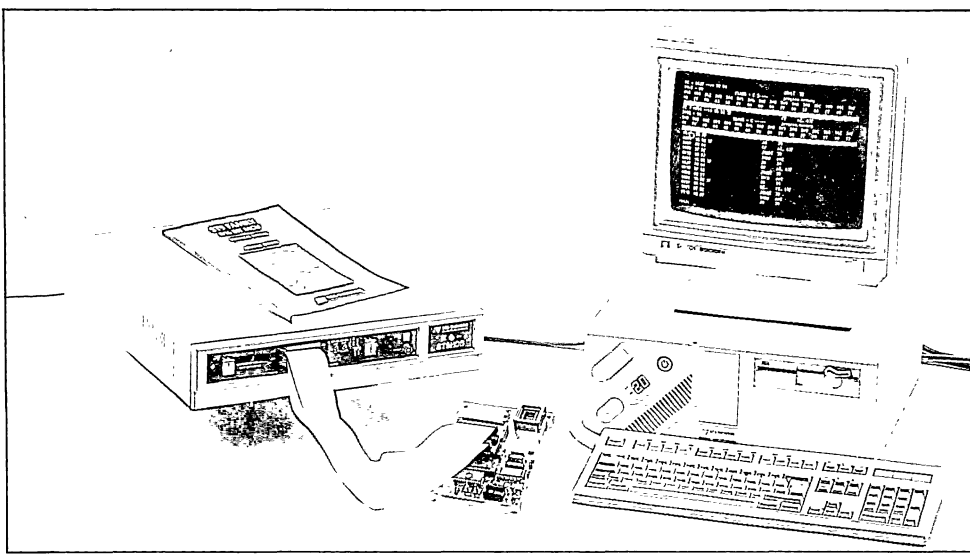
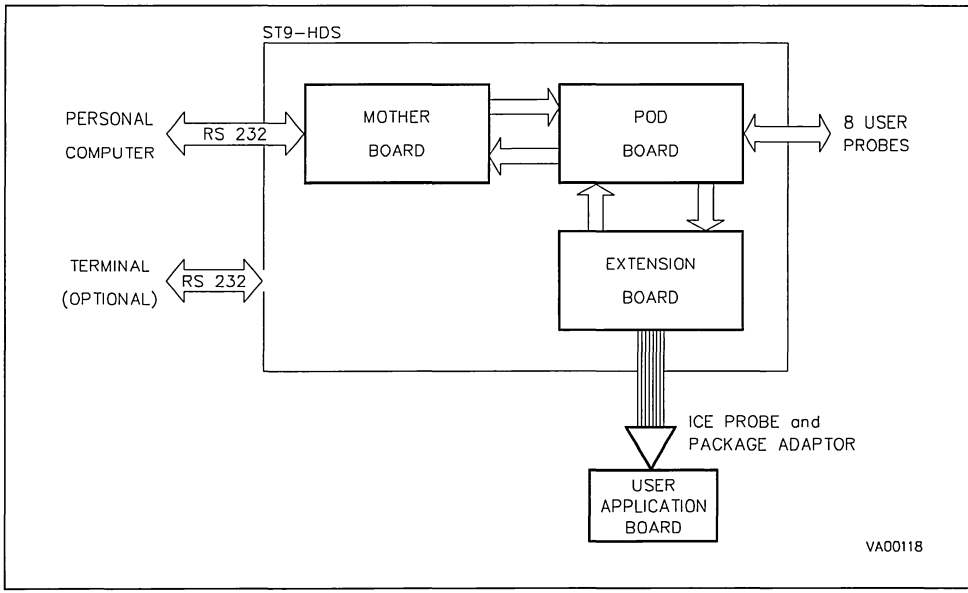


Figure 1. ST9-HDS System Configuration



HARDWARE DESCRIPTION

The Interface and Control Unit (ICU) contains most of the circuitry necessary to control the ST9-HDS, with the exception of the circuits specifically related to the ST9 family of microcontrollers. The ICU provides the control logic for monitoring the execution of programs, setting memory breakpoints, recording signal events, and handling the communication with the host computer. The board contains the following hardware resources:

- Private Microcontroller: A private microcontroller controls the operation of the emulator, allowing execution of an emulated program to run without interference.
- 128K of Fast Static Memory: 128 Kbytes of fast static memory are available for use by the emulated microcontroller.
- Memory Management Circuitry: Memory may be mapped in groups of 512 bytes as Internal/External to the Emulated ST9 or, System/User Supplied, and ReadOnly/ReadWrite, or as Non Existent. (Certain versions also allow memory to be mapped as EEPROM memory.)
- Memory Breakpoints: 4 Hardware controlled memory breakpoints are available to the user and may be combined in AND, OR, SEQUENTIAL, or DELAY mode. Each of the four breakpoints is associated with a breakpoint counter which may be used simply as a counter or used to flag an event only after the associated event has occurred n times.
- Real Time Trace Circuitry: Real Time Trace Circuitry keeps track of a 2K by 40 bit buffer.

The ST9 Emulator POD contains the core of the circuits required to emulate any member of the ST9 family of microcontrollers. This board is responsible for providing the interface with the Extension Board, providing the interface with the Interface Control Unit, sending out signals on reset identifying the emulated device as a ROM-less or ROM-maskable device, managing the opcode fetch signals, generating the clock for the Extension Board, controlling the standalone mode option, decoding the pod addresses, managing the register breakpoints, controlling the idle/run logic, generating wait cycles, and accessing the 8 user probes. The Pod Board contains the following features:

- Programmable Crystal Oscillator: An on board programmable crystal oscillator, as well as the possibility of using an external clock via a BNC connector, allow the user the option of selecting the emulated CPU clock frequency.
- Standalone Logic: The Hardware Development System may be operated in Standalone Mode, that is independently of the Host computer.
- Wait Cycle Generator: A Wait Cycle Generator allows the user to assign from 0 to 7 wait cycles to any block of 512 memory bytes defined as external.

HARDWARE DESCRIPTION (Continued)

- Register Breakpoints: 2 Hardware controlled register breakpoints are available to the user and may be combined in AND or OR mode.

The Extension Board contains the circuitry controlling all the special functions and peripherals of the ST9 being emulated. Since each version of the ST9 has different peripherals and access to different I/O ports, the Extension Board will be different for each version of the ST9.

However, the basic design of the Extension Board will remain the same, allowing the extension board to be easily configured for any future or existing version of the ST9. In general, the Extension Board contains the components and circuitry which emulate the ST9 (Core, I/O ports, and peripherals), interfaces with the ICE Connectors, and sends information to the Pod Board. Either a 220V/50Hz or 110V/60Hz Power Supply is included in the emulator to provide the emulator with all necessary power.

SOFTWARE DEBUGGING PACKAGE

The ST9 Symbolic Debugger is a software tool which allows the user to have complete control of the ST9 Hardware Development System. The Debugger must be used on an IBM PC/XT/AT or compatible that is connected to the ST9-HDS by means of an RS-232 serial communication cable. The following features are provided by the ST9 Symbolic Debugger:

- Debugger Compatibility: The debugger has a command line syntax compatible with the SIMST9 Software Simulator and SDBST9, the debugger for the ST9 Evaluation Module.
- Commands: A wide range of commands are available for displaying and setting memory and registers according to different formats.
- Powerful Symbol Handler: A Powerful Symbol Handler allows the user to define symbols and macros, extract them from symbol table files, and save them in symbol table files.
- Symbolic On-Line Assembler/Disassembler: The debugger provides a symbolic on-line assembler and disassembler.
- Full Screen Video Mode: Full screen video modes are available for Memory, Register, and Single Step Display.
- Symbolic Trace: Trace memory is disassembled into assembler mnemonics.
- Macros and Conditional Block Constructs: Macro commands and conditional block constructs are available for use in automated debugging sessions.
- On-Line Help: An On-line help facility is available in the debugger to give a listing of the complete command set as well as specific information on any of the commands.
- Configuration and Documentation: Log, dump and command file capability allows for easy documentation and configuration.
- Powerful Command Interpreter: A powerful command interpreter allows for the evaluation of complex expressions involving numbers, addresses, memory and register contents, and I/O channel data.

The ST9 Symbolic Debugger accepts inputs from the Software Development Package which includes the following:

- ST9 Macro Assembler (AST9)
- ST9 Linker/Loader (LST9)
- ST9 Library Archiver (ARST9)
- ST9 Software Simulator (SIMST9)

The Software Development Package is available separately, or with the Hardware Development System.

Figure 2. SDBST9 Command Summary

ARCHIVE	Archive symbols and macros
ASM	On-line assembler
BASE	Change base of numbers
BYE	Exit from debugger program
CLOSE	Close I/O channel
CM	Compare memory
DEFINE	Define symbols and macros
DISASM	On-line symbolic disassembler
DM	Display memory
DO	Execute macro
DR	Display register
DUMP	Save current setup
FM	Fill memory
FR	Fill registers
GO	Execute program
HELP	On-line help
IF	Conditional command execution
LISTSYMBOL	List symbols and macros
LOAD	Load program/data from file
MAP	Set/display memory mapping
MB	Modify breakpoint
MM	Move block of memory
MRB	Modify register breakpoint
NEXT	Execute program steps
OPEN	Open I/O channel
PRINT	Print strings and values
QUIT	Terminate command execution
RESET	Reset emulated CPU
SAVE	Save program/data into file
SB	Set/display memory breakpoints
SEARCH	Search a pattern in memory
SET	Set/reset options
SM	Set memory
SR	Set/display registers
SRB	Set/display register breakpoints
TRACE	Display trace
UNDEFINE	Remove symbols
USE	Execute commands from a file
VE	View execution (video mode)
VM	View memory (video mode)
VR	View registers (video mode)
WR	Display current working register set
<value>	Evaluate expression
?	Display symbols having a given value

SOFTWARE DEVELOPMENT TOOLS FOR ST9 MCU FAMILY

- ST9 Macro Assembler
- ST9 Linker/Loader
- ST9 Library Archiver
- ST9 Software Simulation

GENERAL DESCRIPTION

Full software development is achieved using the ST9 Software Development Tools. This follows for the optional C Compiler, through the High Level Macro Assembler, Linker/Loader, Library Archiver and Software Simulator.

ST9 Macro Assembler

The ST9 Macro Assembler accepts one or more source files written in ST9 assembly language and transforms them into linkable object files. The assembler recognizes the use of symbols, macros, pseudo-instructions, pseudo-macros, and conditional assembly directives.

ST9 Linker/Loader

The ST9 Linker/Loader combines a number of object files into a single program, associating an absolute address to each section of code, and resolving any external references. LST9 may be used to generate: a binary or hexadecimal output module, a map file, and an object file.

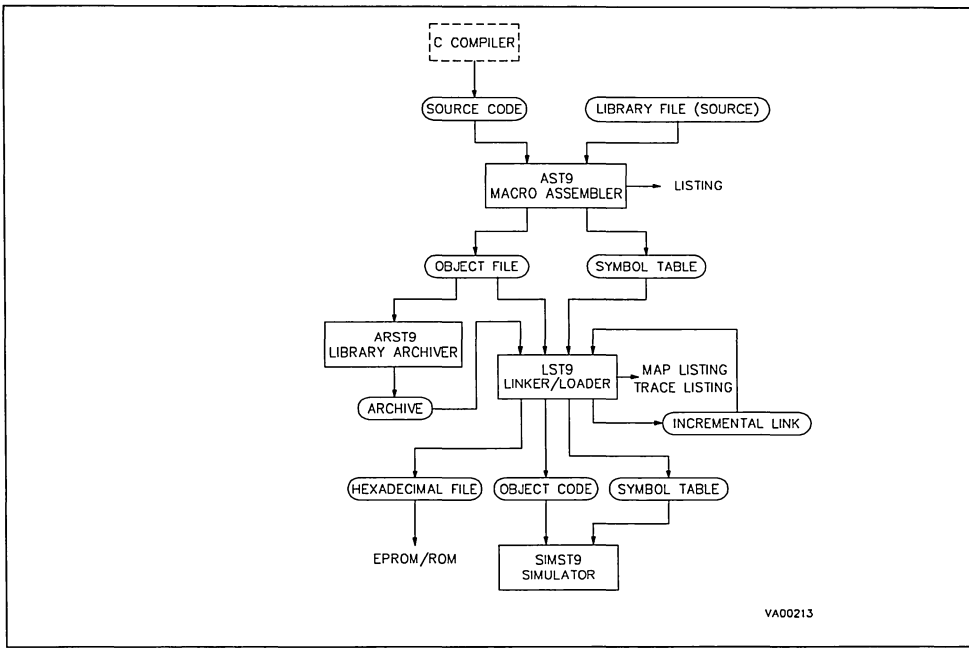
ST9 Library Archiver

The ST9 Library Archiver maintains libraries of software object files, allowing the user to develop standard modules for repetitive use.

ST9 Software Simulator

The ST9 Software Simulator allows the user to debug and execute any program written for any member of the ST9 family of microncontrollers without the aid of additional hardware. The simulator functionally duplicates the operation of the ST9 and completely supports the instruction set.

Figure 1. Development Flow Chart



AST9 - ST9 MACRO ASSEMBLER

- Accepts one or more source files written in ST9 assembly language and produces an object file, a listing file, an alphabetical symbol table, and error diagnostics
- Resulting object files are linkable and relocatable
- Supports program segmenting directives
- Recognizes user defined macros, macro libraries, Conditional assembly directives, pseudo-instructions, and pseudo-macros
- Supports indirect command files

General Description

The ST9 Macro Assembler (AST9) accepts one or more source files written in ST9 assembly language and transforms them into linkable object files. Modules written in assembly language are much easier to write, read, and debug than the equivalent machine code. Furthermore, the assemblers use of symbols, macros, pseudo-instructions, pseudo-macros, and conditional assembly directives, allows for even easier program development.

Figure 2. AST9 Pseudo-Instructions

```
.ascii      stores a string as a sequence of ascii codes
.asciz      same as above followed by a null character
.blkb       allocate bytes of data storage
.blkw       allocate words of data storage
.bss        defines segment as type bss (uninitialized data)
.byte       stores successive bytes of data
.data       defines segment as type data
.defstr     defines a string identifier
.endc       end of a conditional assembly block
.endm       end of a macro
.error      user defined assembly error
.extern     defines symbols as external
.global     defines specified symbols as global
.ifc        beginning of a conditional assembly block
.library    add files to macro-library file list
.list       enables listing of specified fields
.macro      defines a macro
.mcall      specifies which macros must be called from library
.mnarg      assigns to a symbol the number of arguments
            defined in a macro call
.mexit      end of a macro expansion
.nlist      disables listing of specified fields
.org        set current location counter
.page       start a new listing page
.pl         set listing page length
.sbttl      assign subtitle to current section
.text       define segment of type text
.title      assign title to the document
.word       store successive words of data
```

AST9 - ST9 MACRO ASSEMBLER (Continued)

Figure 3. AST9 Pseudo Macros

```
jxcc symbol
if [conditional expression] {macrobody}
if [cond expr] {macrobody} else {macro2}
while [cond expr] {macro}
do {macro} while [cond expr]
loop [loopvar] {macro}
switch [cp] {
        case cp1: macro
        case cp2: macro
        default: macro
}
break
begin [arg1,arg2,...] {macro}
proc procname [arg1,arg2,...] {macro}
return
```

LST9 - ST9 LINKER/LOADER

- Links modules generated by the ST9 Macro Assembler (AST9) encourages modular programming
- Supports indirect command files
- Supports 3 sections (text, data, and bss) which may be relocated and loaded at different addresses. Allows the user to specify the mapping of object files into different pages (supports 8Mbyte addressing of the ST9050).
- Extensive symbol manipulation. produces alphabetically or numerically sorted symbol tables for addresses, registers, or specifically for SIMST9 and SDBST9, strips the symbol table of local symbols, global symbols, or both, allows definition and tracing of symbols.
- Produces binary or hexadecimal output modules
- Generates a map file
- Supports incremental linking
- Resolves references to external symbols and searches libraries for necessary modules
- Provides self explanatory error and warning messages.
- Displays the version number and information about the various phases of linking

General Description

The ST9 Linker/Loader (LST9) is responsible for combining a number of object files into a single program, associating an absolute address to each section of code, and resolving any external references.

LST9 can be used to create either a binary or hexadecimal output module to be used by the ST9. The linker/loader will also produce a map file of the resulting object which gives information about the registers, pages, modules, and labels, or an object file which may be used as an input to another call to the linker.

This software program allows the user to develop modular programs, which may then be combined and addressed as defined by the user. Program modularity allows for easier design and testing, as well as promotes re-use of standard modules.

ARST9 - ST9 LIBRARY ARCHIVER

- Edits libraries by adding, deleting, moving, or replacing files
- Prints a listing of the names of all files in a library, or the table of contents for each file in a library
- Prints a file contained in a library, or extracts it for use without modifying the library
- Libraries may be called by LST9 to resolve external references.

General Description

The ST9 Library Archiver (ARST9) maintains libraries of software object files, allowing the user to develop standard modules for repetitive use. Once a module has been inserted into a library, any application may call the module. The ST9 Linker/Loader (LST9) will only call the portions of each library that are needed to resolve any external references.

SIMST9 - ST9 SOFTWARE SIMULATOR

- Supports symbolic debugging and execution of any program written for the ST9 family of microcontrollers on an IBM PC/XT/AT or compatible computer, without the aid of additional hardware.
- Functionally duplicates the operation of the ST9 family of microcontrollers, and supports the complete instruction set.
- Host Memory may be mapped in groups of 1K byte as Read-only, Read-write, or Non Existent.
- A series of simulator status commands give the user the option of selecting the simulated CPU clock frequency, creating a log of the simulator session, tracing the executed instructions, or enabling the breakpoints and traps.
- The simulator has a command line syntax compatible with SDBST9, the debugger for the ST9-HDS Hardware Development System, and EVMST9, the debugger for the ST9-EVM Evaluation Module.
- A powerful symbol handler allows the user to define symbols and macros, extract them from symbol table files, and save them in symbol table files.
- Full screen video modes are available for Memory, Register, and Single Step Display.

SIMST9 - ST9 SOFTWARE SIMULATOR (Continued)

- An On-line help facility is available to give a listing of the complete command set as well as specific information on any of the commands.
- Dump and command file capability allow for simulator session retrieval and easy configuration.
- The simulator provides a symbolic on-line assembler and disassembler.
- A powerful command interpreter allows for the evaluation of complex expressions involving numbers, addresses, memory and register contents, and I/O channel data.
- 128 software breakpoints and 128 software traps are available to the user.
- A trace is kept during program execution which may be displayed afterwards with the traced instructions disassembled into assembler mnemonics.
- A wide range of commands are available for displaying and setting memory and registers according to different formats.
- Macro commands and conditional block constructs are available for use in automated debugging sessions.
- I/O channels can be opened for simulation of I/O peripheral functions.
- Interrupts may be defined and set pending to simulate the occurrence of an interrupt.
- A simulated clock will use the user assigned clock frequency to calculate the real time execution of a program. The clock may be displayed or changed by the user to perform time measurements.

General Description

SIMST9 allows the user to debug and execute any program written for any of the current and future members of the ST9 family of microcontrollers, without the aid of additional hardware. The simulator functionally duplicates the operation of the ST9 and completely supports the instruction set. I/O channels may be opened, read, and written, in order to simulate the I/O functions of peripherals, while interrupts may be set, and then set pending, in order to simulate the handling of interrupts. The simulator uses the clock frequency assigned by the user, along with the number of clock cycles needed by each instruction to keep track of the real time execution speed.

Figure 4. SIMST9 Command Summary

```
ARCHIVE          Archive symbols and macros
ASM              On-line assembler
BASE             Change base of numbers
BYE              Exit from simulator program
CLOSE            Close I/O channel
CM               Compare memory
DEFINE           Define symbols and macros
DEFINT           Define Interrupts
DISASM           On-line symbolic disassembler
DM               Display memory
DO               Execute macro
DR               Display register
DUMP             Save simulator status
FM               Fill memory
FR               Fill registers
GO               Execute program
HELP             On-line help
IF               Conditional command execution
INTERRUPT        Simulate interrupt
```

Figure 4. SIMST9 Command Summary (Continued)

```
LISTSYMBOL      List symbols and macros
LOAD            Load program/data from file
MAP             Set/display memory mapping
MB              Modify breakpoint
MM              Move block of memory
MT              Modify trap
NEXT            Execute program steps
OPEN            Open I/O channel
PRINT           Print strings and values
QUIT            Terminate command execution
RESET           Reset simulated CPU
RESTORE         Restore dump file
SAVE            Save program/data into file
SB              Set/display memory breakpoints
SEARCH          Search a pattern in memory
SET             Set/reset options
SM              Set memory
SR              Set/display registers
ST              Set/display traps
TIME            Set/display user clock counter
TRACE           Display trace
UNDEFINE        Remove symbols
USE             Execute commands from a file
VE              View execution (video mode)
VM              View memory (video mode)
VR              View registers (video mode)
WR              Display current working register set
<value>         Evaluate expression
?               Display symbols having a given value
```

ST9-C

ANSI C COMPILER FOR ST9 MCU FAMILY

- Upgraded KERNIGHAN AND RITCHIE C definition, respecting ANSI standard X3.159.
- Optimisation stages using artificial intelligence techniques (calculation of costs in terms of code size and execution time).
- Versions available for IBM PC or compatible under MS-DOS 3.1 and higher, SUN 3 and SUN 4 (SPARC station) under the UNIX operating system and for VAX and microVax under the VMS operating system.
- All standard types allowed (char, int, short, long, signed or unsigned).
- "Float" respecting IEEE 754 standard and "Double" types allowed.
- Many library functions implemented in assembler code for increased code and execution time efficiency e.g. string handling, conversion, I/O routines.
- Generates an assembly language source file, interleaving C lines and assembly language lines.
- Direct access to the Register File of the ST9, allowing access to all on-chip peripherals and features of the ST9.
- Extensions for Real Time Interrupt handling.
- Pre-processor included for standardisation and increased readabilty and portability.
- Available with Macro-Assembler, Linker and Symbolic Software Simulator.
- Fully compatible with the ST9 Hardware Development System supporting symbolic debug and source code high-level debugger.

GENERAL DESCRIPTION

The ST9 ANSI C Compiler allows the programmer to write C source code and produce assembly langage source programs. Used with the assembler/linker, it allows the possibility to generate object code executable for all members of the ST9 microcontroller family. The generated object code may be used for symbolic debugging with the software simulator and hardware debugger/emulator, to generate test EPROM devices for prototyping, or to produce ROM mask data. It takes into account all the advanced features of the ST9 family (interrupt, Register File access, memory pages access). The high-level language C Compiler has been designed to provide the greatest flexibility of use.

The user can either run the complete software with only one simple command, or run each step of the compiler separately: pre-processor, analyser, coder, optimizer.

The ST9 ANSI C Compiler is delivered with a standard initialisation file to be linked with the customer application. This file allows the setting of BSS and DATA sections and stack pointers, as well as peripheral startup code.

STANDARD

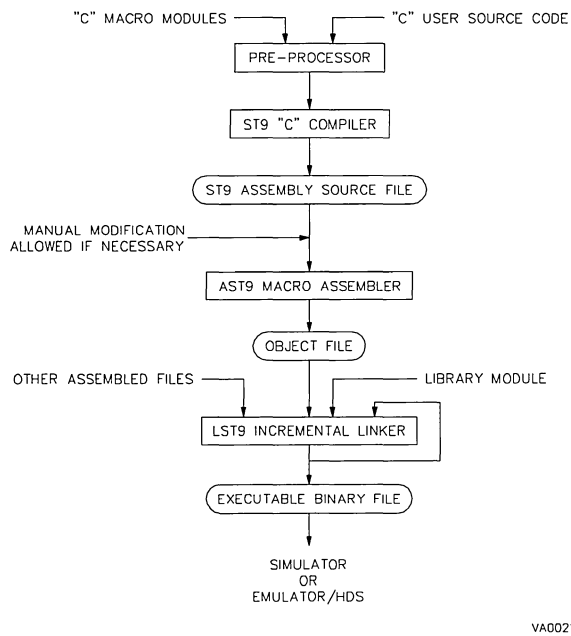
The ST9 ANSI C Compiler is an implementation of the X3.159 ANSI standard (issued from X3J11 draft proposal), which includes and exceeds the Kernighan and Ritchie specification. For example : "CONST" and "VOLATILE" qualifiers and function prototyping.

The ST9 ANSI C Compiler implements the features most often needed by microcontroller developments: interrupt handling, Register File access, far function declarations.

LICENSE

The ST9 ANSI C Compiler is delivered under license for one user only. Upgrading of new releases will be made to each registered user, free of charge, for a duration of 12 months starting from the date of the return of the Registration Card.

Figure 1. ST9-C Flow Chart



DATASHEETS

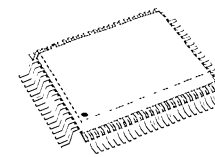
SGS-THOMSON MICROELECTRONICS

ST9040

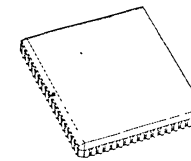
16K ROM HCMOS MCU WITH EEPROM, RAM AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time : 500ns (12MHz internal)
- Internal Memory :
  ROM 16K bytes
  RAM 256 bytes
  EEPROM 512 bytes
  224 general purpose registers available as RAM, accumulators or index registers (register file)
- 80-pin PQFP package for ST9040Q
- 68-lead PLCC package for ST9040C
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- Up to 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Windowed and One Time Programmable EPROM parts available for prototyping and pre-production development phases
- Upward compatible with ST9030, ST9032 and ST9036



PQFP80



PLCC68

(Ordering Information at the end of the Datasheet)

January 1993

Figure 1. 80 Pin PQFP Package

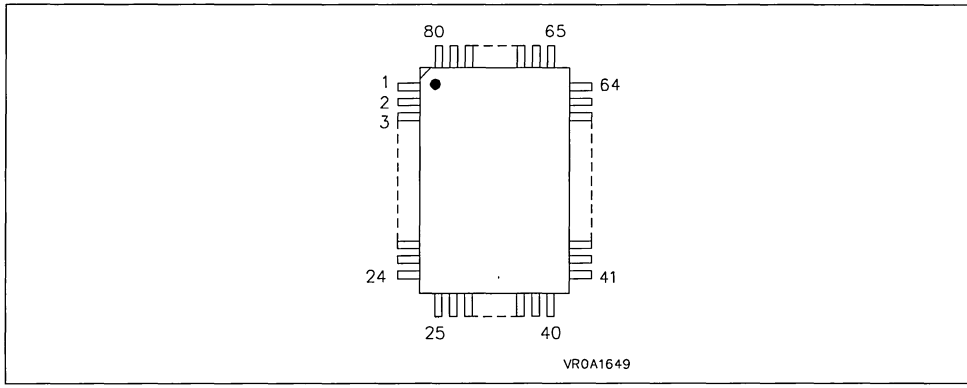


Table 1. ST9040Q Pin Description

Pin	Name
1	AV_{SS}
2	AV_{SS}
3	NC
4	P44/Ain
5	P57
6	P56
7	P55
8	P54
9	INT7
10	INT0
11	P53
12	NC
13	P52
14	P51
15	P50
16	OSCOUT
17	V_{SS}
18	V_{SS}
19	NC
20	OSCIN
21	$\overline{RESET}$
22	P37/T1OUTB
23	P36/T1INB
24	P35/T1OUTA

Pin	Name
25	P34/T1INA
26	P33/T0OUTB
27	P32/T0INB
28	P31/T0OUTA
29	P30/P/$\overline{D}$/T0INA
30	P17/A15
31	P16/A14
32	NC
33	P15/A13
34	P14/A12
35	P13/A11
36	P12/A10
37	P11/A9
38	P10/A8
39	P00/A0/D0
40	P01/A1/D1

Pin	Name
64	P20/NMI
63	NC
62	V_{SS}
61	P70/SIN
60	P71/SOUT
59	P72/INT4/TXCLK /CLKOUT
58	P73/INT5 /RXCLK/ADTRG
57	P74/P/$\overline{D}$/INT6
56	P75/$\overline{WAIT}$
55	P76/WDOUT /$\overline{BUSREQ}$
54	P77/WDIN /$\overline{BUSACK}$
53	R/$\overline{W}$
52	NC
51	$\overline{DS}$
50	$\overline{AS}$
49	NC
48	V_{DD}
47	V_{DD}
46	P07/A7/D7
45	P06/A6/D6
44	P05/A5/D5
43	P04/A4/D4
42	P03/A3/D3
41	P02/A2/D2

Pin	Name
80	AV_{DD}
79	NC
78	P47/Ain7
77	P46/Ain6
76	P45/Ain5
75	P43/Ain3
74	P42/Ain2
73	P41/Ain1
72	P40/Ain0
71	P27/RRDY5
70	P26/INT3 /$\overline{RDSTB5}$/P/$\overline{D}$
69	P25/WRRDY5
68	P24/INT1 /$\overline{WRSTB5}$
67	P23/SDO
66	P22/INT2/SCK
65	P21/SDI/P/$\overline{D}$

Figure 2. 68 Pin PLCC Package

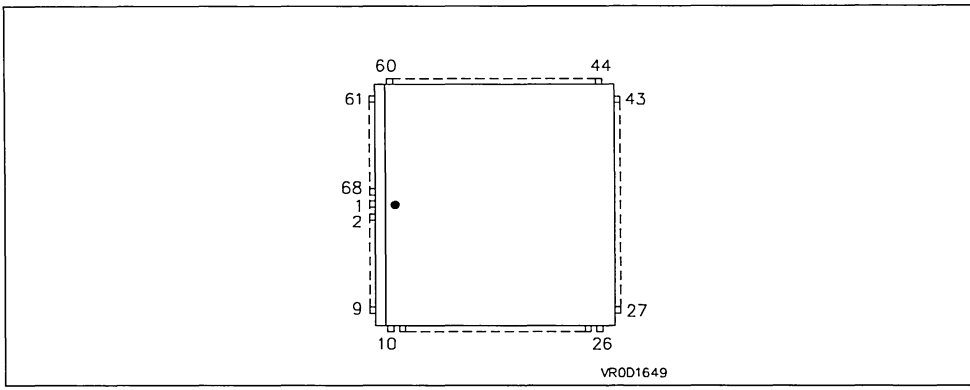


Table 2. ST9040C Pin Description

Pin	Name
61	P44/Ain4
62	P57
63	P56
64	P55
65	P54
66	INT7
67	INT0
68	P53
λ 1	P52
2	P51
3	P50
4	OSCOUT
5	V_{SS}
6	OSCIN
7	$\overline{RESET}$
8	P37/T1OUTB
9	P36/T1INB

Pin	Name
10	P35/T1OUTA
11	P34/T1INA
12	P33/T0OUTB
13	P32/T0INB
14	P31/T0OUTA
15	P30/P/$\overline{D}$/T0INA
16	P17/A15
17	P16/A14
18	P15/A13
19	P14/A12
20	P13/A11
21	P12/A10
22	P11/A9
23	P10/A8
24	P00/A0/D0
25	P01/A1/D1
26	P02/A2/D2

Pin	Name
43	P70/SIN
42	P71/SOUT
41	P72/CLKOUT /TXCLK/INT4
40	P73/ADTRG /RXCLK/INT5
39	P74/P/$\overline{D}$/INT6
38	$\overline{P75/WAIT}$
37	P76/WDOUT /$\overline{BUSREQ}$
36	P77/WDIN /BUSACK
35	R/W
34	DS
33	$\overline{AS}$
32	V_{DD}
31	P07/A7/D7
30	P06/A6/D6
29	P05/A5/D5
28	P04/A4/D4
27	P03/A3/D3

Pin	Name
60	AV_{SS}
59	AV_{DD}
58	P47/Ain7
57	P46/Ain6
56	P45/Ain5
55	P43/Ain3
54	P42/Ain2
53	P41/Ain1
52	P40/Ain0
51	P27/RRDY5
50	P26/INT3 /RDSTB5/P/D
49	P25/WRRDY5
48	P24/INT1 /WRSTB5
47	P23/SDO
46	P22/INT2/SCK
45	P21/SDI/P/$\overline{D}$
44	P20/NMI

1.1 GENERAL DESCRIPTION

The ST9040 is a ROM member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROM device is fully compatible with its EPROM version, which may be used for the prototyping and pre-production phases of development, and can be configured as: a standalone microcontroller with 16K bytes of on-chip ROM, a microcontroller able to manage up to 112K bytes of external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

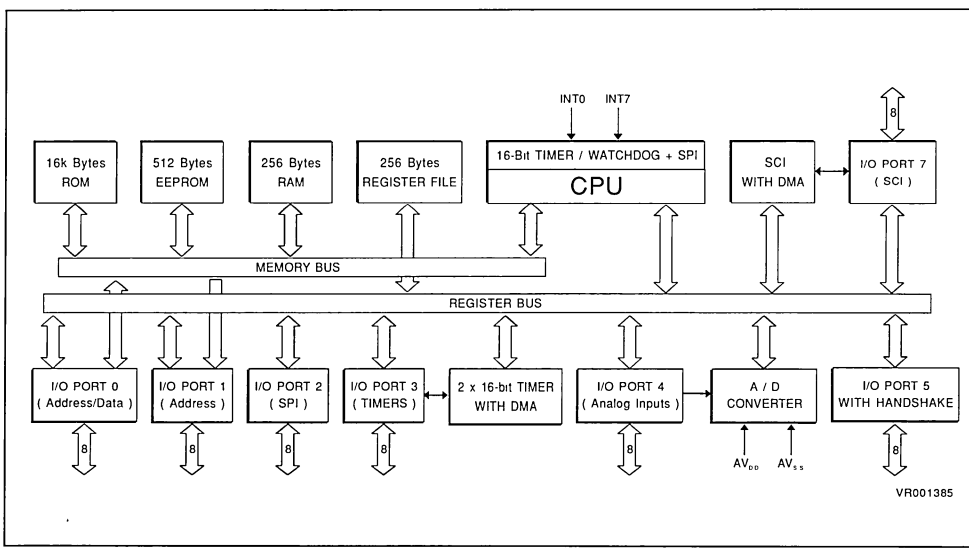
The nucleus of the ST9040 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set. The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST9040 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer. In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375,000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

Figure 3. ST9040 Block Diagram



1.2 PIN DESCRIPTION

$\overline{AS}$. *Address Strobe (output, active low, 3-state).* Address Strobe is pulsed low once at the beginning of each memory cycle. The rising edge of $\overline{AS}$ indicates that address, Read/Write (R/$\overline{W}$), and Data Memory signals are valid for program or data memory transfers. Under program control, $\overline{AS}$ can be placed in a high-impedance state along with Port 0 and Port 1, Data Strobe ($\overline{DS}$) and R/$\overline{W}$.

$\overline{DS}$. *Data Strobe (output, active low, 3-state).* Data Strobe provides the timing for data movement to or from Port 0 for each memory transfer. During a write cycle, data out is valid at the leading edge of $\overline{DS}$. During a read cycle, Data In must be valid prior to the trailing edge of $\overline{DS}$. When the ST9040 accesses on-chip memory, $\overline{DS}$ is held high during the whole memory cycle. It can be placed in a high impedance state along with Port 0, Port 1, $\overline{AS}$ and R/$\overline{W}$.

R/$\overline{W}$. *Read/Write (output, 3-state).* Read/Write determines the direction of data transfer for external memory transactions. R/$\overline{W}$ is low when writing to external program or data memory, and high for all other transactions. It can be placed in a high impedance state along with Port 0, Port 1, $\overline{AS}$ and $\overline{DS}$.

$\overline{RESET}$. *Reset (input, active low).* The ST9 is initialised by the Reset signal. With the deactivation of $\overline{RESET}$, program execution begins from the Program memory location pointed to by the vector contained in program memory locations 00h and 01h.

OSCIN, OSCOUT. *Oscillator (input and output).* These pins connect a parallel-resonant crystal (24MHz maximum), or an external source to the on-chip clock oscillator and buffer. OSCIN is the input of the oscillator inverter and internal clock generator; OSCOUT is the output of the oscillator inverter.

AV_{DD}. Analog V_{DD} of the Analog to Digital Converter.

AV_{SS}. Analog V_{SS} of the Analog to Digital Converter. *Must be tied to V_{SS}.*

V_{DD}. Main Power Supply Voltage (5V ± 10%)

V_{SS}. Digital Circuit Ground.

P0.0-P0.7, P1.0-P1.7, P2.0-P2.7 P3.0-P3.7, P4.0-P4.7, P5.0-P5.7, P7.0-P7.7 *I/O Port Lines (Input/Output, TTL or CMOS compatible).* 56 lines grouped into I/O ports of 8 bits, bit programmable under program control as general purpose I/O or as alternate functions.

1.2.1 I/O Port Alternate Functions

Each pin of the I/O ports of the ST9040 may assume software programmable Alternative Functions as shown in the Pin Configuration Drawings. Table 1-3 shows the Functions allocated to each I/O Port pins and a summary of packages for which they are available.

PIN DESCRIPTION (Continued)

Table 3. ST9040 I/O Port Alternate Function Summary

I/O PORT	Name	Function	Alternate Function	Pin Assignment	
Port. bit				PLCC	PQFP
P0.0	A0/D0	I/O	Address/Data bit 0 mux	24	39
P0.1	A1/D1	I/O	Address/Data bit 1 mux	25	40
P0.2	A2/D2	I/O	Address/Data bit 2 mux	26	41
P0.3	A3/D3	I/O	Address/Data bit 3 mux	27	42
P0.4	A4/D4	I/O	Address/Data bit 4 mux	28	43
P0.5	A5/D5	I/O	Address/Data bit 5 mux	29	44
P0.6	A6/D6	I/O	Address/Data bit 6 mux	30	45
P0.7	A7/D7	I/O	Address/Data bit 7 mux	31	46
P1.0	A8	O	Address bit 8	23	38
P1.1	A9	O	Address bit 9	22	37
P1.2	A10	O	Address bit 10	21	36
P1.3	A11	O	Address bit 11	20	35
P1.4	A12	O	Address bit 12	19	34
P1.5	A13	O	Address bit 13	18	33
P1.6	A14	O	Address bit 14	17	31
P1.7	A15	O	Address bit 15	16	30
P2.0	NMI	I	Non-Maskable Interrupt	44	64
P2.0	ROMless	I	ROMless Select (Mask option)	44	64
P2.1	$P/\overline{D}$	O	Program/Data Space Select	45	65
P2.1	SDI	I	SPI Serial Data Out	45	65
P2.2	INT2	I	External Interrupt 2	46	66
P2.2	SCK	O	SPI Serial Clock	46	66
P2.3	SDO	O	SPI Serial Data In	47	67
P2.4	INT1	I	External Interrupt 1	48	68
P2.4	$\overline{WRSTB5}$	I	Handshake Write Strobe P5	48	68
P2.5	WRRDY5	O	Handshake Write Ready P5	49	69
P2.6	INT3	I	External Interrupt 3	50	70
P2.6	$\overline{RDSTB5}$	I	Handshake Read Strobe P5	50	70
P2.6	$P/\overline{D}$	O	Program/Data Space Select	50	70
P2.7	RDRDY5	O	Handshake Read Ready P5	51	71
P3.0	T0INA	I	MF Timer 0 Input A	15	29
P3.0	$P/\overline{D}$	O	Program/Data Space Select	15	29
P3.1	T0OUTA	O	MF Timer 0 Output A	14	28
P3.2	T0INB	I	MF Timer 0 Input B	13	27
P3.3	T0OUTB	O	MF Timer 0 Output B	12	26
P3.4	T1INA	I	MF Timer 1 Input A	11	25

PIN DESCRIPTION (Continued)

Table 3. ST9040 I/O Port Alternate Function Summary(Continued)

I/O PORT	Name	Function	Alternate Function	Pin Assignment	
Port. bit				PLCC	PQFP
P3.5	T1OUTA	O	MF Timer 1 Output A	10	24
P3.6	T1INB	I	MF Timer 1 Input B	9	23
P3.7	T1OUTB	O	MF Timer 1 Output B	8	22
P4.0	Ain0	I	A/D Analog Input 0	52	72
P4.1	Ain1	I	A/D Analog Input 1	53	73
P4.2	Ain2	I	A/D Analog Input 2	54	74
P4.3	Ain3	I	A/D Analog Input 3	55	75
P4.4	Ain4	I	A/D Analog Input 4	61	4
P4.5	Ain5	I	A/D Analog Input 5	56	76
P4.6	Ain6	I	A/D Analog Input 6	57	77
P4.7	Ain7	I	A/D Analog Input 7	58	78
P5.0		I/O	I/O Handshake Port 5	3	15
P5.1		I/O	I/O Handshake Port 5	2	14
P5.2		I/O	I/O Handshake Port 5	1	13
P5.3		I/O	I/O Handshake Port 5	68	11
P5.4		I/O	I/O Handshake Port 5	65	8
P5.5		I/O	I/O Handshake Port 5	64	7
P5.6		I/O	I/O Handshake Port 5	63	6
P5.7		I/O	I/O Handshake Port 5	62	5
P7.0	SIN	I	SCI Serial Input	43	61
P7.1	SOUT	O	SCI Serial Output	42	60
P7.1	ROMless	I	ROMless Select (Mask option)	42	60
P7.2	INT4	I	External Interrupt 4	41	59
P7.2	TXCLK	I	SCI Transmit Clock Input	41	59
P7.2	CLKOUT	O	SCI Byte Sync Clock Output	41	59
P7.3	INT5	I	External Interrupt 5	40	58
P7.3	RXCLK	I	SCI Receive Clock Input	40	58
P7.3	ADTRG	I	A/D Conversion Trigger	40	58
P7.4	INT6	I	External Interrupt 6	39	57
P7.4	$P/\overline{D}$	O	Program/Data Space Select	39	57
P7.5	$\overline{WAIT}$	I	External Wait Input	38	56
P7.6	WDOUT	O	T/WD Output	37	55
P7.6	$\overline{BUSREQ}$	I	External Bus Request	37	55
P7.7	WDIN	I	T/WD Input	36	54
P7.7	$\overline{BUSACK}$	O	External Bus Acknowledge	36	54

2 CORE ARCHITECTURE

2.1 CORE ARCHITECTURE

The Core or Central Processing Unit (CPU) of the ST9 includes the 8 bit Arithmetic Logic Unit and the 16 bit Program Counter, System and User Stack Pointers. The microcoded Instruction Set is highly optimised for both byte (8 bit) and word (16 bit) data, BCD and Boolean data types, with 14 addressing modes.
Three independent buses are controlled by the Core, a 16 bit Memory bus, an 8 bit Register addressing bus and a 6 bit Interrupt/DMA bus connected to the interrupt and DMA controllers in the on-chip peripherals and the Core. This multiple bus architecture allows a high degree of pipelining and parallel operation, giving the ST9 its efficiency in both numerical calculations and communication with the on-chip peripherals.

2.2 ADDRESS SPACES

The ST9 has three separate address spaces:

- Register File: 240 8-bit registers plus up to 64 pages of 16 bytes each, located in the on-chip peripherals.
- Data memory with up to 64K (65536) bytes
- Program memory with up to 64K (65536) bytes

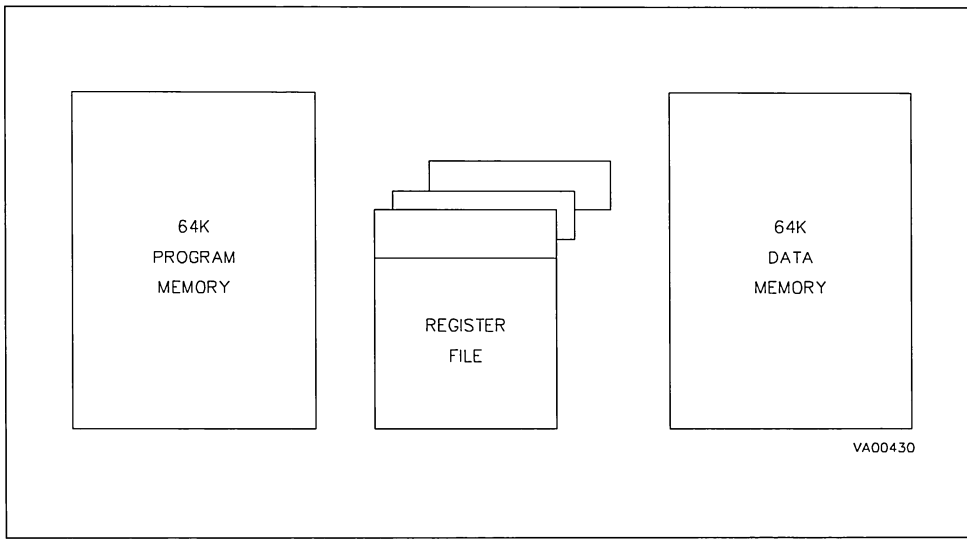
The Data and Program memory spaces will be addressed in further detail in the next section.

2.2.1 Register File

The Register File consists of:

- 224 general purpose registers R0 to R223
- 16 system registers in the System Group (R224 to R239).
- I/O pages depending on the configuration of the ST9, each containing up to 16 registers, with paging facilities based on the top group (R240 to R255).

Figure 2-1. Address Spaces



ADDRESS SPACES (Continued)

Figure 2-2. Register Grouping

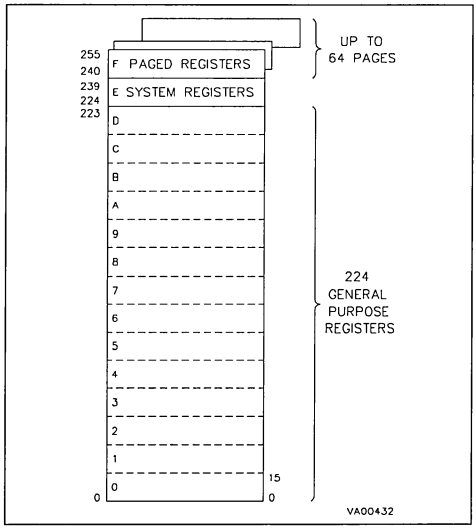


Figure 2-3. Page Pointer Configuration

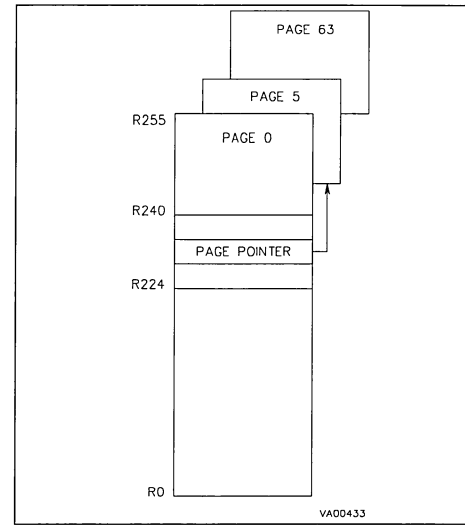
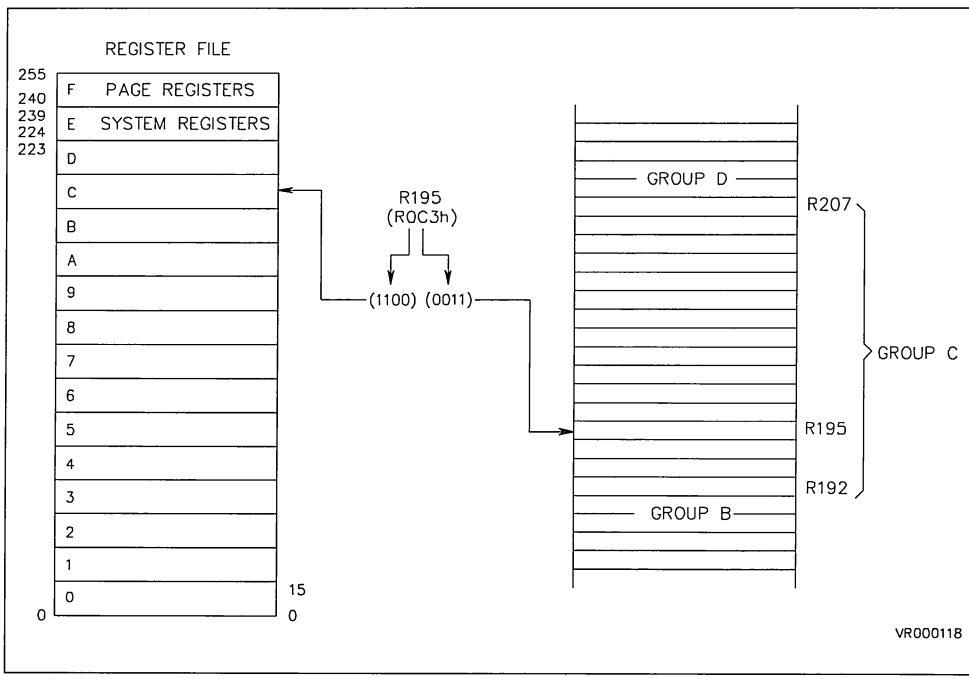


Figure 2-4. Addressing the Register File



ADDRESS SPACES (Continued)

2.2.2 Addressing Registers

All registers in the Register File and pages can be specified by using a decimal, hex or binary address, e.g. R231, RE7h or R11100111b is the same register.
The registers can be referred to by their hexadecimal group address, so that registers R0-R15 form group 0, R160-R175 form group A and so on.

Working Register Addresses

The 8-bit register address is formed by 2 nibbles, for example, for register R195 or RC3h or R11000011, 1100 specifies the 13th group (i.e. group C) and 0011 specifies the 3rd register in that group.

Working registers are addressed by supplying the least significant nibble in the instruction and adding it to the most significant nibble found in the Register Pointer (R233). Working register addressing is shown in Figures 2-4.

System Registers

The 16 system registers at addresses R224 to R239 form Group E.
The system registers are addressable using any of the 4 register addressing modes and the most significant nibble will, in all cases, be 14 (0Eh).

Paged Registers

There are a maximum of 64 pages each containing 16 registers. These are addressed using the register addressing modes with the addition of the Page Pointer register, R234. This register selects the page to be addressed in group F and once set, does not need to be changed if two or more registers on the same page are to be addressed in succession.

Therefore if the Page Pointer, R234, is set to 5, the instructions

```
spp 5
ld R242, r4
```

will load the contents of working register r4 into the third register (R242) of page 5.

These paged registers hold data and control registers related to the on-chip peripherals, and thus the configuration depends upon the peripheral organisation of each ST9 family member. i.e. pages only exist if the peripheral exists.

Available pages are shown in Table 2-2.

2.2.3 Input/Output Ports

The Input/Output ports are located in two areas. The port registers for Ports 0-5 are located at the bottom of the System register group in locations R224 to R229, while port 6 and 7 are located in page three, in registers 251 and 255 respectively.

Each Port has three associated Control registers, which determine the individual pin modes (I/O, Open-Drain etc). These registers are located in pages 2 and 3 .

Table 2-1. Register File Organization

Hex. Address	Decimal Address	Function	Register File Group
F0-FF	240-255	Paged Registers	Group F
E0-EF	224-239	System Registers	Group E
D0-DF	208-223	General Purpose Registers	Group D
C0-CF	192-207		Group C
B0-BF	176-191		Group B
A0-AF	160-175		Group A
90-9F	144-159		Group 9
80-8F	128-143		Group 8
70-7F	112-127		Group 7
60-6F	96-111		Group 6
50-5F	80-95		Group 5
40-4F	64-79		Group 4
30-3F	48-63		Group 3
20-2F	32-47		Group 2
10-1F	16-31		Group 1
00-0F	00-15		Group 0

ADDRESS SPACES (Continued)

Table 2-2. Group F Peripheral Organization

Applicable for ST9040

DEC	DEC HEX	00 00	02 02	03 03	08 08	09 09	10 0A	24 18	63 3F	
R255	RFF	RESERVED	RESERVED	PORT 7	MFT 1	RESERVED	MFT 0	RESERVED	A/D	RFF
R254	RFE	MSPI	PORT 3							RFE
R253	RFD									RFD
R252	RFC	WCR								RFC
R251	RFB	T/WD	RESERVED	RESERVED						RFB
R250	RFA		PORT 2							RFA
R249	RF9									RF9
R248	RF8					MFT				RF8
R247	RF7	EXT INT	RESERVED	PORT 5		MFT 1				RF7
R246	RF6		PORT1							RF6
R245	RF5									RF5
R244	RF4									RF4
R243	RF3		RESERVED	RESERVED		MFT0				RF3
R242	RF2		PORT 0	PORT 4						RF2
R241	RF1	EEPROMCR								RF1
R240	RF0	RESERVED								RF0

2.3 SYSTEM REGISTERS

Following is the description of System Registers. For PORT0 to PORT5 Registers, please refer to I/O Port Chapter.

Figure 2-5. System Registers

R239 (EFh)	SYS. STACK POINTER LOW
R238 (EEh)	SYS. STACK POINTER HIGH
R237 (EDh)	USER STACK POINTER LOW
R236 (ECh)	USER STACK POINTER HIGH
R235 (EBh)	MODE REGISTER
R234 (EAh)	PAGE POINTER
R233 (E9h)	REGISTER POINTER 1
R232 (E8h)	REGISTER POINTER 0
R231 (E7h)	FLAGS
R230 (E6h)	CENTRAL INT. CNTL REG
R229 (E5h)	PORT5
R228 (E4h)	PORT4
R227 (E3h)	PORT3
R226 (E2h)	PORT2
R225 (E1h)	PORT1
R224 (E0h)	PORT0

2.3.1 Central Interrupt Control Register

This Register CICR is located in the system Register Group at the address R230 (E6h). Please refer to "INTERRUPT" and "DMA" chapters in order to get the background of the ST9 interrupt philosophy.

CICR R230 (E6h) Sytem Read/Write
Central Interrupt Control Register

Reset Value : 1000 0111

7							0
GCEN	TLIP	TLI	IEN	IAM	CPL2	CPL1	CPL0

b7 = **GCEN:** *Global Counter Enable.* This bit is the Global Counter Enable of the Multifunction Timers. The GCEN bit is ANDed with the CE (Counter Enable) bit of the Timer Control Register (explained in the Timer chapter) in order to enable the Timers when both bits are set. This bit is set after the Reset cycle.

b6 = **TLIP:** *Top Level Interrupt Pending.* This bit is automatically set when a Top Level Interrupt Request is recognized. This bit can also be set by Software in order to simulate a Top Level Interrupt Request.

b5 = **TLI:** *Top Level Interrrupt bit.* When this bit is set, a Top Level interrupt request is acknowledged depending on the IEN bit and the TLNM bit (in Nested Interrupt Control Register). If the TLM bit is reset the top level interrupt acknowledgement depends on the TLNM alone.

b4 = **IEN:** *Enable Interrupt.* This bit, (when set), allows interrupts to be accepted. When reset no interrupts other than the NMI can be acknowledged. It is cleared by interrupt acknowledgement for concurrent mode and set by interrupt return (`iret`). It can be managed by hardware and software (`ei` and `di` instruction).

b3 = **IAM:** *Interrupt Arbitration Mode.* This bit covers the selection of the two arbitration modes, the Concurrent Mode being indicated by the value "0" and the Fully Automatic Nested Mode by the value "1". This bit is under software control.

b2-b0 = **CPL2-CPL0:** *Current Priority Level.* These three bits record the priority level of the interrupt presently under service (i.e. the Current Priority Level, CPL). For these priority levels 000 is the highest priority and 111 is the lowest priority. The CPL bits can be set by hardware or software and give the reference by which following interrupts are either left pending or able to interrupt the current interrupt. When the present interrupt is replaced by one of a greater priority, the current priority value is automatically stored until required.

SYSTEM REGISTERS (Continued)

2.3.2 Flag Register

The Flag Register contains 8 flags indicating the status of the ST9. During an interrupt the flag register is automatically stored in the system stack area and recalled at the end of the interrupt service routine so that the ST9 is returned to the original status. This occurs for all interrupts and, when operating in the nested mode, up to seven versions of the flag register may be stored.

FLAGR R231 (E7h) System Read/Write Flag Register

Reset value: undefined

7							0
C	Z	S	V	DA	H	UF	DP

b7 = **C:** *Carry Flag.* The carry flag C is affected by the following instructions:

Addition (`add, addw, adc, adcw`),
Subtraction (`sub, subw, sbc, sbcw`),
Compare (`cp, cpw`),
Shift Right Arithmetic (`sra, sraw`),
Rotate (`rrc, rrcw, rlc, rlcw, ror, rol`),
Decimal Adjust (`da`),
Multiply and Divide (`mul, div, divws`).

When set, it generally indicates a carry out of the most significant bit position of the register being used as an accumulator (bit 7 for byte and bit 15 for word operations).

The carry flag can be set by the Set Carry Flag (`scf`) instruction, cleared by the Reset Carry Flag (`rcf`) instruction, and complemented (changed to "0" if "1", and vice versa) by the Complement Carry Flag (`ccf`) instruction.

b6 = **Z:** *Zero Flag.* The Zero flag is affected by the following instructions:

Addition (`add, addw, adc, adcw`),
Subtraction (`sub, subw, sbc, sbcw`),
Compare (`cp, cpw`),
Shift Right Arithmetic (`sra, sraw`),
Rotate (`rrc, rrcw, rlc, rlcw, ror, rol`),
Decimal Adjust (`da`),
Multiply and Divide (`mul, div, divws`),
Logical (`and, andw, or, orw, xor, xorw, cpl`),
Increment and Decrement (`inc, incw, dec, decw`),
Test (`tm, tmw, tcm, tcmw, btset`).

In most cases, the Zero flag is set when the register being used as an accumulator register is zero, following one of the above operations.

b5 = **S:** *Sign Flag.* The Sign flag is affected by the same instructions as the Zero flag.

The Sign flag is set when bit 7 (for byte operation) or bit 15 (for word operation) of the register used as an accumulator is one.

b4 = **V:** *Overflow Flag.* The Overflow flag is affected by the same instructions as the Zero and Sign flags.

When set, the Overflow flag indicates that a two's-complement number, in a result register, is in error, since it has exceeded the largest (or is less than the smallest), number that can be represented in twos-complement notation.

b3 = **DA:** *Decimal Adjust Flag.* The Decimal Adjust flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag is used to specify which type of instruction was executed last, so that the subsequent Decimal Adjust (`da`) operation can perform its function correctly.
The Decimal Adjust flag cannot normally be used as a test condition by the programmer.

b2 = **H:** *Half Carry Flag.* The Half Carry flag indicates a carry out of (or a borrow into) bit 3, as the result of adding or subtracting two 8-bit bytes, each representing two BCD digits. The Half Carry flag is used by the Decimal Adjust (`da`) instruction to convert the binary result of a previous addition or subtraction into the correct BCD result.
Like the Decimal Adjust flag, this flag is not normally accessed by the user.

b1 = **UF:** *User Flag.* Bit 1 in the flag register (UF) is available to the user, but it must be set or cleared by an instruction.

b0 = **DP:** *Data/Program Memory Flag.* This bit in the flag register indicates which memory area is addressed. Its value is affected by the Set Data Memory (`sdm`) and Set Program Memory (`spm`) instructions.

If the bit is set, the ST9 addresses the Data Memory Area; when the bit is cleared, the ST9 addresses the Program Memory Area. By reading this bit, the user can verify in which memory area the processor is working. The user writes this bit with the `sdm` or `spm` instructions.

SYSTEM REGISTERS (Continued)

2.3.3 Register Pointing Techniques

Two registers, R232 and R233, within the system register group, are available for register pointing. R232 and R233 may be used together as a single pointer for a 16 register working space or separately for two 8 register spaces, in which case R232 becomes Register Pointer 0 (RP0) and R233 becomes Register Pointer 1 (RP1).

The instructions `srp`, `srp0` and `srp1` (the Set Register Pointer instructions) automatically inform the ST9 whether the Register File is to operate with a single 16-register group or two 8-register groups. The `srp0` and `srp1` instructions automatically set the twin 8-register group mode while the `srp` instruction sets the single 16-register group mode. There is no limitation on the order or positions of these chosen register groups other than they must be on 8 or 16 register boundaries.

The addressing of working registers involves use of the Register Pointer value plus an offset value given by the number of the addressed working register.

When addressing a register, the most significant nibble (bits 4-7) gives the group address and the least significant nibble (bits 0-3) gives the register within that group.

REGISTER POINTER 0

RP0 R232 (E8h) System Read/Write Register Pointer 0

Reset Value : undefined

7							0
RG7	RG6	RG5	RG4	RG3	RPS	D1	D0

b7-b3 = **RG7-RG3:** *Register Group number.* These bits contain the number (from 0 to 31) of the group of working registers indicated in the instructions `srp0` or `srp`. When using a 16-register group, a number between 0 and 31 must be used in the `srp` instruction indicating one of the two adjacent 8-register group of working registers used. RG7 is the MSB.

b2 = **RPS:** *Register Pointer Selector.* This bit is set by the instructions `srp0` and `srp1` to indicate that a double register pointing mode is used. Otherwise, the instruction `srp` resets the RPS bit to zero to indicate that a single register pointing mode is used.

b1,b0 = **D1,D0:** These bits are fixed by hardware to zero and are not affected by any writing instruction trying to modify their value.

REGISTER POINTER 1

RP1 R233 (E9h) System Read/Write Register Pointer 1

Reset Value : undefined

7							0
RG7	RG6	RG5	RG4	RG3	RPS	D1	D0

This register is used only with double register pointing mode; otherwise, using single register pointing mode, the RP1R register has to be considered as reserved and not usable as a general purpose register.

b7-b3 = **RG7-RG3:** *Register Group number.* These bits contain the number (from 0 to 31) of the group of 8 working registers indicated in the instructions `srp1`. Bit 7 is the MSB.

b2 = **RPS:** *Register Pointer Selector.* This bit is automatically set by the instructions `srp0` and `srp1` to indicate that a double register pointing mode is used. Otherwise the instruction `srp` reset the RPS bit to zero to indicate that a single register pointing mode is used.

b1,b0 = **D1,D0:** These bits are hardware fixed to zero and are not affected by any writing instruction trying to modify their value.

Note. If working in twin 8-register group mode but only using `srp0` (i.e. only using one 8-register group) the unused register (R233) is to be considered as reserved and not usable as a general purpose register.

The group of registers immediately below the system registers (i.e. group D, R208-R223) can only be accessed via the Register Pointers. To address group D then, it is necessary to set the Register Pointer to group D and then use the addressing procedure for working registers. The programmer is required to remember that the group D should be used as a stacking area. This point is also covered in the Stack Pointers paragraph.

SYSTEM REGISTERS (Continued)

EXAMPLES

Using the Single 16 Register Group

When the system is operating in the single 16-register group mode, the registers are referred to as r0-r15. In this mode, the offset value (i.e. the number of the working register referred to) is supplied in the address (preceded by a small r, e.g. r5) and is added to the Register Pointer 0 value to give the absolute address.

For example, if the Register Pointer contains the value 70h, then working register r7 would have the absolute address, R77h.

In this mode, the single 16-registers group will always start from the lowest even number equal or lower to the number given in the instruction.

Example: `srp #3` is equivalent to `srp #2`.

Using the Twin 8-Register Group

When working in the twin working group mode, the registers pointed by Register Pointer 0 (RP0R), are referred as r0-r7 and those pointed by Register Pointer 1 (RP1R), are referred to as r8-r15, regardless of their absolute addresses. In this mode, when operating with the first 8 working registers (i.e. r0 - r7) the working register number acts as an offset which is added to the value in Register Pointer 0.

So if Register Pointer 0 contains the value 96, then working register 0 has the absolute address 96, working register 5 has the absolute address 101, and so on. The second group of working registers, r8-r15, has the offset values 0 to 7 respectively (i.e. r8 has the offset value 0, r9 has the offset value 1, and so on), this offset value being added to the value in Register Pointer 1.

For example, given that the value in Register Pointer 1 is 32, then working register 12 supplies an offset value of 4 (given by 12 minus 8) to the value in Register Pointer 1 to give an absolute address of 36.

Figure 2-6. Single 16 Register pointing Mode

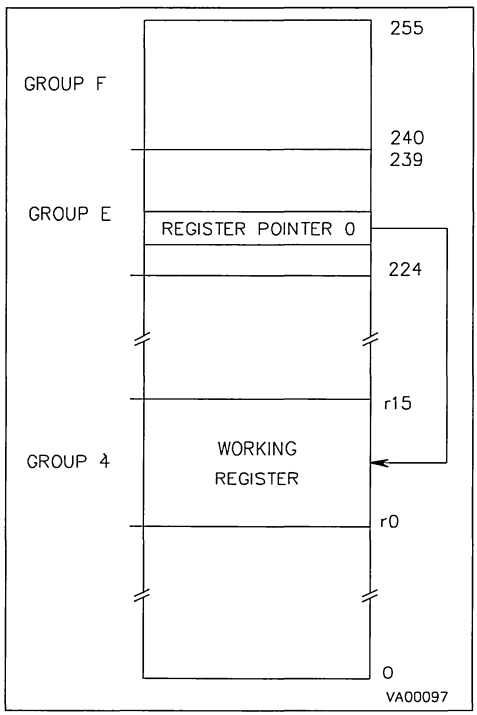
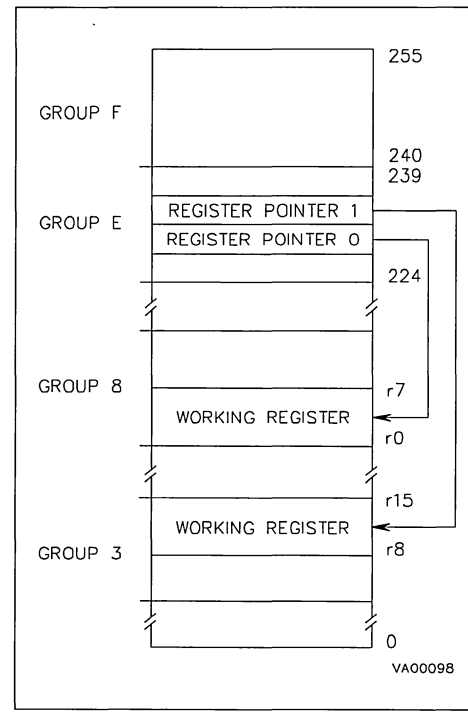


Figure 2-7. Double Register pointing Mode



SYSTEM REGISTERS (Continued)

2.3.4 Page Configuration

The pages are available to be used for the storage of control information (such as interrupt vector pointers) relevant to particular peripherals. There are up to 64 pages (each with 16 registers) based on registers R240-R255. These paged registers are addressable via the page pointer register (PPR), which is system register R234.

To address a paged register the page pointer register (R234) must be loaded with the relevant page number using the `spp` instruction (Set Page Pointer) and subsequently any address from the top (F) group (R240-R255) will be referred to that page.

For example if register 23 contains the value 44, the following sequence loads the third register R242 on page 5 with the value 44.

```
spp 5
ld R242, R23
```

PPR R234 (EAh) System Read/Write Page Pointer Register

Reset value : undefined

7							0
PP7	PP6	PP5	PP4	PP3	PP2	D1	D0

b7-b2 = **PP7-PP2:** *Page Pointer.* These bits contain the number (between 0 to 63) of the page chosen by the instruction `ssp` (Set Page Pointer). PP7 is the MSB of the page address. Once the page pointer has been set, there is no need to refresh it unless a different page is required.

b1-b0 = **D1,D0:** These bits are fixed by hardware to zero and are not affected by any writing instruction trying to modify their value.

PAGE 0 contains the control registers of:

- the external interrupt
- the watchdog timer
- the wait logic states
- the serial peripheral interface (SPI)
- the EEPROM (ST9040/E40/T40)

2.3.5 Mode Registers

This register MODER is located in the System Register Group at the address 235.

Using this register it is possible:

- to select either internal or external System and User Stack area,
- to manage the clock frequency
- to enable the Bus request and Wait signals when interfacing external memory.

MODER R235 (EBh) System Read/Write Mode Register

Reset value : 1110 0000

7							0
SSP	USP	DIV2	PRS2	PRS1	PRS0	BRQEN	HIMP

b7 = **SSP:** *System Stack Pointer.* This bit selects internal (in the Register File) or external (in the external Data Memory) System Stack area, logical "1" for internal, and logical "0" for external. After Reset the value of this bit is "1".

b6 = **USP:** *User Stack Pointer.* Same as bit 7 for the User Stack Pointer;

b5 = **DIV2:** *OSCIN Clock Divided by 2.* This bit controls the divide by 2 circuit which operates on the OSCIN Clock. A logical "1" value means that the OSCIN clock is internally divided by 2, and a logical "0" value means that no division of the OSCIN Clock occurs.

b4-b2 = **PRS2-PRS0:** *ST9 CPUCLK Prescaler.* These bits load the prescaling module of the internal clock (INTCLK). The prescaling value selects the frequency of the ST9 clock, which can be divided by 1 to 8. See Clock chapter for more information.

b1 = **BRQEN:** *Bus Request Enable.* This bit is a software enable of an External Bus Request. When set to "1", it enables a Bus Request on the BUSREQ pin.

b0 = **HIMP:** *High Impedance Enable.* When Port 0 and/or Port 1 are programmed as multiplexed address and Data lines to interface external Program and/or Data Memory, these lines and the Memory interface control lines can be forced into the High Impedance state by setting to "1" the HIMP bit. When this bit is reset, it has no effect on P0 and P1 lines.

If Port 1 is declared as an address AND as an I/O port (example: P10 ... P14 = Address, and P15 ... P17 = I/O), HIMP has no effect on the I/O lines (in the previous example: P15 ... P17).

SYSTEM REGISTERS (Continued)

2.3.6 Stack Pointers

There are two separate, double register stack pointers available (named System Stack Pointer and User Stack Pointer), both of which can address registers or memory.

The stack pointers point to the bottom of the stacks which are filled using the `push` commands and emptied using the `pop` commands. The stack pointer is automatically pre-decremented when data is "pushed in" and post-incremented when data is "popped out".

For example, the register address space is selected for a stack and the corresponding stack pointer register contains 220. When a byte of data is "pushed" into the stack, the stack pointer register is decremented to 219, then the data byte is "loaded" into register 219. Conversely, if a stack pointer register contains 189 and a byte of data is "popped" out, the byte of data is then extracted from the stack and then the stack pointer register is incremented to 190.

The `push` and `pop` commands used to manage the system stack area are made applicable to the user stack by adding the suffix U, while to use a stack instruction for a word a W is added.

For example `push` inserts data into the system stack, but an added U indicates the user stack and W means a word, so the instruction `pushuw` loads a word into the bottom of the user stack.

If the User Stack Pointer register contains 223 (working in register space) the instruction `pushuw` will decrement User Stack Pointer register to 222 and then load a word into register R222 and R221.

When bytes (or words) are "popped out" the values in those registers are left unchanged until fresh data is loaded into those locations. Thus when data is "popped" out from a stack area, the stack content remains unchanged.

Note. Stacks must not be located in the pages or the system register area.

The System Stack area and The System Stack Pointer

The System Stack area is used for the storage of temporarily suspended system and/or control registers, i.e. the Flag register and the Program counter, while interrupts are being serviced. For subroutine execution only the Program Counter needs to be saved in the System stack area.

There are two situations when this occurs automatically, one being when an interrupt occurs and the other when the instruction call subroutine is used. When the system stack area is in the Register File, the stack pointer, which points to the bottom of the stack, only needs one byte for addressing, in which case the System Stack Pointer Low Register (R239) is sufficient for addressing purposes. As a result the System Stack Pointer High Register (R238) becomes redundant BUT must be considered as reserved (please refer also to "spirious" memory access section). Clearly when the stack is external a full word address is necessary and so both registers are used to point, the even register providing the MSB and the odd register providing the LSB.

The User Stack area and User Stack Pointer

The User Stack area is completely free from all interference from automatic operations and so it provides a totally user controlled stacking area, that area being in any part of the memory which is of a RAM nature, or the first 14 groups of the general Register File i.e. not in the System register or Paged group.

The User Stack Pointer consists of two registers, R236 and R237, which are both used for addressing an external stack, while, when stacking in the Register File, the User Stack Pointer High Register, R236, becomes redundant but must be considered as reserved.

SYSTEM REGISTERS (Continued)

Stack location

Care is necessary when managing stacks as there is no limit to stack sizes apart from the bottom of any address space in which the stack is placed. Consequently programmers are advised to use a stack pointer value as high as possible, particularly when using the Register File as a stacking area. This will also benefit programmers who may locate the stacks in group D using, for example the instruction `ld R237, #223` which loads the value 223 into the User Stack Pointer Low Register. The Programmer will not need to remember to set the Register Pointer to 208 to gain access to registers in the D-group, a problem outlined in Register Pointing Techniques paragraph.

Stacks may be located anywhere in the first 14 groups of the Register File (internal stacks) or the data memory (external stacks). It is not necessary to set the data memory using the instruction `sdm` as external stack instructions automatically use the data memory.

Figure 2-8. System and/or User Stack in Register Stack Mode

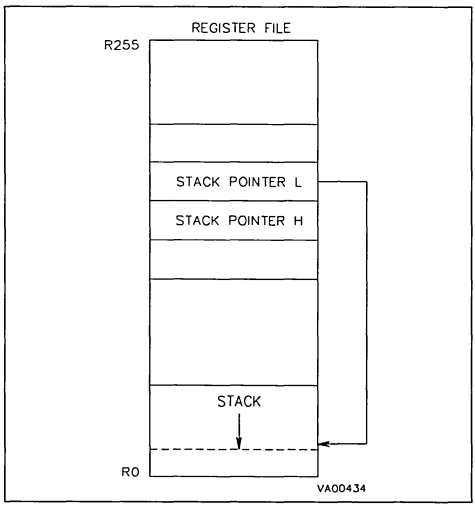
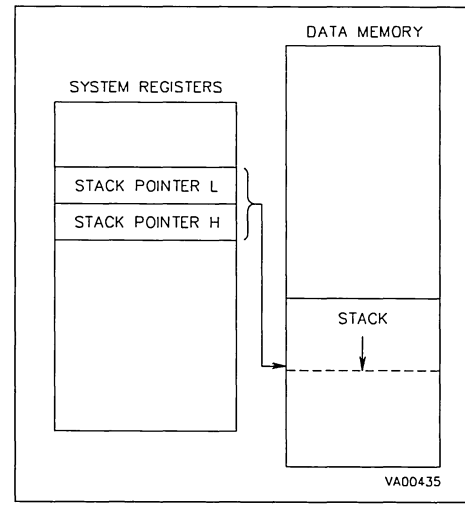


Figure 2-9. System and/or User Stack in Register Stack Mode



USP R236 (ECh) Read/Write
User Stack Pointer High Byte
Reset value: undefined

7							0
USP15	USP14	USP13	USP12	USP11	USP10	USP9	USP8

USP R237 (EDh) Read/Write
User Stack Pointer Low Byte
Reset value: undefined

7							0
USP7	USP6	USP5	USP4	USP3	USP2	USP1	USP0

SSP R238 (EEh) Read/Write
System Stack Pointer High Byte
Reset value: undefined

7							0
SSP15	SSP14	SSP13	SSP12	SSP11	SSP10	SSP9	SSP8

SSP R239 (EFh) Read/Write
System Stack Pointer Low Byte
Reset value: undefined

7							0
SSP7	SSP6	SSP5	SSP4	SSP3	SSP2	SSP1	SSP0

3 MEMORY

3.1 INTRODUCTION

The memory of the ST9 is divided into two spaces:

- Data memory with up to 64K (65536) bytes
- Program memory with up to 64K (65536) bytes

Thus, there is a total of 128K bytes of addressable memory space.

The 16K bytes of on-chip ROM memory of the ST9040 are selected at memory addresses 0 through 3FFFh (hexadecimal) in the PROGRAM space.

The DATA space includes the 512 bytes of on-chip EEPROM at addresses 0 through 1FFh and the 256 bytes of on-chip RAM memory at addresses 200h through 2FFh.

Off-chip memory, addressed using the multiplexed address and data buses (Ports 0 and 1) may be divided into the Program and Data spaces by the external decoding of the Program/Data select pin (P/$\overline{D}$) available as an Alternate function output, allowing the full 128K byte memory.

The memory spaces are selected by the execution of the `sdm` and `spm` instructions (Set Data Memory and Set Program Memory, respectively). There is no need to use either of these instructions again until the memory area required is to be changed. This requirement is not necessary in two cases: first, when operating with external stacks (the Data memory is automatically selected) and, secondly, when using the memory indirect to memory indirect post-increment addressing mode (the memory types are specified in the instructions: `ldpp, ldpd, lddp, lddd`).

Program instructions and data in the immediate addressing mode are always read from the Progam space. For example:

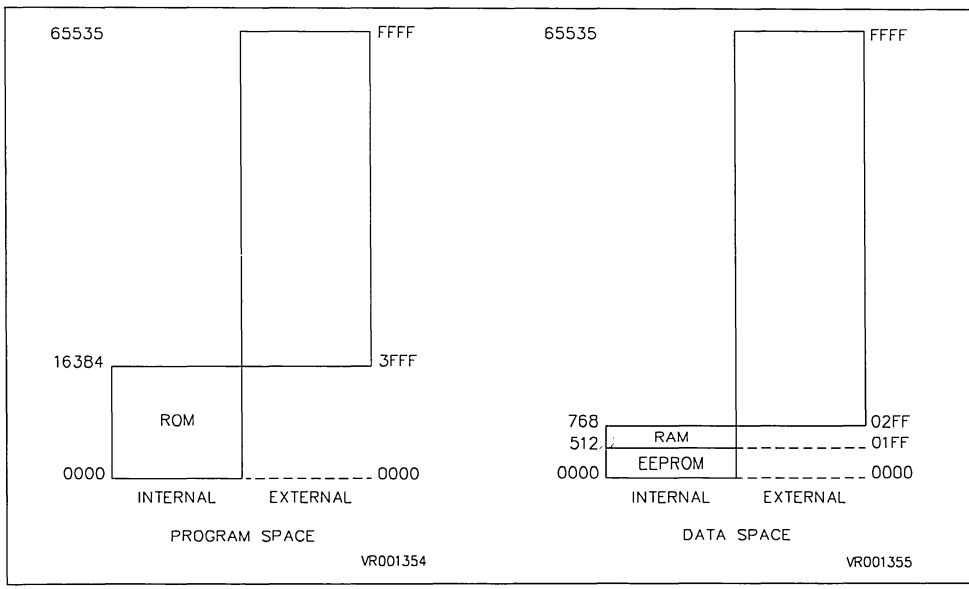
```
sdm
ld  R80,#99
```

will load R80 with the value 99 decimal from the program space.

Either the Data Memory or the Program Memory, both external or internal, can be addressed using any of the memory addressing modes.

The 16 bit memory address may be supplied directly using the absolute memory location address or indirectly using a pair of registers. In addition the address can be given by an indexed mode when a short (byte) or long (word) offset is added to an indirect base word address.

Figure 3-1. Memory Map



3.2 PROGRAM SPACE DEFINITION

The Program memory space of the ST9040, from the 16K bytes of on-chip ROM memory (0 through 3FFFh) to the full 64K bytes with off-chip memory expansion is fully available to the user. At addresses greater than the first 16K bytes of Program space, external memory cycles are automatically executed for instruction fetches.

The first 256 memory locations from address 0 to FFh hold the Reset Vector, the Top-Level (Pseudo Non-Maskable) interrupt, the Divide by Zero Trap Routine vector and, optionally, the interrupt vector table for use with the on-chip peripherals and the external interrupt sources. Apart from this case no other part of the Program memory has a predetermined function.

Each vector is contained in two consecutive byte locations, the high order address held in the lower (even) byte, the low order address held in the upper (odd) byte, forming the address which is loaded into the Program Counter when selected by the interrupt vector provided by the interrupt source. This should point to the relevant Interrupt Service routine provided by the user for immediate response to the interrupt.

Table 3-1. First 6 Bytes of Program Space

0	Address high of Power on Reset routine
1	Address low of Power on Reset routine
2	Address high of Divide by zero trap Subroutine
3	Address low of Divide by zero trap Subroutine
4	Address high of Top Level Interrupt routine
5	Address low of Top Level Interrupt routine

3.3 ROMLESS OPTION SUMMARY

In the event of a program revision being required after the development of a ROM-based device, a mask option is available which enables the reconfiguration of the memory spaces to give a fully ROMless device. This means that the on-chip program ROM is disabled and ALL PROGRAM memory is seen as external, allowing the use of replacement program code in external ROM memory. The on-chip EEPROM and RAM memory in DATA space is not affected.
For more information on this option, please refer to the section "External Memory Interface".

3.4 DATA SPACE DEFINITION

The Data memory maximum size is 64K bytes and has exactly the same addresses and addressing modes as the Program memory, the spaces being distinguished by the use of the memory setting command (`sdm`, Set Data Memory).

The ST9040 addresses the 512 bytes of on-chip EEPROM memory in the Data Space from addresses 0 to 511 (00h to 1FFh) and the 256 bytes of on-chip RAM memory from addresses 512 to 768 (200h to 2FFh). It may also address up to 64,767 locations of External Data through the External Memory Interface when decoded with the P/$\overline{D}$ pin.
The on-chip general purpose registers may be used as additional RAM memory for minimum chip count systems.

The Data Space is selected by the execution of the `sdm` instruction. All subsequent operand and stack memory references will access the Data Space.

When a separate Data Space is not provided, data may also be stored in external RAM or ROM memory within the Program Space.

3.5 EEPROM

3.5.1 Introduction

The EEPROM memory provides user-programmable non-volatile memory on-chip, allowing fast and reliable storage of user data. As there is also no off-chip access required, as for an external serial EEPROM, high security levels can be achieved.

The EEPROM memory is read as normal RAM memory at Data Space addresses 0 to 1FFh, however one WAIT cycle is automatically added for a Read cycle, while a byte write cycle to the EEPROM will cause the start of an ERASE/WRITE cycle at the addressed location. Word (16 bit) writes are not allowed.

The programming cycle is self-timed, with a typical programming time of 6ms. The voltage necessary for programming the EEPROM is internally generated with a +18V charge pump circuit.

Up to 16 bytes of data may be programmed into the EEPROM during the same write cycle by using the PARALLEL WRITE function.

A standby mode is also available which disables all power consumption sources within the EEPROM for low power requirements. When STBY is high, any attempt to access the EEPROM memory will produce unpredictable results. After the re-enabling of the EEPROM, a delay of 6 INTCLK cycles must be allowed before the selection of the EEPROM.

EEPROM (Continued)

The EEPROM of the ST9040 has been implemented in a high reliability technology developed by SGS-THOMSON, this, together with the double bit structure, allow 300k Erase/Write cycles and 10 year data retention to be achieved on a microcontroller.

Control of the EEPROM is performed through one register mapped at register address R241 in Page 0.

3.5.2 EEPROM Programming Procedure

The programming of a byte of EEPROM memory is equivalent to writing a byte into a RAM location after verifying that EEBUSY bit is low. Instructions operating on word data (16 bits) will not access the EEPROM.

The EEPROM ENABLE bit EEWEN must first be set before writing to the EEPROM. When this bit is low, attempts to write data to the EEPROM have no affect, this prevents any spurious memory accesses from affecting the data in the EEPROM.

Termination of the write operation can be detected by polling on the EEBUSY status bit, or by interrupt, taking the interrupt vector from the External Interrupt 4 channel. The selection of the interrupt is made by EEPROM Interrupt enable bit EEIEN. It should be noted that the Mask bit of External Interrupt 4 should be set, and the Interrupt Pending bit reset, before the setting of EEIEN to prevent unwanted interrupts. A delay (eg a `nop` instruction) should also be included between the operations on the mask and pending bits of External Interrupt 4.

If polling on EEBUSY is used, a delay of 6 INTCLK clock cycles is necessary after the end of programming, this can be a `nop` instruction or, normally, the required time to test the EEBUSY bit and to branch to the next instruction will be sufficient. While EEBUSY is active, any attempt to access the EEPROM matrix will be aborted and the data read will be invalid. EEBUSY is a read only bit and cannot be reset by the user if active.

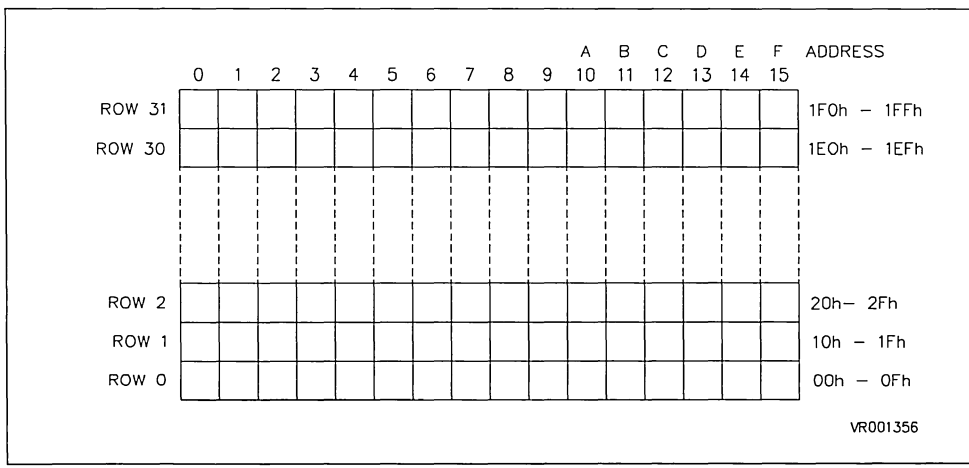
An erased bit of the EEPROM memory will read as a logic "0", while a programmed cell will be read as a logic "1". For applications requiring the highest level of reliability, the Verify Mode, set by EEPROM control register bit VRFY, allows the reading of the EEPROM memory cells with a reduced gate voltage (typically 20%). If the EEPROM memory cell has been correctly programmed, a logic "1" will be read with the reduced voltage, otherwise a logic "0" will be read.

3.5.3 Parallel Programming Procedure

Parallel programming is a feature of the EEPROM macrocell. One up to sixteen bytes of a same row can be programmed at once.

The constraint is that each of the bytes occur in the same ROW of the EEPROM memory (A4 constant, A3-A0 variable). To operate this mode, the Parallel Mode enable bit, PLLEN, must be set. The data written is then latched into buffers (at the addresses specified, which may be non-sequential) and then transferred to the EEPROM memory by the setting of the PLLST bit of the control register. Both PLLST and PLLEN are internally reset at the end of the programming cycle. Any attempt to read the EEPROM memory when PLLEN is set will give invalid data. In the event that the data in the buffer latches is not required to be written into the memory by the setting of PLLST, the correct way to terminate

Figure 3-2. EEPROM Parallel Programming Rows



EEPROM (Continued)

the operation is to reset PLLEN and to perform a dummy read of the EEPROM memory. This termination will clear all data present in the latches.

3.5.4 EEPROM Programming Voltage

No external Vpp voltage is required, an internal 18Volt charge-pump gives the required energy by a dedicated oscillator pumping at a typical frequency of 5MHz, regardless of the external clock.

3.5.5 EEPROM Programming Time

No timing routine is required to control the programming time as dedicated circuitry takes care of the EEPROM programming time (The typical programming time is 6ms).

3.5.6 EEPROM Interrupt Management

At the end of each write procedure the EEPROM sends an interrupt request (if EEIEN bit is set). The EEPROM shares its interrupt channel with the external interrupt source INT4, from which the priority level is derived.

Care must be taken when EEIEN is reset. The associated external interrupt channel must be disabled (by reseting bit 4 of EIMR, R244) along with reseting the interrupt pending bit (bit 4 of EIPR, R243) to prevent unwanted interrupts. A delay instruction (at least 1 `nop` instruction) must be inserted between these two operations

WARNING. *The content of the EEPROM of the ST9040 family after the out-going test at SGS-THOMSON's manufacturing location is not guaranteed.*

3.5.7 EEPROM Control Register

EECR R241 (F1h) Page 0 Read/Write
(except EEBUSY: read only)
EEPROM Control Register

Reset value : 0000 0000b (00h)

7							0
0	VERIFY	EESTBY	EEIEN	PLLST	PLLEN	EEBUSY	EEWEN

bit 7 = **B7:** This bit is forced to "0" after reset and **MUST** not be modified by the user.

bit 6 = **VERIFY:** *Set Verify mode.* Verify (active high) is used to activate the verify mode.

The verify mode provides a guarentee of good retention of the programmed bit. When active, the reading voltage on the cell gate is decreased from 1.2V to 0.0V, decreasing the current from the programmed cell by 20%. If the cell is well programmed (to "1"), a "1" will still be read, otherwise a "0" will be read.

Note . The verify mode must not be used during an erasing or a programming cycle).

bit 5 = **EESTBY:** *EEPROM Stand-By.* EESTBY = "1" switches off all power consumption sources inside the EEPROM. Any attempt to access the EEPROM when EESTBY = "1" will produce unpredictable results.

Note. After EESTBY is reset, the user must wait 6 CPUCLK cycles (e.g. 1 `nop` instruction) before selecting the EEPROM.

bit 4 = **EEIEN:** *EEPROM Interrupt Enable.* INTEN = "1" disables the external interrupt source INT4, and enables the EEPROM to send its interrupt request to the central interrupt unit at the end of each write procedure.

bit 3 = **PLLST:** *Parallel Write Start.* Setting PLLST to "1" starts the parallel writing procedure. It can be set only if PLLEN is already set. PLLST is internally reset at the end of the programming sequence.

bit 2 = **PLLEN:** *Parallel write Enable.* Setting PLLEN to "1" enables the parallel writing mode which allows the user to write up to 16 bytes at the same time. PLLEN is internally reset at the end of the programming sequence.

bit 1 = **EEBUSY:** *BUSY.* When this read only bit is high, an EEPROM write operation is in progress and any attempt to access the EEPROM is aborted.

bit 0 = **EEWEN:** *EEPROM Write Enable.* Setting this bit allows programming of the EEPROM, when low a writing attempt has no effect.

4 INTERRUPTS

4.1 INTRODUCTION

The ST9 responds to peripheral events and external events through its Interrupt channels. When such an event occurs, if previously enabled and according to a priority mechanism, the current program execution can be suspended to allow the ST9 to execute a specific response routine. If the event generates an interrupt request, the current program status is saved after the current instruction is completed and the CPU control passes to the Interrupt Service Routine.

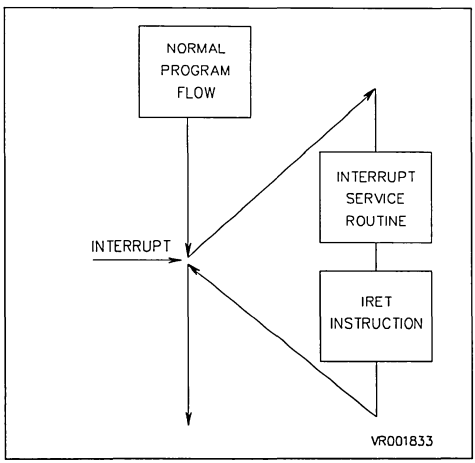
The ST9 CPU can receive requests from the following type of sources:

- On-chip peripherals
- External pins
- Top-Level Pseudo-non-maskable interrupt

According to the on-chip peripheral features, an event occurrence can generate an Interrupt request depending on the selected mode.

Up to eight external interrupt channels, with programmable input trigger edge, are available. In addition, a dedicated interrupt channel, set to the Top-level priority, can be devoted either to the external pin NMI (to provide a Non-Maskable-Interrupt) or to the Timer/Watchdog. Interrupt service routines are addressed through a vector table mapped in Program Memory.

Figure 4-1. Interrupt Flow



4.2 INTERRUPT VECTORIZATION

The ST9 implements an interrupt vectoring structure that allows the on-chip peripheral to identify the location of the first instruction of the Interrupt Service Routine (IVR) automatically.

When the interrupt request is acknowledged, the peripheral interrupt module provides, through its Interrupt Vector Register (IVR), a vector to point into the vector table of locations containing the start addresses of the Interrupt Service Routines (defined by the programmer).

Each peripheral has a specific IVR mapped within its Register File pages.

The Interrupt Vector table, containing the list of the addresses of the Interrupt Service Routines, is located in the first 256 locations of the Program Memory. The first 6 locations of the Program Memory are reserved for:

Address	Content
0	Address high of Power on Reset routine
1	Address low of Power on Reset routine
2	Address high of Divide by zero trap Subroutine
3	Address low of Divide by zero trap Subroutine
4	Address high of Top Level Interrupt routine
5	Address low of Top Level Interrupt routine

With one Interrupt Vector register, it is possible to address more interrupt service routines; in fact, several peripherals share the same interrupt vector register among several interrupt channels. The most significant bits of the vector are user programmable to define the base vector address inside the vector table in the program memory, the least significant bits are controlled by the interrupt module in hardware to select the specific vector.

Note: The first 256 locations of the program memory can contain program code. Other than the Reset vector, they are not exclusively reserved to the vector table.

Warning. *Although the Divide by Zero Trap operates as an interrupt, the FLAG Register is not pushed onto the system Stack automatically. As a result it must be regarded as a subroutine, and the acknowledge routine must end with the* `RET` *instruction.*

INTERRUPT VECTORIZATION (Continued)

Figure 4-2. Vectors and Associated Routines

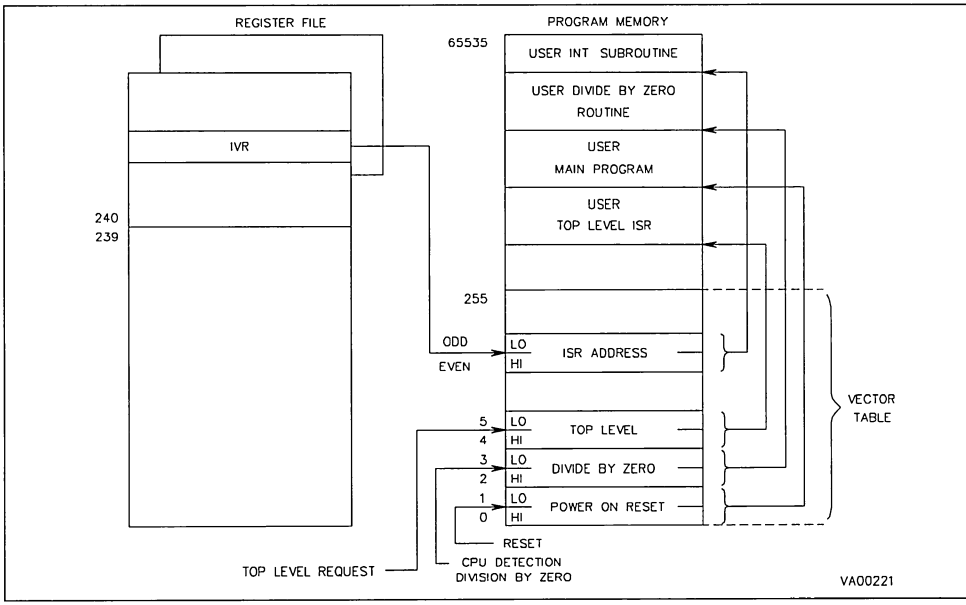
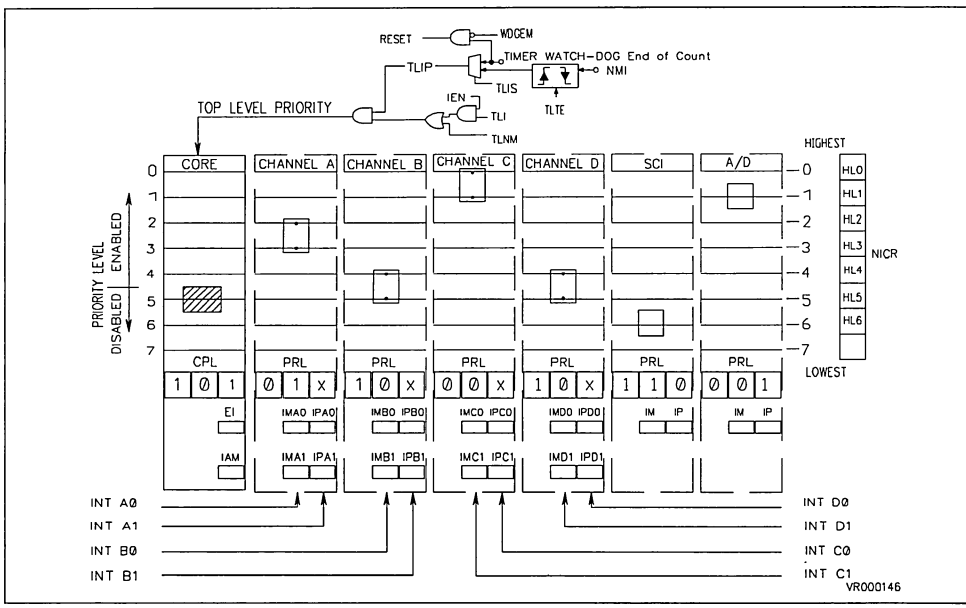


Figure 4-3. Interrupt Architecture, Example of piority Allocations



4.3 INTERRUPT PRIORITY LEVEL ARCHITECTURE

The ST9 supports a fully programmable interrupt priority structure. Figure 4-4 shows a conceptual description.

9 priority levels are available to define the channel priority relationship. Each channel has a 3 bit field, PRL (Priority Level), that defines its priority level among 8 programmable levels. The ninth level (Top Level Priority) is reserved for the Timer/Watchdog or the External Pseudo Non-Maskable Interrupt. The On-chip peripheral channel and the eight external interrupt sources can be programmed within eight priority levels: level 7 has the lowest priority, level 0 has the highest priority.

If several units are located at the same priority level, an internal daisy chain, fixed for each ST9 device, defines the priority relationship within that level.

The PRL bits are used to define the priority level for interrupt requests.

Top level priority interrupt (highest) can be assigned either to the external Pseudo Non-Maskable interrupt or to the internal Timer/Watch-Dog. An Interrupt service routine at this level cannot be interrupted in any arbitration mode. Its mask can be both maskable (TLI) or non-maskable (TLNM).

4.4 PRIORITY LEVEL ARBITRATION

The 3 bits of CPL (Current Priority Level) in the Central Interrupt Control Register contain the priority of the currently running program (CPU priority). CPL is set to 7 (lowest priority) upon reset and can be modified during program execution either by software or automatically by hardware according to the selected Arbitration Mode.

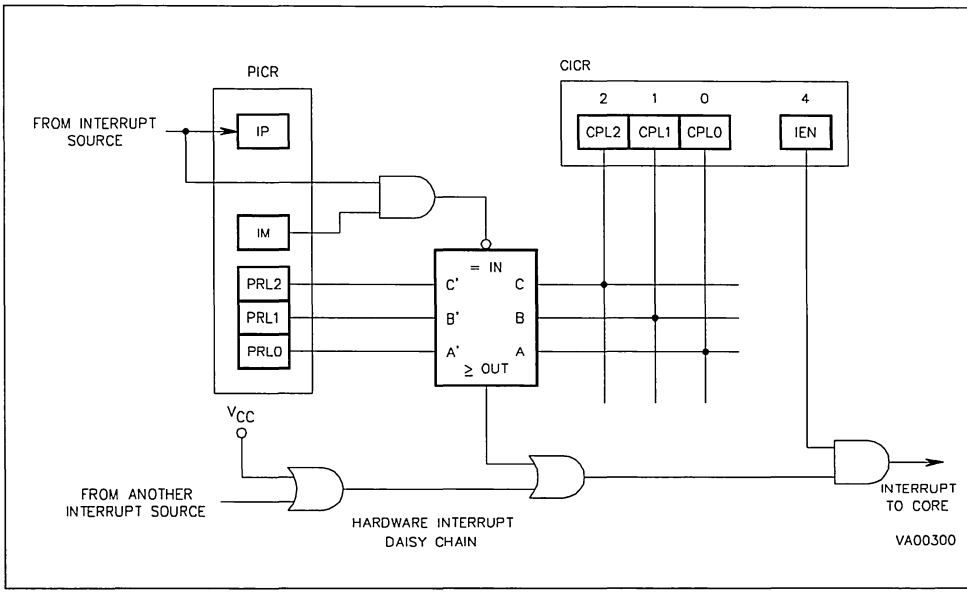
During every instruction an arbitration phase is made between every channel capable of generating an Interrupt, each priority level is compared to all the other requests. If the highest priority request is an interrupt, it must be *higher* than the CPL value in order to be acknowledged.

The priority of the Top Level Interrupt overrides every other priority.

If two or more requests occur at the same instant of time and at the same priority level, an on-chip daisy chain, specific to every ST9 version, selects the channel with the highest position in the chain. The position in the chain is shown in table 4-1.

ST9 provides two interrupt arbitration modes: Concurrent and Nested modes. The Concurrent mode is the standard interrupt arbitration mode while the Nested mode improves the effective interrupt response time when a nesting of the service routines is required according to the request priority levels.

Figure 4-4. Interrupt Logic



PRIORITY LEVEL ARBITRATION (Continued)

The control bit IAM (CICR.3) selects the Concurrent Arbitration mode (when reset to "0") or the Nested Arbitration Mode (when set to "1").

Table 4-1. Daisy Chain Priorities

Applicable for ST9030, ST9032, ST9036, ST9040

Highest Position	INTA0
	INTA1
	INTB0
	INTB1
	INTC0
	INTC1
	INTD0
	INTD1
	TIMER0
	SCI
	A/D
Lowest Position	TIMER1

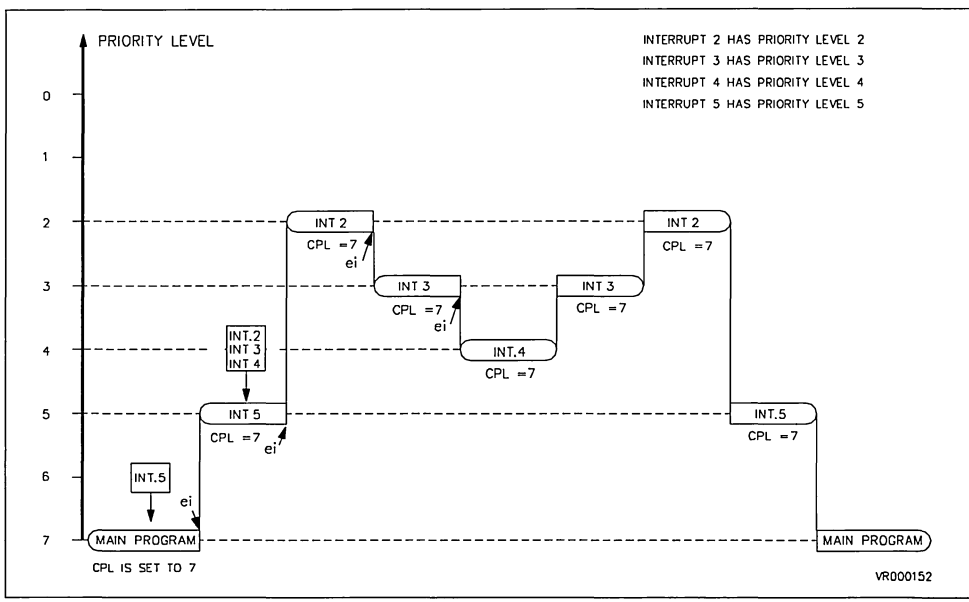
4.4.1 Concurrent Mode

This mode is selected when the IAM bit is cleared (reset condition). The arbitration phase, performed during every instruction, selects the request with the highest priority level.

If the highest priority request is an interrupt request and its priority value is higher than the Current Priority Value CICR.2,1,0 (R230.2,1,0), the interrupt request will be acknowledged at the end of the current instruction. The interrupt Machine Cycle performs the following steps:

- 1. Disables all the maskable interrupt requests by clearing CICR.IEN
- 2. Pushes the PC low byte into the system stack
- 3. Pushes the PC high byte into the system stack
- 4. Pushes the Flag register into the system stack
- 5. Loads the PC with the 16-bit vector stored in the Vector Table, pointed to by the Interrupt Vector Register (IVR).

Figure 4-5. Example of a Sequence of Interrupt Requests with :
- Concurrent mode
- EI set to 1 during the interrupt routine execution



PRIORITY LEVEL ARBITRATION (Continued)

The Interrupt Service Routine must be concluded with the `iret` instruction. The `iret` instruction executes the following operations:

- 1. Pops off the Flag register from the system Stack
- 2. Pops off PC high byte from the system Stack
- 3. Pops off PC low byte from the system Stack
- 4. Enables all the un-masked Interrupts, by setting the CICR.IEN bit

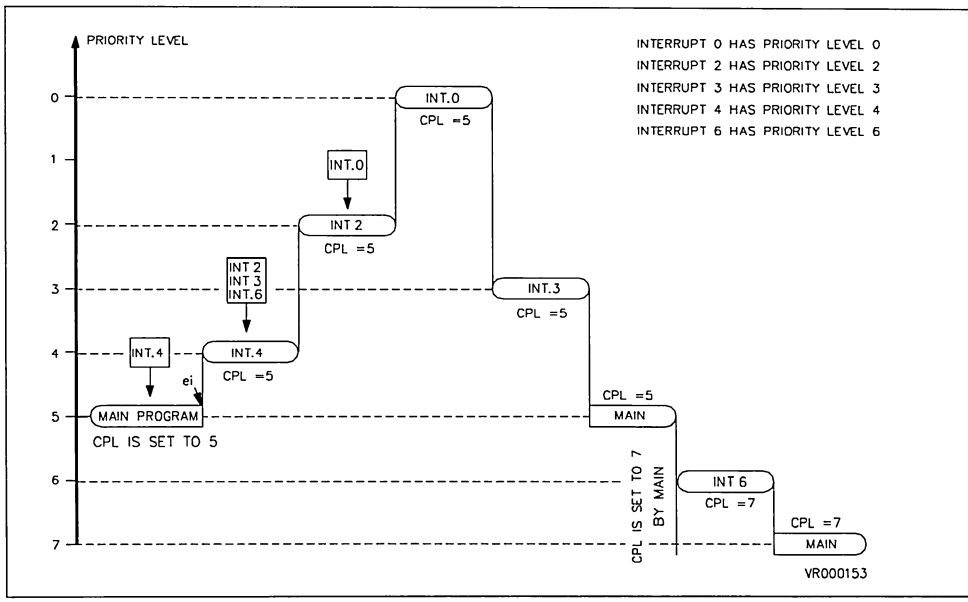
The suspended program execution is thus recovered at the interrupted instruction. All pending interrupts existing, or having occurred during the interrupt service routine execution, remain pending until the Enable Interrupt instruction (even if it is executed during the interrupt service routine).

NOTE: When Concurrent mode is selected, the source priority level is meaningful only during the arbitration phase, where it is compared to all the other priority levels and the CPL, but no trace is kept of its value during the Interrupt Service Routine. Therefore, if other requests are issued, once the global CICR.IEN is enabled again, they will be acknowledged regardless of the Interrupt Service Routine priority value; if no care is taken by the programmer, unpleasant side effects can take place.

A typical case is the following: 3 pending requests with different priority levels (ie 2,3,4) generate requests at the same time (because the associated events occurred during the same instruction). The three interrupt service routines set Interrupt Enable (IEN, CICR.4) by the `ei` instruction at the beginning of the routine to avoid a high interrupt response time to requests with a priority higher than the one under service (usually, the higher the priority, the sooner the routine must be executed). Unfortunately, what will happen in this case is that the three interrupt servicing routines will be executed exactly in the opposite order of their priority. Interrupt routine level 2 will be acknowledged first, then, when the `ei` instruction is executed, it will be interrupted by interrupt routine level 3, which itself will be interrupted by interrupt routine level 4. When interrupt routine level 4 is completed, interrupt routine level 3 will be recovered and finally, interrupt routine level 2.

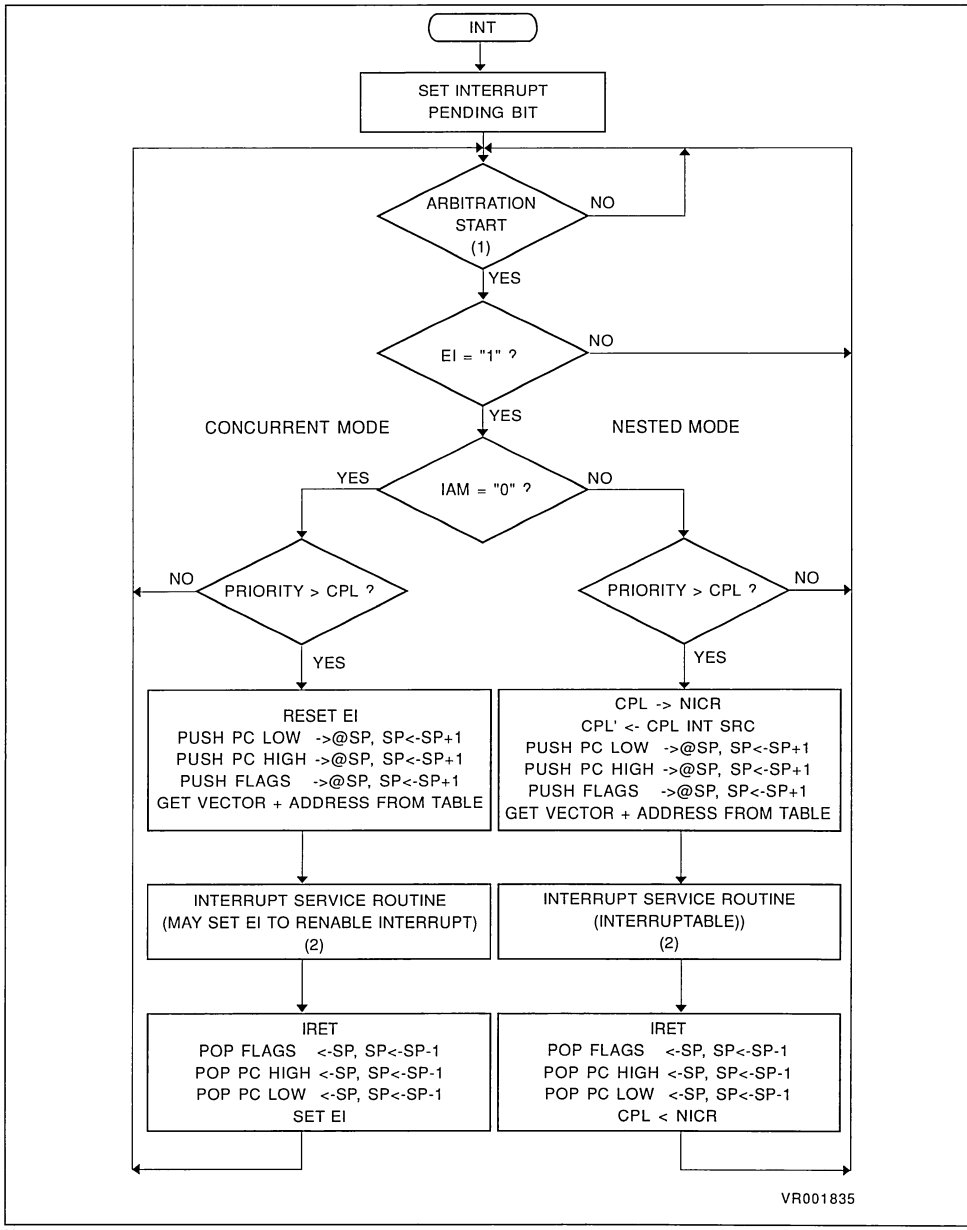
Therefore, **it is recommended, in concurrent mode, to avoid the insertion of the `ei` instruction in the interrupt subroutine**, which can trigger this LIFO (Last In, First Out) sequence of interrupt processing.

Figure 4-6. Example of a Sequence of Interrupt Requests with :
- Concurrent mode
- EI unchanged by the interrupt routines



PRIORITY LEVEL ARBITRATION (Continued)

Figure 4-7. Interrupt Mode Flow-Chart



Notes:

1. The interrupt arbitration starts 6 CPUCLK cycles befor the end of execution of each instruction (5 cycles during WFI).
2 Clear interrupt pending bit

PRIORITY LEVEL ARBITRATION (Continued)

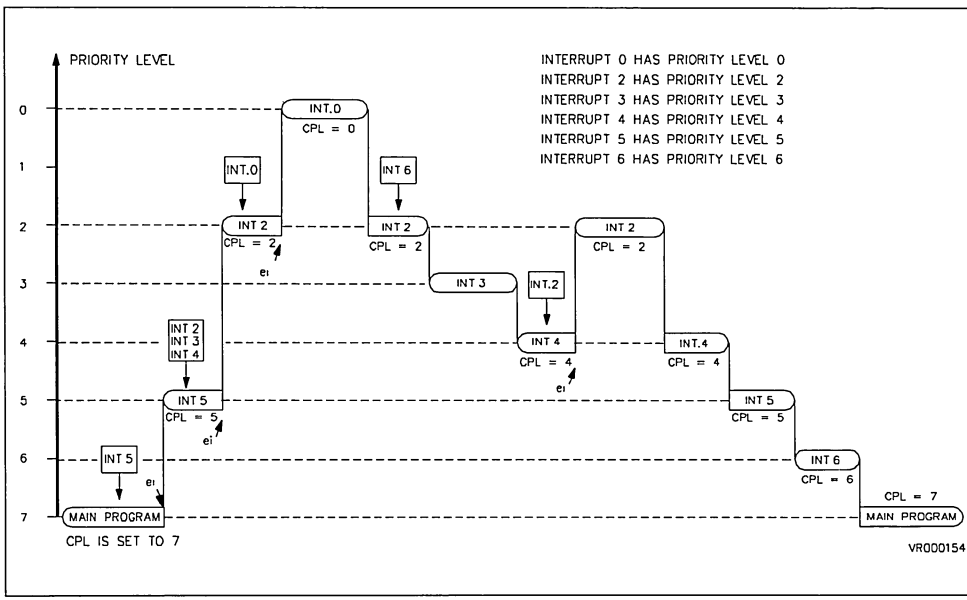
4.4.2 Nested Mode

The difference of the Nested mode to the Concurrent mode consists of the modification of the CPL value during the interrupt processing. The arbitration phase is basically identical to the concurrent Mode, however once the request is acknowledged, the current CPL value is saved in the Nested Interrupt Control Register (NICR, R247 page 0) by setting the NICR bit corresponding to the CPL value (i.e. if the CPL is 3, NICR.3 bit will be set). The CPL value is then updated with the Priority value of the request just acknowledged, in this way the next arbitration cycle will be performed against the priority level of the Service Routine in progress.

The Interrupt Machine Cycle will perform the following steps:

- Disable all the maskable interrupts by clearing IEN
- Save the CPL value into the special stack NICR to hold the priority level of the suspended routine
- Store in CPL the priority level of the acknowledged routine, so that the next request priority will be compared with the one of the routine under service
- Push the PC-low byte into the System Stack
- Push the PC-high byte into the System Stack
- Push the Flag Register into the System Stack

Figure 4-8. Example of a Sequence of Interrupt Requests with :
- Nested mode
- EI set to 1 during the interrupt routine execution



PRIORITY LEVEL ARBITRATION (Continued)

- Load the PC with the vector pointed by IVR.

The `iret` Interrupt Return instruction executes the following steps:

- 1. Pop off the Flag Register from the System Stack
- 2. Pop off the PC-high byte from the System Stack
- 3. Pop off the PC-low byte from the System Stack
- 4. Enable all the unmasked interrupts by setting the IEN bit
- 5. Recover the interrupted routine priority level by popping the value from the special register (NICR) and by copying it into CPL.

The suspended execution is thus recovered at the interrupted instruction.

REMARKS

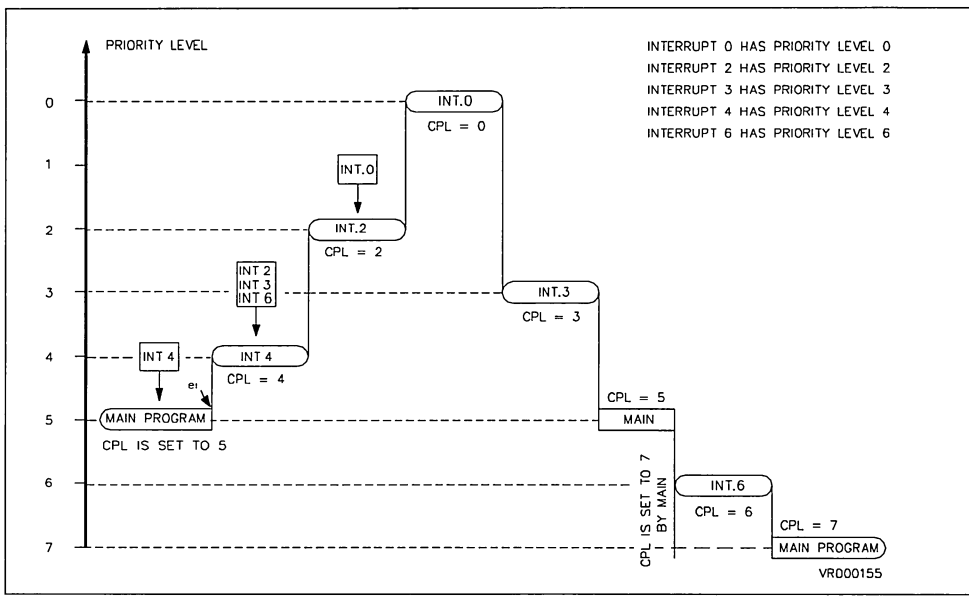
1) Dynamic priority level modification: the main program and routines can be specifically prioritized. Since CPL is represented by 3 bits in a read/write register, it is possible to modify dynamically the current priority value during the program execution. This means that a critical section can have a higher priority with respect to other interrupt requests. Furthermore it is possible to prioritize even the Main Program execution by modifying CPL during its execution.

2) Maximum number of nestings: No more than 8 routines can be nested. If an interrupt routine at level N is being serviced, no other Interrupts located at level N can interrupt it. This guarantees a maximum number of 8 nested levels including the Top Level Interrupt request.

3) Priority level 7: Interrupt requests at level 7 cannot be acknowledged as their priority cannot be higher than the CPL value. This can be of use in a fully polled interrupt environment.

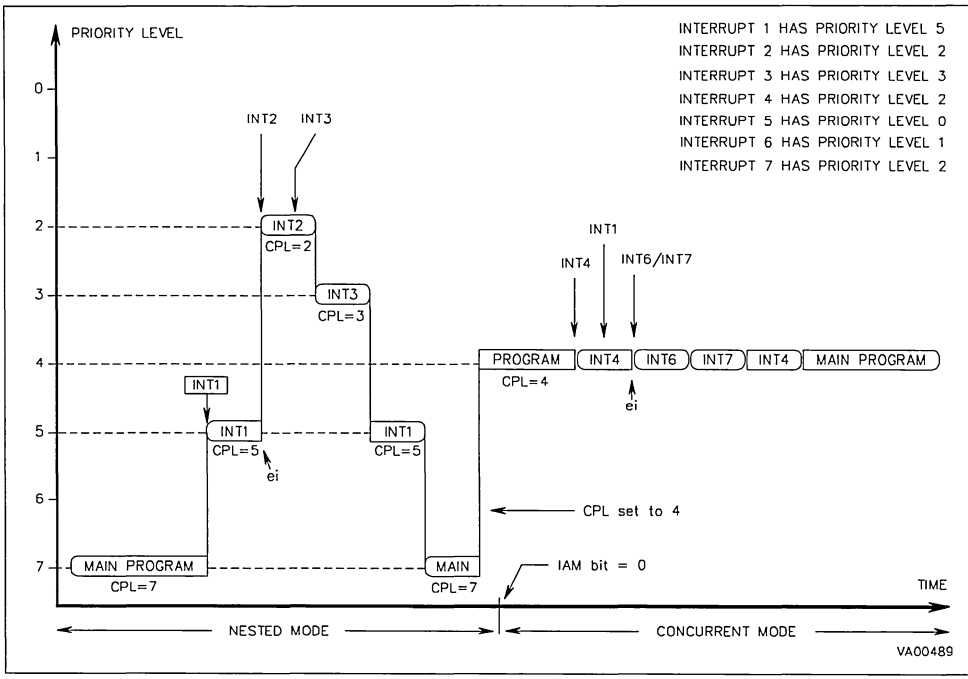
A nested/concurrent mode sequence is given on Figure 4-10. This example clearly shows that Nested and Concurrent modes are defined by the user. Note that here the Y axis is referenced by CPL, instead of the source priority level, and *that Interrupt 1 stays pending*, having a priority level lower than CPL.

Figure 4-9. Example of a Sequence of Interrupt Requests with :
- Nested mode
- EI unchanged by the interrupt routiness



PRIORITY LEVEL ARBITRATION (Continued)

Figure 4-10. Example of a Nested and Concurrent Mode Sequence



4.5 EXTERNAL INTERRUPTS

The standard ST9 core contains 8 external interrupts sources grouped into four pairs.

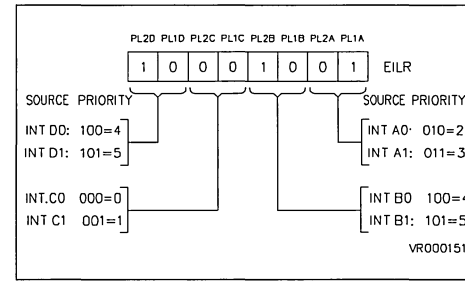
Table 4-2. External Interrupt Channel Grouping

External Interrupt	Channel
INT7 INT6	INTD1 INTD0
INT5 INT4	INTD1 INTD0
INT3 INT2	INTD1 INTD0
INT1 INT0	INTD1 INTD0

Each source has a trigger control bit TEA0,..TED1 (R242,EITR.0,..,7 Page 0) to select triggering on the rising or falling edge of the external pin. If the Trigger control bit is set to "1", the corresponding pending bit IPA0,..,IPD1 (R243,EIPR.0,..,7 Page 0) is set on the input pin rising edge, if it is cleared, the pending bit is set on the falling edge of the input pin. Each source can be individually masked through the corresponding control bit IMA0,..,IMD1 (EIMR.7,..,0). See Figure 4-12.

The priority level of the external interrupt sources can be programmed among the eight priority levels with the control register EIPLR (R245). The priority level of each pair is software defined using the bits PRL2,PRL1. For each pair, the even channel (A0,B0,C0,D0) of the group has the even priority level and the odd channel (A1,B1,C1,D1) has the odd (lower) priority level. Figure 4-11 shows an example of priority levels.

Figure 4-11. Priority Level Examples



- The source of the interrupt channel A0 can be selected between the external pin INT0 (when IA0S = "1", the reset value) or the On-chip Timer/Watchdog peripheral (when IA0S = "0").
- The source of the interrupt channel B0 can be selected between the external pin INT2 (when (SPEN,BMS)=(0,0)) or the on-chip SPI peripheral.
- The source of the interrupt channel C0 can be selected between the external pin INT4 (when EEIEN = "0") and the on-chip EEPROM write completion interrupt (when EEIEN="1").

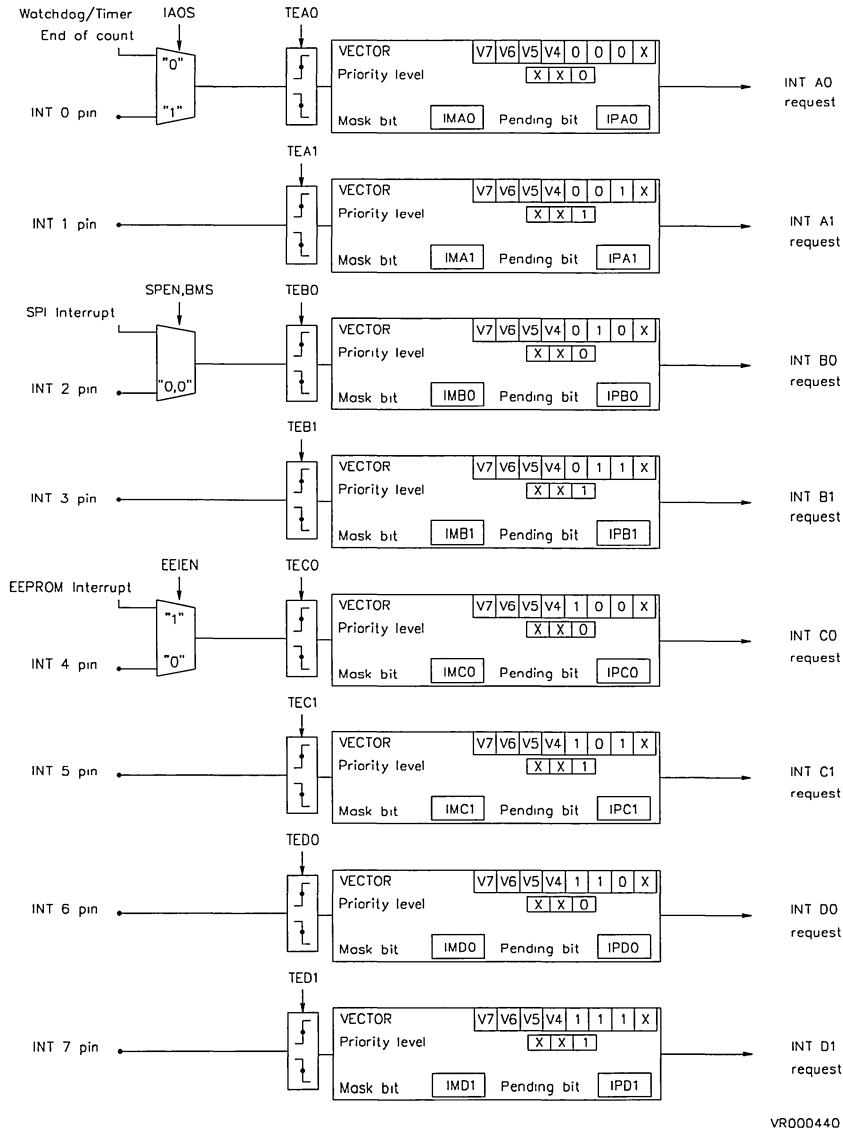
All other interrupt channels have an input pin as source, however, the input line may be multiplexed with an on-chip peripheral I/O or connected to an input pin that performs also other function (as in the case of the handshake feature).

Table 4-3. Internal/External Interrupt Source

Channel	Internal Interrupt Source	External Interrupt Source
INTA0	Timer/Watchdog	INT0
INTB0	SPI Interrupt	INT2
INTC0	EEPROM	INT4

EXTERNAL INTERRUPTS (Continued)

Figure 4-12. External Interrupts Control Bits and Vectors



4.6 TOP LEVEL INTERRUPT

The Top Level Interrupt channel can be assigned either to the external pin NMI or to the Timer/Watchdog according to the status of the control bit EIVR.TLIS (R246.2, Page 0). If this bit is high (the reset condition) the source is the external pin NMI, if it is low, the source is the Timer/ Watchdog End Of Count. When the source is the NMI external pin, the control bit EIVR.TLTEV (R246.3; Page 0) selects between the rising (if set) or falling (if cleared) edge generating the interrupt request. When the selected event occurs, the CICR.TLIP bit (R230.6) is set. Depending on the mask situation, a Top Level Interrupt request may be generated. Two kinds of masks are available, a Maskable mask and a Non-Maskable mask. The first mask is the bit CICR.TLI (R230.5): it can be set or cleared to enable or disable respectively the Top Level Interrupt request. If it is enabled, the global Enable Interrupt bit CICR.IEN (R230.4) must also be enabled in order to allow a Top Level Request.

The second mask NICR.TLNM (R247.7) is a set-only mask. Once set, it enables the Top Level Interrupt request independently of the value of CICR.IEN and it cannot be cleared by program. Only the processor RESET cycle can clear this bit.

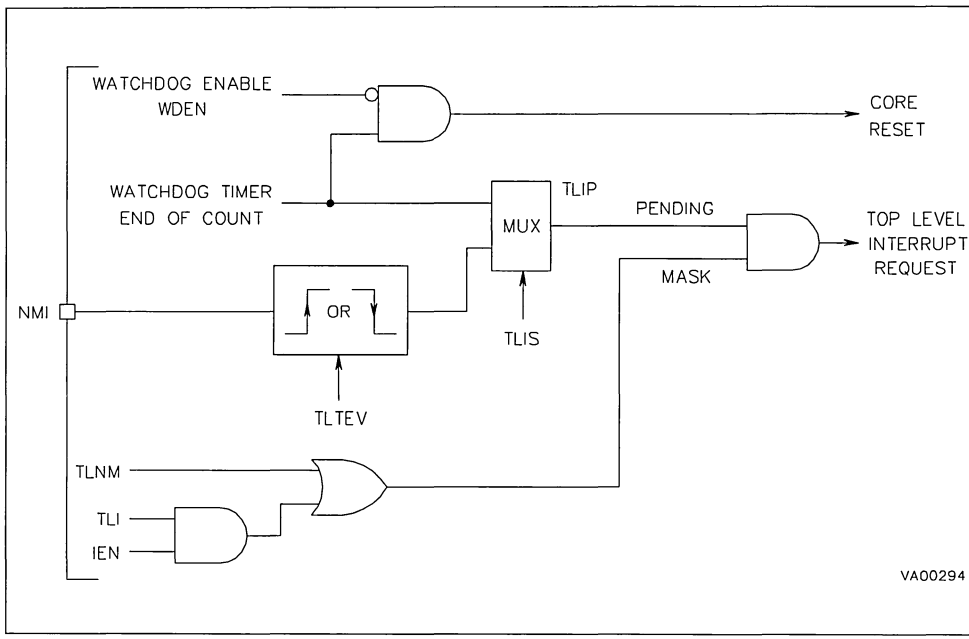
The Top Level Interrupt Service Routine cannot be interrupted by any other interrupt request, in any arbitration mode, even by another Top Level Interrupt request.

Warning. *The interrupt machine cycle of the Top Level Interrupt does not clear the CICR.IEN bit, and the corresponding* `iret` *does not set it.*

4.7 ON-CHIP PERIPHERAL INTERRUPTS

The general structure of the peripheral interrupt unit is described here, however each on-chip peripheral has its own specific interrupt unit containing one or more interrupt channels, or DMA channels. Please refer to the specific peripheral chapter for the description of its interrupt features and control registers.

Figure 4-13. Top Level Interrupt Structure

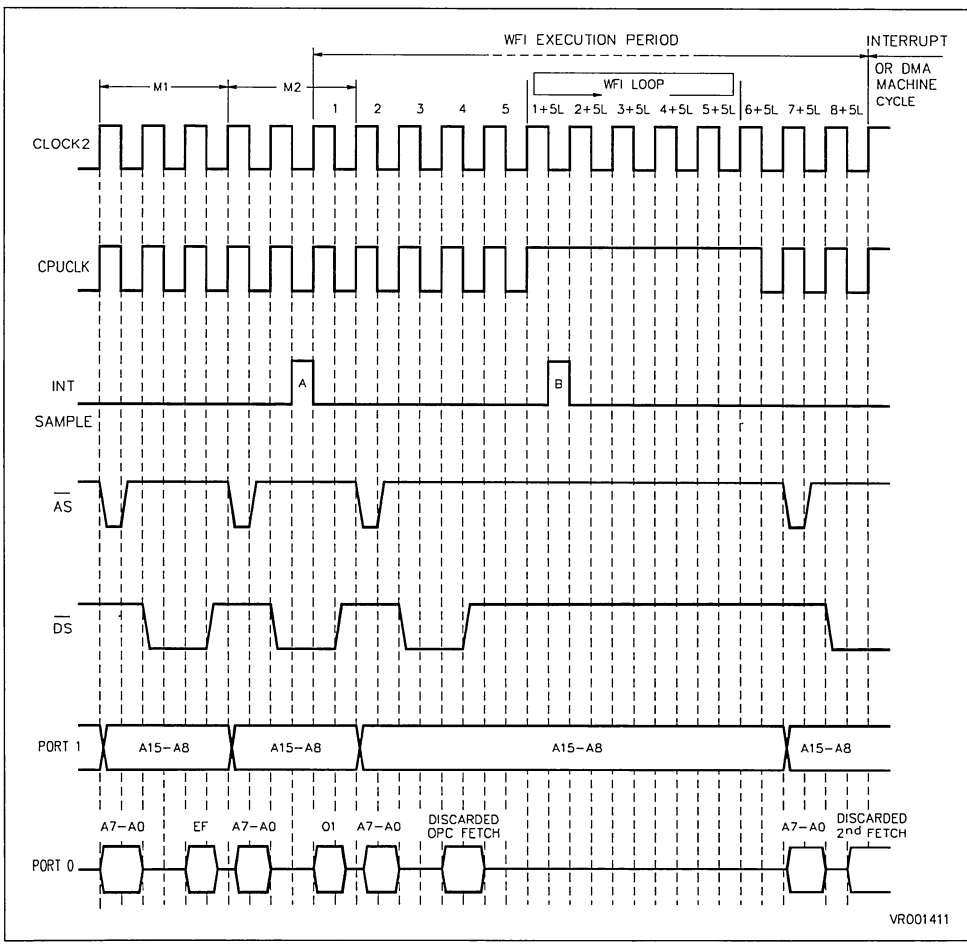


ON-CHIP PERIPHERAL INTERRUPTS (Continued)

The on-chip peripheral interrupt channels provide the following control bits:

- Interrupt Pending bit (IP)
  Set by hardware when the Trigger Event occurs. Can be set/cleared by software to generate/cancel pending interrupts and give the status for Interrupt polling.
- Interrupt Mask bit (IM)
  If IM = "0", no interrupt request is generated. If IM ="1" an interrupt request is generated whenever IP = "1" and CICR.IEN = "1".
- Priority Level (PRL, 3 bits)
  These bits define the source priority level
  PRL=0: the highest priority
  PRL=7: the lowest priority (the interrupt cannot be acknowledged)
- Interrupt Vector Register (IVR, up to 7 bits)
  The IVR points to the vector table which itself contains the interrupt routine start address.

Figure 4-14. Wait For Interrupt Timing



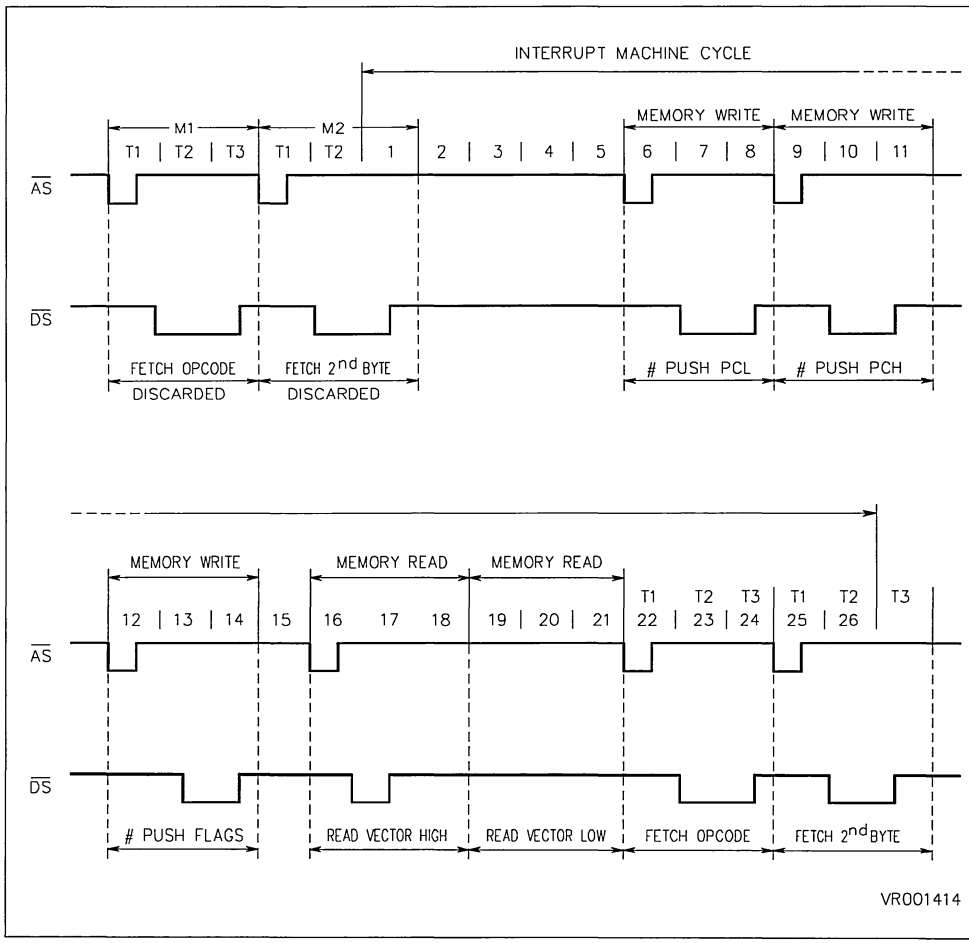
4.8 WAIT FOR INTERRUPT INSTRUCTION

The Wait For Interrupt instruction suspends program execution until an interrupt request is acknowledged. During the WFI instruction, the CPUCLK is halted while INTCLK keeps running. Under this state, the power consumption of the processor is lowered by the CORE power consumption value.

4.9 INTERRUPT RESPONSE TIME

Interrupt requests are sampled 6 CPUCLK cycles before the end of the instruction. If Wait For Interrupt is in progress, requests are sampled every 5 CPUCLK cycles. If the interrupt request comes from an external pin, the programmed event has to be set a minimum of one CPUCLK cycle before the sampling time.

Figure 4-15. Interrupt Acknowledge Timing



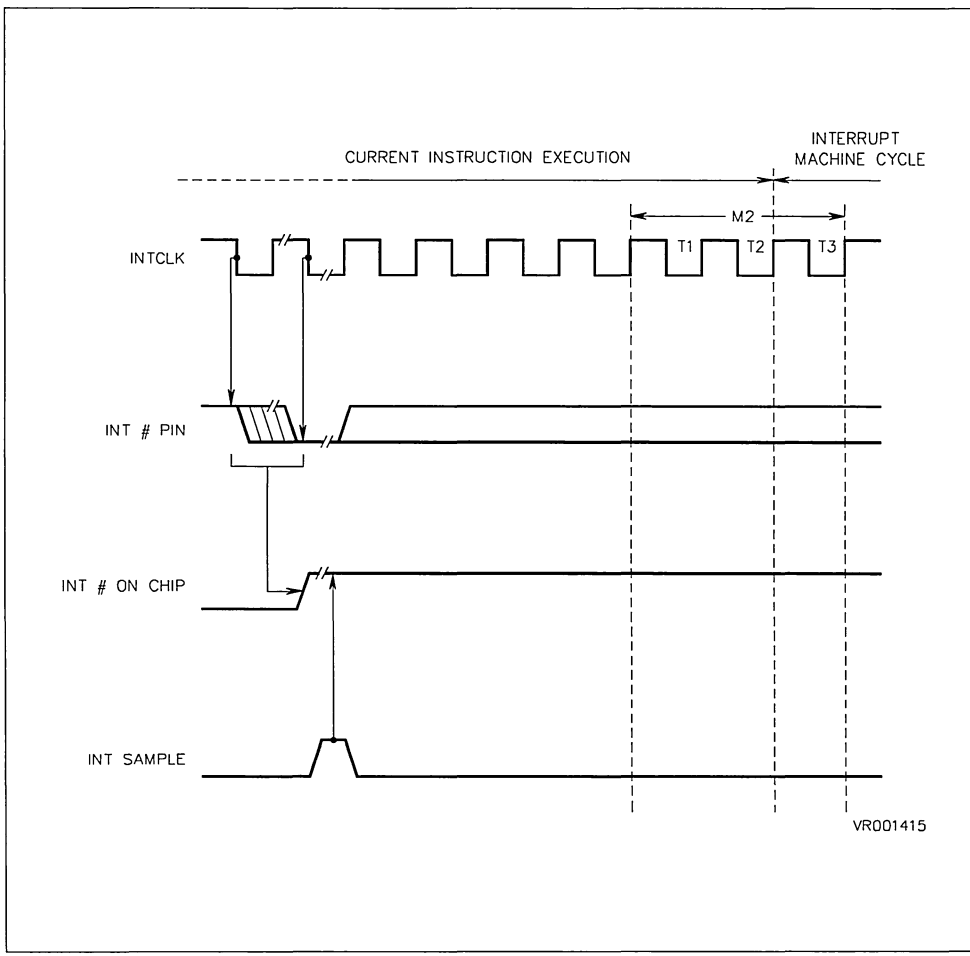
INTERRUPT RESPONSE TIME (Continued)

In order to guarantee the falling/rising edge detection, input signals must be kept low/high for a minimum of one CPUCLK cycle.

An interrupt machine cycle takes 26 internal clock cycles (CPUCLK), with some exceptions as follows:

- 28 internal clock cycles (CPUCLK), if a Wait For Interrupt is in progress
- 32 internal clock cycles (CPUCLK), if the acknowledge cycle follows a DMA transfer with Register File

Figure 4-16. External Interrupt Response Time



4.10 INTERRUPT REGISTERS

CICR R230 (E6h) System Read/Write
Central Interrupt Control Register

Reset value: 1000 0111 (87h)

7							0
GCEN	TLIP	TLI	IEN	IAM	CPL2	CPL1	CPL0

b7 = **GCEN:** *Global Counter Enable bit.* When set the 16 bit MultiFunction Timers are enabled (see Timer Control Register in MULTI FUNCTION TIMER chapter)

b6 = **TLIP:** *Top Level Interrupt Pending bit.* Set by hardware when the Trigger Event occurs. Cleared by hardware when the Top Level Interrupt is acknowledged.

b5 = **TLI:** *Top Level Interrupt bit.* If TLI ="1", and IEN is set, a Top Level Interrupt request is generated as TLIP is set. If TLI = "0", a request is generated only if TLNM is set.

b4 = **IEN:** *Interrupt Enable.* If IEN = "0", no maskable Interrupt requests are generated. This bit is cleared by the interrupt machine cycle and it is set by the IRET instruction of maskable routines.

b3 = **IAM:** *Interrupt Arbitration Mode.* If IAM = "0", Concurrent Arbitration Mode is selected; If IAM = "1" Nested Mode is selected.

b2-b0 = **CPL2, CPL1, CPL0:** *Current Priority Level.* Defines the Current Priority Level under service. CPL=0 is the highest priority. CPL=7 is the lowest priority. This bits may be modified directly by the interrupt hardware when the Nested Interrupt Mode is used.

EITR R242 (F2h) Page 0 Read/Write
External Interrupt Trigger Event Register

Reset value: XXXX 0000 (00h)

7							0
TED1	TED0	TEC1	TEC0	TEB1	TEB0	TEA1	TEA0

If TExy bit is set, the pending bit will be set upon the rising edge of the input signal.

If TExy is cleared, the pending bit will be set upon the falling edge of the input signal.

All bits are set/reset only by software.

b7 = **TED1:** *Trigger Event of Interrupt Channel D1*

b6 = **TED0:** *Trigger Event of Interrupt Channel D0*

b5 = **TEC1:** *Trigger Event of Interrupt Channel C1*

b4 = **TEC0:** *Trigger Event of Interrupt Channel C0*

b3 = **TEB1:** *Trigger Event of Interrupt Channel B1*

b2 = **TEB0:** *Trigger Event of Interrupt Channel B0*

b1 = **TEA1:** *Trigger Event of Interrupt Channel A1*

b0 = **TEA0:** *Trigger Event of Interrupt Channel A0*

IDPR R243 (F3h) Page 0 Read/Write
External Interrupt Pending Register

Reset value: 0000 0000 (00h)

7							0
IPD1	IPD0	IPC1	IPC0	IPB1	IPB0	IPA1	IPA0

b7 = **IPD1:** *Interrupt Pending bit Channel D1*

b6 = **IPD0:** *Interrupt Pending bit Channel D0*

b5 = **IPC1:** *Interrupt Pending bit Channel C1*

b4 = **IPC0:** *Interrupt Pending bit Channel C0*

b3 = **IPB1:** *Interrupt Pending bit Channel B1*

b2 = **IPB0:** *Interrupt Pending bit Channel B0*

b1 = **IPA1:** *Interrupt Pending bit Channel A1*

b0 = **IPA0:** *Interrupt Pending bit Channel A0*

IP bits are hardware set upon the occurence of the trigger event and are reset by the interrupt acknowledge machine cycle.

Note. IP bits may be set by the programmer to implement a software interrupt.

INTERRUPT REGISTERS (Continued)

EIMR R244 (F4h) Page 0 Read/Write
External Interrupt Mask-bit Register

Reset value: 0000 0000 (00h)

7							0
IMD1	IMD0	IMC1	IMC0	IMB1	IMB0	IMA1	IMA0

EIMR bits are set/reset by software

When the IM bit is set (and the global IEN is enabled), an interrupt request is generated if the corresponding IP bit is set. When IM = "0", no request will be generated.

- IMxy = "1": an interrupt request can be acknowledged (depending on IEN)
- IMxy = "0": an interrupt request is masked.

b7 = **IMD1:** *Interrupt Mask of Interrupt Channel D1*

b6 = **IMD0:** *Interrupt Mask of Interrupt Channel D0*

b5 = **IMC1:** *Interrupt Mask of Interrupt Channel C1*

b4 = **IMC0:** *Interrupt Mask of Interrupt Channel C0*

b3 = **IMB1:** *Interrupt Mask of Interrupt Channel B1*

b2 = **IMB0:** *Interrupt Mask of Interrupt Channel B0*

b1 = **IMA1:** *Interrupt Mask of Interrupt Channel A1*

b0 = **IMA0:** *Interrupt Mask of Interrupt Channel A0*

EIPLR R245 (F5h) Page 0 Read/Write
External Interrupt Priority Level Register

Reset value: 1111 1111 (FFh)

7							0
PL2D	PL1D	PL2C	PL1C	PL2B	PL1B	PL2A	PL1A

EIPLR bits are set/reset by software

b7-b6 = **PL1D, PL2D:** *Priority level for the Group INTD0, INTD1*

b5-b4 = **PL1C, PL2C:** *Priority level for the Group INTC0, INTC1*

b3-b2 = **PL1B, PL2B:** *Priority level for the Group INTB0, INTB1*

b1-b0 = **PL1A, PL2A:** *Priority level for the Group INTA0, INTA1*

EIVR R246 (F6h) Page 0 Read/Write
External Interrupt Vector Register

Reset value: xxxx 0110 (X6h)

7							0
V7	V6	V5	V4	TLTEV	TLIS	IAOS	EWEN

b7-b4 = **V7 to V4:** *Most significant nibble of External Interrupt Vector.* Not initialized by reset.

b3 = **TLTEV:** *Top Level Trigger Event bit* When set, the Top Level event is triggered on rising edge of NMI input pin. Triggering on the falling edge of the NMI input pin is activated when this bit is "0" (reset value)

b2 = **TLIS:** *Top Level Input Selection bit* This bit selects the source of the Top Level Interrupt between the external NMI pin (when "1", the reset value) and the Timer/Watchdog End of Count (when "0").

b1 = **IA0S:** *Interrupt A0 Selection bit* When set, the External Interrupt pin is selected as the External Interrupt Channel A0 source. When reset the source is the Timer/Watchdog End of Count interrupt.

b0 = **EWEN:** *External Wait Enable bit* When set, this bit enables the WAIT input pin to stretch the external memory access cycle. For more details of the WAIT mode, the reader should refer to the Clock and Wait chapter or External memory Interface chapter.

NICR R247 (F7h) Page 0 Read/Write
Nested Interrupt Control Register

Reset value: 0000 0000 (00h)

7							0
TLNM	HL6	HL5	HL4	HL3	HL2	HL1	HL0

b7 = **TLNM:** *Top Level Not Maskable.*

If TLNM = "1", a top level request is generated as TLIP is set. Once TLNM is set, it can be cleared only with an hardware reset

bx = **HLx:** *Hold Level x* These bits are set to "1" when, in Nested Mode, an interrupt service routine at level x is interrupted from a request with higher priority (other than the Top Level interrupt request). It is cleared by the `iret` execution when the routine at level x is recovered.

5 ON-CHIP DMA

5.1 INTRODUCTION

The ST9 includes on chip Direct Memory Access (DMA) channels to provide high-speed data transactions between peripherals and memory or the Register File. Multi-channel DMA is fully supported by peripherals with their own controller and DMA channel(s). Each DMA channel transfers data to or from contiguous locations of the Register File, Program Memory or Data Memory. The maximum number of transactions that each DMA channel can perform is 222 if the Register File is selected, or 65536 if Program or Data Memory is selected.

The DMA controller in the Peripheral uses an indirect addressing mechanism to DMA Pointers and Counter Registers stored in the Register File. This is the reason that the maximum number of transactions for Register File is 222, as two Registers are allocated for the Pointer and Counter. Register pairs are used for memory pointers and counters to offer the full 65536 byte and count capability.

The ST9 supports a fully programmable DMA priority structure included within the interrupt structure.

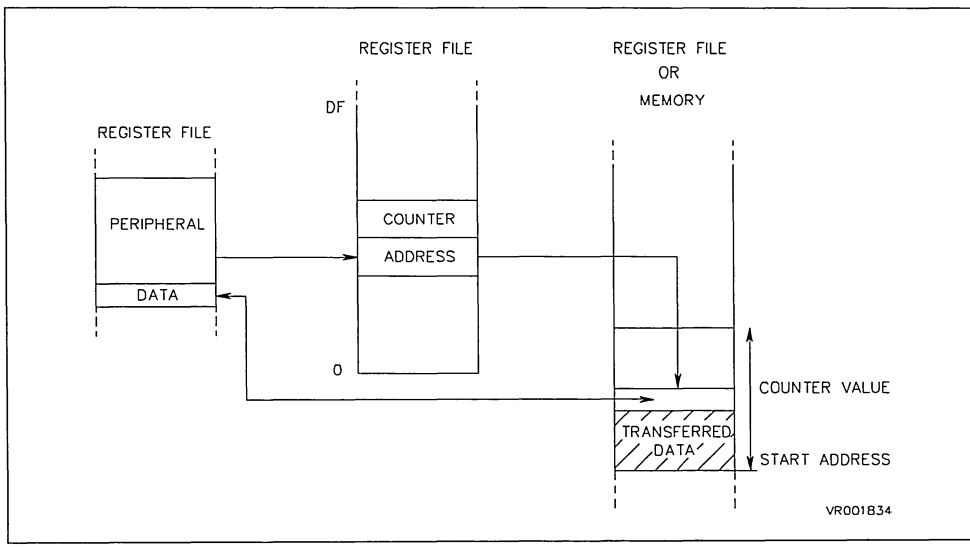
5.2 DMA PRIORITY LEVEL ARCHITECTURE

The 8 priority levels used for interrupts are also used to prioritize the DMA requests, which are arbitrated in the same arbitration phase as interrupt requests. If the event occurrence requires a DMA transaction, this will take place at the end of the current instruction execution. When an interrupt and a DMA request occur simultaneously, on the same priority level, the DMA request is serviced before the interrupt.

Note however that an interrupt priority request must be *higher* than the CPL value in order to be acknowledged. For a DMA transaction request, it must be *equal to* or *higher than* the CPL value in order to be executed.
Thus only DMA transaction requests can be acknowledged when CPL = 7.

DMA requests do not modify the CPL value as the DMA transaction is non-interruptable.

Figure 5-1. DMA Overview



DMA TRANSACTIONS (Continued)

Figure 5-2. DMA Between Registers and peripherals

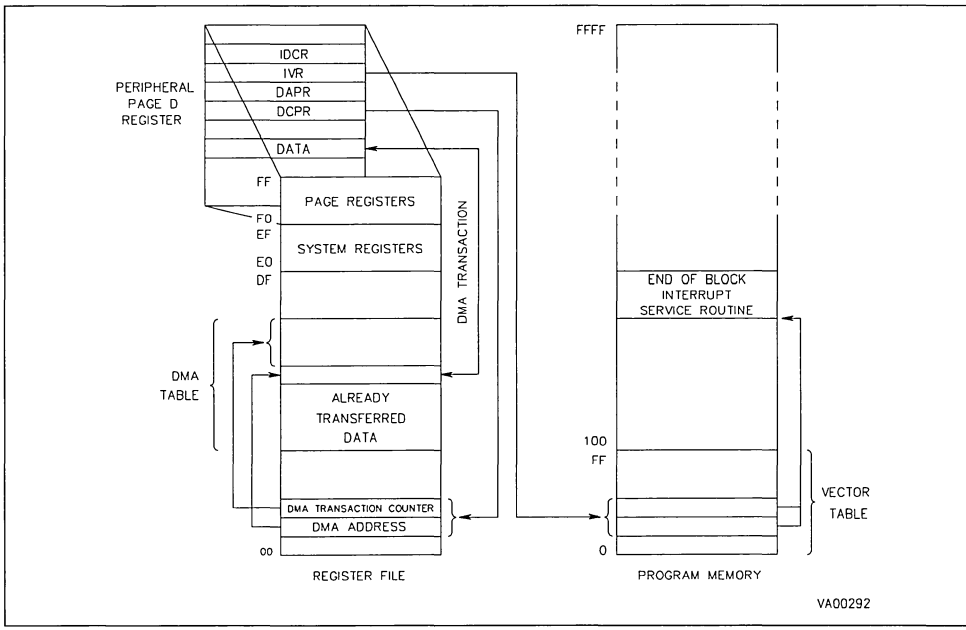
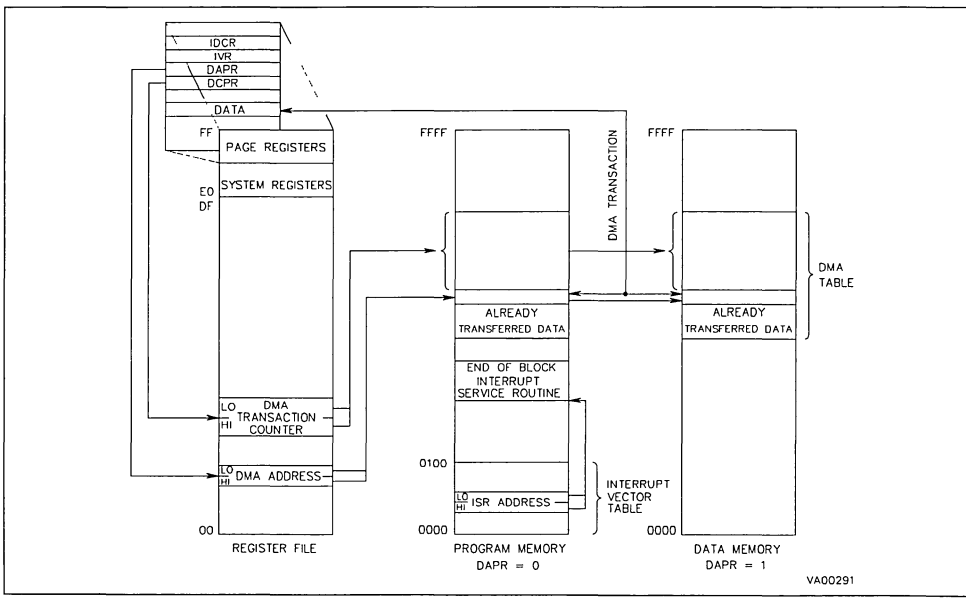


Figure 5-3. DMA Between Memory and peripherals



5.3 DMA TRANSACTIONS

The purpose of on-chip DMA channel is to transfer a block of data from/to the peripheral to/from Register File or Memory. Each DMA transfer consists of three operations:

- A load from/to the peripheral data register to a location of Register File (or Memory) addressed through the DMA Address Register (or Register pair)
- A post-increment of the DMA Address Register (or Register pair)
- A post-decrement of the DMA transaction counter, which contains the number of transactions that have still to be performed.

If the DMA transaction is made between **the peripheral and the Register File** (Figure 5-2), one register is required to hold the DMA Address and one to hold the DMA transaction counter. These two registers must be located in the Register File: the DMA Address Register in the even addressed register, the DMA transaction Counter in the following register (odd address). They are pointed to by the DMA Transaction Counter Pointer Register (DCPR) located in the page registers of the peripheral. In order to select the DMA transaction with the Register File, the control bit DCPR.RM (bit 0 of DCPR) must be set.

The Transaction Counter Register must be initialized with the number of DMA transfers to perform and will be decremented after each transaction. The DMA Address Register must be initialized with the starting address of the DMA table in the Register File, and is increased after each transaction. These two registers must be located between addresses 00h and DFh of the Register File.

If the transaction is made between **the peripheral and Memory Space (Program or Data Memory)**, a register pair (16 bits) is required for the DMA Address and for the DMA transaction Counter (Figure 5-3. Thus, two register pairs must be located in the Register File. The DMA Transaction Counter is pointed to by the DMA Transaction Counter Pointer Register (DCPR), the DMA Address is pointed to by the DMA Address Pointer Register (DAPR), both DCPR and DAPR are located in the page registers of the peripheral. When selecting the DMA transaction with memory, the control bit DCPR.RM (bit 0 of DCPR) must be cleared to "0".

To select between Program or Data Memory, the control bit DAPR.DP (bit 0 of DAPR) must be cleared or set respectively.

Once the DMA table is completed (the transaction counter reaches 0 value), an Interrupt request to the CPU is generated.

The DMA transaction Counter must be initialized with the number of transactions to perform and will be decremented after each transaction. The DMA Address must be initialized with the starting address of the DMA table and is increased after each transaction. These two register pairs must be located betweeen addresses 00h and DFh of the Register File.

Once a DMA channel is initialized, a transfer can start. The direction of the transfer (data from/to peripheral to/from memory or Register File) is automatically defined by the type of peripheral and programming mode.

When the Request Pending bit is set by a hardware event (or by software), and the DMA Mask bit is set, a DMA request is generated. If the Priority Level of the DMA source is higher than or equal to the Priority Level under service (CPL) the DMA transfer is executed at the end of the current instruction. DMA transfer reads/writes data from/to the location pointed by the DMA Address Register, increments the DMA Address register and decrements the Transaction Counter Register. When the content of the Transaction Counter is decremented to zero, the DMA Mask bit (DM) is cleared and an interrupt request is generated according to the Interrupt Mask bit (End of Block interrupt). This End-of-Block interrupt request is taken into account depending on the PRL value.

***WARNING**. DMA request are not acknowledged if the top level interrupt service is in progress.*

5.4 DMA CYCLE TIME

DMA and Interrupt requests are sampled 6 INTCLK cycles before the end of the instruction. If Wait For Interrupt is in progress, requests are sampled every 5 INTCLK cycles. DMA transactions are executed if their priority allows it.

A DMA transfer with the Register file takes 8 CPUCLK cycles, except when the Wait For Interrupt is in progress (10 CPUCLK cycles).

A DMA transfer with the memory takes 16 CPUCLK cycles except when the Wait For Interrupt is in progress (18 CPUCLK cycles).

Figure 5-4. DMA Transaction to Memory

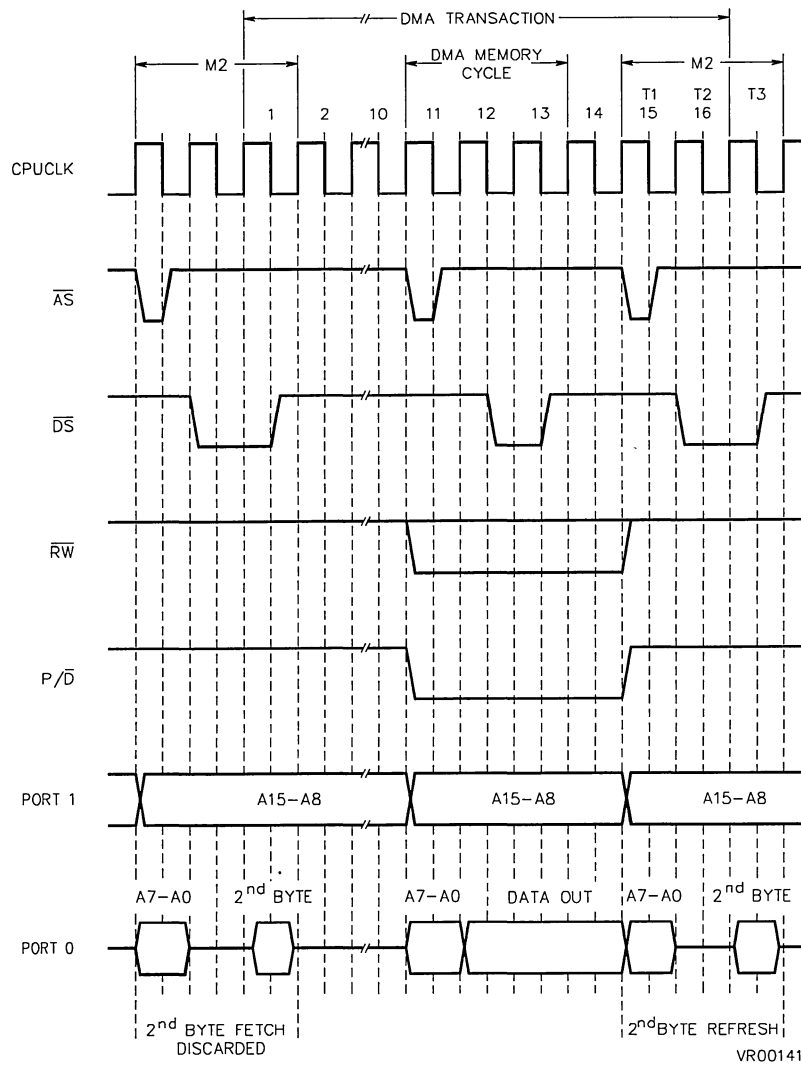
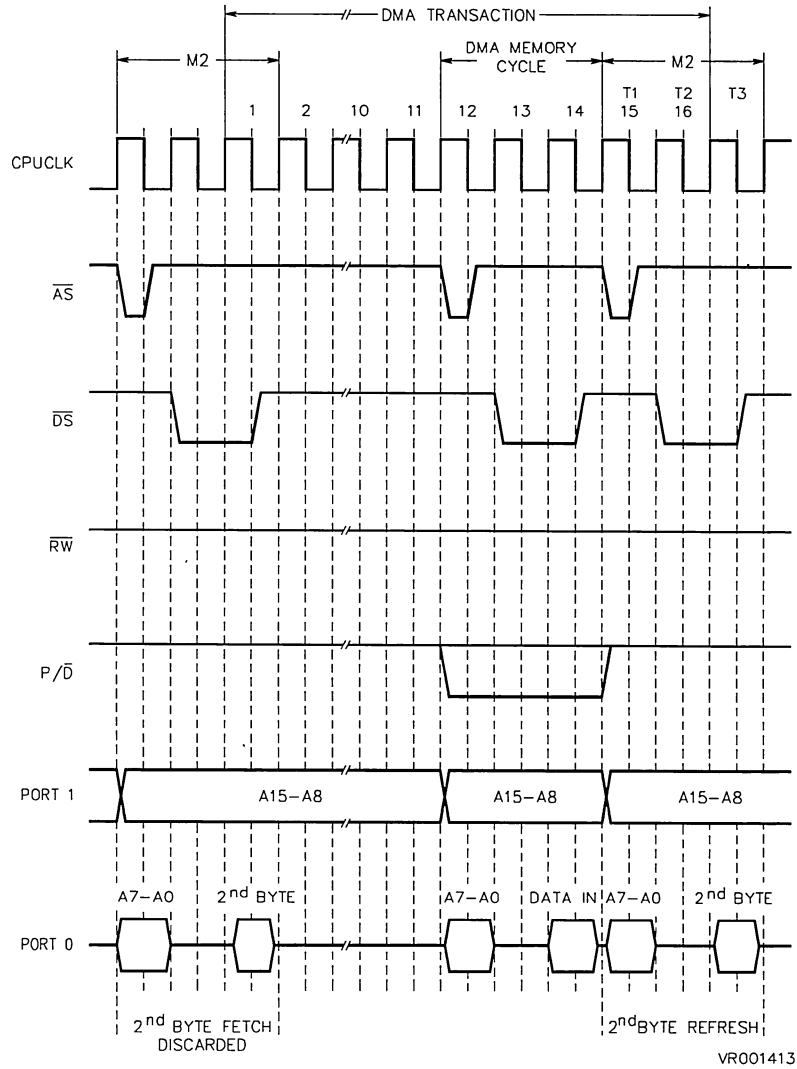


Figure 5-5. DMA Transaction from Memory



5.5 THE SWAP-MODE

An extra feature of ST9 DMA channels of some peripherals (i.e the MultiFunction TIMER) is the SWAP mode. This feature allows transaction from two DMA tables alternatively. All the DMA descriptors in the Register File are thus doubled. Two DMA transaction counters and two DMA address pointers allow the definition of two fully independent tables (they only have to belong the same space, Register file or Data memory or Program memory). The DMA transaction is programmed to start on one of the two tables (say table 0) and, at the end of block, the DMA controller automatically swaps to the other table (table 1) by pointing to the other DMA descriptors. In this case, the DMA mask (DM bit) control bit is not cleared, but the End Of Block interrupt request is generated to allow the optional updating of the first data table (table 0).

Until the swap mode is not disabled, the DMA controller will continue to swap between DMA Table 0 and DMA Table 1.

5.6 DMA REGISTERS

As each peripheral DMA channel has its own control registers, the following register list should be considered as a general example. The names and register bit allocation shown here may be different from those found in the peripheral chapters.

DCPR Address set by Peripheral Read/Write
DMA Counter Pointer Register

Reset value: undefined

7							0
C7	C6	C5	C4	C3	C2	C1	RM

b7-b1 = **C7-C1:** *DMA Transfer Counter Register(s) Address*

b0 = **RM:** *Register File/Memory Selector* If set, the DMA transactions are done with the Register File; if cleared, the DMA transactions are done with the Program or Data memory (see DAPR.DP)

IDCR Address set by Peripheral Read/Write
Generic Peripheral Interrupt and DMA Control

Reset value: undefined

7							0
		IP	DM	IM	PRL2	PRL1	PRL0

b5 = **IP:** *Interrupt Pending.* Set by hardware when the Trigger Event occurs. Cleared by hardware when the request is acknowledged for DMA cycles and external interrupts only. Can be set/cleared by software in order to generate/cancel a pending request. Identical in function to IP of ICR.

b4 = **DM:** *DMA Mask.* If DM = "0" no DMA request is generated when the trigger event occurs. This bit is cleared whenever the transaction counter reaches zero (unless SWAP mode is active).

b3 = **IM:** *Interrupt Mask.* If IM = "0" no interrupt request is generated. If IM = "1" DMA requests depend on DM bit value as shown above.

b2-b0 = **PRL2, PRL1, PRL0:** *Priority Level* Definition of the source priority level. PRL = 0 is highest priority. If PRL = 7, no interrupt can be acknowledged, DMA requests will be.

DAPR Address set by Peripheral Read/Write
DMA Address Pointer Register

Reset value: undefined

7							0
A7	A6	A5	A4	A3	A2	A1	DP

b7-b1 = **A7-A1:** *DMA Address Register(s) Address*

b0 = **DP:** *Data/Program Memory Selector:* (DAPR.RM is "0") if set the DMA transactions are made with the Data Memory; if cleared the DMA transactions are made with the Program Memory.

6 CLOCK

6.1 INTRODUCTION

The ST9 Clock generator module generates the internal clock for the ST9 core and the on-chip peripherals. The Clock generator can be driven by an external crystal circuit, connected to the OSCIN and OSCOUT pins, or by an external pulse generator, connected to OSCIN.

6.2 CLOCK MANAGEMENT

The oscillator circuit generates an internal clock signal with the same period and phase as at the OSCIN input pin. The maximum frequency allowed is 24MHz.

As shown in Figure 6-6, the CLOCK1 signal drives a programmable divider by two. If the control bit MODER.DIV2 (R235.5) is set, the internal clock CLOCK2 is CLOCK1 divided by two; otherwise, if DIV2 bit is cleared, the clock signal CLOCK2 has the same period and phase as CLOCK1. CLOCK2 drives the internal clock INTCLK delivered to all ST9 on-chip peripherals and acts as the central timebase for all timing functions (eg Multifunction Timer or Serial Communications Interface Baud Rate generator).
The maximum frequency allowed for INTCLK is 12MHz. For internal operating frequencies above 8MHz, it is recommended to work with the Clock Divider active in order to provide a duty cycle of 50% for INTCLK.

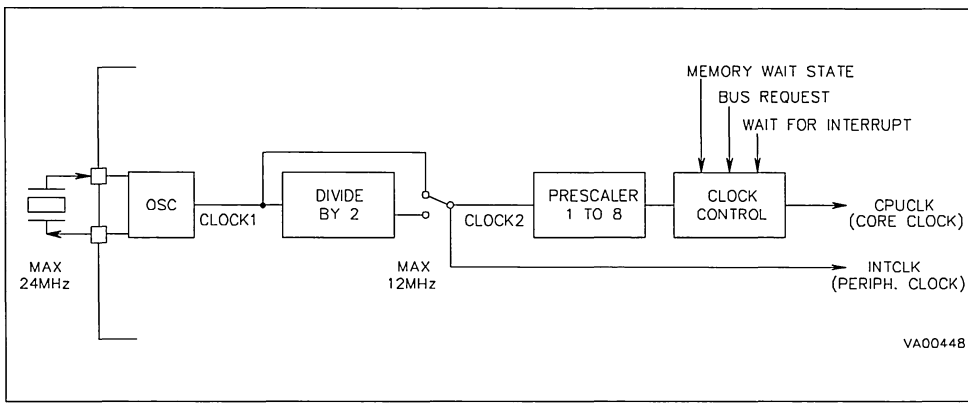
CLOCK2 also drives a programmable prescaler which generates the basic time base, CPUCLK, for the instruction executer of the ST9 core. This allows the user to slow the program execution time to reduce power dissipation, and to locally speed up certain code segments for time critical routines. The internal peripherals are not affected by the CPUCLK prescaler. The prescaler value divides the input clock by the value programmed in the control bits MODER.PRS2,1,0 (R235.4,3,2). If the prescaler value is zero, no prescale is made, thus CPUCLK has the same period and phase as CLOCK2 and INTCLK. If the value is different from 0, the prescaling is equal to the value plus one, ranging thus from two (PRS2,1,0 = 1) to eight (PRS2,1,0 = 7). The clock generated is shown in Figure 6-2. It must be noticed that the prescaling of the clock does not keep the duty cycle to 50%, but stretches the high level of the clock until completion.

When External Memory Wait (or Bus Request or Wait for Interrupt) events occur, CPUCLK is stretched on the high level for the whole period required by the function.

Note. The added wait cycles refer to the INTCLK frequency and not the original CPUCLK.

Figure 6-3 shows an example of a memory access cycle with the CPUCLK prescaled by 2 and with 5 added Wait states.

Figure 6-1. Peripheral and Core Clocks



6.3 CLOCK CONTROL REGISTER

The ST9 clock division by 2 and the clock prescaling are controlled by the MODER register.

Note. This register contains bits with other functions. Only the bits relating to control of the clock are shown here.

MODER R235 (EBh) System Read/Write Mode Register

Reset Value : 1110 0000 (E0h)

7							0
X	X	DIV2	PRS2	PRS1	PRS0	X	X

b5 = **DIV2:** *OSCIN Divided by 2.* This bit controls the divide by 2 circuit which operates on the OSCIN Clock. A logical "1" value means that the OSCIN clock is internally divided by 2, and a logical "0" value means that no division of the OSCIN Clock occurs.

b4-b2 = **PRS2, PRS1, PRS0:** *Prescaling of ST9 Clock.* These bits define the prescaler value used to prescale the CPUCLK from INTCLK. When these three bits are reset, the CPUCLK is not prescaled, and is equal to INTCLK; in all other cases, the internal clock is prescaled by the value of (PRS2,1,0 + 1).

Figure 6-2. Core Clock Prescaling

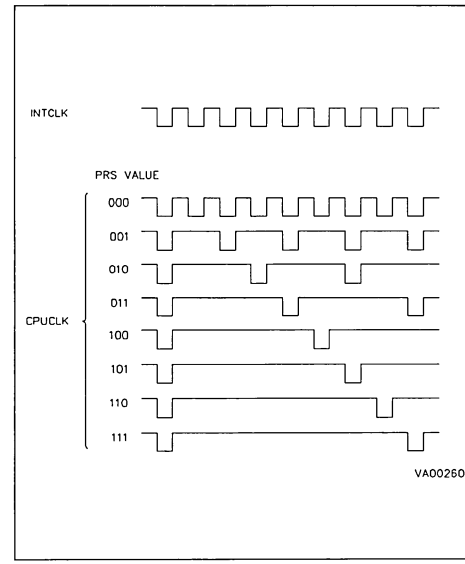
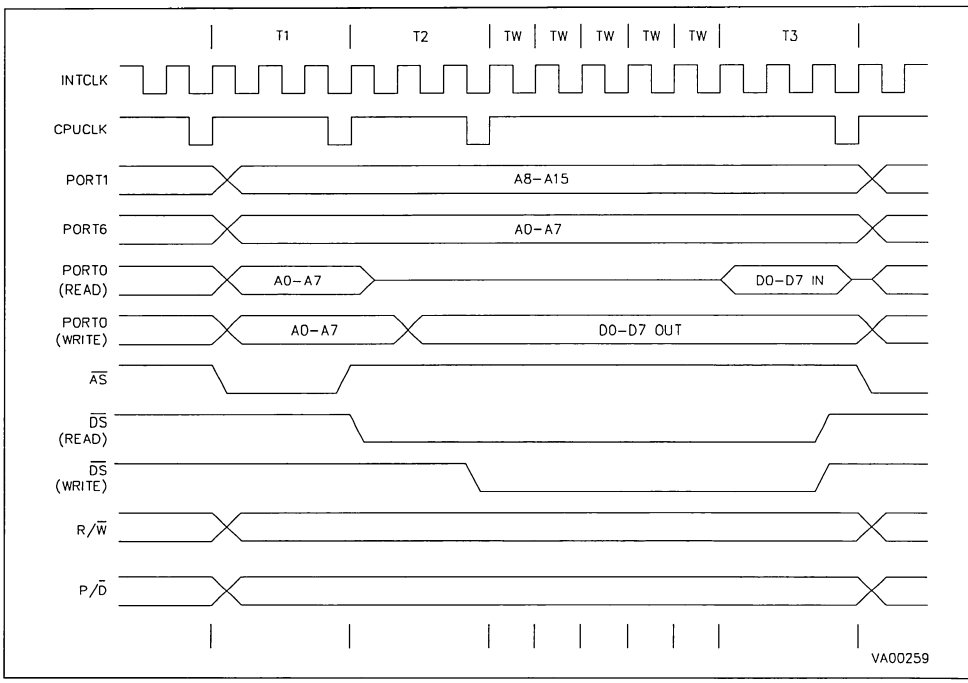


Figure 6-3. Memory Access with a Clock Prescaler by 2 and 5 Wait Cycle



6.4 OSCILLATOR CHARACTERISTICS

The on-chip oscillator circuit (Figure 6-4) is an inverting gate circuit.

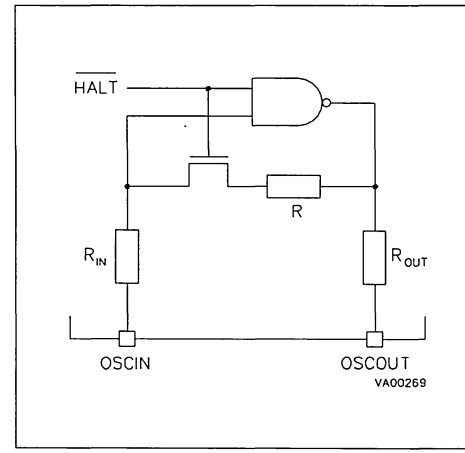
Note. Owing to the Q factor required, Ceramic Resonators may not provide a reliable oscillator source.

In Halt mode, set by means of the HALT instruction, the parallel resistor R is disconnected and the oscillator is disabled, forcing the internal clock CLOCK1 to a high level and OSCOUT to a low level.

To exit the HALT condition and restart the oscillator, an external RESET pulse is required of a minimum duration of 12ms (Figure 6-7).

It must be noted that if the Timer/Watchdog watchdog function is enabled, a HALT instruction will not disable the oscillator. This to avoid stopping the watchdog if, by an error, a HALT code is executed. When this occurs, the ST9 CPU falls into an endless loop ended by the watchdog (or external) reset.

Figure 6-4. Internal Oscillator Schematic



Note: 300Ω < RIN < 700Ω
ROUT< 50W R> 1.5MΩ R > 50Ω

Figure 6-5. Crystal Oscillator

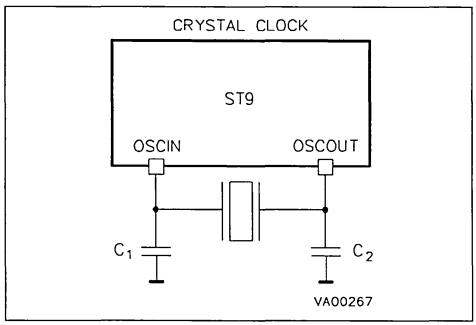


Figure 6-6. External Clock

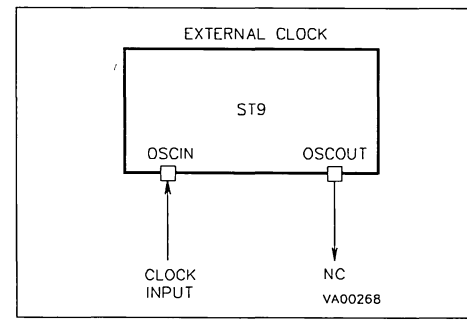


Table 6-1. Crystal Specification (C0 < 7pF)

Frequency (MHz)	C1=C2=56pF Rs Max	C1=C2=47pF Rs Max	C1=C2=22pF Rs Max
24	20	25	70
16	40	60	150
12	80	100	250
8	180	240	600
4	700	800	600

Table 6-2. Oscillator Transductance

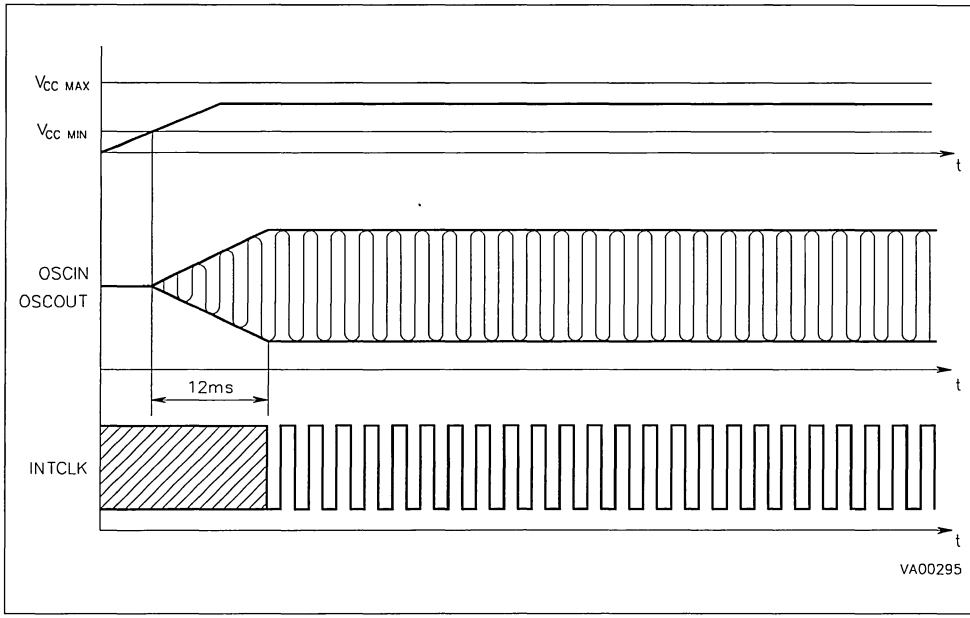
gm	Min	Typ	Max
mA/V	3	5.8	9.5

Legend:
Rs. Parasitic Series Resistance of the quartz crystal (upper limit)
C0: Parasitic capacitance of the quartz crystal (upper limit, < 7pF)
C1, C2: Maximum Total Capacitances on pins OSCIN and OSCOUT(the value includes the external capacitance tied to the pin plus the parasitic capacitance of the board and of the device)
gm Transconductance of the oscillator.

Notes. The tables are relative to the fundamental quartz crystal only (not ceramic resonator)

OSCILLATOR CHARACTERISTICS (Continued)

Figure 6-7. Oscillator Start-up Sequence



7 RESET

7.1 INTRODUCTION

The processor Reset overrides all the other conditions and forces the ST9 to the reset state. During Reset, the internal registers are set to the reset value, as shown in Table 7-1 for the system and Page 0 Registers and the I/O pins are set to the Bidirectional Weak Pull-up mode (see Warning). The programmer must then initialize the ST9 system and peripheral control registers to give the required functions.

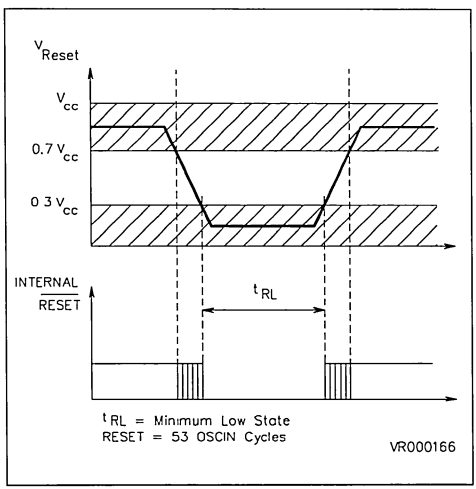
7.2 RESET GENERATION

The reset condition can be generated by the external pin $\overline{\text{RESET}}$ or by the on-chip Timer/Watchdog.

The on-chip Timer/Watchdog generates a reset condition if the watchdog mode is enabled (WCR.WDEN cleared, R252 page 0), and if the programmed period elapses without the specific code (AAh,55h) written to the appropriate register. The input pin RESET is not driven low by the on-chip reset generated by the Timer/Watchdog.

During reset, the $\overline{\text{DS}}$ output signal is kept low and the $\overline{\text{AS}}$ output is toggled with the crystal frequency (input at OSCIN) divided by 32. This condition is recognized by off-chip Z-bus peripherals as a reset condition.

Figure 7-1. Signal to be applied on Reset Pin



7.3 RESET PIN TIMING

The $\overline{\text{RESET}}$ pin has a Schmitt trigger input circuit with hysteresis. The internal reset is generated by the external pin synchronized with the internal clock. The power up reset circuit must keep the $\overline{\text{RESET}}$ input low for a minimum of the crystal startup period plus 53 crystal periods.

Once the $\overline{\text{RESET}}$ pin reaches a logical "1", the processor exits from the reset status after 67 crystal periods from the rising edge ($\overline{\text{DS}}$ is set). The processor then fetches from Program Memory locations 0 and 1 (power-on reset vector) and begins program execution from the address contained in the vector. If the ST9 is a ROMLESS version, without on-chip program memory, ports Port0, Port1 (and Port6 for ST905x family) are set to external memory mode (i.e Alternate Function) and the memory accesses are made to external Program memory with wait cycles insertion.

WARNING: *I/O pins are set to the Weak Pull-up mode during the Reset cycle. This state is forced during the reset sequence, but the I/O pins can be in a random state for up to 64 crystal periods. The application circuit must take this into account if it can lead to critical situations in the external circuitry.*

7.4 PROCESSOR SYNCHRONIZATION UNDER RESET

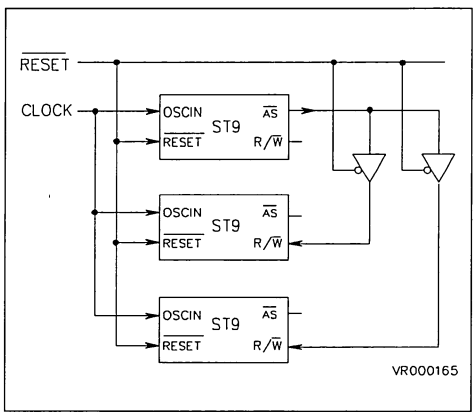
During reset, a specific procedure has been implemented to synchronize two or more oscillators in a multi-micro ST9 based system, for example a majority voting high reliability system. Figure 7-2 shows the principle schematic for the multi-microprocessor synchronization. The master processor delivers the synchronous signal, output at its $\overline{\text{AS}}$ pin, to the R/$\overline{\text{W}}$ pin of the slave processors. The R/$\overline{\text{W}}$ pin is, under reset status, set to input with a weak (10kΩ typical) pull up resistor. The slave processor(s) synchronizes its internal clock phase with the clock received at its R/$\overline{\text{W}}$ pin. To guarantee the phase synchronization, the reset status must be at least 32x31 = 992 crystal periods. All the processors must have the same input clock.

RESET (Continued)

Table 7-1. Internal Registers Reset Values

Register Number	System Register	Reset Value	Page 0 Register	Reset Value
F	(SSPLR)	undefined	Reserved	
E	(SSPHR)	undefined	(SPICR)	00h
D	(USPLR)	undefined	(SPIDR)	undefined
C	(USPHR)	undefined	(WCR)	7Fh
B	(MODER)	E0h	(WDTCR)	12h
A	(Page Ptr)	undefined	(WDTPR)	undefined
9	(Reg Ptr 1)	undefined	(WDTLR)	undefined
8	(Reg Ptr 0)	undefined	(WDTHR)	undefined
7	(FLAGR)	undefined	(NICR)	00h
6	(CICR)	87h	(EIVR)	x2h
5	(PORT5)	FFh	(EIPLR)	FFh
4	(PORT4)	FFh	(EIMR)	00h
3	(PORT3)	FFh	(EIPR)	00h
2	(PORT2)	FFh	(EITR)	00h
1	(PORT1)	FFh	(EEPROM)	xx00 0000b
0	(PORT0)	FFh	Reserved	

Figure 7-2. Synchronization Under Reset

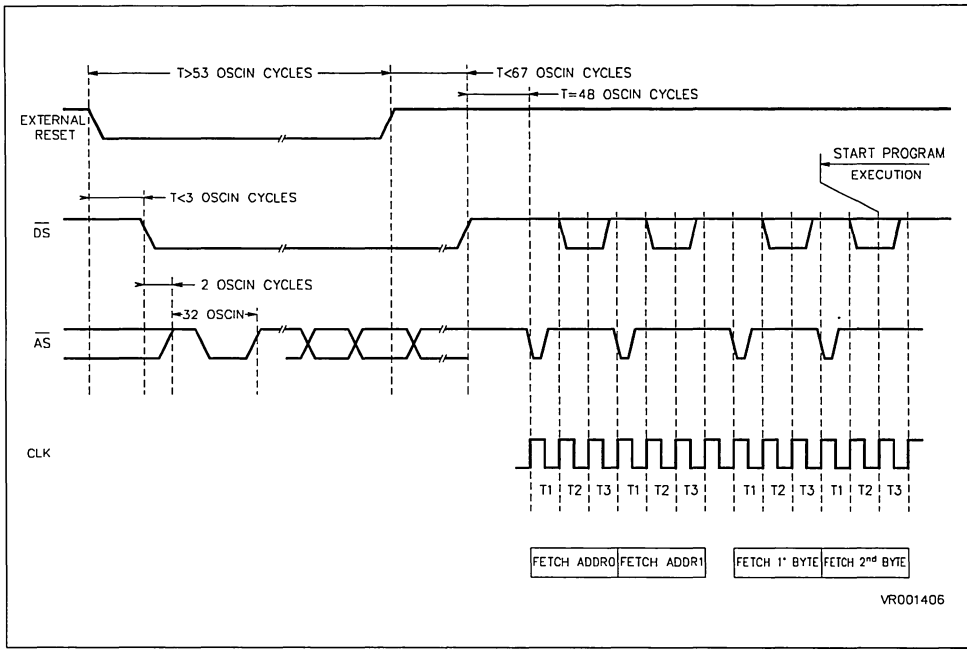


7.5 EPROM PROGRAMMING PIN

The ST9 versions with on-chip EPROM memory require an external programming voltage Vpp to perform the programming procedure. The Vpp voltage must be applied to the RESET pin during the whole programming phase. Refer to the EPROM Programming Board Manual for specifications.

RESET (Continued)

Figure 7-3. Exit From Reset Timing



8 EXTERNAL MEMORY INTERFACE

8.1 INTRODUCTION

In the event of an application requiring more ROM space than available on-chip, or for easier program management and customization with external memory or peripherals, the ST9 microcontroller provides an external memory interface. The external memory interface provides the memory lines and timing and status control signals, plus enhanced features including programmable memory wait cycles, bus request/acknowledge cycles and shared memory bus access control.

The ST9 Memory Control Unit automatically recognizes if a memory location belongs to on-chip memory. When the memory location is on-chip, it performs a machine cycle without $\overline{DS}$ generation, and the access is performed on-chip. If the location does not belong to on-chip memory, an access to off-chip memory is performed (generating the $\overline{DS}$ low pulse) through the Ports 0, 1 (and 6 for ST905x family).

During Reset, $\overline{AS}$ and $\overline{DS}$ are driven to perform external peripherals reset and to implement, in conjunction with the R/$\overline{W}$ pin, a multi-microprocessor synchronization procedure (see Clock and Reset chapters).

8.2 CONTROL SIGNALS

$\overline{AS}$
Address Strobe (Output, Active low, Tristate). The rising edge of $\overline{AS}$ indicates that Memory Address, R/$\overline{W}$ and P/$\overline{D}$ Memory signals are valid.
$\overline{AS}$ is released in high-impedance during a Bus acknowledge cycle or under processor control by setting the HIMP bit (MODER.0).

$\overline{DS}$
Data Strobe (Output, Active low, Tristate). Data Strobe provides the memory data timing during external memory access cycle. When internal memory is accessed, $\overline{DS}$ is kept high during the whole memory cycle. During an External memory write cycle, the data output at Port 0 is valid when $\overline{DS}$ is active. During a read cycle, the data at Port 0 must be valid before the trailing edge of $\overline{DS}$.
$\overline{DS}$ is released in high-impedance during a Bus Acknowledge cycle or under processor control by setting the HIMP bit (MODER.0).

R/$\overline{W}$
Read/Write (Output, Active low, Tristate). The R/$\overline{W}$ output signal identifies the type of memory cycle. When R/$\overline{W}$ = "1", the memory cycle is a Memory Read cycle; when R/$\overline{W}$ = "0", it is a Memory Write Cycle. R/$\overline{W}$ output signal is defined at the beginning of the memory cycle and is stable until the next Memory cycle.
R/$\overline{W}$ is released in high-impedance during Bus acknowledge cycle or under processor control by setting the HIMP bit.

P/$\overline{D}$
Program/Data Memory (Alternate Function Output, Active low). The P/$\overline{D}$ output signal selects between Program and Data Memory. When P/$\overline{D}$ = "1", the memory referenced by the processor is the Program Memory; when P/$\overline{D}$ = "0", the memory referenced is the Data Memory. The P/$\overline{D}$ output signal is defined at the beginning of the memory cycle and is stable until the next Memory cycle.
P/$\overline{D}$ is enabled by software as the Alternate Function output of a parallel port bit (refer to the Pin Configuration and Alternate Function tables to identify the specific port and pin).

$\overline{WAIT}$
External Memory Wait (Input, Active low). The $\overline{WAIT}$ input signal indicates to the ST9 that the external memory requires more time to complete the memory access cycle. The memory cycle will then be stretched. $\overline{WAIT}$ is enabled by setting EWEN (EIVR.0 R246 Page 0).

$\overline{BREQ}$
Bus Request (Input, Active low). The $\overline{BREQ}$ input signal indicates to the ST9 that a bus request has tried or is trying to gain control of the memory bus. $\overline{BREQ}$ is enabled by setting BRQEN (MODER.1 R235).

$\overline{BACK}$
Bus Acknowledge (Alternate Function Output, Active low). The $\overline{BACK}$ output signal indicates that the ST9 has relinquished control of the memory bus in response to a bus request.

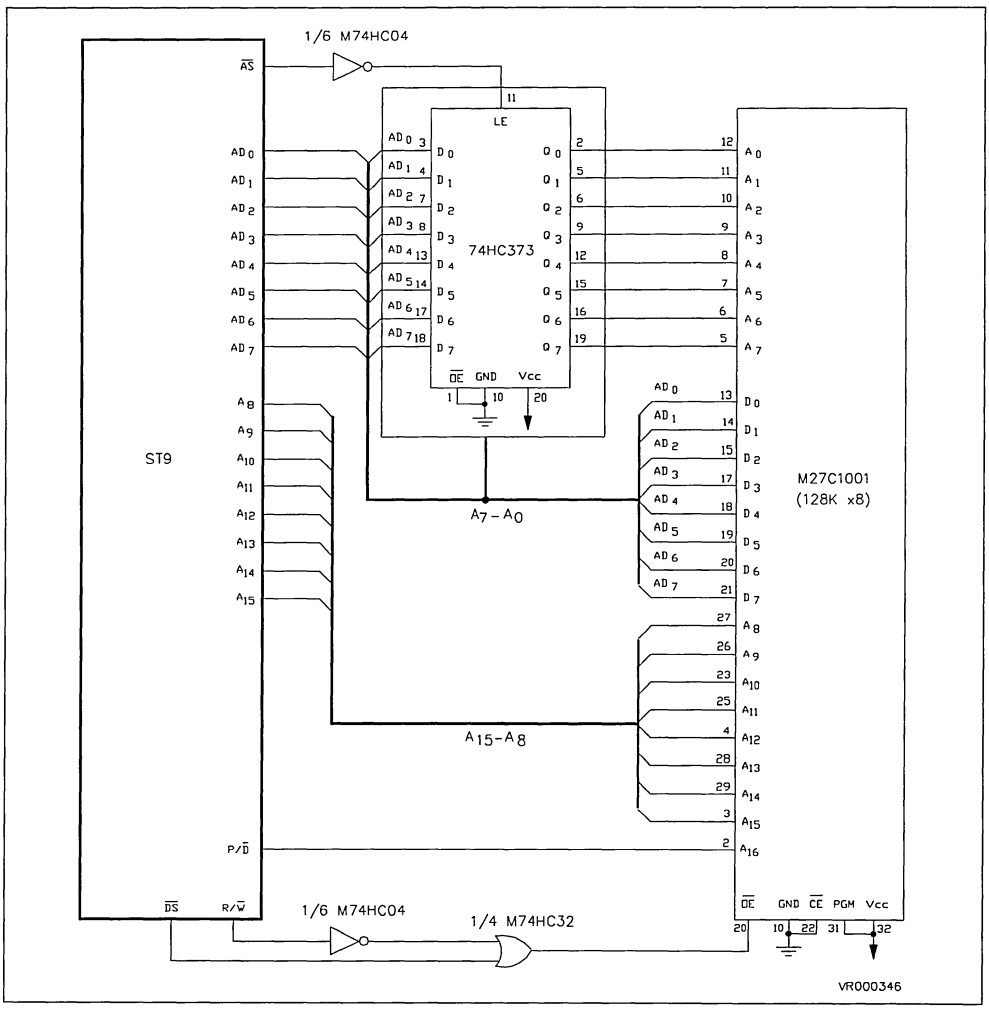
CONTROL SIGNALS (Continued)

P0
Port 0 (Input/Output, Push-Pull/Open-Drain/Weak Pull-up). Port0 can be configured as a bit programmable Parallel I/O port or as External Memory interface for multiplexed Low-Address/Data (A0-7/D0-7).

P1
Port 1 (Input/Output, Push-Pull/Open-Drain/Weak Pull-up). Port1 can be configured as a bit programmable Parallel I/O port or as External Memory interface for the High-Address (A8-A15).

P6 *(When available)*
Port 6 (Input/output, Push-Pull/Open-Drain/Weak Pull-up). This port, when available, can be configured as bit programmable Parallel I/O port or as External Memory interface for the Low-Address (A0-7), allowing a non-multiplexed memory bus capability.

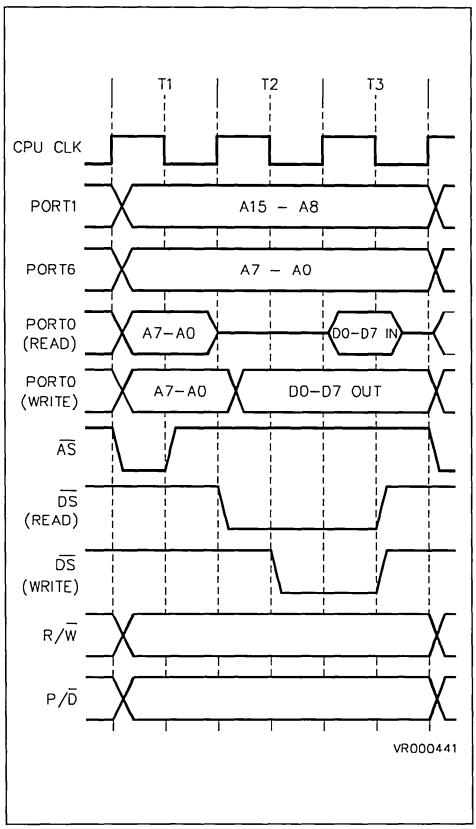
Figure 8-1. ST9 Accessing External Program and Data Memory.



8.3 MEMORY ACCESS CYCLE

Each memory access cycle is composed of three CPUCLK phases: T1, T2, T3. During phase T1, the memory address is output upon $\overline{AS}$ falling edge and is valid upon the rising edge of $\overline{AS}$. Port1 and Port6 maintain the address stable until the next T1 phase.
If the Memory access cycle is a Read cycle, Port0 pins are released to high impedance with the falling edge of $\overline{DS}$ until the next $\overline{AS}$ falling edge.
If the Memory access is a Write cycle, Port0 is held active, the data is output during T2 and is maintained until the next address is output (upon the falling edge of $\overline{AS}$). DS is pulled low during T2 only if the Memory access is an External Memory access. If the memory cycle is a Memory Read, it is pulled low at the beginning of T2. If it is a Memory Write, $\overline{DS}$ is kept low from the middle of T2 until the middle of T3.

Figure 8-2. External Memory Read/Write.



8.4 STRETCHED ACCESS CYCLE

The ST9 can interface to memory with slow access times by automatically inserting additional Wait cycles during the External Memory cycle. On-chip memory accesses do not require $\overline{WAIT}$ cycles and run at the full speed of CPUCLK.

Three Wait cycle sources are available:

- The input pin WAIT from external sources
- The internal programmable Wait cycle generator
- Internal memories with stretched access cycle (EEPROM)

The input pin $\overline{WAIT}$ (when enabled) is sampled on the CPUCLK falling edge of phase T2. If active (low), one INTCLK clock cycle will be added. During the added clock cycle, the $\overline{WAIT}$ pin is sampled again. CPUCLK is stretched for as long as the $\overline{WAIT}$ input is active.

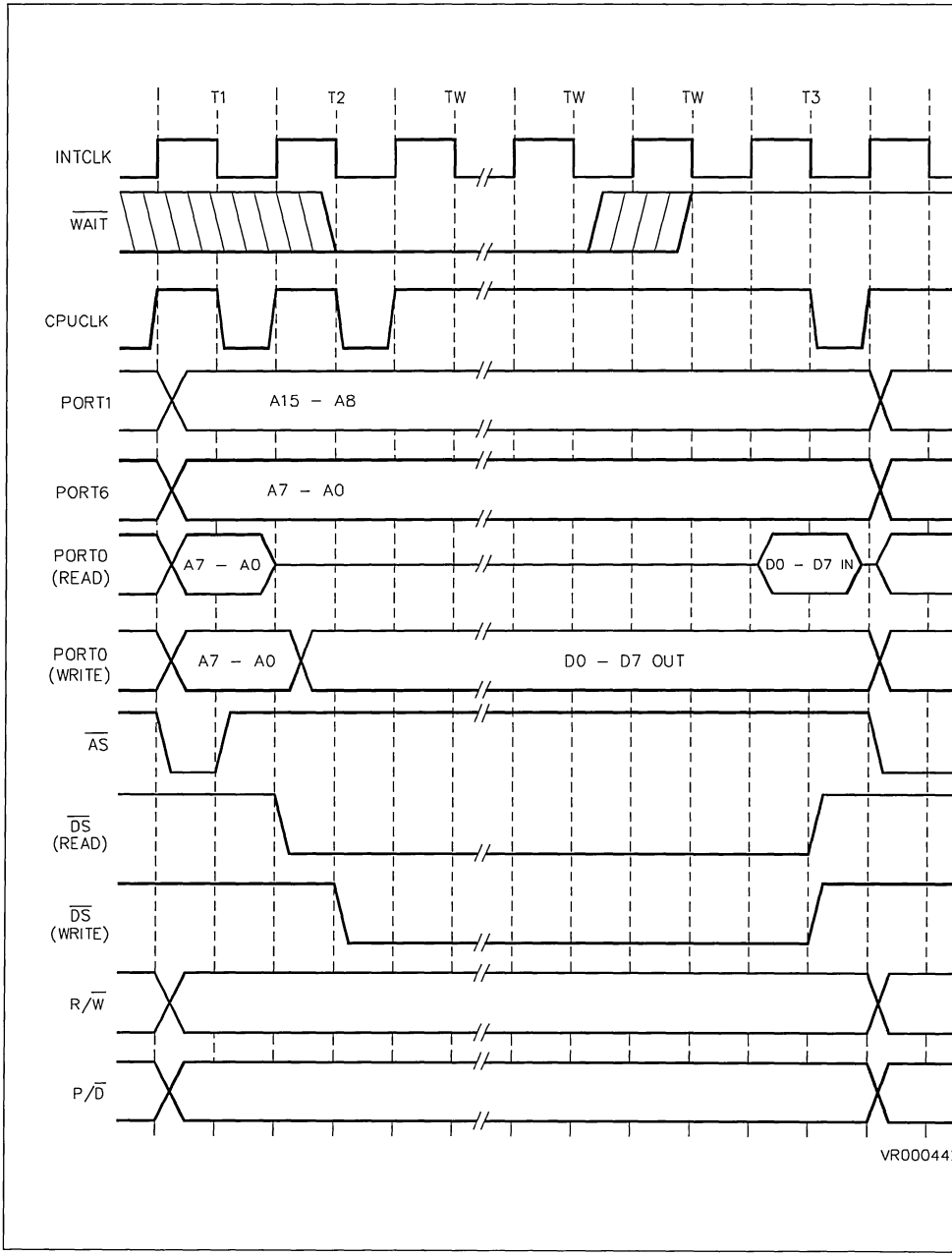
The internal programmable $\overline{WAIT}$ cycle generator allows the extension of the External Memory cycle automatically by the programmed number of $\overline{WAIT}$ cycles. Two three bit fields in the Wait Control Register WCR (R252 Page 0) allow the stretching of Program and Data Memory access cycles independently by 0 to 7 cycles. WPM2,1,0 (WCR.5,4,1) contain the number of Program memory wait cycles to be added, WDM2,1,0 (WCR.2,1,0) contain the number of Data memory wait cycles to be added.

Table 8-1. Number of wait cycles added

WDM2 WPM2	WDM1 WPM1	WDM0 WPM0	Nb of Clock cycle added	Note
0	0	0	0	No Wait cycle
0	0	1	1	
0	1	0	2	
0	1	1	3	
1	0	0	4	
1	0	1	5	
1	1	0	6	
1	1	1	7	Reset Value

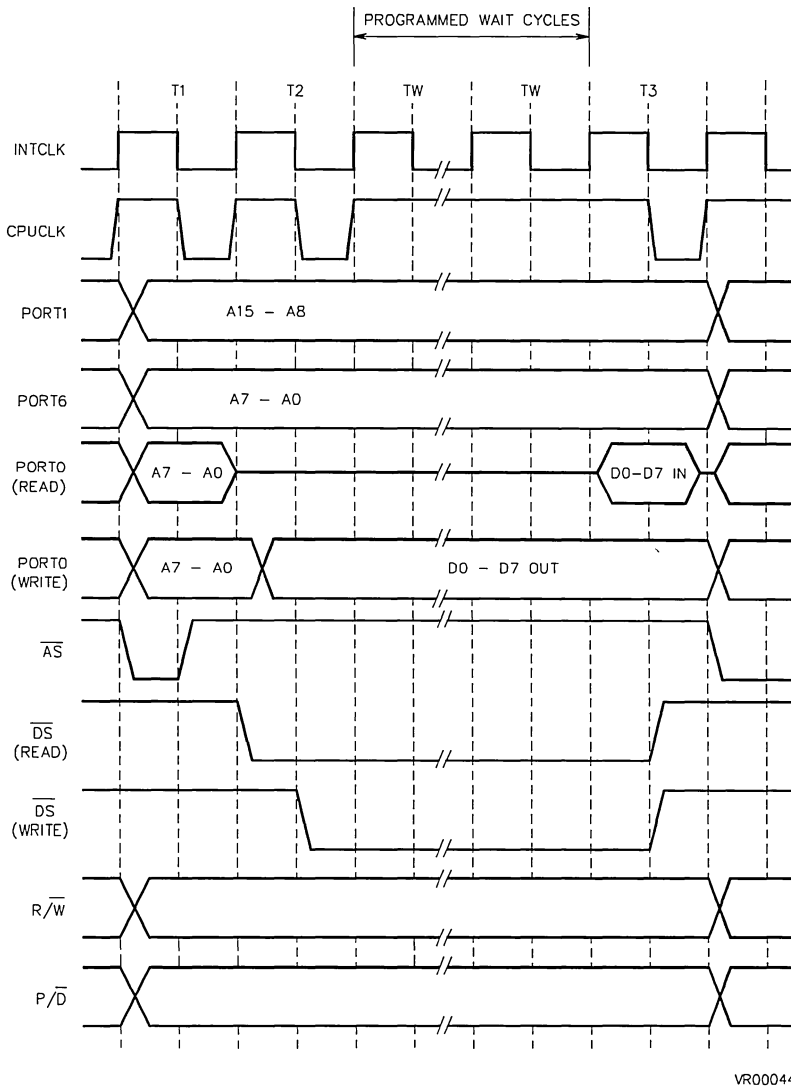
STRETCHED ACCESS CYCLE (Continued)

Figure 8-3. External Memory Read/Write Sequence with External Wait.



STRETCHED ACCESS CYCLE (Continued)

Figure 8-4. External Memory Read/Write Sequence with Programmable Wait.



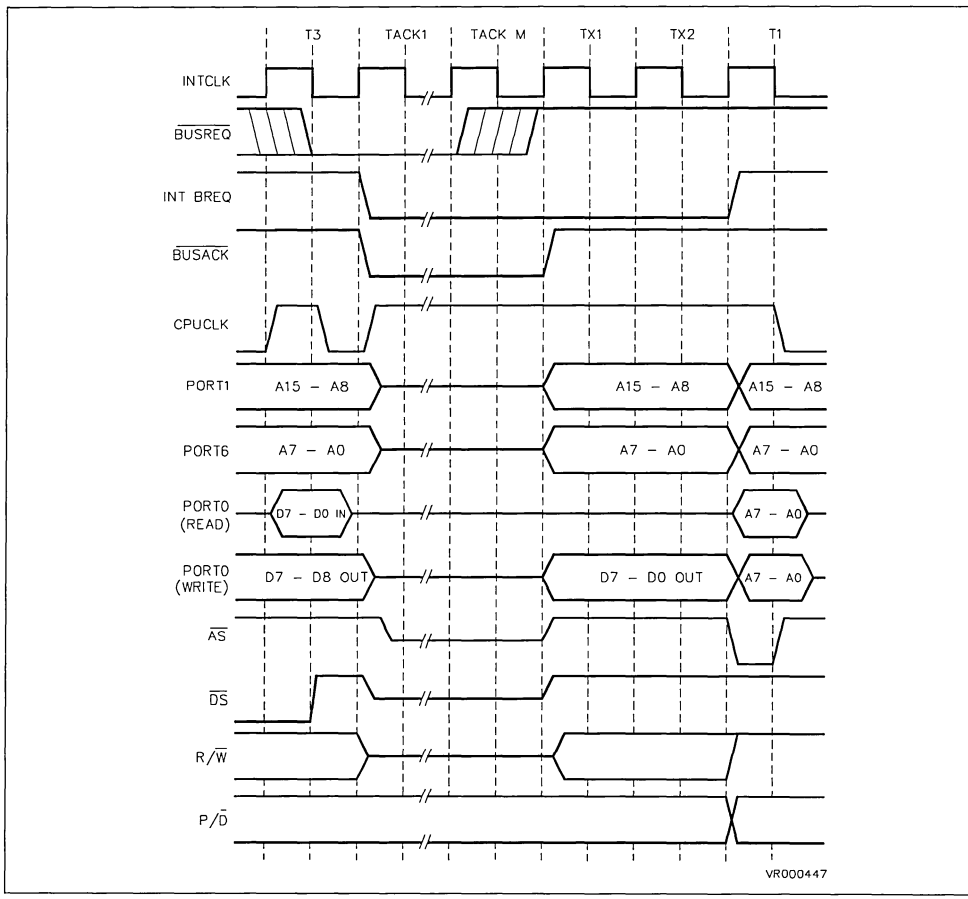
8.5 SHARED BUS

When the ST9 runs in a multi-master bus system, it is necessary to release the bus control to other bus master(s). This operation can be performed by the Bus Request/Acknowledge capability supported by the ST9.

Once enabled by setting BRQEN (MODER.1 R235), $\overline{BREQ}$ is sampled by the ST9 upon the falling edge of the internal clock during the phase T3. When the $\overline{BREQ}$ signal is sampled low, the CPUCLK clock is stretched and the External Memory signals ($\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, P0, P1(, P6 for ST905x)) are released to high-impedance. The input signal $\overline{BREQ}$ is then continuously monitored, and when it is sampled high the External Memory interface pins are driven again by the ST9 after two additional internal clock cycles. These cycles are used to fully drive and propagate the control and data signals through the external memory bus before CPUCLK is restarted.
The output signal $\overline{BACK}$ is driven low during the whole period when the External Memory interface is released to high impedance.

Under the Reset status, the bits of the I/O port(s) associated to $\overline{BREQ}$ and $\overline{BACK}$ are set to Bidirectional Weak pull-up mode and the enable bit BRQEN is cleared. To enable this function, the program must set the $\overline{BACK}$ port as an Alternate Function output and enable (set to "1") the bit BRQEN.

Figure 8-5. Bus Request/Acknowledge Timing.



SHARED BUS (Continued)

When it is required to disable the external bus, but to keep the processor running in the on-chip memory, the external memory bus can be disabled by software programming of the HIMP (MODER.0) control bit. By setting HIMP, the External Memory Interface ($\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ and Port0, Port1 (and Port6), if not configured as I/O lines) is set into a high impedance state. In this way, the external memory bus can be used by another resource (e.g. diagnostic equipment or external programming of system memories) and the ST9 program can continue accessing the on-chip memory. This feature can also be useful for high security applications where the flow and addresses of the on-chip security algorithms must not be shown on the external address pins.

When running in internal memory, disabling the external bus will reduce the noise emitted by the micro.

The disabling of the External Memory Interface by setting HIMP = "1" can be interrupt driven by applying the "Bus Request" input signal to an External Interrupt pin. In this case the bus disable response time will be longer than the automatic system using the $\overline{BREQ}$ request, however the ST9 can continue to execute the program written in the on-chip memory.

8.6 PORTS P0, P1, P6 INITIALIZATION AFTER RESET

The Port 0, Port 1 and Port 6 (for ST905x family) initialization after reset depends on the configuration of the ST9 as shown in Table 8-2.

If the device has on-chip Program memory (ROM or EPROM), the ports (or the existing parts of them) are set to Bidirectional Weak Pull-up Mode.

Table 8-2. Port status after Reset

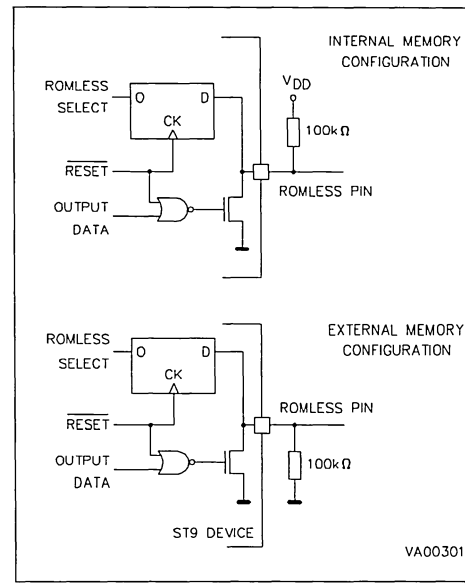
Device	Port 0, 1, 6 Initialization
ROM EPROM	Bidirectional Weak-Pull-Up (PxC0, PxC1, PxC2 = 0,0,0; Data = 1)
ROMLESS	Memory Address and Data Alternate Function Push-Pull (PxC0, PxC1, PxC2 = 1,1,0; Data = 1)

If the device is ROMLESS or a ROM device with the ROMless function enabled, the ports (or the existing part of them) are set to Alternate Function Push-Pull Mode, providing the Address and Data lines to interface to the external Program and Data Memory from Reset.

8.7 ROMLESS FUNCTION

In order to accomodate the use of ROM based ST9 devices in the event of a subsequent ROM code change, a ROMless function may be enabled on a specified Port I/O pin by Mask Option. This function is activated by pulling the ROMless select pin to ground with a 100kohm resistor. This status is latched on the rising edge of the $\overline{RESET}$ pin and, when low, the on-chip PROGRAM memory (ROM or EPROM) is disabled, causing all instruction fetch cycles to be external. On-chip Data memory (RAM or EEPROM) is not affected.
If the ROMless function is enabled by the mask option, and the internal program is to be used, then the ROMless pin must be held to a high level (via a 100kohm resistor to V_{DD}) during the Reset cycle. After the Reset cycle the ROMless pin may be programmed for any I/O or Alternate function.

Figure 8-6. ROMless Selection

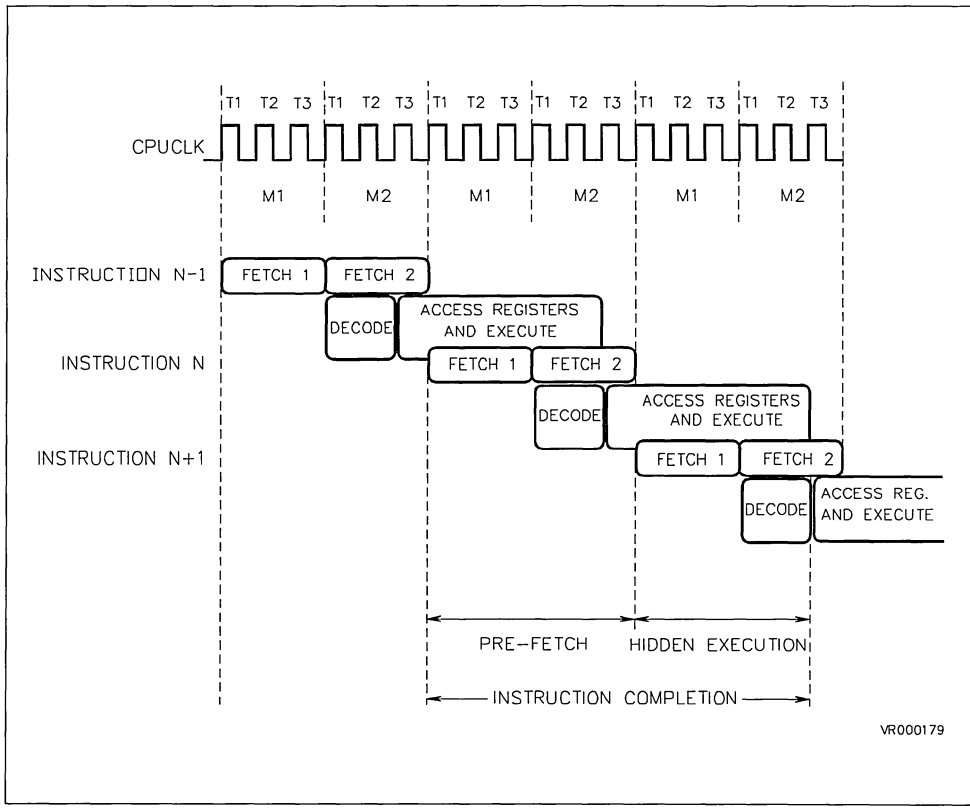


8.8 PIPELINE

The ST9 implements pipe-line stages on instruction fetch and execution in order to increase the execution speed. The instruction execution is in fact hidden by the Memory access cycles: the execution of one instruction is overlapped with the prefetch of the two successive bytes. The fetch of the first byte (opcode) is identified by the machine cycle M1, the fetch of the second byte by M2.

The 2 bytes instructions, whose execution time is 6 CPUCLK cycles, have the instruction execution hidden by the following instruction prefetch. For those instructions that require an execution time longer than the time to prefetch the following bytes, perform memory access during their execution or interrupt the sequential memory access, the pipe is flushed.

Figure 8-7. Instruction Pipe-line Stages



8.9 "SPURIOUS" MEMORY READ ACCESSES

The ST9 in certain cases produces external memory accesses which may be regarded as "Spurious" in their nature. While these do not affect the correct operation of the ST9, these accesses may cause misunderstandings when developing and debugging applications as the signals $\overline{AS}$ and $\overline{DS}$ are produced, and Ports 0, 1, (6 for ST905x) output updated addresses (if used to interface to external memory).

The spurious reading cycle is produced when executing specific instructions. This is one of 4 cases: double reading, reading before writing, reading when the stack is internal or prefetch reading.

- DOUBLE READING
  A memory location read by the ST9 is read two times consecutively (instead of one).

Involved instruction(s):

```
DIV          rr,r   ; divide (16/8)
```

The first byte of the code following `DIV` is fetched two times. The double reading does not occur if the Overflow flag was set by `DIV`, or if Divide by zero was trapped. The P/$\overline{D}$ line remains high during the cycle.

- READING BEFORE WRITING
  A memory location which is to be written to by the ST9 is previously read.

Involved instruction(s):

```
LD           (rr)+,(r)+   ;load (byte)
                          ;Memory, Register
```

The destination memory location is read before being written. The P/$\overline{D}$ line reflects the memory space of the destination memory location.

- READING WHEN THE STACK IS INTERNAL
  If the System and/or User Stack has been programmed to use the Register File, a memory location of Data Space is POPed in parallel.

Involved instruction(s):

```
POP      R   ; POP (byte) from System Stack
POP     (R)  ;      "     "     "
POPU     R   ; POP (byte) from User Stack
POPU    (R)  ;      "     "     "
```

While a byte is being `POP`ed from the Register File, a memory location in Data Space is read in parallel with its address given by SSPHR+SSPLR for `POP` instructions and by USPHR+USPLR for `POPU` instructions. The external data is ignored.

```
POPW     RR   ; POP (word) from System Stack
POPUW    RR   ; POP (word) from User Stack
```

While the high er address byte is being popped from the Register File, a memory location in Data Space is read in parallel with its address given by SSPHR+SSPLR for `POPW` instructions and by USPHR+USPLR for `POPUW` instructions. No spurious reading is made for the lower byte.

```
RET           ; Return from Subroutine
```

While the Program Counter Higher and Lower bytes are `POP`ed from the Register File, two memory locations are read at addresses given by SSPHR+SSPLR.

```
IRET          ; Return from Interrupt
```

While the Program Counter Higher and Lower bytes and the FLAGS are `POP`ed from the Register File, three memory locations are read at addresses given by SSPHR+SSPLR. When working with Internal Stacks, SSPHR and USPHR contents are don't care from the point of view of program execution, but they must be considered RESERVED by the User as the instructions listed in this section perform updating of SSPHR/USPHR, together with the spurious reading.

- PREFETCH READING
  Due to the ST9 Pipeline, instructions which stop the Core or which perform program branches can fetch bytes of the following program code while the pipeline is being flushed.

Involved instruction(s):

```
WFI           ; Wait For Interrupt
```

reads two bytes of the following code.

```
HALT          ; Halt
CALL(rr)      ; Unconditional Call subroutine
```

read one byte of the following code in Program space (P/$\overline{D}$ high).

8.10 REGISTERS

WCR R252 (FCh) Page 0 Read/Write
Wait Control Register

Reset Value: 0111 1111 (7Fh)

7							0
X	WDGEN	WDM2	WDM1	WDM0	WPM2	WPM1	WPM0

b7 = Reserved, reads as a "0".

b6 = **WDGEN:** refer to Timer/Watchdog chapter.

WARNING. Resetting this bit to zero has the effect of setting the Timer/Watchdog to the Watchdod mode. Unless this is desired, this must be set to "1".

b5-b3 = **WDM2-0:** *Data Space Wait Cycles.* These bits contain the number of INTCLK cycles to be added automatically to external Data memory accesses. WDM = 0 gives no additional wait cycles, WDM = 7 provides the maximum 7 INTCLK cycles (this is the reset condition in order to allow the use of slow access time external memory, if faster memory is used, then this value may be modified by the User).

b2-b0 =**WPM2-0:** *Program Space Wait Cycles.* These bits contain the number of INTCLK cycles to be added automatically to external Program memory accesses. WPM = 0 gives no additional wait cycles, WPM = 7 provides the maximum 7 INTCLK cycles (this is the reset condition in order to allow the use of slow access time external memory, if faster memory is used, then this value may be modified by the User).

Note. the number of clock cycles added refer to INTCLK and NOT to CPUCLK.

WARNING. *The Wait Control Register is reset to give the maximum number of Wait cycles for external memory. To give the optimum performance of the ST9 when used in single-chip mode (no external memory) the WDM2,1,0 and WPM2,1,0 bits should be reset to "0".*

9 I/O PORTS

9.1 INTRODUCTION

The ST9 is provided with dedicated lines for input/output. These lines, grouped into 8-bit ports, can be independently programmed to provide parallel input/output, or to carry input/output signals to/from the on-chip peripherals and Core (e.g. Timers and SCI). All ports have active pull-ups and pull-down resistors compatible with TTL loads. In addition, pull-ups can be turned off for open-drain operation and weak pull-ups can be turned on to save off-chip resistive pull-ups. Input buffers can be either TTL or CMOS compatible.

9.2 CONTROL REGISTERS

Each port PX has a Data Register PX, and three associated control registers (PXC0, PXC1, PXC2) which define the port line configuration and allow dynamic change in port configuration during program execution. Ports and control registers are mapped into the Register File as shown in Figure 9-1 Ports and control registers are treated like any other general-purpose register. There are no special instruction for port manipulation, any instruction that addresses a register can address the ports. Data can be directly accessed in the port register, without passing through other memory or "accumulator" locations.

Figure 9-1. I/O Register Map

Applicable to ST9030, ST9032, ST9036, ST9040

GROUP E			PAGE 2			PAGE3	
			Reserved		0FFh	P7DR	R255
		0FEh	P3C2	R254	0FEh	P7C2	R254
		0FDh	P3C1	R253	0FDh	P7C1	R253
		0FCh	P3C0	R252	0FCh	P7C0	R252
			Reserved		0FBh		R251
		0FAh	P2C2	R250	0FAh		R250
		0F9h	P2C1	R249	0F9h		R249
		0F8h	P2C0	R248	0F8h		R248
		0F6h	P1C2	R246	0F6h	P5C2	R246
0E5h	P5DR	0F5h	P1C1	R245	0F5h	P5C1	R245
0E4h	P4DR	0F4h	P1C0	R244	0F4h	P5C0	R244
0E3h	P3DR						
0E2h	P2DR	0F2h	P0C2	R242	0F2h	P4C2	R242
0E1h	P1DR	0F1h	P0C1	R241	0F1h	P4C1	R241
0E0h	P0DR	0F0h	P0C0	R240	0F0h	P4C0	R240

CONTROL REGISTERS (Continued)

During the reset state, all the Ports are set as bidirectional/weak pull-up mode, with the output data register set to FFh. This condition is also held after reset (except for Ports 0, 1 (, 6 for ST905x) in ROM-less devices, see Memory chapter) and can be redefined under software control at any time.

9.3 PORT BIT STRUCTURE AND PROGRAMMING

By programming the control bits PXC0.n and PXC1.n (see Figure 9-2) it is possible to configure bit PX.n as Input, Output, Bidirectional or Alternate Function Output, where X is the number of the I/O port, and n the bit within the port (n = 0 to 7).

When programmed as input, it is possible to select the input level as TTL or CMOS by programming the control bit PXC2.n.

The output buffer can be programmed as Push-pull or Open-drain. A Weak Pull-up configuration can be used when the port bit is programmed as Bidirectional. This is an Open-drain configuration with a high pull-up resistor value (turned on by writing a "1"), to avoid the requirement for external resistances.

The basic structure of the bit PX.n of a general purpose port PX is shown in Figure 9-3.

Independently to the chosen configuration, when the User addresses the port as an destination register of an instruction, the port is written to and the data is transferred from the internal Data Bus into the Output Master Latches. When the port is addressed as a source register for an instruction, the port is read and the data stored in the Input Latch is transferred onto the internal Data Bus.

When PX.n is programmed as Input: (Figure 9-4)

- The Output Buffer is forced tristate
- The data present on the I/O pin is sampled into the Input Latch at the beginning of the execution of the instruction which is accessing the port.
- The data stored in the Output Master Latch is copied into the Output Slave Latch at the end of the execution of each instruction. So if bit PX.n is reconfigured as Output or Bidirectional, the data stored in the Output Slave Latch is reflected on the I/O pin.

Figure 9-2. Control Bits

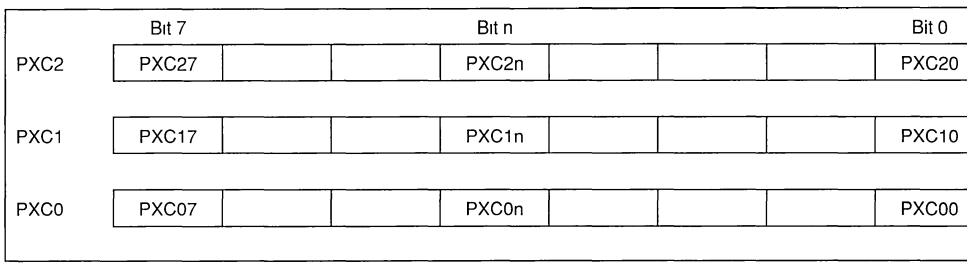


Table 9-1. Port Bit Configuration Table

PXC2n PXC1n PXC0n	1 0 0	0 0 0	1 0 1	0 0 1	1 1 0	0 1 0	1 1 1	0 1 1
PXn Configuration	BID	BID	IN	IN	OUT	OUT	AF	AF
PXn Output	OD	WP	TRI	TRI	OD	PP	OD	PP
PXn Input	TTL	TTL	TTL	CMOS	TTL	TTL	TTL	TTL

Notes:

BID · BIDIRECTIONAL
IN INPUT
OUT OUTPUT
AF · OUTPUT ALTERNATE FUNCTION
TRI . TRISTATE
OD OPEN DRAIN
WP WEAK PULL-UP
PP · PUSH-PULL
TTL TTL STANDARD INPUT
CMOS CMOS STANDARD INPUT

PORT BIT STRUCTURE AND PROGRAMMING (Continued)

Figure 9-3. Basic Structure of an I/O Port Pin (except analog input)

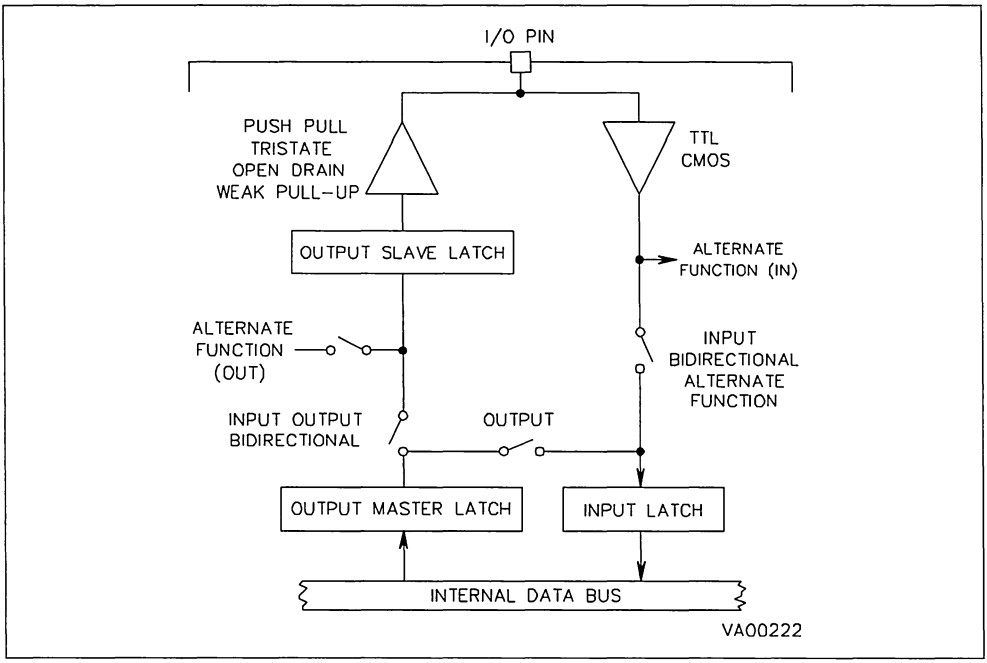


Figure 9-4. Input Configuration

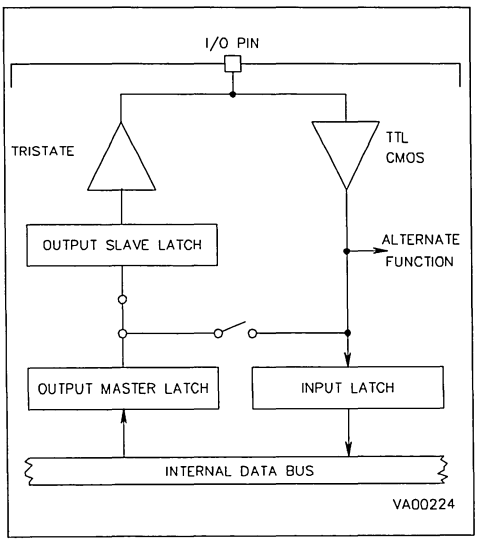
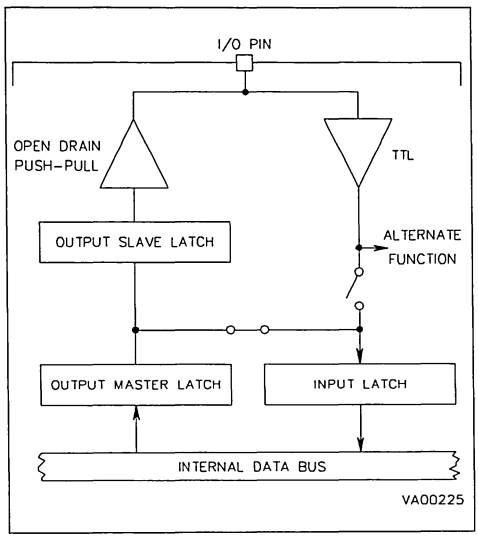


Figure 9-5. Output Configuration



PORT BIT STRUCTURE AND PROGRAMMING (Continued)

When PX.n is programmed as Output: (Figure 9.5)

- The Output Buffer is turned on in an Open-drain or Push-pull configuration
- The data stored in the Output Master Latch is copied both into the Input Latch and into the Output Slave Latch, driving the I/O pin, at the end of the execution of each instruction.

When PX.n is programmed as Bidirectional: (Figure 9-6)

- The Output Buffer is turned on in an Open-drain or Weak Pull-up configuration
- The data present on the I/O pin is sampled into the Input Latch at the beginning of the execution of each instruction
- The data stored in the Output Master Latch is copied into the Output Slave Latch, driving the I/O pin, at the end of the execution of each instruction.

WARNING. Due to the unique feature of the bidirectional mode of reading the external pin instead of the output latch, particular care must be taken with arithmetic/logic and boolean instructions performed on a bidirectional port pin.

These instructions use a read-modify-write sequence, and the result written in the port register depends on the logical level present on the external pin.

This may bring unwanted modifications to the port output register content.

For example:

Port register content	external port value
0Fh	03h

(Bits 3 and 2 are externally forced to 0)

Making a `bset` instruction on bit 7 will return:

Port register content	external port value
83h	83h

(Bits 3 and 2 have been cleared.)

To avoid this situation, it is suggested that all the operations on a port, using at least one bit in bidirectional mode, are performed on a copy of the port register, then transferring the result with a load instruction to the I/O port.

When PX.n is programmed as Alternate Function Output (Figure 9-7) except fo Analog Inputs :

- The Output Buffer is turned on in an Open-drain or Push-pull configuration
- The data present on the I/O pin is sampled into the Input Latch at the beginning of the execution of each instruction
- A signal coming from an on-chip Function is allowed to load the Output Slave Latch driving the I/O pin. Signal timing is under control of the Function. If no Function is connected to PX.n the I/O pin is driven to a high level in Push-pull configuration and is driven to high impedance in open drain configuration.

Figure 9-6. Bidirectional Configuration

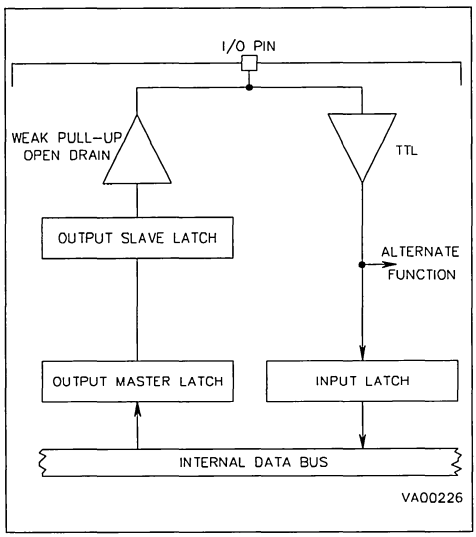
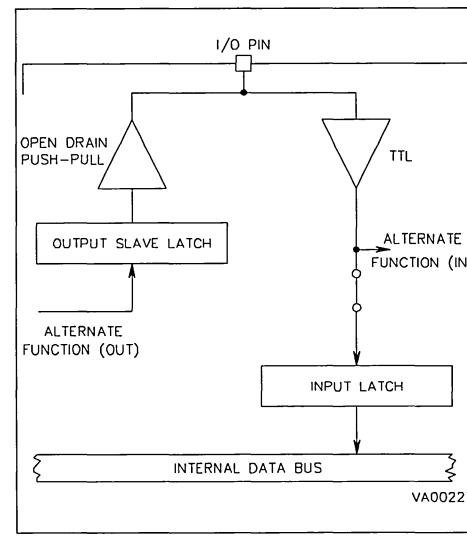


Figure 9-7. Alternate Function Configuration



9.4 ALTERNATE FUNCTION ARCHITECTURE

Each single I/O pin may access three different types of ST9 internal signals:

- Data bus line (I/O)
- 'Alternate Function' Input
- Alternate Function Output

Each pin configuration is made by software, thus allowing the User to choose the type of signal to access a pin. The choice of type of signal is made with the registers PXC2, PXC1, PXC0 of the I/O Port X (Please refer to the previous section for more details)

Pins Declared as an I/O
A pin declared as an I/O is a pin connected to the I/O buffer. In such a case, this pin may either be an Input or an Output or an I/O depending on the value stored in (PXC2, PXC1, PXC0)

Figure 9-8. Example of 3 Alternate Function Inputs

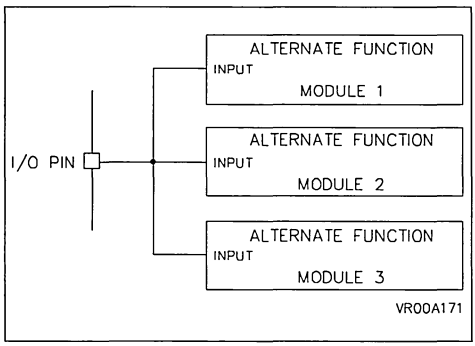
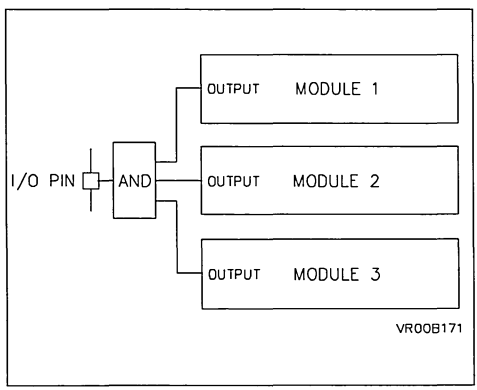


Figure 9-9. Example of 3 Alternate Function Inputs

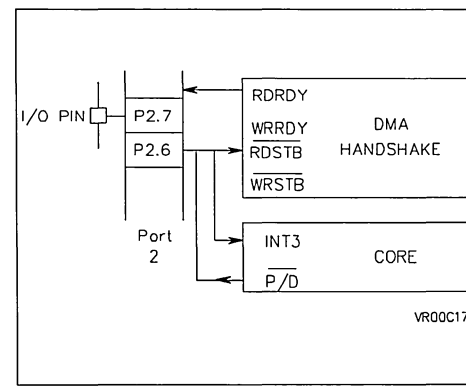


Pin Declared As An 'Alternate Function' Input
A single pin may be directly connected to several Alternate Function inputs. In such a case, the User has to select the required input mode (TTL or CMOS levels) and to enable, by software, the selected Alternate Function module (by enabling it) and unselect all other Alternate Functions (by disabling them).
No specific configuration of the port is required to enable the input Alternate Function, as the input buffer is directly connected to each module using it. As more than one module can use the same input Alternate Function line, it is under User software control to enable a module to use the input Alternate Function.
The digital I/O remains operational even when using the Alternate Function input. The exception to this is for an I/O port bit connected to analog voltages (for the Analog to Digital Converter).

Pin Declared As An Alternate Function Output
A pin declared as an Alternate function output corresponds to (PXC2,PXC1,PXC0) = 1,1,1 or 0,1,1. Several Alternate Function outputs may drive a common pin. In such a case, the Alternate Function output signals are ANDed before driving the common pin. The User has therefore to select, by software, the Alternate Function Output required by enabling it and disabling all other Alternate Function Outputs on the same pin (a disabled Alternate Function Output outputs a "1").

The inputs to on-chip Functions and Alternate Function Outputs are predefined for each I/O pin of an ST9. Please refer to the Alternate Function Table at the beginning of this datasheet for the exact configuration.

Figure 9-10. Example of One I/O Pin Configuration



ALTERNATE FUNCTION (Continued)

General Configuration
A single pin may be used, according to different phases of the software, as an I/O or connected to an input or an Alternate Function output. An example is given in Figure 9-10.

WARNING: When a pin is connected to an Input Function and to an Alternate Function output, the User must be aware of the fact that the Alternate Function output signal always input to the Alternate Function module(s) declared as input(s). Figure 9-10 shows an example where the signal *P/D also enters RDSTB and INT3.*

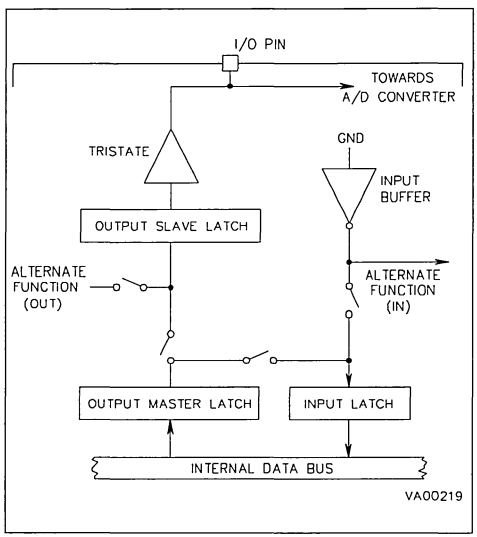
9.5 SPECIAL PORTS

9.5.1 Bit Structure For A/D Converter Inputs

When a port bit is used as input for an on-chip A/D Converter, its structure is modified as shown in Figure 9-11.

The behaviour of this bit is identical to the general purpose bit described in paragraph 9.68 except when it is programmed as Alternate Function. In this case, the Output Buffer is forced Tristate and the input of the Input Buffer is disconnected from the I/O pin and forced low. In this way the I/O pin is free to assume any analog value without causing power consumption in the Input Buffer. The bit **MUST** be programmed to **(PXC2, PXC1, PXC0) = 1,1,1)** to assume this special configuration.

Figure 9-11. A/D Input Port Bit Structure



9.6 I/O STATUS AFTER WFI, HALT AND RESET

The status of the ST9 I/O ports during the Wait For Interrupt, Halt and Reset operational modes is shown in the following table. The External Memory Interface ports are shown separately, however, if only the internal memory is being used and the ports are acting as I/O, the status is the same as shown for the other I/O ports.

- if ROMLESS (ST9 memory is Off-chip):
  P0, P1 are set to push-pull A.F.
  Push-pull, Output value is undefined.
- if not ROMLESS (ROM or EPROM parts)
  P0, P1 and P6 are set to Bidirectional Weak Pull-up, Output value is FFh (all pins high).

WARNING: *I/O pins (other than the ROMless pin, if enabled by mask option) are set to the Weak Pull-up mode during the Reset cycle. This state is forced during the reset sequence, but the I/O pins can be in a random state for up to 64 crystal periods.*
The application circuit must take this into account if it can lead to critical situations in the external circuitry.

Mode	P0	P1 [P6](2)	I/O
WFI	HIgh Impedance	Next Address	No Affect (clocks output from ST9 running)
HALT	HIgh Impedance	Next Address	No Affect (clocks output from ST9 stopped)
RESET	Note 1		Bidirectional Weak Pull-up except: ROMless = weak weak pull-up

Notes
1 P0 and P1 (when used to provide non-multiplexed low order address) setup depends on the ROMLESS condition
2 for ST905x

10 HANDSHAKE/DMA CONTROLLER

10.1 INTRODUCTION

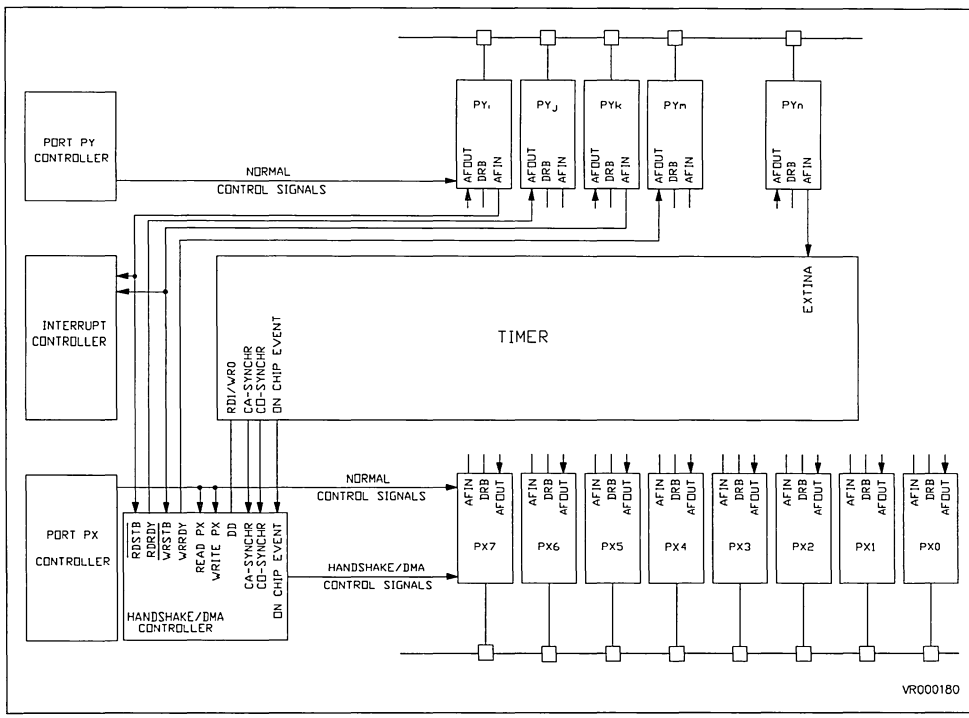
The handshake module allows the User to configure an I/O Port under handshake control or to support DMA operations, driven by an on-chip 16 bit Multifunction TIMER, between Data/Program Memory or Register File and an I/O port.

The module supports data exchange with handshake through port PX (where PX is predefined by the ST9 configuration) with up to 4 handshake lines: 2 Outputs (RDRDY and WRRDY) connected as Alternate Function Outputs and 2 Inputs ($\overline{RDSTB}$ and $\overline{WRSTB}$).
Input, Output and Bidirectional Handshake modes are available.

Input Functions $\overline{RDSTB}$ and $\overline{WRSTB}$ are always associated to external interrupt channels. To synchronize handshake protocols generating interrupt requests (as the following paragraph will show) the User must program the interrupt control register and the vector associated to the used line(s) ($\overline{RDSTB}$ and/or $\overline{WRSTB}$). The active high output lines RDRDY and WRRDY are held high when not active in order to allow the Alternate Function Output connection of other ST9 peripherals.

DMA transfers can move data from Data/Program Memory or Register File to the I/O Port with Handshake capability or viceversa, using either the Multifunction Timer CAPT0 or COMP0 DMA Channels. In Figure 10-1 the four on-chip lines that connect the module to the on-chip Multifunction Timer to support DMA transfers are shown (DD (Data Direction), CO_SYNCHR(COmpare SYNCHronism), CA_SYNCHR (CApture SYNCHronism) and On Chip Event).

Figure 10-1. Handshake/DMA Controller Module Block Diagram



10.2 PROGRAMMABLE HANDSHAKE MODES

10.2.1 Input Handshake

Two Input Handshake Modes are available to synchronise input transitions on port bits programmed as Input or Bidirectional. Output or Alternate Function bits are not affected.
In the timings, READ PORT is an ST9 internal signal that transfers data from the Input Latches onto the Data bus.

Two Lines Input Handshake

When this mode is selected WRRDY is set to indicate that data can be loaded into the Input Latches of the Input and Bidirectional port pins. Data present on the pins is sampled when the peripheral forces a low level on $\overline{\text{WRSTB}}$.

When a rising edge on $\overline{\text{WRSTB}}$ occurs, WRRDY goes low signifying that the Input Latch is full and further loading must be inhibited until the ST9 reads the port. When the port register is read, WRRDY is set. Both low and high levels on $\overline{\text{WRSTB}}$ must last at least one INTCLK cycle.

The User is suggested to program the External Interrupt Channel associated with the $\overline{\text{WRSTB}}$ line to generate an interrupt request when a rising edge occurs. The ST9 can thus, in the course of its interrupt service routine, read the data furnished by the peripheral as soon as it is available.

Figure 10-2. Two Line Input Handshake

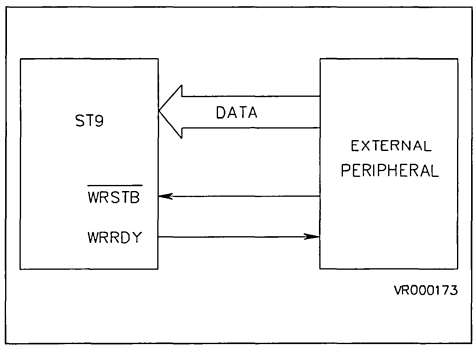
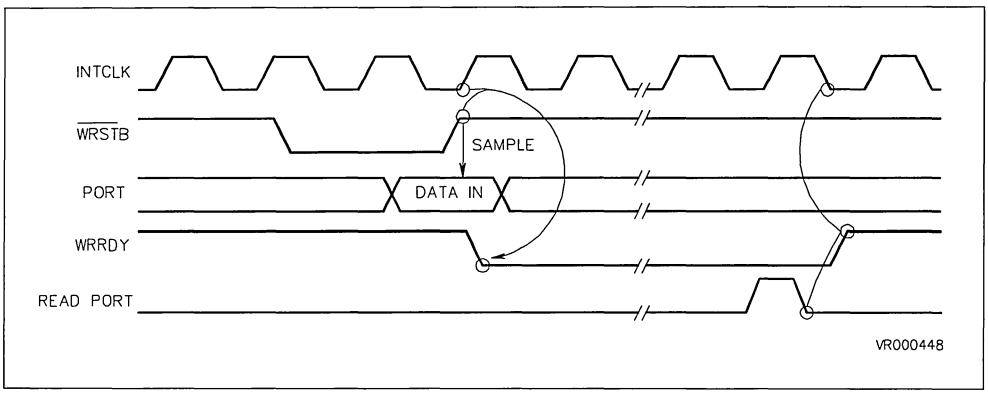


Figure 10-3. Two Line Input Handshake Timing



HANDSHAKE MODES (Continued)

One Line Input Handshake

Figures 10-4 and 10-5 illustrate the timing associated with the One Line (WRRDY) Input Handshake Mode.

When this mode is selected the ST9 sets WRRDY to indicate that data can be loaded into the Input Latches of the Input and Bidirectional port pins.

Data present on the pins is continuously sampled. When the ST9 is reading the port WRRDY goes low. As data is strobed into the port only when WRRDY goes high, the forced low state of WRRDY will prevent the Input Latch data from changing while ST9 is reading the port. When the ST9 read cycle finishes, WRRDY is set.

Figure 10-4. One Line Input Handshake

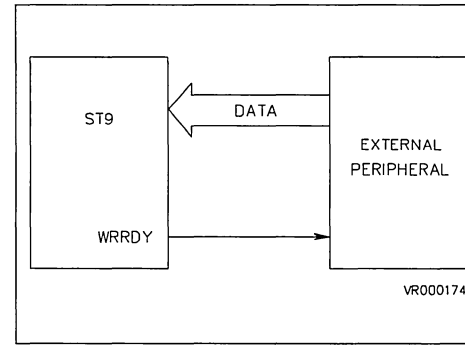
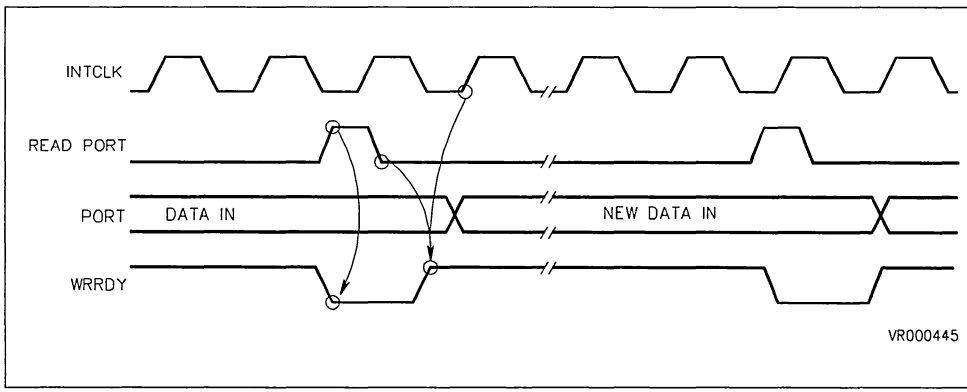


Figure 10-5. One Line Input Handshake Timing



HANDSHAKE MODES (Continued)

10.2.2 Output Handshake

Two Output Handshake Modes are available to synchronize output transitions on port bits programmed as Output or Bidirectional. I/O pins programmed as Input or Alternate Function Output are not affected.

In the timing diagrams, WRITE PORT is the internal signal that transfers data from the Internal Data Bus into the Port Output Master Latches.

Two Lines Output Handshake

Figure 10-7 illustrates the timing associated with the Two Lines (RDRDY, $\overline{\text{RDSTB}}$) Output Handshake Mode (Figure 10-6).

When this mode is selected RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written, RDRDY is set to indicate that data is ready for the peripheral device. In most systems the rising edge of RDRDY can be used as a latching signal in the peripheral device. RDRDY will remain high until a rising edge is received on $\overline{\text{RDSTB}}$ indicating that the peripheral has taken the data. Both low and high level on $\overline{\text{RDSTB}}$ must last at least one ST9 INTCLK cycle. The User is suggested to program the External Interrupt Channel associated with the $\overline{\text{RDSTB}}$ line to generate an interrupt request when a rising edge occurs. The ST9 can thus, in the course of its interrupt service routine, furnish new data as soon as the previous data is taken by the peripheral.

Figure 10-6. Two Line Output Handshake

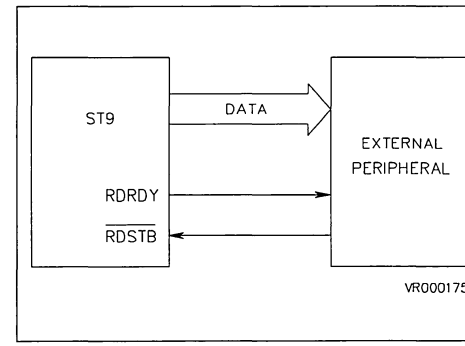
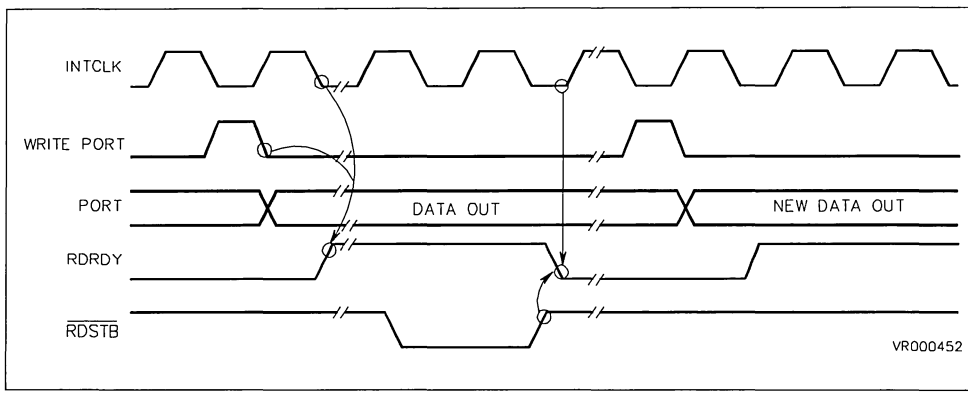


Figure 10-7. Two Line Output Handshake Timing



HANDSHAKE MODES (Continued)

One Line Output Handshake

Figure 10-9 illustrates the timing associated with the One Line (RDRDY) Output Handshake Mode Figure 10-8.

When this mode is selected RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written to, RDRDY is set to indicate that data is ready for the peripheral device. In most systems the rising edge of RDRDY can be used as a latching signal in the peripheral device. No peripheral acknowledge is waited for. While ST9 is writing into the Output Slave Latches RDRDY goes low, RDRDY is set again when the new data is ready on the port pins.

Figure 10-8. One Line Output Handshake

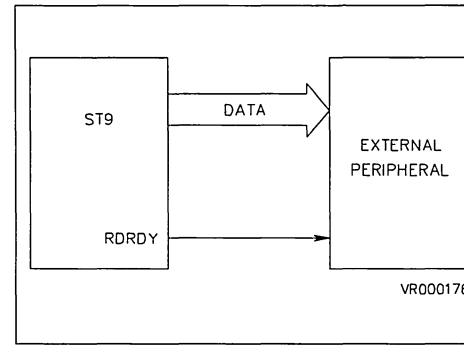
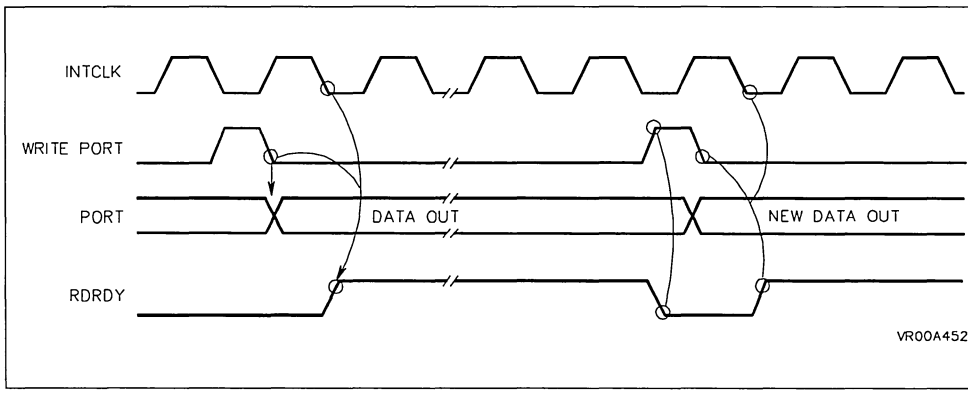


Figure 10-9. One Line Output Handshake Timing



HANDSHAKE MODES (Continued)

10.2.3 Bidirectional Handshake

A Bidirectional Handshake Mode is available to synchronise bidirectional transitions on Port bits programmed as Bidirectional. When this mode is selected, the Output Buffer configuration of Bidirectional port pins programmed as Weak Pull-up become Push-pull. Open-drain configuration is not modified. I/O bits set to Input, Output or Alternate Function Output are not affected.

Figure 10-11 illustrates the timing associated with the Bidirectional Handshake Mode. This mode is a combination of Two Lines Output Mode and Two Lines Input Mode using all four handshake lines, two for output (RDRDY, $\overline{\text{RDSTB}}$) and two for input control (WRRDY, $\overline{\text{WRSTB}}$). In the timing INTCLK is the ST9 internal not stretched clock, WRITE PORT is the signal that transfers data from the Internal Data Bus into the port Output Master Latches and READ PORT is the signal that transfers data from the Input Latches onto the Data Bus. When Bidirectional Handshake mode is selected the Output Buffers of the Bidirectional port pins are forced tristate, WRRDY is set to indicate that data can be loaded into the Input Latches and RDRDY is reset to indicate that no significant data is present in the Output Slave Latches.

Input Transitions. Data present on the pins is sampled when the peripheral forces a low level on $\overline{\text{WRSTB}}$. When a rising edge on $\overline{\text{WRSTB}}$ occurs, WRRDY goes low signifying that the Input Latches are full and further loading must be inhibited until the ST9 reads the port. When the port register is read, WRRDY is set. Both low and high levels on $\overline{\text{WRSTB}}$ must last at least one ST9 INTCLK cycle.

The User is suggested to program the External interrupt Channel associated with the $\overline{\text{WRSTB}}$ line to generate an interrupt request when a rising edge occurs.The ST9 can thus, in the course of its interrupt service routine, read the data furnished by the peripheral as soon as it is available.

Figure 10-10. Four Line Bidirectional

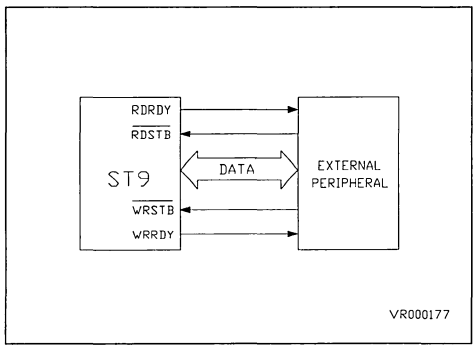
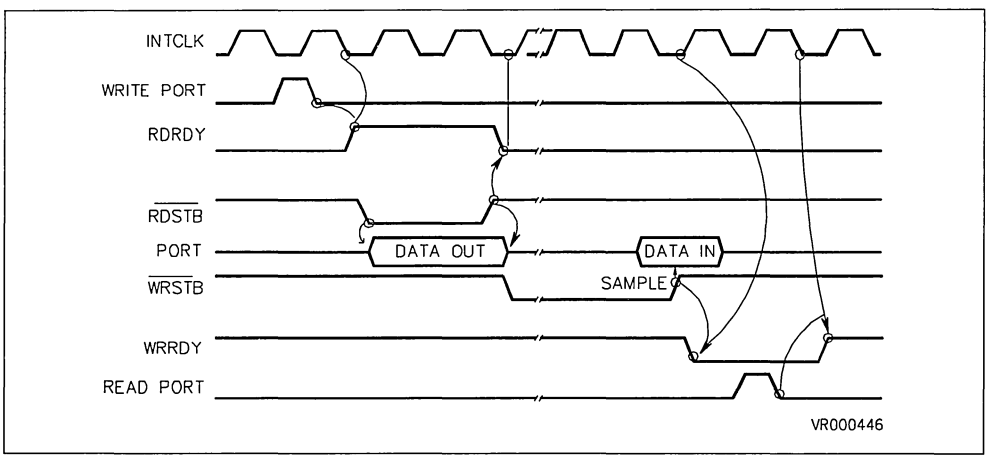


Figure 10-11. Bidirectional Handshake Timing



HANDSHAKE MODES (Continued)

Output Transitions. When the Output Slave Latches are written to, RDRDY is set to indicate that data is ready for the peripheral device. When $\overline{RDSTB}$ goes low, data is allowed out onto the port pins. When a rising edge is received on $\overline{RDSTB}$, indicating that the peripheral has taken the data, the Output Buffers are forced tristate and RDRDY goes low. Both low and high level on $\overline{RDSTB}$ must last at least one INTCLK cycle.

The User is suggested to program the External Interrupt Channel associated to the $\overline{RDSTB}$ line to generate an interrupt request when a rising edge occurs; The ST9 can thus, in the course of its interrupt service routine, write new data into the Output Slave Latches as soon as the previous data is taken by the peripheral.

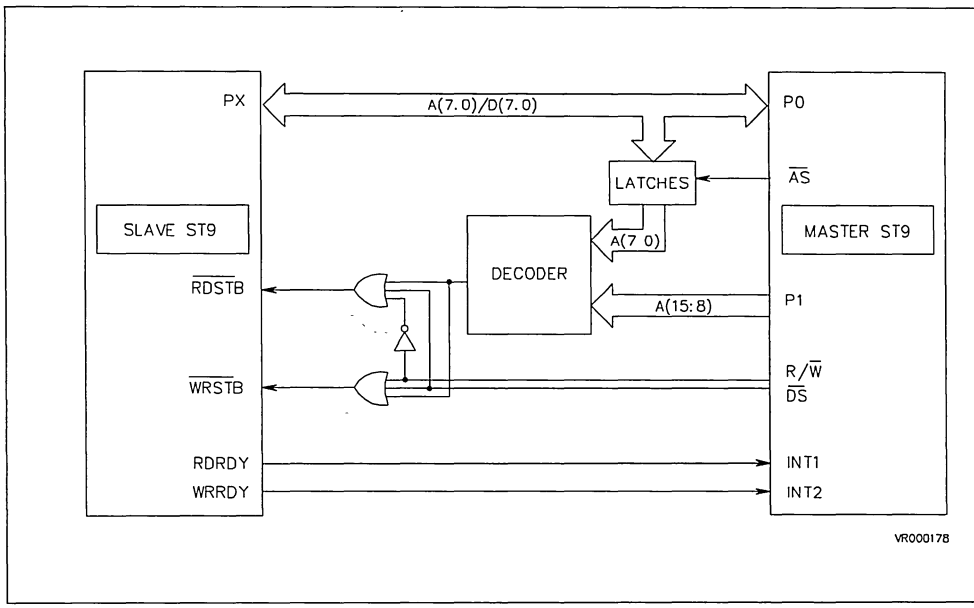
10.2.4 Application example: Mapping an ST9 onto the memory bus of another ST9

Figure 10-12 shows a possible application of the bidirectional handshake protocol, used to connect an ST9 as a slave of another (master) ST9.

PX of the slave ST9 is connected to the Address/Data Memory Bus of the master ST9. A decoder enables, with a low level, the generation of $\overline{RDSTB}$ or $\overline{WRSTB}$ when $\overline{DS}$ is low and the master is reading from, or writing to, the memory.

To synchronize data transfers with the slave, the master ST9 uses RDRDY and WRRDY as External Interrupt Sources, programmed to generate an interrupt request when a rising edge occurs. The slave ST9 interrupts the master raising RDRDY when new data is ready in the port Output Slave Latches and raising WRRDY when the Input Latches can be filled with new data. According to the interrupt request received, the master ST9 can read the ready data from the slave (RDRDY interrupt routine) or write other data into the slave (WRRDY interrupt routine).

Figure 10-12. Bidirectional Application Example



10.3 PROGRAMMABLE DMA MODES

The Handshake Module supports DMA operations controlled by either the CAPT0 or COMP0 DMA Channel of a Multifunction Timer. The User enables this function writing a "0" in the DEN bit in the HDCTL register and selects the DMA Channel by writing the DCH bit: "0" for CAPT0, "1" for COMP0.

When the CAPT0 Channel is chosen, the DD bit selects the Data Direction: "0" to move data from Data/Program Memory or Register File to the port (DMA Output), "1" to perform the opposite transfer (DMA Input). Signal CA_SYNCHR is sent by the Timer to the Handshake/DMA Controller for writing the port Output Master Latches or reading the Input latches (depending on DD), during the DMA operations when a capture occurs on the Timer.

If the Handshake section of the module is enabled, the data transfer from the Output Master Latches into the Output Slave Latches (Output Strobe, for pins programmed as Output or Bidirectional) or from the Pins into the Input Latches (Input Strobe, for pins programmed as Input or Bidirectional) is controlled by the logic supporting the chosen Handshake protocol.

If no Handshake is programmed the User can choose how to drive the Output or Input Strobe by writing the DST bit: a "0" leaves the Strobes under the normal port control, according to the chosen port bit configuration, a "1" selects the On Chip Event generated by the Timer as the Output or Input Strobe.

When the COMP0 Channel is selected, DMA output transfers are only allowed independent of DD, and CO_SYNCHR is used for output Master Latch. If Handshake is disabled, DST selects how to control the Output Strobe. If enabled, the Handshake controls the Output Strobe.

10.3.1 DMA Transfers Driven By Timer CAPT0 Channel With Handshake

The following descriptions are made assuming that DMA transfers are driven by Multifunction Timer 0. The following table shows the DMA Port capabilities of the ST9 family:

MultiFunction Timer	Handshake Port
0	5

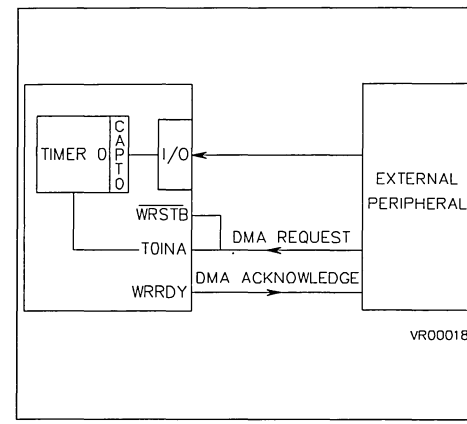
10.3.2 DMA Input transfers with two line input handshake

When

- Two Lines Input Handshake mode is selected (HS7="1", HS6="0", HS5="1")
- the port is enabled to support DMA input transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DD="1", DCH="0")
- the Handshake $\overline{\text{WRSTB}}$ line is connected off-chip to the Timer T0INA line
- T0INA DMA requests are enabled on rising edges
- $\overline{\text{WRSTB}}$ interrupt requests are disabled, data transfers on port pins programmed as Input (or Bidirectional) can be synchronized using the Handshake $\overline{\text{WRSTB}}$ line as DMA Request and the WRRDY line as DMA Acknowledge.

WRRDY is set to indicate that data can be loaded into the Input Latches of the Input (or Bidirectional) port pins. Data present on the port pins is sampled when the peripheral forces a low level on $\overline{\text{WRSTB}}$. When a rising edge on $\overline{\text{WRSTB}}$ (T0INA) occurs WRRDY goes low, signifying that the Input Latches are full and further loading must be inhibited until the ST9 reads the port, and a DMA request is issued. When the port register is read, during the DMA transfer, WRRDY is set.

Figure 10-13. DMA with 2 Line Input Handshake Mode



PROGRAMMABLE DMA MODES (Continued)

10.3.3 DMA output transfers with two lines output handshake

When

- Two Lines Output Handshake is selected (HS7="1", HS6="1", HS5="0")
- the port is enabled to support DMA output transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DD="0", DCH="0")
- the Handshake $\overline{\text{RDSTB}}$ line is connected off-chip to the Timer T0INA line
- T0INA DMA requests are enabled on rising edges
- $\overline{\text{RDSTB}}$ interrupt requests are disabled

data transfers on port pins programmed as Output (or Bidirectional) can be synchronized when using the Handshake $\overline{\text{RDSTB}}$ and RDRDY lines as DMA Request and DMA Acknowledge.

When Two Lines Output Handshake is selected, RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written, RDRDY is set to indicate that data is ready for the peripheral device. The first data value, whose usual meaning is that ST9 is ready to provide the following data by DMA transfers, is normally written by the DMA initialization routine.

When a rising edge is received on $\overline{\text{RDSTB}}$ (T0INA), indicating that the peripheral has taken the data, RDRDY is reset and a DMA request is issued to get the next data. When the ST9 Output Slave Latches are written, during the DMA transfer, RDRDY is set again. If the User wants to get data from ST9 as soon as $\overline{\text{RDSTB}}$ goes low, external latches clocked by $\overline{\text{RDSTB}}$ can be added to create a pipeline stage, that is at each $\overline{\text{RDSTB}}$ low pulse on the falling edge the peripheral gets data transferred into the port by the previous DMA transfer and on the rising edge a DMA request is issued to get the next data.

10.3.4 DMA input transfers with one line input handshake

When

- One Line Input Handshake is selected (HS7="0", HS6="0", HS5="1")
- the port is enabled to support DMA input operations driven by the Timer CAPT0 DMA Channel (DEN="0", DD="1", DCH="0")
- the Timer T0INA DMA requests are enabled on rising (or falling) edges data transfers on port pins programmed as Input (or Bidirectional) can be synchronized by using the Timer T0INA line as DMA Request, and the Handshake WRRDY line as DMA Acknowledge.

When One Line Input Handshake is selected WRRDY is set to indicate that data can be loaded into the Input Latches of the Input and Bidirectional port pins. Data present on the port pins is continuously sampled. While ST9 is reading the port, during the DMA transfer requested by a rising (or falling) edge on the Timer T0INA line, WRRDY goes low. If data is strobed into the port only when WRRDY is high, the forced low state of WRRDY will prevent Input Latches data from changing while ST9 is reading the port. When ST9 reading cycle finishes, WRRDY is set.

Figure 10-14. DMA output transfers with 2 lines output handshake

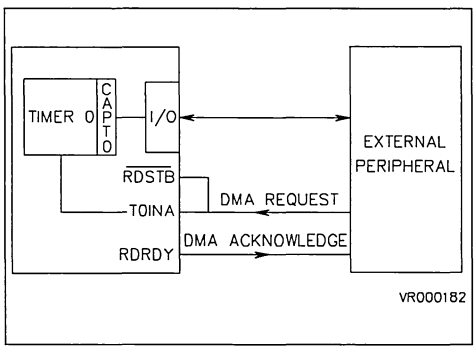
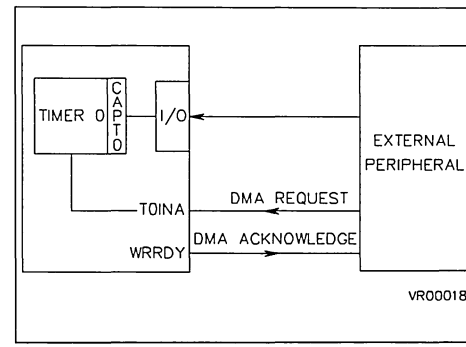


Figure 10-15. DMA input transfers with one line input handshake



PROGRAMMABLE DMA MODES (Continued)

10.3.5 DMA output transfers with one line output handshake

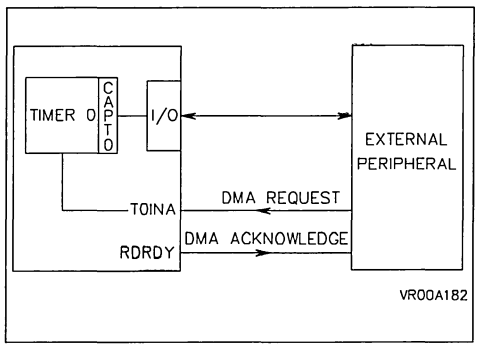
When

- One Line Output Handshake is selected (HS7="0", HS6="1", HS5="0")
- the port is enabled to support DMA output transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DD="0", DCH="0")
- the Timer T0INA DMA requests are enabled on rising (or falling) edges

data transfers on port pins programmed as Output or Bidirectional can be synchronized using the Timer T0INA line as DMA Request, and the Handshake RDRDY line as DMA Acknowledge. RDRDY is reset to indicate that no significant data is present on the Output (or Bidirectional) port pins. When the ST9 Output Slave Latches are written, RDRDY is set to indicate that data are ready for the peripheral device. The first data, whose usual meaning is that the ST9 is ready to provide the following data by DMA transfers, is normally written by the DMA initialization routine.

When a rising (or falling) edge is received on T0INA, a DMA request is issued to get the next data. While ST9 is writing into the Output Slave Latches, during the DMA transfer, RDRDY goes low. RDRDY is set again when the new data is ready on the port pins.

Figure 10-16. DMA output transfers with one line output handshake



10.3.6 DMA input/output transfers with bidirectional handshake

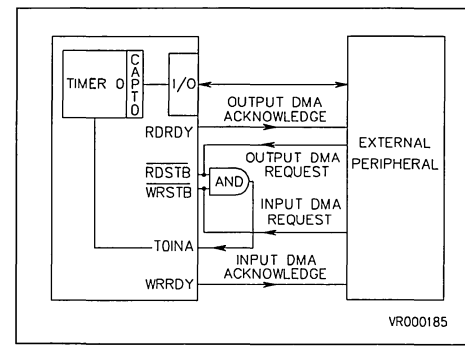
When

- Bidirectional Handshake is selected (HS7="X", HS6="0", HS5="0")
- the port is enabled to support DMA transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DCH="0")
- the Handshake $\overline{WRSTB}$ and $\overline{RDSTB}$ lines are ANDed and connected off-chip to the Timer T0INA line
- T0INA DMA requests are enabled on rising edges
- $\overline{WRSTB}$ and $\overline{RDSTB}$ interrupt requests are disabled

data transfers on port pins programmed as Bidirectional can be synchronized using the Handshake $\overline{WRSTB}$ and WRRDY lines as DMA Request and DMA Acknowledge for DMA Input transfers (DD="1") and the Handshake $\overline{RDSTB}$ and RDRDY lines as DMA Request and DMA Acknowledge for DMA Output transfers (DD="0").

DMA Input Transfers. When Bidirectional Handshake is selected WRRDY is set to indicate that data can be loaded into the Input Latches of the Bidirectional port pins. Data present on the pins is sampled when the peripheral forces a low level on $\overline{WRSTB}$. When a rising edge on $\overline{WRSTB}$ (T0INA) occurs WRRDY goes low, signifying that the Input Latches are full and further loading must be inhibited until the ST9 reads the port, and a DMA request is issued. When the port register is read, during the DMA transfer, WRRDY is set.

Figure 10-17. DMA input/output transfers with bidirectional handshake



PROGRAMMABLE DMA MODES (Continued)

DMA Output Transfers. When Bidirectional Handshake is selected, RDRDY is reset to indicate that no significant data is present on the Bidirectional port pins. When the Output Slave Latches are written, RDRDY is set to indicate that data is ready for the peripheral device. The first data, whose usual meaning is that ST9 is ready to provide the following data by DMA transfers, is normally written by the DMA initialization routine.

When $\overline{\text{RDSTB}}$ goes low data is allowed onto the port pins. When a rising edge is received on $\overline{\text{RDSTB}}$ (T0INA), indicating that the peripheral has taken the data, the Output Buffers are forced tristate, RDRDY is reset and a DMA request is issued to get the next data. When the Output Slave Latches are written during the DMA transfer, RDRDY is set again.

In the output data flow there is one pipeline stage, that is at each $\overline{\text{RDSTB}}$ low pulse on the falling edge the peripheral gets data transferred into the port by the previous DMA transfer and on the rising edge issues a DMA request to get the next data.

Example. As the direction of DMA transfers is controlled by software, the User must define a protocol to control the sequence of input/output data transfers.

The initialization routine defines the direction (DD) of the first DMA transfer and the address and size of the data buffer (Pointer and Counter associated to the DMA Channel). In the interrupt routine called when the DMA Transaction Counter = 0, the User must define the new address and size of the data buffer and can change (according to the chosen protocol) the direction of next DMA operations.

Figure 10-18 shows how the application example of Figure 10-17 (an ST9 connected as a slave of another ST9) is modified when data transfer from/to the slave ST9 is performed by DMA transfers.

10.3.7 DMA Transfers Driven By Timer Comp0 Channel With Handshake

DMA output transfers with one line output handshake

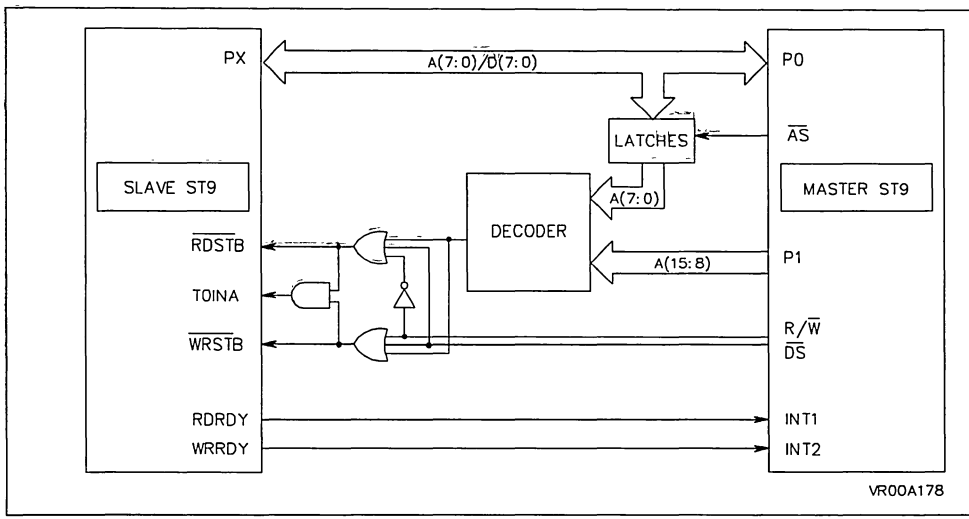
when

- One Line Output Handshake is selected (HS7="0", HS6="1", HS5="0")
- the port is enabled to support DMA output transfers driven by the Timer COMP0 DMA Channel (DEN="0", DCH="1")

data transferred by DMA transfers on port pins programmed as Output or Bidirectional can be strobed using the Handshake RDRDY line.

When One Line Output Handshake is selected RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written RDRDY is set. The rising edge of RDRDY can be used as a latching signal. At every DMA transfer triggered by the COMP0 event new data is written into the port. While data is changing on the Output Slave Latches, RDRDY goes low. RDRDY is set again when the new data is ready on the port pins.

Figure 10-18. Bidirectional Application Example With DMA Transfer



Slave Latches, RDRDY goes low. RDRDY is set again when the new data is ready on the port pins.

10.4 HANDSHAKE/DMA CONTROL REGISTERS

To program the Handshake and DMA modes, the User has to write the Handshake/DMA Control register (HDCTL) according to the table shown in page 80. The different handshake protocols and the Port behaviour during DMA operations are explain in the previous paragraphs.

HDCTLx Read/Write
Handshake/DMA Control Register

Reset Value: 1111 1111 (0FFh)

7							0
HS7	HS6	HS5	DEN	DD	DST	DCH	1

b7-b5 = **HS7, HS6, HS5:** *Handshake Mode Selection.* These bits allow selection of the Handshake direction and the number of lines used in the handshake as shown in the following table.

b4 = **DEN:** *DMA Enable.* This bit (when reset) enables the DMA function with handshake through I/O Port 5. DMA is disabled when this bit = "1".

b3 = **DD:** *DMA Data Direction.* The direction of the DMA transfers through I/O Port 5 is set by this bit. A "1" sets DMA Input and a "0" sets DMA Output.

b2 = **DST:** *DMA Strobe.* This bit, when set, enables the use of the Multifunction Timer 0 On-Chip Event to trigger the DMA transaction.

b1 = **DCH:** *DMA Channel* When DST is set, allowing the DMA transactions to be triggered by Multifunction Timer 0, DCH selects the MFT source, a "1" selects the COMP0 source, a "0" selects the CAPT0 source.

Table 10-1. Module Configuration Table

Handshake Modes		HS7	HS6	HS5
Disabled		X	1	1
Output	(2 lines)	1	1	0
Output	(1 line)	0	1	0
Input	(2 lines)	1	0	1
Input	(1 line)	0	0	1
Bidirectional	(2 lines)	X	0	0

CONTROL REGISTERS (Continued)

Figure 10-19. Handshake/DMA Control Registers

Applicable for ST9030, ST9032, ST9036, ST9040

	PAGE 0			PAGE 2			PAGE3	
			0FFh	Reserved	R255	0FFh	P7DR	R255
			0FEh	P3C2	R254	0FEh	P7C2	R254
			0FDh	P3C1	R253	0FDh	P7C1	R253
			0FCh	P3C0	R252	0FCh	P7C0	R252
			0FBh	Reserved	R251	0FBh		R251
			0FAh	P2C2	R250	0FAh		R250
			0F9h	P2C1	R249	0F9h		R249
			0F8h	P2C0	R248	0F8h		R248
						0F7h		R247
			0F6h	P1C2	R246	0F6h	P5C2	R246
0E5h	P5DR	R229	0F5h	P1C1	R245	0F5h	P5C1	R245
0E4h	P4DR	R228	0F4h	P1C0	R244	0F4h	P5C0	R244
0E3h	P3DR	R227				0F3h		R243
0E2h	P2DR	R226	0F2h	P0C2	R242	0F2h	P4C2	R242
0E1h	P1DR	R225	0F1h	P0C1	R241	0F1h	P4C1	R241
0E0h	P0DR	R224	0F0h	P0C0	R240	0F0h	P4C0	R240

11 SERIAL PERIPHERAL INTERFACE

11.1 INTRODUCTION

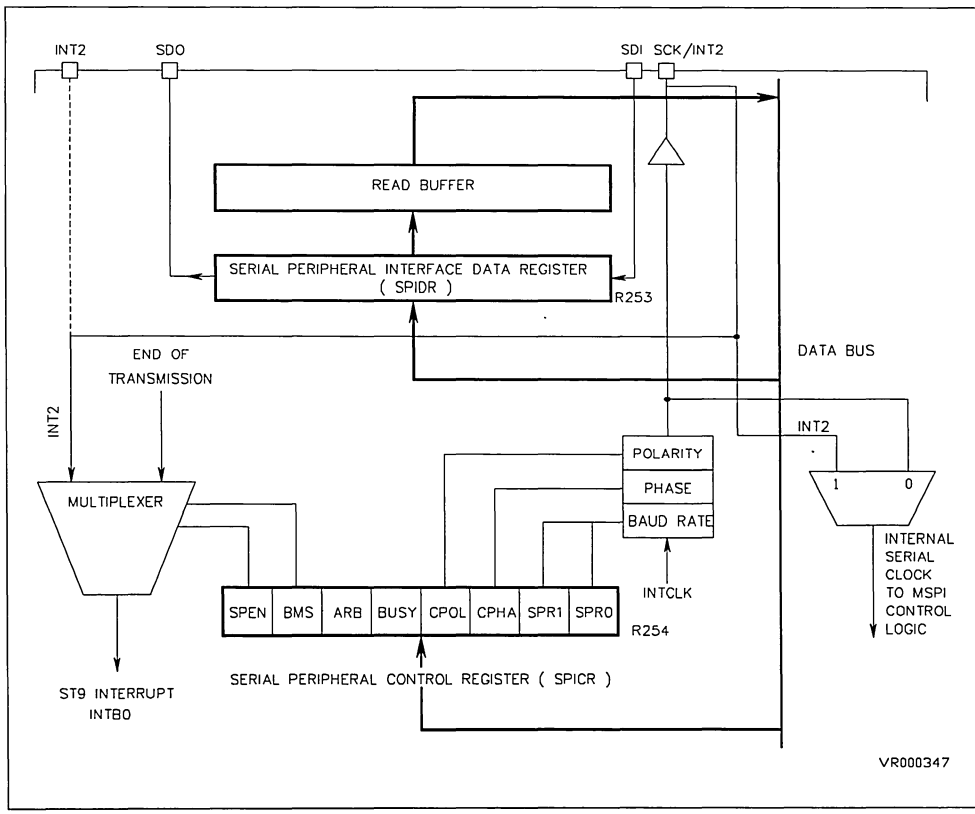
The Serial Peripheral Interface (SPI) is integrated into the Core module of the ST9 and provides a general purpose shift register based peripheral allowing several external peripherals to be linked through an SPI protocol bus. In addition, special modes allow reduced software overhead with I^2C-bus and IM-bus Communication standards.

The SPI uses 3 lines comprising Serial Data In (SDI) and Alternate Function outputs Serial Data Out (SDO) and Synchronous Serial Clock (SCK). Additional I/O pins may act as device selects or IM-bus address ident signals.

Its Main Features are:

- Full duplex 3-wire synchronous transfer
- Master operation only
- 1.5MHz max bit transfer frequency (INTCLK = 12MHz)
- 4 Programmable bit rates
- Programmable clock polarity and phase
- Busy Flag
- End of transmission interrupt
- Additional hardware to facilitate more complex protocols

Figure 11-1. Block Diagram



11.2 FUNCTIONAL DESCRIPTION

The SPI, when enabled, receives input data from the ST9 Core internal data bus into SPIDR, and originates the Serial Clock (SCK) based upon dividing of the internal processor clock (INTCLK). The data is parallel loaded into the 8 bit shift register (from the internal bus) during a write cycle and then shifted out serially through the SDO pin (Most Significant bit first) to the slave device, which responds by sending its data to the master device via the SDI pin. This implies full duplex transmission with data-out and data-in both synchronized with the same clock signal. Thus the transmitted byte is replaced by the byte received, eliminating the need to have separate "Tx empty" and "Rx full" status bits.

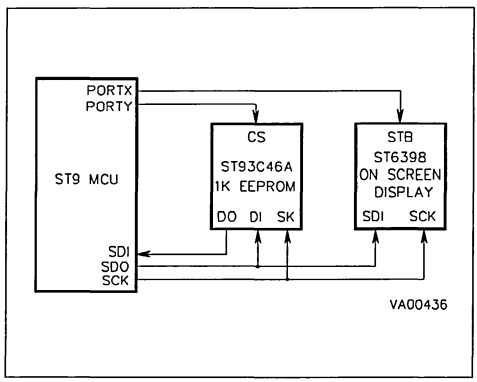
When the shift register is loaded, data is parallel transferred to the read buffer and data becomes available for the ST9 during a following read cycle.

The SPI requires three pins on an I/O port:

SCK	Serial Clock signal
SDO	Serial Data Out
SDI	Serial Data In

An additional output bit of an I/O port may be used to perform the slave chip select signal.

Figure 11-2. A Typical SPI Network



11.2.1 Input Signal Description

Serial Data In (SDI)

Data is transferred serially from a slave to a master on this line, most significant bit first. In an S-BUS/I^2C-bus configuration, SDI line senses the value forced on the data line (by SDO or by another peripheral connected to the S-bus/I^2C-bus environment).

11.2.2 Output Signal Description

Serial Data Out (SDO)

The SDO pin is configured as an output for the master device. This is obtained by programming the corresponding I/O pin as an output alternate function. Data is transferred serially from a master to a slave on SDO, most significant bit first. This pin is forced to the high impedance state when the SPI is disabled and is set to "1" when arbitration is lost (during an S-bus/I^2C-bus protocol transmission). The master device always allows data to be applied on the SDO line one half cycle before the clock edge in order to latch the data for the slave device.

Master Serial Clock (SCK)

The master device uses SCK to latch the incoming data on the SDI line. This pin is forced to a high impedance state when SPI is disabled (SPEN, SPICR.7 = "0"), in order to avoid clock contention from different masters in a multi-master system.
The master device generates SCK from INTCLK. SCK is used to synchronize the transfer of data both in and out of the device through its SDI and SDO pins. The SCK type and its relationship to data are controlled by the CPOL and CPHA bits in the Serial Peripheral Control Register.
This input is provided with a digital filter which cleans spikes lasting less than one INTCLK period.

Two bits (SPR1 and SPR0) in the Serial Peripheral Control Register, SPICR (R254) select the clock rate. Four frequencies can be selected, two in a high frequency range (mostly used with the SPI protocol) and two in a medium frequency range (mostly used for more complex protocols).

11.3 INTERRUPT STRUCTURE

SPI peripheral is associated with external interrupt channel B0 (pin INT2). Multiplexing between the external pin and SPI internal source is controlled by the SPEN and BMS bits according to the following table.

The two possible SPI interrupt sources are: End of transmission (after each byte) and S-bus/I²C-bus start condition. Care should be taken when toggling SPEN or/and BMS bits from (0,0) status, this should be done by masking the interrupt channel B0 (reset of EIMR.IMB0, bit 2 of External Interrupt Mask Register). Furthermore it is necessary to clear possible spurious requests on the corresponding channel by resetting the interrupt pending bit EIPR.IPB0 (bit 2 of External Interrupts Pending Register).

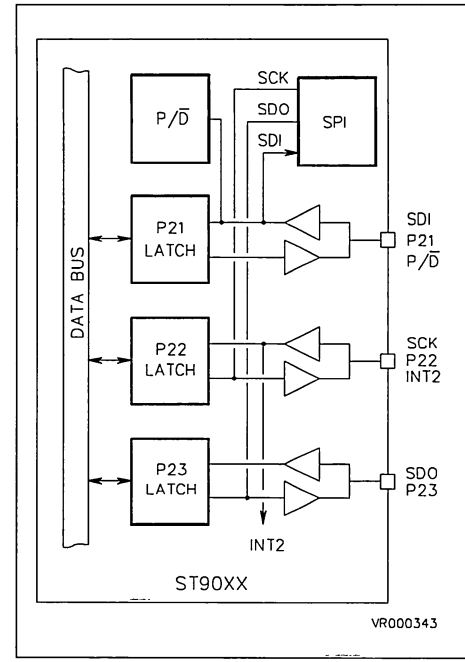
The INT2 input Function is always mapped together with the SCK input Function to allow start/stop bit detection when using S-bus/I²C-bus protocols.

A delay instruction (e.g. a NOP instruction) should be inserted between the SPEN toggle instruction and the interrupt pending bit reset instruction.

Table 11-1. Interrupt Configuration

SPEN	BMS	Interrupt Source
0	0	External channel INT2
0	1	S-bus/I²C bus start or stop condition
1	X	End of one byte transmission

Figure 11-3. SPI I/O Pins



11.4 SPI REGISTERS

SPI uses two registers mapped on page 0 of the register file:

SPIDR R253 (FDh) Page 0 Read/Write
SPI Data Register (R253)

Reset Value: 0000 0000b (00h)

7							0
D7	D6	D5	D4	D3	D2	D1	D0

b7-b0 = **D0-D7:** *SPI Data Bits.* This register contains the data transmitted and received by the SPI. Data is transmitted b7 first, and receives incoming data into b0. Transmission is started by writing to this register.

SPICR R255 (FEh) Page 0 Read/Write
SPI Control Register (R254)

Reset Value: 0000 0000b (00h)

7							0
SPEN	BMS	ARB	BUSY	CPOL	CPHA	SPR1	SPR0

b7 = **SPEN:** *Serial Peripheral Enable.* When set, the two alternate functions SCK and SDO are enabled. When disabled, SCK and SDO are kept in high impedance. Furthermore, SPEN affects the selection of the source for interrupt channel B0. Transmission will start by simply writing the data into the SPIDR Register.

b6 = **BMS:** *S-bus/I²C-bus Mode Selector.* This bit should be set to "1" when the SPI is used in an S-bus/I²C-bus protocol. It enables S-bus/I²C-bus arbitration, clock synchronization and Start/ Stop detection.
When this bit is reset to "0", a reinitialisation of the SPI logic is performed allowing recovery procedures after a R_X/T_X failure. BMS (and SPEN) affects the selection of the source for interrupt channel B0.

b5 = **ARB:** *Arbitration flag bit.* This bit is set when the SPI, in S-bus/I²C-bus mode, loses arbitration, and is reset when an S-bus/I²C-bus stop condition is detected. ARB can be reset by software. When ARB is set automatically, the SDO pin is set to high value until a write instruction on SPIDR is performed.

b4 = **BUSY:** *SPI Busy Flag.* BUSY flag is set when a transmission is in process. This bit allows the user to monitor the SPI status by polling its value.

b3 = **CPOL:** *Transmission Clock Polarity.* CPOL controls the normal or steady state value of the clock when data is not being transferred.

As the SCK line is held in a high impedance state when the SPI is disabled (SPEN = "0"), the SCK pin must be connected to V_{SS} or V_{CC} through a resistor according to the CPOL state. Polarity should be selected during the reset routine according to the value set into all peripherals and must not be changed during program execution.

b2 = **CPHA:** *Transmission Clock Phase.* CPHA controls the relationship between the data on the SDI and SDO pins and the clock produced at the SCK pin. CPHA bit selects the clock edge which captures data and allows it to change state. It has its greatest impact on the first bit transmitted (MSB) because it does (or does not) allow a clock transition before the first data capture edge.
Figure 11-5 shows the relationship between CPHA, CPOL and SCK, and indicates active clock edges and strobe times.

CPOL	CPHA	SCK on Figure 11-5
0	0	(a)
0	1	(b)
1	0	(c)
1	1	(d)

b1-b0 = **SPR1,SPR0:** *SPI Rate.* These two bits select one (out of four) baud rates to be used as SCK.

SPR1	SPR0	Clock Divider	SCK Frequency (INTCLK = 12MHz)	
0	0	8	1500kHz	(T = 0.67µs)
0	1	16	750kHz	(T = 1.33µs)
1	0	128	93.75kHz	(T = 10.66µs)
1	1	256	46.87kHz	(T = 21.33µs)

11.5 WORKING with DIFFERENT PROTOCOLS

The SPI peripheral offers the following facilities to work with S-bus/I²C-bus and IM-bus protocols:

- Interrupt request on start/stop detection
- Hardware clock synchronisation
- Arbitration lost flag with an automatic set of data line

Note that the I/O bit associated to the SPI should be returned to a defined state as a normal I/O pin before changing the SPI protocol.

The following paragraphs provide information to manage these protocols.

11.5.1 I²C-bus Interface

I²C-bus is a two-wire bidirectional data-bus, the two lines being SDA (Serial DAta) and SCL (Serial CLock). Both are open drain lines to allow arbitration. As shown in figure 11-6, data is toggled with clock low and Start and Stop conditions are detected when a high to low (start) or a low to high (stop) transition on the SDA line occurs with the SCL line high.

Each transmission consists of nine clock pulses (SCL line). The first 8 pulses transmit the byte (msb first), the ninth is used by the receiver to acknowledge.

Figure 11-4. S-Bus/I²C-bus Peripheral Compatibility without S-Bus Chip Select

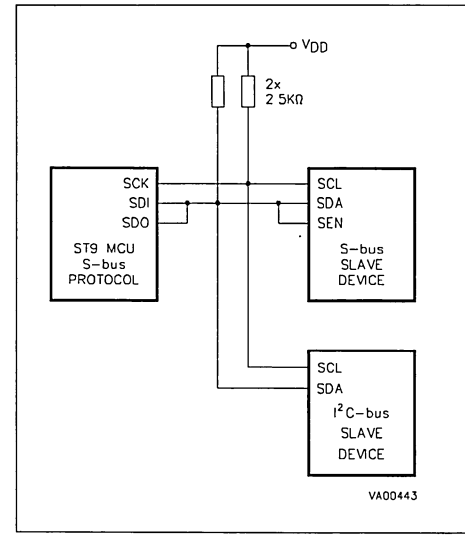
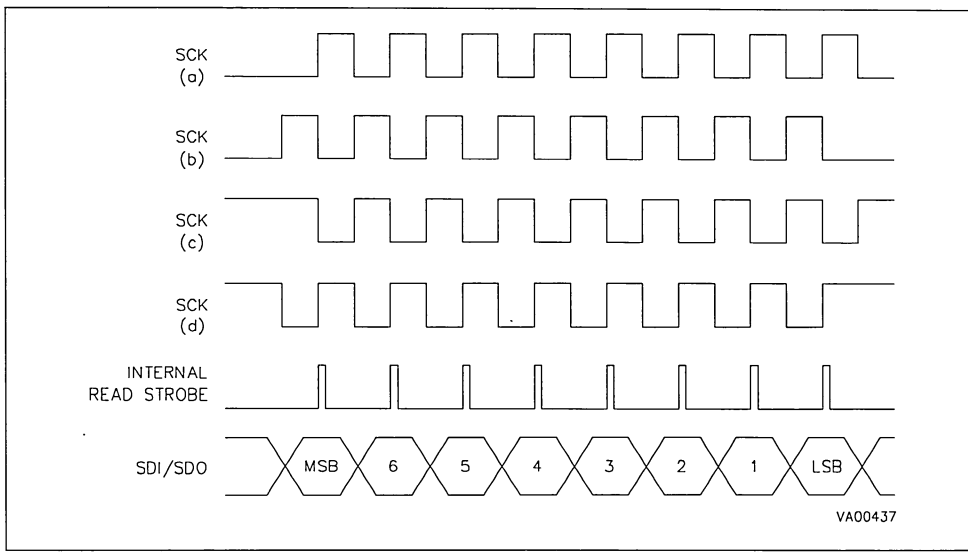


Figure 11-5. SPI Data and Clock Timing

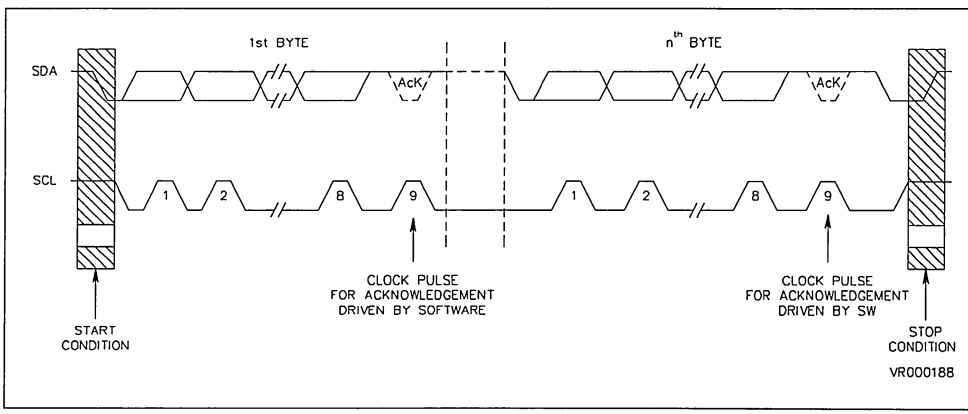


DIFFERENT PROTOCOLS (Continued)

Table 11-2. Typical I^2C-bus Sequences

Phase	Software	Hardware	Notes
INITIALIZE	SPICR.CPOL, CPHA = 0, 0 SPICR.SPEN = 0 SPICR.BMS = 1 SCK pin set as AF output SDI pin set as input Set SDO port bit to 1	SCK, SDO IN HI-Z SCL, SDA = 1, 1	Set polarity and phase SPI disable START/STOP interrupt Enable
START	SDO pin set as output Open Drain Set SDO port bit to 0	SDA = 0, SCL = 1 interrupt request	START condition receiver START detection
TRANSMISSION	SPICR.SPEN = 1 SDO pin as Alternate Function output load data into SPIDR	SCL = 0 Start transmission interrupt request	Managed by interrupt routine load FFh when receiving end of transmission detection
ACKNOWLEDGE	SPICR.SPEN = 0 Poll SDA line Set SDA line SPICR.SPEN = 1	SCK, SDO in HI-Z SCL, SDA = 1 SCL = 0	SPI disable only if transmitting only if receiving only if transmitting
STOP	SDO pin set as output Open Drain SPICR.SPEN = 0 Set SDO port bit to 1	SDA = 1 interrupt request	STOP condition

Figure 11-6. SPI Data and Clock Timing



DIFFERENT PROTOCOLS (Continued)

The data on the SDA line is sampled with the low to high transition on the SCL line.

SPI Working With I²C-bus

To use the SPI with the I²C-bus protocol, the SCK line is used as SCL, the SDI and SDO lines, externally wired-OR'd, are used as SDA. All the output pins must be configured as open drain (see Figure 11-6).

Table 11-2 shows the typical I²C-bus sequence divided in 5 phases: initialize, start, transmission, acknowledge and stop.

Software and hardware will take care of each phase. A master to slave transmission can be managed as example according to the following table.

During the transmission phase, the following I²C-bus features are also supported by hardware.

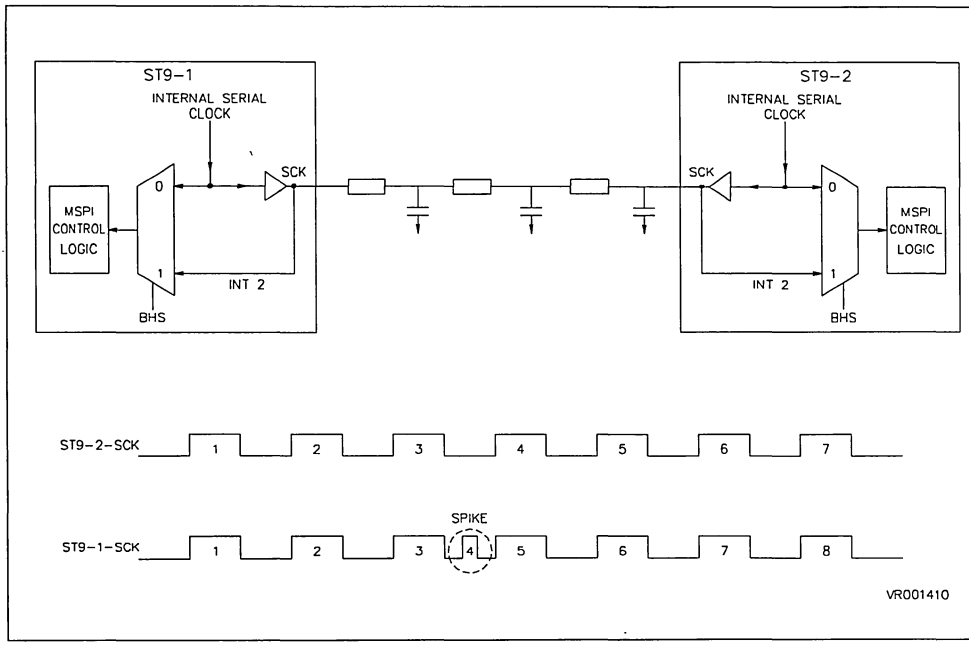
Clock Synchronization

In a multimaster I²C-bus system, when more masters generate their own clock, synchronization is needed. The first master which releases the SCL line stops internal counting, restarting only when the SCL line goes high (released by all the other masters). In this way, devices using different clock sources and different frequencies can be interfaced.

Arbitration Lost

When more masters are sending data on SDA line, the following mechanism is performed: if the transmitter sends a "1" and SDA line is forced low by another device the ARB flag (SPICR.5) is set and the SDO buffer is "switched off. (ARB is reset and SDO buffer is "switched on" when SPIDR is written to again). When BMS is set to "1" the peripheral clock is supplied through the INT2 line by the external clock line (SCL). Due to potential noise spikes (which must last longer than one INTCLK period to be detected), RX or TX may gain a clock pulse.
Referring to Figure 11-7, if ST9-1 detects a noise spike and gains a clock pulse, it will stop its transmission in advance and hold the clock line low causing ST9-2 to be frozen at the 7th bit. To exit and recover from this condition the BMS bit must be reset to "0", this will cause the reset of the SPI logic, aborting the current transmission. An End of Transmission interrupt is generated after this reset sequence.

Figure 11-7. SPI Arbitration



DIFFERENT PROTOCOLS (Continued)

11.5.2 S-Bus Interface

S-bus is a three-wire bidirectional data-bus, with functional features similar to I^2C-bus. Differently from I^2C-bus, the START/STOP conditions are given by encoding the information on 3 wires instead of 2, as shown in Figure 11-8. The additional line is referred as SEN.

SPI Working With S-bus

The S-bus protocol uses the same pin configuration as I^2C-bus for generating the SCL and SDA lines. The additional SEN line is managed through a standard ST9 I/O port under software control (see Figure 11-9).

Figure 11-8. Mixed S-bus and I^2C-bus system

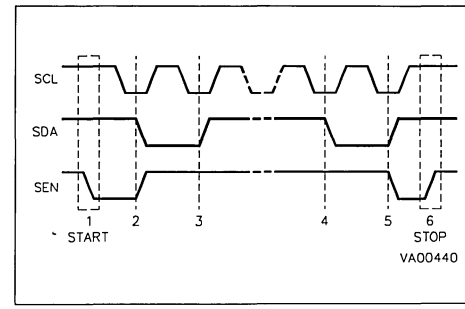


Figure 11-9. S-bus Configuration

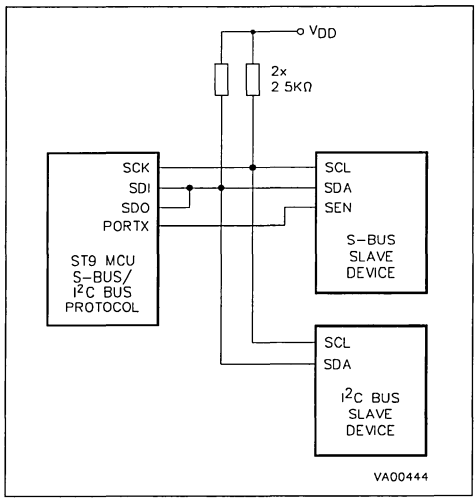
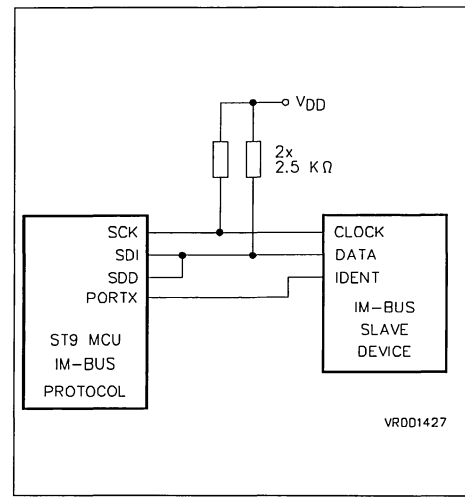


Figure 11-10. ST9 and InterMetal Peripheral



DIFFERENT PROTOCOLS (Continued)

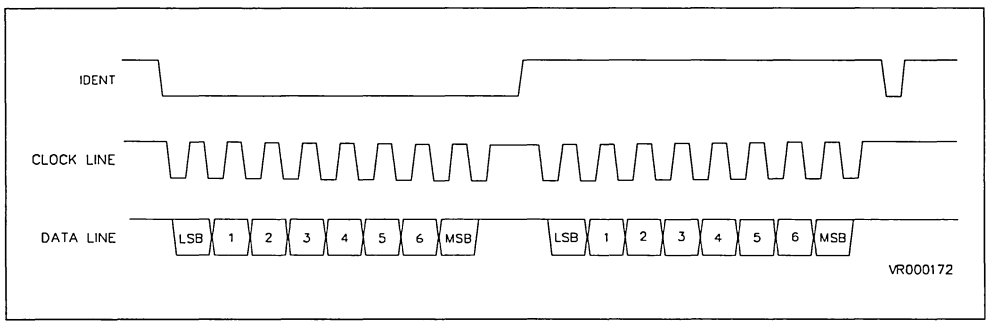
11.5.3 IM-Bus Interface

The IM-bus has a bidirectional data line and a clock line, and in addition it requires an IDENT line that distinguishes an address from a data byte (Figure 11-11). Unlike the I^2C-bus protocol, the IM-bus protocol sends the least significant bit first, this requires a software routine which reverses the bit order before sending, and after receiving a data byte. Figure 11-10 shows the connections for an IM-bus peripheral to an ST9 SPI. The SDO and SDI pins are connected to the bidirectional data pin of the peripheral device. The SDO alternate function is set in Open Drain (external 2.5KΩ pull-up resistors are required).

With this type of configuration, data is sent to the peripheral by writing the data byte to SPIDR. To receive data from the peripheral, the User should write FFh into SPIDR in order to generate the shift clock pulses. As the SDO line is set to the Open Drain configuration, the incoming data bits that are set to one do not affect the SDO/SDI line status (which defaults to a high level due to the FFh in the transmit register), while incoming bits that are set to "0" pull the input line low.

In software it is necessary to initialise the ST9 SPI with CPOL and CPHA set to "1", "1". By using a general purpose I/O as the IDENT line and forcing it to a logical "0" when writing to SPIDR, an address is sent (or read). Then, by setting this bit to a logical "1" and writing to SPIDR, data is sent to the peripheral. When all the address and data pairs are sent it is necessary to drive the IDENT line low and high to create a short pulse. In this way the stop condition is generated.

Figure 11-11. IM bus Timing



12 TIMER/WATCHDOG

12.1 INTRODUCTION

A programmable 16-bit down counter with an 8-bit prescaler is included in the ST9 Core. This Timer can be programmed to be used as a general purpose 16-bit Timer, with associated input and output pins for timing functions, or as a Watchdog Timer offering security against possible processor malfunctions due to hardware or software failures.

The Timer/Watchdog functions can use inputs from an external pin and an Alternate Function output of an I/O Port. The Input pin can be used in one of the four programmable input modes:

- event counter,
- gated external input mode,
- triggerable input mode,
- retriggerable input mode.

The output pin can be used to generate a square or a Pulse Width Modulated signal.

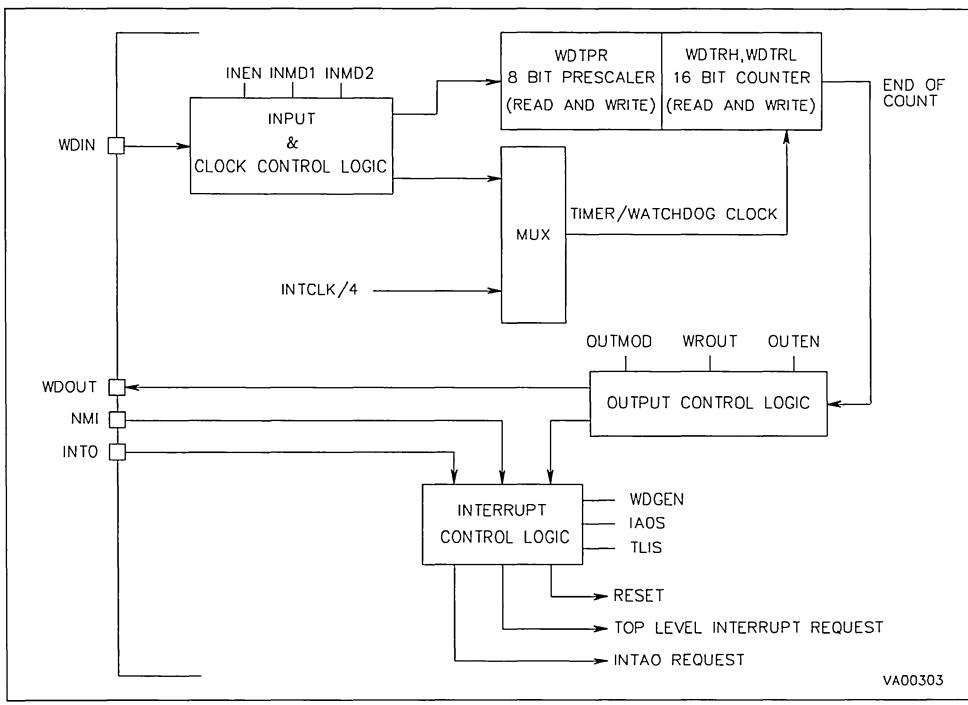
An interrupt generated by the unit (when running as a 16-bit Timer/counter and not as Watchdog) can be used as a Top Level Interrupt or as an interrupt source connected to channel A0 of the external interrupt structure (replacing the INT1 interrupt input).

The clock for the counter can be driven either by an external clock or an internal clock equal to INTCLK divided by 4.
When using an external 24MHz crystal (INTCLK = 12MHz), the End Of Count rate is:

5.59 sec. for Max. Count (Timer Const. = FFFFh, Prescaler Const. = FFh)

333 nsec. for Min. Count (Timer Const. = 0000h, Prescaler Const. = 00h)

Figure 12-1. Block Diagram



12.2 FUNCTIONAL DESCRIPTION

12.2.1 Timer/Counter Input Modes

Setting the Input Enable (INEN) bit enables the input mode which is selected via the INMD1 and INMD2 bits. When INEN is reset to zero, the input section is disabled and the values of INMD1 and INMD2 are don't-care.

Event Counter Mode
(INMD1 = "0", INMD2 = "0")

The Timer is driven by the signal applied to the input pin which acts as an external clock. The unit works therefore as an event counter. The event is a high to low transition of the input signal.
Spacing between trailing edges should be at least 350ns (i.e. the maximum Watchdog Timer input frequency is 2.9MHz with INTCLK = 12MHz).

Gated Input Mode
(INMD1 = "0", INMD2 = "1")

The Timer uses the Watchdog internal clock (INTCLK divided by 4) and starts and stops the Timer according to the input pin. When the status of the Input pin is High the Timer Watchdog count operation proceeds, and when Low, counting is stopped.

Retriggerable Input Mode
(INMD1 = "1", INMD2 = "1")

A Timer/Watchdog start is caused by:
a) a set of the Start-Stop bit, or
b) a High to Low (low trigger) transition on the input pin.

In order to stop the Timer, it is only necessary to reset the Start-Stop bit to zero.

Triggerable Input Mode
(INMD1 = "1", INMD2 = "0")

In this mode when the Timer is running (TIMER/WATCHDOG internal clock), a High to Low transition of the input pin causes the counting to start from the initial value. When the Timer is stopped (ST_SP bit equal to zero), a High to Low transition of the input pin has no effect.

12.2.2 Timer/Watchdog Output Modes

OUTPUT modes are selected using 2 bits of WDTCR (R251): OUTEN (Output Enable) and OUTMD (Output Mode).

When OUTMD = "0", the Timer outputs a signal with a frequency equal to half the End Of Count repetition rate. With INTCLK = 12MHz, this allows generation of a square wave with a period ranging from 666ns to 11.18 seconds.

The value of the WROUT bit is transferred to the output pin at the End Of Count and the value is held until the next End of Count when OUTMD = "1". This allows the user to generate PWM signals, by modifying the status of WROUT between End of Count events, based on software counters decremented on the Timer/Watchdog interrupt.

OUTEN = "1" enables the output function selected via OUTMD

When OUTEN = "0", the output is disabled and the output pin is held at a "1" level to allow several alternate functions on the same pin.

12.2.3 Timer/Counter Control

Start/Stop

ST_SP (WDTCR.7) enables down-counting. An instruction which sets this bit will cause the Timer to start at the beginning of the following instruction. Resetting this bit will stop the counter.

If the counter is stopped and restarted, counting will resume from the last value unless a new constant has been entered in the Timer registers. A new constant can be written with the counter running. The new value will be loaded at the following End Of Count (EOC).

WARNING: *In order to prevent incorrect counting of the Timer/Watchdog, the prescaler (WDTPR) and counter (WDTRL, WDTRH) registers must be initialised before the starting of the Timer/Watchdog. If this is not done, counting will start with the reset (un-initialised) values.*

Single/Continuous Mode

SINGLE MODE: At End Of Count the Timer stops, reloads the constant, and resets the Start/Stop bit (WDTCR.6) (user may check the current status by reading this bit). Restarting is done by setting the Start/Stop bit. Note that the Timer constant is reloaded only if it has been modified during the stop period.

CONTINUOUS MODE: At End Of Count the counter automatically reloads the constant and restarts. It is stopped only if the Start/Stop bit is reset. This Mode bit can be written with the Timer stopped or running. It is possible to toggle the S_C bit and start the counter with the same instruction.

FUNCTIONAL DESCRIPTION (Continued)

12.2.4 Timer/Watchdog Mode

In this mode (WDGEN = "0") the counter generates a fixed time basis. When End Of Count is reached the Timer generates a system Reset.

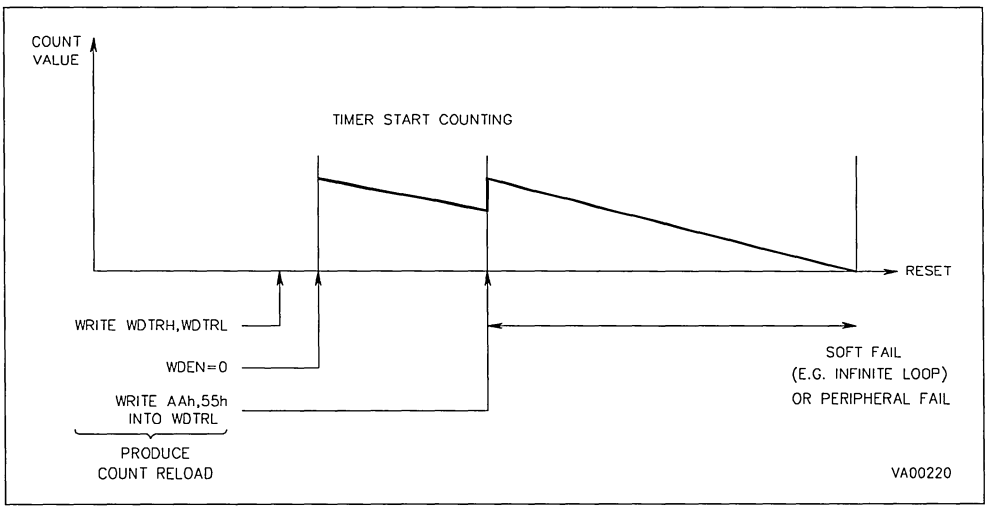
The time base is user-defined and must be written in the Timer registers before entering Watchdog mode. In Watchdog mode it is possible to modify only the Prescaler Constant. This new value will be loaded when the counter restarts.
Resetting WDGEN (bit 6 of the Wait Control Register) causes the counter to start regardless of the value of the Start-Stop. In order to prevent a system reset the sequence AAh, 55h should be entered in WDTLR (Watchdog Timer register low). Once the writing of 55h has been performed the Timer reloads the constant and counting restarts from the preset value.

The minimum time between the writing of the AAh and 55h codes is zero, i.e. the writing is sequential, and the maximum time is given by the Watchdog timeout period.

In Watchdog-mode a `halt` instruction is regarded as illegal. Execution of the `halt` instruction stops further core execution by the CPU and interrupt acknowledgment, but does not stop INTCLK or CPUCLK or the Watchdog Timer, which will cause a System Reset when reaching the End of Count. Furthermore ST_SP, S_C and input mode selection bits are "don't-care". Hence regardless of their status, the counter always runs in Continuous Mode driven by the internal clock.

The Output mode should not be enabled since that particular mode of operation is meaningless.

Figure 12-2. Timer /Watchdog in Watchdog Mode



12.3 TIMER/WATCHDOG INTERRUPT

When enabled, the Timer/Watchdog will issue an interrupt request at every End Of Count.

A pair of control bits, IA0S (EIVR.1, Interrupt A0 selection bit) and TLIS (EIVR.2, Top Level Input Selection bit) allow the selection of 2 interrupt sources (the Timer/Watchdog End of Count or an external pin) in two different ways, as a top level non maskable interrupt (Software Reset) or as a source for channel A0 of the external interrupt logic.

In the Watchdog mode the End Of Count always causes a system reset.

A block diagram of the interrupt logic is given in Figure 12-3 (Note: software traps can be generated by setting the appropriate interrupt pending bit):

The following table shows all the possible configurations of the interrupt/reset sources which involve the Timer/Watchdog:

Figure 12-3. Interrupt Sources

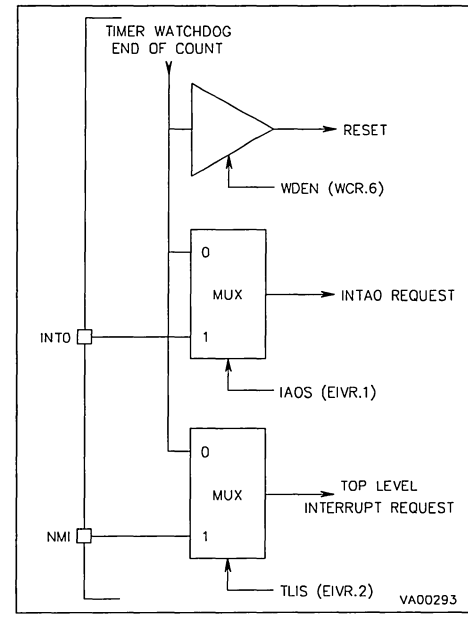


Table 12-1. Interrupt Configuration

Control Bits			Enabled Sources			Watchdog Timer Status
WDGEN	IAOS	TLIS	Reset	INTA0	Top Level	
0	0	0	WDG/Ext Reset	SW TRAP	SW TRAP	Watchdog
0	0	1	WDG/Ext Reset	SW TRAP	Ext Pin	Watchdog
0	1	0	WDG/Ext Reset	Ext Pin	SW TRAP	Watchdog
0	1	1	WDG/Ext Reset	Ext Pin	Ext Pin	Watchdog
1	0	0	Ext Reset	Timer	Timer	Timer
1	0	1	Ext Reset	Timer	Ext Pin	Timer
1	1	0	Ext Reset	Ext Pin	Timer	Timer
1	1	1	Ext Reset	Ext Pin	Ext Pin	Timer

Note.
WDG = Watchdog function
SW TRAP = Software Trap

12.4 TIMER/WATCHDOG REGISTERS

The Timer/Watchdog has 4 registers mapped into Group F, Page 0 of the Register File.

WDTHR (R248): Timer/Watchdog Counter High Register
WDTLR (R249): Timer/Watchdog Counter Low Register
WDTPR (R250): Timer/Watchdog Prescaler Register
WDTCR (R251): Timer/Watchdog Control Register

Three additional control bits are mapped in the following registers of Page 0:

- watchdog mode enable, WCR.6
- top level interrupt selection, EIVR.2
- interrupt A0 channel selection, EIVR.1

Note: The registers containing these bits also contain other functions. Only the bits relevant to the operation of the Timer/Watchdog are shown here.

Counter Registers

This 16 bit register is used to load the 16 bit counter value. The registers can be read or written "on the fly".

WDTHR R248 (F8h) Page 0 Read/Write
Timer/Watchdog Counter Register, High byte

Reset value: undefined

7							0
R15	R14	R13	R12	R11	R10	R9	R8

WDTLR R249 (F9h) Page 0 Read/Write
Timer/Watchdog Counter Register, Low byte.

Reset value: undefined

7							0
R7	R6	R5	R4	R3	R2	R1	R0

WDTPR R250 (FAh) Page 0 Read/Write
Timer/Watchdog Prescaler Register

Reset value: undefined

7							0
PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0

b7-b0 = **PR7-PR0:** *Timer/Watchdog Prescaler.* The value stored in this Register is used to select the prescaling factor from 1 (loading 00h) to 256 (loading FFh).

WARNING. *In order to prevent incorrect counting of the Timer/Watchdog, the prescaler (WDTPR) and counter (WDTRL, WDTRH) registers must be initialised before the starting of the Timer/Watchdog. If this is not done, counting will start with the reset (un-initialised) values.*

WDTCR R251 (FBh) Page 0 Read/Write
Timer/Watchdog Control Register

Reset value: 0001 0010 (12h)

7							0
ST_SP	S_C	INMD1	INMD2	INEN	OUTMD	WROUT	OUTEN

b7 = **ST_SP:** *Start/Stop Bit.* Setting this bit to a "1" starts the counting operation (see Warning above). When this bit is "0", the counter is stopped (reset status)

b6 = **S_C:** *Single/Continuous.* When this bit is set, the counter operates in Single Count Mode. Continuous Mode is set when this bit is "0"

b5-b4 = **INMD1, INMD2:** *Input mode selection bits.*

b3 = **INEN:** *Input Enable.* This bit enables ("1") and disables ("0") the input section

b2 = **OUTMD:** *Output Mode.* When this bit is "1", and the output is enabled, the value of WROUT is transferred to the output pin on every End Of Count. When "0", the output is toggled on every End of Count

b1 = **WROUT:** *WROUT bit.* The status of this bit is transferred to the Output pin when OUTMD = "1", it is user definable to allow PWM output (at reset WROUT = "1")

b0 = **OUTEN:** *Output Enable bit.* The output is enabled by setting this bit to "1", and disabled by resetting to "0"

TIMER/WATCHDOG REGISTERS (Continued)

WCR R252 (FCh) Page 0 Read/Write

Wait Control Register

Reset value: 0111 1111 (7Fh)

7							0
X	WDGEN	X	X	X	X	X	X

b6 = **WDGEN:** *Watchdog Enable Bit (active low).* Resetting this bit to zero via software enters the Watchdog mode. Once reset, it cannot be set to "1" by the user program. At system reset, the Watchdog mode is disabled

EIVR R246 (F6h) Page 0 Read/Write

External Interrupt Vector Register

Reset value: xxxx 0110 (X6h)

7							0
X	X	X	X	X	TLIS	IAOS	X

b2 = **TLIS:** *Top Level Input Selection bit.* This bit selects the Top Level interrupt source. When "0", the Top Level interrupt source is the Watchdog/Timer end of count, when = "1", it is the external pin NMI.

b1 = **IAOS:** *Interrupt channel A0 Selection Bit.* This bit allows the Timer/Watchdog interrupt to channel through the external Interrupt A0 source, allowing the setting of user-defined priority levels.

WARNING. *To avoid spurious interrupt requests, an access to the IAOS bit must be made only when the interrupt logic is disabled (i.e. after the DI instruction). It is also necessary to clear a possible interrupt pending request on channel A0 before enabling this interrupt channel. A delay instruction (e.g. a NOP instruction) must be inserted between the reset of the interrupt pending bit and the IAOS write instruction.*

13 MULTIFUNCTION TIMER

13.1 INTRODUCTION

The Multifunction Timer is a 16-bit Up/Down counter, driven by the output of an 8-bit prescaler which may be driven by INTCLK/3 (giving a minimum timing resolution of 250ns at INTCLK = 12 MHz) or by an external source.
This timer is supported by two 16-bit Comparison Registers (CMP0R, CMP1R) for generating timed functions and two 16-bit Capture/(re)Load Registers (REG0R, REG1R) for timing and variable timebase functions. These features coupled with 2 input pins (TxINA and TxINB) and 2 Alternate Function output pins (TxOUTA and TxOUTB), where x = the number of the Timer, give the Timer 12 operating modes including automatic PWM generation and frequency measurement.

Several functional configurations are possible, e.g.:

- 2 input captures on two different external lines and 2 independent output compare functions (counter in free running mode), or 1 output compare on a fixed repetition rate.
- 1 input capture, 1 counter reload and 2 independent output compares.
- 2 alternate autoreloads and 2 independent output compares.
- 2 alternate captures on the same external line and 2 independent output compares on a fixed repetition rate.

When two timers are present on ST9 chip, a combined mode is available.

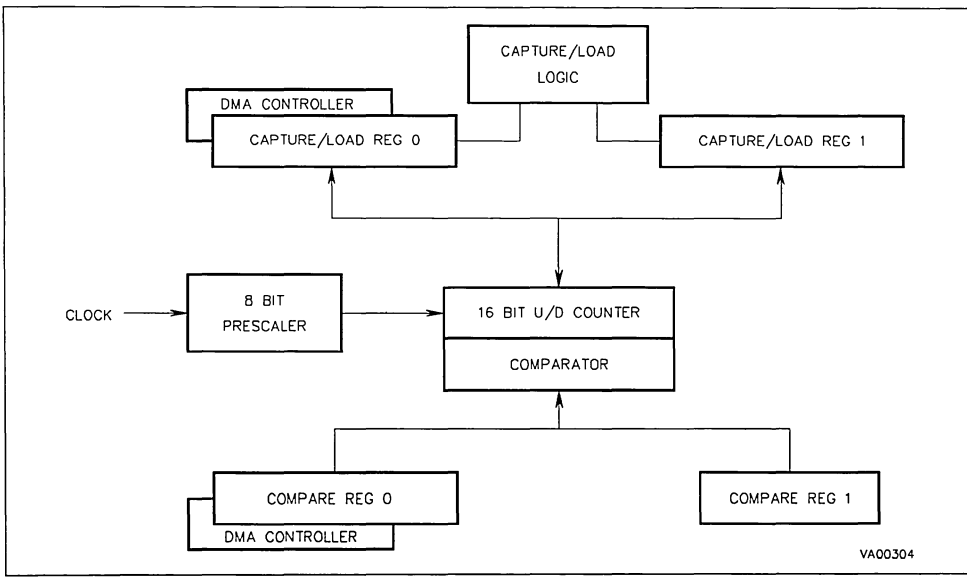
Four internal signals are also available for timing of on-chip functions: the On Chip Event signal can be used to control other peripherals on the chip itself, and 3 other signals which can be internally connected to I/O port(s) in order to allow automatic, timed, DMA transfers.

The two external inputs (TxINA/TxINB) of the timer can be individually programmed to catch a particular external configuration, i.e.:

- rising edge
- falling edge
- rising and falling edges

The configuration of each input is fixed by the Input Control Register (ICR).

Figure 13-1. MFT Simplified Block Diagram



INTRODUCTION (Continued)

Each of the two output pins (TxOUTA/TxOUTB) can be driven from any of three possible sources:

- Compare Register 0 logic
- Compare Register 1 logic
- Overflow/Underflow logic

Each of these three sources can cause one of the following four effects, independently, on each of the two outputs:

- Nop
- Set
- Reset
- Toggle

Furthermore an additional on-chip Event signal can be generated by two of the three sources mentioned above, i.e. Over/Underflow event and Compare 0 event. This signal can be used internally as synchronism for another on-chip peripheral or as strobe for an I/O port (see I/O port chapter).

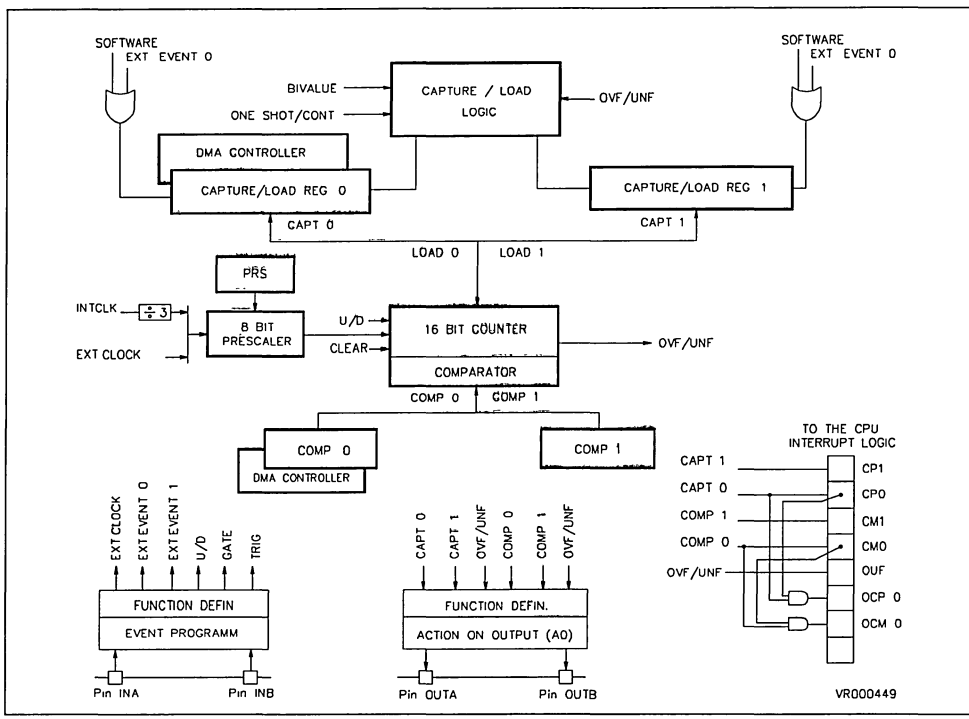
Five maskable interrupt sources referring to an End Of Count condition, 2 input captures and 2 output compares, can generate 3 different interrupt requests (with hardware fixed priority), pointing to 3 interrupt routine vectors.

Two independent DMA channels are available for a MFtimer and can be used for quick data flow operations. Each DMA request (associated to a capture on REG0R register, or a compare on CMP0R register) has priority on the INT request generated by the same source.

Each DMA channel can be employed in external transfers to/from memory from/to an I/O port using three internal lines (one for setting the data flow direction, and two for the transfer synchronization).

A SWAP mode is also available to allow high speed continuous transfers (see Interrupt and DMA chapter).

Figure 13-2. Detailed Block Diagram



13.2 FUNCTIONAL DESCRIPTION

The operating modes of the timer can be selected by programming the Timer Control Register (TCR) and the Timer Mode Register (TMR).

13.2.1 One Shot Mode

When the counter generates an overflow (in up-count mode) or an underflow (in down-count mode), i.e. an End Of Count is reached, the counter stops and no counter reload occurs. The counter can be restarted only by an external or software trigger. The One Shot Mode is entered by setting TMR bit CO.

13.2.2 Continuous Mode

Whenever the counter reaches an End Of Count, the counting sequence is automatically restarted and the counter is reloaded from REG0R (or REG1R when selected in Biload Mode). Continuous Mode is entered by resetting TMR bit C0.

13.2.3 Trigger And Retrigger Modes

A trigger event may be generated either by software action (setting either CP0 or CP1 bit in timer register FLAGR), or by an external source which may be programmed to be active on the rising edge, the falling edge or both, using the fields A0-A1 and B0-B1 in ICR.

In One Shot and Trigger Mode, every trigger event (used as a reload and start count) arriving before an End Of Count, is masked. In One Shot and Retrigger Mode, every trigger (used as a reload and start count) received while the counter is running automatically reloads the counter from REG0R (or REG1R when the register is selected in Biload Mode). Trigger/Retrigger Mode is set by the REN bit in TMR.

TxINA input refers to REG0R and TxINB input refers to REG1R.

WARNING. *If the Trigger Mode is selected when the counter is in Continuous Mode, then every trigger to reload the counter starting value is disabled, so it is not possible to synchronize the counting cycle by hardware or software.*

13.2.4 Gate Mode

In this mode the counting operation is performed only when the external gate input is active (logical state "0"). The selection of TxINA or TxINB input as gate input is made through IN0-IN3 bits in ICR.

13.2.5 Capture Mode

REG0R and REG1R registers may be independently set in Capture Mode by setting RM0 or RM1 in TMR, so that a capture of the current count value can be performed either on REG0R or REG1R, via software action (by setting CP0 or CP1 in the FLAGR register) or a programmable event on the external input pins.

WARNING. *Care should be taken when two software captures have to be performed on the same register. In this case, at least one extra instruction must be present between the first CP0/CP1 bit set and the subsequent CP0/CP1 bit reset.*

13.2.6 Up/Down Mode

The counter can count up or down depending on the state of the UDC bit (Software Up/Down) in TCR, or on the configuration of the external input pins, which have priority over UDC (see Input pin assignment in ICR). When read, the UDCS bit always returns the counter up/down current status (see also the Up/Down Autodiscrimination mode in the Input Pin Assignment Section).

13.2.7 Free Running Mode

The timer performs full range counting (in up or down mode) without reloading from REG0R at an End Of Count. This mode is automatically selected either in Bicapture Mode or by setting REG0R for capture function (Continuous Mode must also be set). In Autoclear Mode, free running with modulo less than 2^{16} may be obtained (see Autoclear Mode).

FUNCTIONAL DESCRIPTION (Continued)

13.2.8 Monitor Mode

When RM1 bit in TMR is reset and the timer is not in Bivalue Mode, then REG1R acts as monitor, reproducing the current U/D counter content enabling the ST9 to read the counter "on the fly".

13.2.9 Autoclear Mode

A clear command forces the counter to the value 0000h or 0FFFFh, when counting in up or down count mode respectively. The counter reset may be obtained either directly, through CCL bit in TCR, or by entering the Autoclear Mode, through CCP0 and CCMP0 fields in TCR.

Every capture performed on REG0R (if CCP0 = "1"), or every successful compare performed by CMP0R (if CCMP0 = "1"), clears the counter and reloads the prescaler.

The Clear On Capture mode allows the direct measurement of delta time between successive captures on REG0R, while the Clear On Compare mode allows free running with modulo less than 2^{16}.

13.2.10 Bivalue Mode

Depending on the value of RM0 bit in TMR, the Biload Mode (RM0 = "0") or the Bicapture Mode (RM0 = "1") can be selected as explained in the following table:

Table 13-1. Bivalues Modes

TMR bits			Timer Operating Modes
RM0	RM1	BM	
0	X	1	BiLoad mode
1	X	1	BiCapture Mode

A) Biload Mode

The Biload Mode is entered by selecting the Bivalue Mode (BM = "1" in TMR) and programming REG0R as a reload register (RM0 = "0" in TMR).

At any End Of Count, the counter reloading is performed alternately from REG0R and REG1R, (a low level for BM bit always sets REG0R as the current register, so that, after a Low to High transition of BM bit, the first reload is always from REG0R).

Every software or external trigger event on REG0R performs a reload from REG0R resetting the Biload cycle. In One Shot mode (reload made by a software or external trigger), the reload is always from REG0R.

B) Bicapture Mode

The Bicapture Mode is entered selecting the Bivalue Mode (BM = "1" in TMR) and programming REG0R as a capture register (RM0 = "1" in TMR).

Every capture event, software simulated (by setting CP0 flag) or from the TxINA input line, captures the current counter value alternately into REG0R and REG1R. A low level for BM bit always sets REG0R as current register, so that the first capture, after setting BM bit, is always into REG0R.

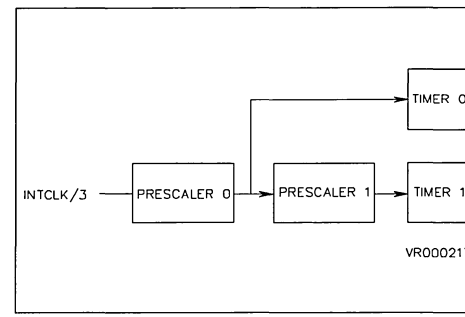
13.2.11 Parallel Mode

When there are two timers on ST9 chip, the parallel mode is entered with ECK ="1" in TMR of Timer 1. Timer 1 prescaler input is internally connected to the Timer-0 prescaler output. Timer 0 prescaler input is connected to the system clock line.

By loading the Prescaler Register of Timer 1 with the value 00h the two timers (Timer 0 and Timer 1) are driven by the same frequency in parallel mode.

13.2.12 Autodiscriminator Mode

Figure 13-3. Parallel Mode Description



The phase difference sign of two overlapped pulses (respectively on TxINB and TxINA) generates a one step up(down) count, so that the up/down control and the counter clock are both external. The setting of the UDC bit in the TCR register has no effect in this configuration.

13.3 iNPUT PIN ASSIGNMENT

The two external inputs (TxINA and TxINB) of the timer.can be individually configured to catch a particular external event (i.e. rising edge, falling edge, rising and falling edges) by programming the two relevant bits (A0, A1 and B0, B1) for each input in the external Input Control Register (ICR).

The 16 different functional modes of the two external inputs can be selected by programming IN0 - IN3 bits of the ICR as explained in the following table.

Table 13-2. Input Pin Function

I C Reg. IN3-IN0 bits	TxINA Input Function	TxINB Input Function
0000	not used	not used
0001	not used	Trigger
0010	Gate	not used
0011	Gate	Trigger
0100	not used	Ext. Clock
0101	Trigger	not used
0110	Gate	Ext. Clock
0111	Trigger	Trigger
1000	Clock Up	Clock Down
1001	Up/Down	Ext. Clock
1010	Trigger Up	Trigger Down
1011	Up/Down	not used
1100	Autodiscr.	Autodiscr.
1101	Trigger	Ext. Clock
1110	Ext. Clock	Trigger
1111	Trigger	Gate

Some choices in the external input pin assignment are defined in conjunction with RM0 and RM1 bits in TMR.

For input pin assignment codes using the input pins as Trigger Inputs (except for code 1010, Trigger Up:Trigger Down):

- a trigger signal on TxINA input pin performs an U/D counter load if RM0 ="0", or an external capture if RM0 = "1".
- a trigger signal on TxINB input pin always performs an external capture on REG1R. The TxINB input pin is disabled when the Bivalue Mode is set.

Note. For proper operation of the External Input pins, the following must be observed:

– the minimum external clock/trigger pulse width cannot be less than the system clock (INTCLK) period if the input pin is programmed as rising or falling edge sensitive.

– the minimum external clock/trigger pulse width cannot be less than the prescaler clock period (INTCLK/3) if the input pin is programmed as rising and falling edges sensitive (valid also in Autodiscrimination mode). - the minimum delay between two clock/trigger pulse active edges must be greater than the prescaler clock period (INTCLK/3), while the minimum delay between two consecutive clock/trigger pulses must be greater than the system clock (INTCLK) period.

– the minimum gate pulse width must be at least twice the prescaler clock period (INTCLK/3).

– in Autodiscrimination mode, the minimum delay between the input pin A pulse edge (inside the input pin B pulse) and the edges of the input pin B pulse, must be at least the system clock (INTCLK) period.

– if a number N of external pulses must be counted using a Compare Register of a Timer in External Clock mode, then the Compare Register used must be loaded with the value [X +/- (N-1)], where X is the starting counter value and the sign is chosen depending if in Up or Down count mode respectively.

The sixteen external input functional modes available (referring to Table 13-2) are:

13.3.1 TxINA = I/O - TxINB = I/O

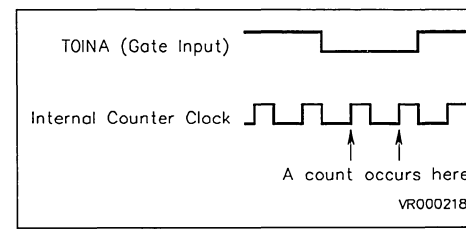
Input pins A and B are not used by the Timer. The counter clock is internally generated and the up/down control may be made only by software action through the UDC (Software Up/Down) bit in the TCR register.

13.3.2 TxINA = I/O - TxINB = Trigger

The signal applied to input pin B acts as a trigger signal on REG1R register. The prescaler clock is internally generated and the up/down control may be made only by software action through the UDC bit in the TCR register.

13.3.3 TxINA = Gate - TxINB = I/O

The signal applied to input pin A acts as a gate signal for the internal clock (i.e. the counter runs only when the gate signal is at a low level). The counter clock is internally generated and the up/down control may be made only by software action through the UDC bit in the TCR register.



INPUT PIN ASSIGNMENT (Continued)

13.3.4 TxINA = Gate - TxINB = Trigger

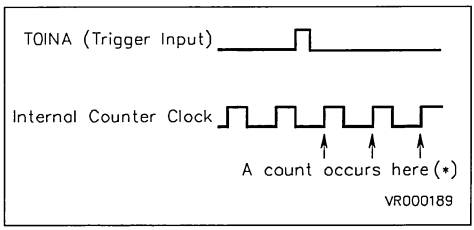
Both input pins A and B are connected to the timer, with the resulting effect of combining the actions due to the above explained configurations.

13.3.5 TxINA = I/O - TxINB = Ext. Clock

The signal applied to input pin B is used as the external clock for the prescaler. The up/down control may be made only by software action through the UDC bit in the TCR register.

13.3.6 TxINA = Trigger - TxINB = I/O

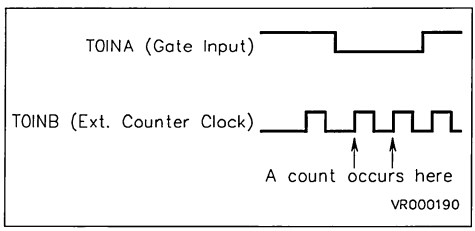
The signal applied to input pin A acts as a trigger signal on REG0R register performing the action for which the register was programmed (i.e. a reload or capture). The prescaler clock is internally generated and the up/down control may be made only by software action through the UDC bit in the TCR register.



(*) The timer is in One shot mode and REG0R in Reload mode

13.3.7 TxINA = Gate - TxINB = Ext. Clock

The signal applied to input pin B, gated by the signal applied to input pin A, acts as external clock for the prescaler. The up/down control may be made only by software action through the UDC bit in the TCR register.

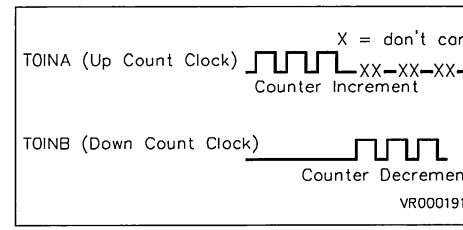


13.3.8 TxINA = Trigger - TxINB = Trigger

The signal applied to input pin A (or B) acts as trigger signal for the REG0R (or REG1R) register performing the action for which the register has been programmed. The counter clock is internally generated and the up/down control may be made only by software action through the UDC bit in the TCR register.

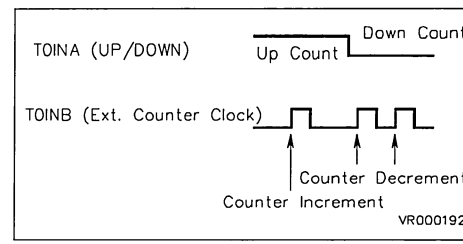
13.3.9 TxINA = Clock Up - TxINB = Clock Down

The pulse received on input pin A (or B) performs a one step up (or down) count, so that the counter clock and the up/down control are external. Setting the UDC bit in the TCR register has no effect in this configuration while input pin B has priority on input pin A.



13.3.10 TxINA = Up/Down - TxINB = Ext Clock

An High (or Low) level of the signal applied on input pin A sets the counter in the up (or down) count mode, while the signal applied to input pin B is used as clock for the prescaler. Setting the UDC bit in the TCR register has no effect in this configuration.

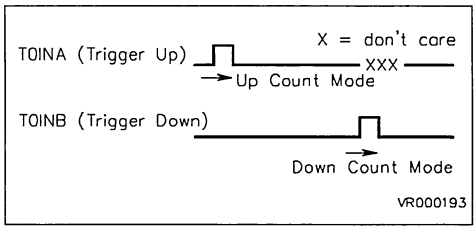


13.3.11 TxINA = Trigger Up - TxINB = Trigger Down

Up/down control is performed through both input pins A and B. A pulse on input pin A sets the up count mode, while a pulse on input pin B (which has priority on input pin A) sets the down count

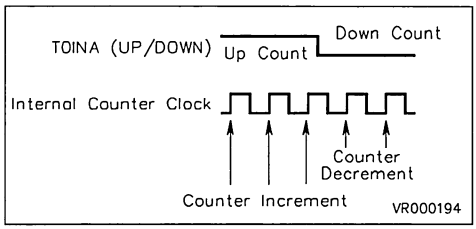
INPUT PIN ASSIGNMENT (Continued)

mode. The counter clock is internally generated while setting the UDC bit in the TCR register has no effect in this configuration.



13.3.12 TxINA = Up/Down - TxINB = I/O

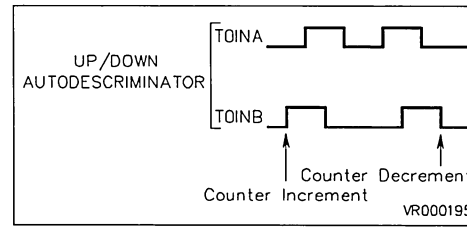
An High (or Low) level of the signal applied on input pin A sets the counter in the up (or down) count mode. The counter clock is internally generated. Setting the UDC bit in the TCR register has no effect in this configuration.



13.3.13 Autodiscrimination Mode

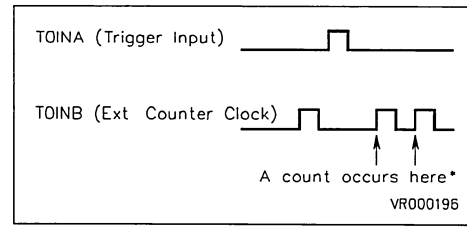
The phase between two pulses (respectively on input pin B and input pin A) generates a one step up (or down) count, so that the up/down control and the counter clock are both external. Thus, if the rising edge of TxINB arrives when TxINA is at level "0" the timer is incremented (no action if the rising edge of TxINB arrives when TxINA is at level "1"). If the falling edge of TxINB arrives when TxINA is at level "0" the timer is decremented (no action if the falling edge of TxINB arrives when TxINA is at level "1").

Setting the UDC bit in the TCR register has no effect in this configuration.



13.3.14 TxINA = Trigger - TxINB = Ext. Clock

The signal applied to input pin A acts as a trigger signal on REG0R register performing the action for which the register was programmed (i.e. a reload or capture), while the signal applied to input pin B is used as clock for the prescaler.



(*) The timer is in One shot mode and REG0R in reload mode

13.3.15 TxINA = Ext. Clock - TxINB = Trigger

The signal applied to input pin B acts as a trigger, performing a capture on REG1R register, while the signal applied to the input pin A is used as clock for the prescaler.

13.3.16 TxINA = Trigger - TxINB = Gate

The signal applied to input pin A acts as a trigger signal on REG0R register performing the action for which the register was programmed (i.e. a reload or capture), while the signal applied to input pin B acts as a gate signal for the internal clock (i.e. the counter runs only when the gate signal is at a low level).

13.4 OUTPUT PIN ASSIGNMENT

Two external outputs are available for each timer when programmed as Alternate Function Outputs of the I/O pins.

Two registers for every timer, Output A Control Register (OACR) and Output B Control Register (OBCR) define the driver for the outputs and the actions to be performed.

Each of the two output pins can be driven from any of the three possible sources:

- Compare Register 0 event logic
- Compare Register 1 event logic
- Overflow/Underflow event logic.

Each of these three sources can cause one of the following four effects on any of the two outputs:

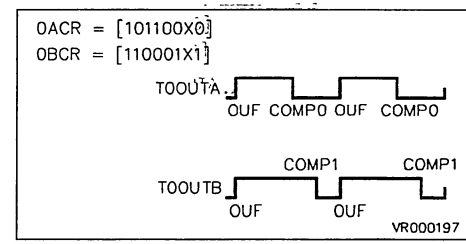
- Nop
- Set
- Reset
- Toggle.

Furthermore an On Chip Event signal can be driven by two of the three sources: the Over/Underflow event and Compare 0 event by programming the CEV bit of the OACR register and the OEV bit of OBCR register respectively. This signal can be used for another on-chip peripheral or as strobe for an I/O port (see Handshake chapter).

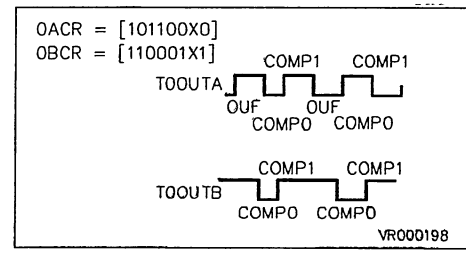
Output Waveforms

Depending on the different programmed values of OACR and OBCR the following example waveforms can be generated on TxOUTA and TxOUTB pins.

Configuration where TxOUTA is driven by Over/Underflow (OUF) and Compare 0 event (CM0), while TxOUTB is driven by the Over/Underflow and Compare 1 event (CM1).
OACR is programmed with TxOUTA preset to "0", OUF sets TxOUTA, CM0 resets TxOUTA and CM1 does not affect the output.
OBCR is programmed with TxOUTB preset to "0", OUF sets TxOUTB, CM1 resets TxOUTB while CM0 does not affect the output.

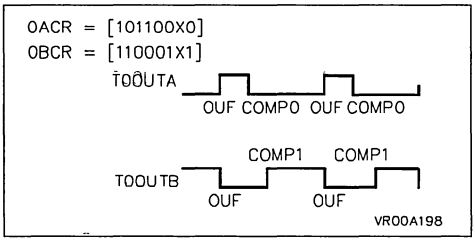


Configuration where TxOUTA is driven by Over/Underflow, Compare 0 and Compare 1, while TxOUTB is driven by both Compare 0 and Compare 1.
OACR is programmed with TxOUTA preset to "0". OUF toggles the Output 0 as do CM0 and CM1.
OBCR is programmed with TxOUTB preset to "1". OUF does not affect the output while CM0 resets TxOUTB and CM1 sets it.

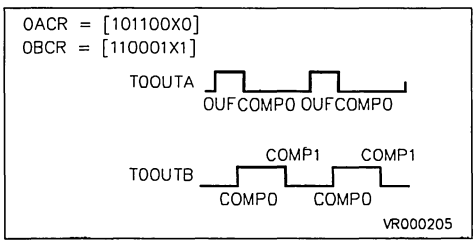


OUTPUT PIN ASSIGNMENT (Continued)

Configuration where TxOUTA is driven by Over/Underflow and Compare 0, while TxOUTB is driven by Over/Underflow and Compare 1.
OACR is programmed with TxOUTA preset to "0". OUF sets TxOUTA while CM0 resets it and CM1 has no affect.
OBCR is programmed with TxOUTB preset to "1". OUF toggles TxOUTB, CM1 sets it and CM0 has no affect.



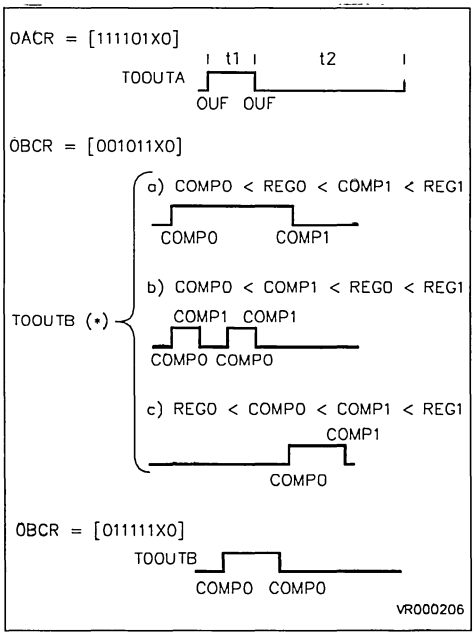
Configuration where TxOUTA is driven by Over/Underflow and Compare 0, while TxOUTB is driven by Compare 0 and 1.
OACR is programmed with TxOUTA preset to "1". OUF sets TxOUTA, CM0 resets it and CM1 has no affect.
OBCR is programmed with TxOUTB preset to "0". OUF has no affect, CM0 sets TxOUTB and CM1 toggles it.



Output Waveform Samples In Biload Mode

TxOUTA is programmed to monitor the two time intervals (t1 and t2) of the Biload Mode while TxOUTB is independent from the Over/Underflow and is driven by the different values of Compare 0 and Compare 1.
OACR is programmed with TxOUTA preset to "0". OUF toggles the output and CM0 and CM1 do not affect TxOUTA.
OBCR is programmed with TxOUTB preset to "0". OUF has no effect, while CM1 resets TxOUTB and CM0 sets it.

Depending on the CM1/CM0 values, three different example waveforms have been drawn starting from the above mentioned configuration of OBCR. In the last case, with a different programmed value of OBCR, only Compare 0 drives TxOUTB, toggling the output.



Note (*) Depending on the CMP1R/CMP0R values

13.5 INTERRUPT AND DMA

13.5.1 Timer Interrupt

The timer has 5 different Interrupt sources, grouped into 3 independent groups, assigned to the following Interrupt vectors:

Table 13-3. Timer Interrupt Structure

Interrupt Source	Vector Address
COMP 0 COMP 1	xxxx x110
CAPT 0 CAPT 1	xxxx x100
Overflow/Underflow	xxxx x000

The three least significant bits of the vector pointer address represent the relative priority assigned to each group, (000 value is the highest priority level) and are fixed by hardware depending on the source which generates the interrupt request. The 5 most significant bits are programmed by the user in the Interrupt Vector Register (IVR) of each Timer.

Each source can be masked by a dedicated bit in the Interrupt/DMA Mask Register (IDMR) of each timer, as well as a global mask enable bit (IDMR.7), masking all interrupts.

If an interrupt request (CM0 or CP0) happens before the corresponding pending bit is reset, an overrun condition occurs. This condition is flagged in two dedicated overrun bits, concerning the Comp0 and Capt0 sources, and placed in the Timer Flag Register (FLAGR).

13.5.2 Timer DMA

Two Independent DMA channels, associated to Compare 0 and Capture 0 sources, respectively allow DMA transfers from Register File/Memory to Comp0 Register and vice versa from Capt0 Register to Register File/Memory (also transfers in/from Memory from/into an I/O port are available). Their priority is hardware set as follows:

- Compare 0 Destination Lower Priority
- Capture 0 Source Higher Priority

The two DMA request sources are independently maskable by two DMA Mask bits, mapped in the Timer Interrupt/DMA Mask register (IDMR).

The two End of Block procedures, associated to each Interrupt mask and DMA mask combination, follow the standard architecture as shown in the Interrupt and DMA chapters.

13.5.3 DMA Pointers

The 6 programmable most significant bits of the Timer Address and Counter Pointer registers (DAPR-DCPR) are common to both channels (Comp0 and Capt0 sources). As a consequence, the Comp0 and Capt0 Address pointers are mapped by pair in the Register File, as well as the Comp0 and Capt0 DMA Counter pair.

The different address specification, in order to point either Capt0 or Comp0 pointers, is provided by the Timer according to the channel under service (replacing the address bit 1 with "0" for CAPT0 or with "1" for COMP0), when D0 bit on DCPR register is equal to zero (Word address in Register File). In this condition (register with program/data memory transfer), the pointers will be split in two groups of adjacent Address pointer and Counter pairs respectively.

In the case of register to register transfers (selected by programming the value "1" into bit 0 of the DCPR register), only one pair of pointers are required and the pointers are mapped into one group of adjacent positions.

DAPR (the DMA/Address Pointer Register) in this case in not used, but must be considered reserved.

INTERRUPT AND DMA (Continued)

Figure 13-4. Map Pointer for Register to Prog/Data Memory Transfer

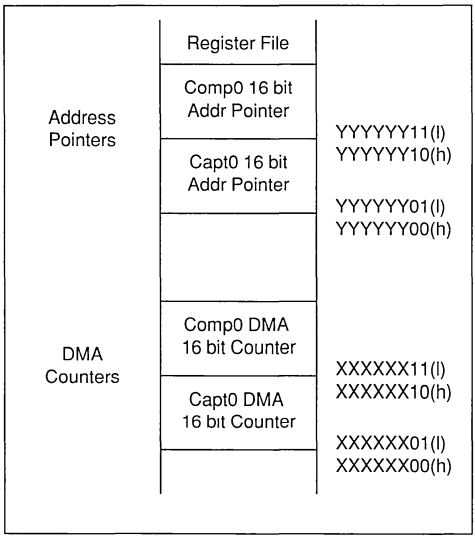
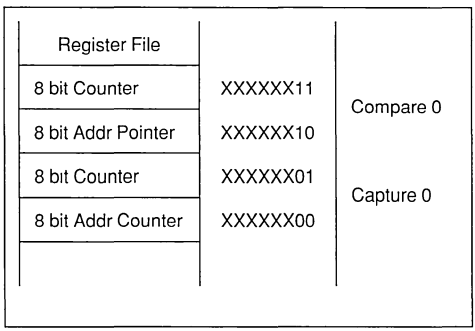


Figure 13-5. Map Pointer for Register to Register Transfer



13.5.4 Priority During The DMA Transactions

Each Timer DMA transaction is a 16-bit operation, therefore two different bytes must be transferred subsequently. This is accomplished by two DMA transfers. In order to speed up each word transfer, the second byte transfer is executed by forcing automatically the peripheral priority to the highest level (000) regardless to the previous set level. It will be then restored to the original value after executing this transfer. Furthermore, once one request is being served, its hardware priority is kept at the highest level regardless to the other Timer internal sources, i.e. once a Comp0 request is being served, it keeps a higher priority on the Capt0 channel, even if a Capt0 request occurs between the two byte transfers.

13.5.5 The DMA Swap Mode

After a complete data table transfer, the transaction counter is reset and an End Of Block condition occurs, the block transfer is completed.

The End Of Block Interrupt routine has at this point to reload both address and counter pointers of the channel referred by the End Of Block interrupt source if the application requires a continuous high speed data flow. This procedure causes speed limitations because of the time consumed by the reload routine.

The SWAP feature overcomes this drawback, allowing high speed continuous transfers. Bit 2 of the Timer Address and Counter Pointer registers (DAPR-DCPR), toggles after any End Of Block condition, alternately providing odd and even address (D2-D7) for the pair of pointers, thus pointing to an updated pair, after a block has been completely transferred. This allows the User to be updating or reading the first block, and to update the pointer values while the second is being transferred. These two toggle bits are software writable and readable, mapped in DCPR bit 2 for the CM0 channel, and in DAPR bit 2 for the CP0 channel (though a DMA event on a channel, in Swap mode, modifies a field in DAPR and DCPR common to both channels, the DAPR/DCPR content used in the transfer is always the bit related to the correct channel).

The SWAP mode can be enabled by a control bit placed in the Interrupt Control Register.

***WARNING**: this mode is always set for both channel (CM0 and CP0).*

INTERRUPT AND DMA (Continued)

13.5.6 The DMA End Of Block Interrupt Routine

This Interrupt request is generated after each block transfer (EOB) and its priority is the same as assigned in the usual Interrupt request, for the two channels. As a consequence, they will be served only when no DMA request occurs, and will be submitted to a possible OUF Interrupt request, which has higher priority.

Here is a typical EOB procedure (with swap mode enabled):

- Toggle bit test and Jump.
- Pointers (odd or even depending on toggle bit status) reload.
- Reset EOB bit: this bit must be reset only after the old couple of pointers has been restored, so that, if a new EOB condition occurs, the next pointers are ready to be swapped.
- Verify the software protection condition.
- Read the corresponding Overrun bit: this makes the user sure that NO DMA request has been lost in the meantime.
- Return.

WARNING: *The EOB bits are read/write bits only for testing reasons. Writing a logical "1" by software (when SWEN bit is set) will cause a spurious interrupt request. During normal operation, these bits must only be reset by software.*

13.5.7 DMA Software Protection

A second EOB condition may occur before the first EOB routine is completed, this would cause a not yet updated pointer couple to be addressed, with consequent overwriting of memory. To prevent these errors, a protection mechanism is provided, such that the attempted setting of the EOB bit before it has been reset by software will cause the DMA mask on that channel to be reset (DMA disabled), locking any further DMA operation. As shown above, this mask bit should always be checked in each EOB routine, to ensure all DMA transfers are properly served.

13.6 TIMER DMA EXTERNAL MODES ON I/O PORTS

Each Timer DMA channel can also be employed in external transfers to/from memory from/to an I/O port. In this case only Byte transfers are executed for any request. Two control bits (DCTS and DCTD) in the Interrupt/DMA Control Register (IDCR) set each channel in INT/EXT (Internal = Register to Memory/External = Memory to/from I/O ports) mode.

The relevant I/O port must then be programmed in DMA mode and the right direction of the port chosen by the HDCxR register of that port (see Handshake chapter).

The two modes, however, are not the same for both channels as explained in the following section.

13.6.1 CM0 Channel External Mode

This mode is enabled when DCTD (DMA Compare Transaction Destination) bit is equal to "1" in the IDCR register.

This mode allows only Output transfers, from Register File/memory to the I/O port, under a request caused by a CM0 event or a software request (writing "1" in the CM0 flag). An application for this is a data flow under DMA to be output at fixed times.

The synchronization with the I/O port is accomplished by an internal signal, active when the data to be transfered is present on the internal Data Bus. If programmed, the on-chip event pulse can also be generated and used to strobe the output data on the selected handshake port.

In either case the DMA Output mode must be selected in the HDCTL Register of the port (see Handshake chapter).

13.6.2 CP0 Channel In External Mode

This mode is enabled when DCTS (DMA Capture Transaction Source) bit is equal to "1" in the IDCR register.

This mode allows bi-directional transfers controlled (when the I/O port is programmed in DMA Input/Output mode in the HDCTL register) by the value of the DD bit of the HDCTL register (the DD bit selects the DMA input or DMA Output mode).

The DMA request can be either an External CPT0 request (Timer External input A) or a software request (by writing "1" in the CP0 Flag).

This, along with a further internal synchronization signal, generated by the Timer Unit, allows handshake operations managed by the I/O port while the direction of the data to read or write on the I/O port is fixed by the value of the DD bit in the HDCTL register.

MODES ON I/O PORTS (Continued)

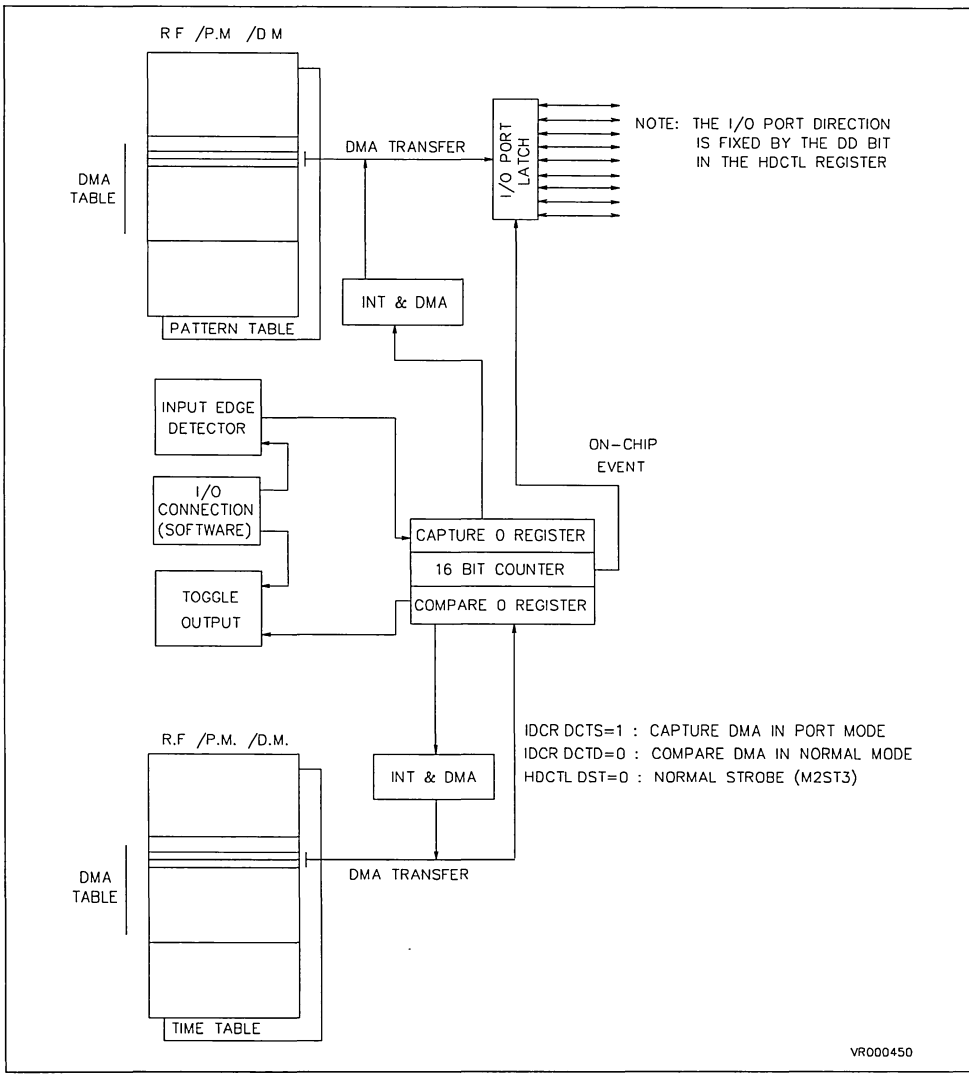
13.6.3 DMA Channel Synchronization

A CP0 DMA request can be generated also by a CM0 event, simply by setting the Timer External Input A on rising and falling edges sensitive, connecting it by hardware or software (though the IOCR register) to the Timer OUT 0, and programming the CM0 action as output toggle.

This will cause a CP0 request to be generated after each CM0 condition, thus synchronizing the 2 DMA channels (see the following application example).

The DCTS bit must be set and DCTD bit must be reset in the IDCR register. Figure 13-6 shows an example of two channel synchronisation. A new byte will be sent out through the I/O port at an interval specified by the COMP0 value mapped in the look-up table.

Figure 13-6. Timer DMA Channels Synchronization



13.7 REGISTER DESCRIPTION

Twenty control and data registers are associated to each Multifunction timer, and are located in the Group F I/O pages of the ST9 Register File.

The registers of the Multifunction Timers are located in the I/O pages as follows:

Note that unused registers must be regarded as reserved registers.

In the following pages there is a detailed description of every register with the meaning and the function of every bit. The register is referred without the absolute address which is depending on the number of the timer used (of course the configuration and the functions of the internal bits of i.e. TCR - TIM0 are the same of TCR - TIM1 and so on.

Table 13-4. Multifunction Timer Register Map

	Page 10 (0Ah)		Page 9 (09h)		Page 8 (08h)
R255	IMDR - TIM0	FF			IMDR - TIM1
R254	FLAGR - TIM0	FE			FLAGR - TIM1
R253	OBCR - TIM0	FD			OBCR - TIM1
R252	OACR - TIM0	FC			OACR - TIM1
R251	PRSR - TIM0	FB			PRSR - TIM1
R250	ICR - TIM0	FA			ICR - TIM1
R249	TMR - TIM0	F9			TMR - TIM1
R248	TCR - TIM0	F8	IOCR		TCR - TIM1
R247	CMP1LR - TIM0	F7	IDCR - TIM1		CMP1LR - TIM1
R246	CMP1HR - TIM0	F6	IVR - TIM1		CMP1HR - TIM1
R245	CMP0LR - TIM0	F5	DAPR - TIM1		CMP0LR - TIM1
R244	CMP0HR - TIM0	F4	DCPR - TIM1		CMP0HR - TIM1
R243	REG1LR - TIM0	F3	IDCR - TIM0		REG1LR - TIM1
R242	REG1HR - TIM0	F2	IVR - TIM0		REG1HR - TIM1
R241	REG0LR - TIM0	F1	DAPR - TIM0		REG0LR - TIM1
R240	REG0HR - TIM0	F0	DCPR - TIM0		REG0HR - TIM1

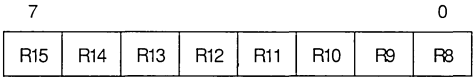
REGISTER DESCRIPTION (Continued)

13.7.1 Register 0 (REG0R) Registers

This pair of registers (REG0LR and REG0HR) is used to capture values from the U/D counter or to load preset values into the U/D counter.

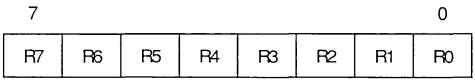
REG0HR R240 (F0h) Read/Write
Capture Load Register 0 (High)

Reset value: undefined



REG0LR R241 (F1h) Read/Write
Capture Load Register 0 (Low)

Reset value: undefined

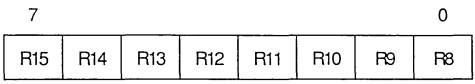


13.7.2 Register 1 (REG1R) Registers

This pair of registers (REG1LR and REG1HR) is used (as REG0R) to capture values from the U/D counter or to load preset values into the U/D counter.

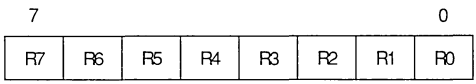
REG1HR R242 (F2h) Read/Write
Capture Load Register 1 (High)

Reset value: undefined



REG1LR R243 (F3h) Read/Write
Capture Load Register 1 (Low)

Reset value: undefined

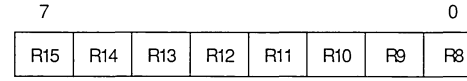


13.7.3 Compare 0 (CMP0R) Registers

This pair of Registers (CMP0L and CMP0H) is used to store 16-bit values to be compared to the U/D counter content.

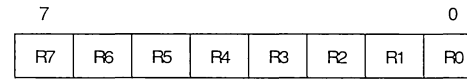
CMP0HR R244 (F4h) Read/Write
Compare 0 Register (High)

Reset value: undefined



CMP0LR R245 (F5h) Read/Write
Compare 0 Register (Low)

Reset value: undefined

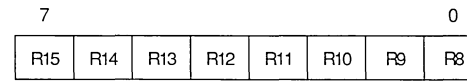


13.7.4 Compare 1 (CMP1R) Registers

This pair of Registers (CMP1L and CMP1H) is used (as CMP0R) to store 16-bit values to be compared to the U/D counter content.

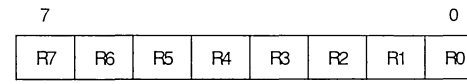
CMP1HR R246 (F6h) Read/Write
Compare 1 Register (High)

Reset value: undefined



CMP1LR R247 (F7h) Read/Write
Compare 1 Register (Low)

Reset value: undefined



REGISTER DESCRIPTION (Continued)

13.7.5 Timer Control Register (TCR)

This register is used to control the status of the timer.

TCR R248 (F8h) Read/Write
Timer Control Register

Reset value: 0000 0xxxb

7							0
CEN	CCP0	CCMP0	CCL	UDC	UDCS	OF0	CS

b7 = **CEN:** *Counter Enable.* This bit is ANDed with the Global Counter Enable bit (GCEN bit on R230 - Central Interrupt Control Register; the GCEN bit is set after the Reset cycle). Setting the CEN bit starts the counter and prescaler (without reload). When this bit is reset, the counter and prescaler stop.

b6 = **CCP0:** *Clear on Capture.* When this bit is set, a clear of the counter and a reload of the prescaler are performed on REG0R or REG1R capture. No effect when this bit is reset.

b5 = **CCMP0:** *Clear on Compare.* When this bit is set, a clear of the counter and a reload of the prescaler are performed on CMP0R compare. No effect when this bit is reset.

b4 = **CCL:** *Counter clear.* When this bit is set, the counter is cleared without generation of interrupt request. No effect when this bit is reset.

b3 = **UDC:** *Software Up/Down.* When the direction of the counter is not fixed by TxINA and/or TxINB (see par. 10.3) it can be software controlled by the UDC bit. Setting the UDC bit selects the Up mode counting. Resetting this bit the Down counting is performed.

b2 = **UDCS:** *Up/Down Count status.* This bit is read only and monitors the direction of the counter. Reading "1" means that the counter is using the Up mode counting. Reading "0" means that the Down mode counting is in use.

b1 = **OF0:** *OVF/UNF state.* This bit is read only and is set if an Overflow or an Underflow occurs during a Capture on Register 0.

b0 = **CS:** *Counter Status.* This bit is read only and monitors the status of the counter. Reading "1" means that the counter is running. Reading "0" indicates that the counter is halted.

13.7.6 Timer Mode Register (TMR)

This register is used to select the operating mode of the timer.

TMR R249 (F9h) Read/Write
Timer Mode Register

Reset value: 0000 0000b (00h)

7							0
OE1	OE0	BM	RM1	RM0	ECK	REN	CO

b7 = **OE1:** *Output 1 Enable.* Setting this bit enables the Output 1 (TxOUTB) of the relevant timer. When this bit is reset, the TxOUTB is disabled and forced to the logic state "1". The relevant I/O bit must also be set to Alternate Function.

b6 = **OE0:** *Output 0 Enable.* Setting this bit enables the Output 0 (TxOUTA) of the relevant timer. When this bit is reset, the TxOUTA is disabled and forced to the logic state "1". The relevant I/O bit must also be set to Alternate Function.

b5 = **BM:** *Bivalue Mode.* This bit enables the Bivalue mode when is set. When the bit is reset, the Bivalue mode is disabled. After that, depending on the value of RM0 bit (TMR - bit 3), the Biload or Bicapture mode is selected.

b4 = **RM1:** *REG1R mode.* When this bit is set, the REG1R can be used to capture the value of the counter. When the bit is reset, the REG1R monitors the value of the counter. The selection performed by this bit has no effect when the Bivalue Mode is enabled.

b3 = **RM0:** *REG0R mode.* When this bit is set, the REG0R can be used to capture the value of the counter (also the Bicapture mode can be selected if the BM bit is equal to 1). When the bit is reset, the REG0R can be used to load the new value of the counter (also the Biload mode can be selected if the BM bit is equal to "1").

b2 = **ECK:** *Timer clocking mode.* This bit selects the clock source which drives the prescaler. When the ECK bit is reset, either the Internal or External clock is used depending on IN0 - IN3 configuration in ICR. When ECK bit is set, different functions are performed depending on the number of the relevant timer. For odd timers (Timer 1, Timer 3 and so on) setting the ECK bit enables the Parallel mode where the prescaler of the odd timer is driven by the prescaler output of the even timer.

REGISTER DESCRIPTION (Continued)

b1 = **REN:** *Retrigger mode.* When this bit is reset, the Retriggerable mode is enabled. When the bit is set, this operating mode is disabled.

b0 = **CO:** *Continous/One shot mode.* When this bit is reset, the Continuous mode is selected (with autoreload on condition). The bit must be set to select the one shot mode. The following table summarizes the different operating modes depending on the values of RM0, RM1 and BM bits.

Table 13-5. Timer Operating Modes

TMR Bits			Timer Operating Modes
BM	RM1	RM0	
1	X	0	Biload mode
1	X	1	Bicapture mode
0	0	0	Load from REG0R and Monitor on REG1R
0	1	0	Load from REG0R and Capture on REG1R
0	0	1	Capture on REG0R and Monitor on REG1R
0	1	1	Capture on REG0R and REG1R

13.7.7 External Input Control Register(ICR)

By this register it is possible to program the function and the operation to be performed on TxINA and TxINB inputs.

ICR R250 (FAh) Read/Write
External Input Control Register

Reset value: 0000 xxxxb (0Xh)

7							0
IN3	IN2	IN1	IN0	A0	A1	B0	B1

b7-b4 = **IN3,IN2,IN1,IN0:** *Input pin assignment.* The different functions of TxINA and TxINB inputs of every timer can be selected by IN0 - IN3 bits as explained below.

b3-b2 = **A0, A1:** *TxINA event programming.* The following TxINA configurations can be selected according to the values of A0 and A1 bits:

b1-b0 = **B0, B1:** *TxINB event programming.* The following TxINB configurations can be selected according to the values of B0 and B1 bits:

A0/B0	A1/B1	TxINA/TxINB Configuration
0	0	No operation
0	1	Falling edge sensitive
1	0	Rising edge sensitive
1	1	Rising and falling edges

I C Reg. IN3-IN0 bits	TxINA Input Function	TxINB Input Function
0000	not used	not used
0001	not used	Trigger
0010	Gate	not used
0011	Gate	Trigger
0100	not used	Ext. Clock
0101	Trigger	not used
0110	Gate	Ext. Clock
0111	Trigger	Trigger
1000	Clock Up	Clock Down
1001	Up/Down	Ext. Clock
1010	Trigger Up	Trigger Down
1011	Up/Down	not used
1100	Autodiscr.	Autodiscr.
1101	Trigger	Ext. Clock
1110	Ext. Clock	Trigger
1111	Trigger	Gate

REGISTER DESCRIPTION (Continued)

13.7.8 Prescaler Register (PRSR)

This register holds the preset value for the 8-bit prescaler. The PRSR content may be modified at any time, but it will be loaded into the prescaler at the following prescaler underflow, or as a consequence of a counter reload (either by software or upon external request). On an external RESET condition, the prescaler is automatically loaded with the 00h value, so that the prescaler divides by 1 and the maximum counter clock is generated (OSCIN frequency divided by 6 when MODER.5 = DIV2 bit is set).

PRSR R251 (FBh) Read/Write
Prescaler Register

Reset value: 0000 0000b (00h)

7							0
P7	P6	P5	P4	P3	P2	P1	P0

The binary value stored (by programmer) in the PRSR register is equal to [divider value - 1]. For example, loading PRSR with 24 makes the prescaler divide by 25.

13.7.9 Output A Control Register (OACR)

This register selects the sources that can perform actions on a TxOUTA pin. TxOUTA can be driven from any of three possible sources:

- OVF/UNF being an Overflow or Underflow event on the U/D counter,
- COMP0 being a successful compare event on CMP0R register, and
- COMP1 being a successful compare event on CMP1R.

By programming bits B0 and B1 of the relevant source can cause one of the following four effects on TxOUTA (which can be preset previously):

B0	B1	Event
0	0	Set
0	1	Toggle
1	0	Reset
1	1	Nop

Note: In any case of contemporary events the action will be taken which results from 'ANDing' the B1-B0 fields Through this register the action of COMP0 on the on-chip event can be also selected

OACR R252 (FCh) Read/Write
Output A Control Register

Reset value: xxxx xx0xb

7							0
B0	B1	B0	B1	B0	B1	CEV	OP
< COMP0 >		< COMP1 >		< OVF/UNF >			

b7-b6 = **B0, B1:** *Control bits of COMP0.* Control bits for event driven by COMP0.

b5-b4 = **B0, B1:** *Control bits of COMP1.* Control bits for event driven by COMP1.

b3-b2 = **B0, B1:** *Control bits of OVF/UNF.* Control bits for event driven by OVF/UNF.

b1 = **CEV:** *On-Chip Event on CMP0R.* When this bit is set, a successful compare on CMP0R activates the on-chip event signal (a single pulse is generated). No action when this bit is reset.

b0 = **OP:** *Control bit of TxOUTA preset.* The value of this bit is the preset value of TxOUTA output pin. Reading this bit returns the current state of the TxOUTA output pin (i.e. useful when this output is selected in toggle mode).

REGISTER DESCRIPTION (Continued)

13.7.10 Output B Control Register (OBCR)

This register selects the sources that can perform actions on TxOUTB output pin. TxOUTB can be driven from any of three possible sources:

- OVF/UNF being an Overflow or Underflow event on the U/D counter,
- COMP0 being a successful compare event on CMP0R register, and
- COMP1 being a successful compare event on CMP1R.

By programming bits B0 and B1 of the relevant source can cause one of the following four effects on TxOUTB (which can be previously preset):

B0	B1	Event
0	0	Set
0	1	Toggle
1	0	Reset
1	1	Nop

Note: In any case of contemporary events the action will be taken which results from 'ANDing' the B1-B0 fields Through this register the action of Overflow/Underflow on the on-chip event can be also selected

OBCR R253 (FDh) Read/Write
Output B Control Register

Reset value: xxxx xx0xb

7							0
B0	B1	B0	B1	B0	B1	OEV	OP
< COMP0 >		< COMP1 >		< OVF/UNF >			

b7-b6 = **B0, B1:** *control bits of COMP0.* Control bits for event driven by COMP0.

b5-b4 = **B0, B1:** *control bits of COMP1.* Control bits for event driven by COMP1.

b3-b2 = **B0, B1:** *control bits of OVF/UNF.* Control bits for event driven by OVF/UNF.

b1 = **OEV:** *On-Chip Event on OVF/UNF.* When this bit is set, a successful Overflow/Underflow activates the on-chip event signal (a single pulse is generated). No action when this bit is reset.

b0 = **OP:** *control bit of TxOUTB preset.* The value of this bit is the preset value of TxOUTB output pin. Reading this bit, it returns the current state of the TxOUTB output pin (i.e. useful when this output is selected in toggle mode).

13.7.11 Flag Register (FLAGR)

This register contains the flags of the successful captures or comparisons together with the Overflow/Underflow and overrunning indications. Also the mode of the Interrupt on capture can be selected. By writing into the capture flags it is possible to generate software captures. It is necessary to clear the capture flag before subsequent sofware captures can be generated. By reading this register, user can know which source has generated an interrupt (several sources may share the same interrupt vector).

FLAGR R254 (FEh) Read/Write
Flags Register

Reset value: 0000 0000b (00h)

7							0
CP0	CP1	CM0	CM1	OUF	OCP0	OCM0	A0

b7 = **CP0:** *Flag on Capture 0.* This bit is set after a capture on REG0R register. Writing "1" acts as a software load/capture from/on REG0R.

b6 = **CP1:** *Flag on Capture 1.* This bit is set after a capture on REG1R register. Writing "1" acts as a software capture on REG1R, except when in Bi-capture mode.

b5 = **CM0:** *Flag on Compare 0.* This bit is set after a successful compare on CMP0R register.

b4 = **CM1:** *Flag on Compare 1.* This bit is set after a successful compare on CMP1R register.

b3 = **OUF:** *Flag on Overflow/Underflow.* This bit is set after a counter Over/Underflow condition.

b2 = **OCP0:** *Flag of overrun on Capture 0.* This bit is set when more than one INT/DMA request occurs before having reset the event flag CP0 or whenever a capture is software simulated.

b1 = **OCM0:** *Flag of overrun on Compare 0.* This bit is set when more than one INT/DMA request occurs before having reset the event flag CM0.

b0 = **A0:** *Capture Interrupt Function.* When this bit is set the Interrupt is generated by an AND function of REG0R/REG1R captures while when the A0 bit is reset, the Interrupt is generated by an OR function of REG0R/REG1R captures.

REGISTER DESCRIPTION (Continued)

13.7.12 Interrupt/DMA Mask Register (IDMR)

This register contains the Global Timer Interrupt enable bit and the INT/DMA enable bits of the following events:

- Capture on REG0R (CP0 field),
- Capture on REG1R (CP1I bit - only Interrupt mask),
- Compare on CMP0R (CM0 field),
- Compare on CMP1R (CM1I bit- only Interrupt mask), and
- Overflow/Underflow (OUI bit - only Interrupt mask).

IDMR R255 (FFh) Read/Write
Interrupt/DMA Mask Register

Reset value: 0000 0000b (00h)

7							0
GTIEN	CP0D	CP0I	CP1I	CM0D	CM0I	CM1I	OUI

< CP0 > <CP1> < CM0 > <CM1>

b7 = **GTIEN:** *Global Timer Interrupt Enable.* When this bit is set, all the Interrupts (of the enabled sources) of the timer are enabled. When the bit is reset, all the Interrupts of timer are disabled.

b6 = **CP0D:** *Capture 0 DMA Mask.* Capture on REG0R DMA is enabled when CP0D = "1."

b5 = **CP0I:** *Capture 0 Interrupt Mask.* Capture on REG0R interrupt is enabled when CP0I = "1".

b4 = **CP1I:** *Capture 1 Interrupt Mask.* Capture on REG1R interrupt is enabled when CP1I = "1".

b3 = **CM0D:** *Compare 0 DMA Mask.* Compare on CMP0R DMA is enabled when CM0D = "1".

b3 = **CM0I:** *Compare 0 Interrupt Mask.* Compare on CMP0R interrupt is enabled when CM0I = "1".

b1 = **CM1I:** *Compare 1 Interrupt Mask.* Compare on CMP1R interrupt is enabled when CM1I = "1".

b0 = **OUI:** *Overflow/Underflow Interrupt Mask.* Overflow/Underflow condition interrupt is enabled when OUI = "1".

Note. The following Registers show in square brackets ([ ]) the Register address in the case of an odd numbered (1, 3, 5...) Multifunction Timer being available on-chip. If only one Timer is present these addresses may be ignored.

13.7.13 DMA Counter Pointer Register (DCPR)

This register is not used only as DMA Counter pointer but also to define the DMA area and the DMA source.

DCPR R240 (F0h)[R244 (F4h)] Read/Write
DMA Counter Pointer Register

Reset value: undefined

7							0
D7	D6	D5	D4	D3	D2	DMA SRCE	REG MEM

b7-b2 = **D7-D2:** *MSB of DMA counter register address.* Those bits contain the most significant bits of the DMA counter register address and are user programmable. Though user programmable, the D2 bit may be hardware toggled if the Swap mode is set for the Timer DMA section related to Compare 0 channel.

b1 = **DMA-SRCE:** *DMA source selection (hardware programmed).* This bit is hardware fixed by the Timer DMA logic and is set if the DMA destination is a Compare on CMP0R register and reset if the DMA source is a Capture on REG0R register.

b0 = **REG/MEM:** *DMA area selection.* When this bit is set, it selects the Source/Destination of the DMA area from/into Register File while when it is reset, the Source/Destination of the DMA area is from/to the External Program or Data Memory (according with the value of D0 bit in DAPR).

REGISTER DESCRIPTION (Continued)

13.7.14 DMA Address Pointer Register (DAPR)

This register is not used only as DMA Address pointer but also to define the DMA area and the DMA source.

DAPR R241 (F1h)[R245 (F5h)] Read/Write
DMA Address Pointer Register

Reset value: undefined

7							0
D7	D6	D5	D4	D3	D2	DMA SRCE	PRG/ DAT

b7-b2 = **D7-D2:** *MSB of DMA Address register location.* Those bits contain the most significant bits of the DMA Address register location and are user programmable. Through user programmable, the bit D2 may be hardware toggled if the Swap mode is set for the Timer DMA section related to Capture 0 channel.

b1 = **DMA-SRCE:** *DMA source selection (hardware programmed).* This bit is hardware fixed by the Timer DMA logic and is set if the DMA destination is a Compare on CMP0R register and reset if the DMA source is a Capture on REG0R register.

b0 = **PRG/DAT:** *DMA memory selection.* When this bit is set it selects the Source/Destination of the DMA area from/into Data Memory while when it is reset the Source/Destination of the DMA area is from/into the External Program Memory (according with the value of D0 bit in DCPR).

REG.MEM	PRG/DAT	DMA Source/Destination
0	0	Program memory
0	1	Data memory
1	0	Register file
1	1	Register file

13.7.15 Interrupt Vector Register (IVR)

This register is used as a vector pointing to the 16-bit interrupt vectors in the program memory which contain the starting addresses of the three interrupt subroutines managed by every timer.

Only one Interrupt Vector Register is available for every timer and is able to manage the three interrupt groups because the 3 least significant bits are fixed by hardware depending on the group which generated the interrupt request.

In order to understand which request generated the interrupt inside the same group, the FLAGR register can be used to check the relevant flag of the interrupt source.

IVR R242 (F2h) [R246 (F6h)] Read/Write
Interrupt Vector Register

Reset value: xxxx xxx0b

7							0
V4	V3	V2	V1	V0	W1	W0	D0

b7-b3 = **V4 - V0:** *MSB of the Vector address.* These bits are user programmable and contain the five most significant bits of the Timer interrupt vector addresses in the program memory. In any case, an 8-bit address can be used to indicate the Timer interrupt vector locations because they are within the first 256 locations of the program memory (see Interrupt and DMA chapters).

b2-b1 = **W1 - W0:** *Vector Address bits.* These bits are equivalent to bit 1 and bit 2 of the Timer interrupt vector addresses in the program memory. They are fixed by hardware depending on the group of sources which generated the interrupt request as follows:

b0 = **D0.** This bit is fixed by hardware. It always returns the value "0" if read.

W1	W0	Interrupt Source
0	0	Overflow/Underflow even interrupts
0	1	Not available
1	0	Capture event interrupts
1	1	Compare event interrupts

REGISTER DESCRIPTION (Continued)

13.7.16 Interrupt/DMA Control Register (IDCR)

This register is used to control the Interrupt and DMA priority level, the DMA transfer source and destination and the Swap mode. This register contains also the two End Of Block bits.

IDCR R243 (F3h) [R247 (F7h)] Read/Write
Interrupt/DMA Control Register

Reset value: 1100 0111b (C7h)

7							0
CPE	CME	DCTS	DCTD	SWEN	PL2	PL1	PL0

b7 = **CPE:** *Capture 0 EOB.* This bit is set by hardware when the End Of Block condition is reached during a Capture 0 DMA operation with the Swap mode enabled. When the Swap mode is disabled (SWEN bit = "0") the CPE bit is forced by hardware to "1".

b6 = **CME:** *Compare 0 EOB.* This bit is set by hardware when the End Of Block condition is reached during a Compare 0 DMA operation with the Swap mode enabled. When the Swap mode is disabled (SWEN bit = "0") the CME bit is forced by hardware to "1".

b5 = **DCTS:** *DMA Capture Transfer Source.* This bit selects the source of the DMA operation related to the channel associated to the Capture 0. When the DCTS bit is reset the selected source is the REG0R register. When the DCTS bit is set the ST9 port is selected as DMA transfer source (with this DMA channel the ST9 port can also be destination depending on the value of the DD bit in the HDCTL register of the port - see I/O port chapter 9).

b4 = **DCTD:** *DMA Compare Transfer Destination.* This bit selects the destination of the DMA operation related to the channel associated to the Compare 0. When this bit is reset, the selected destination is the CMP0R register. When the bit is set, the ST9 port is selected as DMA transfer destination.

b3 = **SWEN:** *Swap function Enable.* When this bit is set, the Swap function is enabled for the two DMA channels. Resetting the SWEN bit disables the Swap mode.

b2-b0 = **PL2 to PL0:** *Interrupt/DMA priority level.* With these three bits it is possible to select the Interrupt and DMA priority level of every single timer within eight different levels (see Interrupt/DMA chapter).

13.7.17 I/O Connection Register (IOCR)

This register allows user to select (or not) an on-chip connection between input A and output A of one same timer.

IOCR R248 (F8h) Read/Write
I/O Connection Register

Reset value: 1111 1100b (FCh)

7							0
						SC1	SC0

b7-b2 = not used.

b1 = **SC1:** *Select Connection Odd.* SC1 selects if connection between TxOUTA and TxINA for ODD timers is made on-chip or externally (physically on pins)

SC1 = "0": TxOUTA and TxINA unconnected

SC1 = "1": TxOUTA and TxINA connected internally

b0 = **SC0:** *Select Connection Even.* SC0 selects if connection between TxOUTA and TxINA for EVEN timers is made on-chip or externally (physically on pins)

SC0="0": TxOUTA and TxINA unconnected

SC0="1": TxOUTA and TxINA connected internally

14 SERIAL COMMUNICATIONS INTERFACE

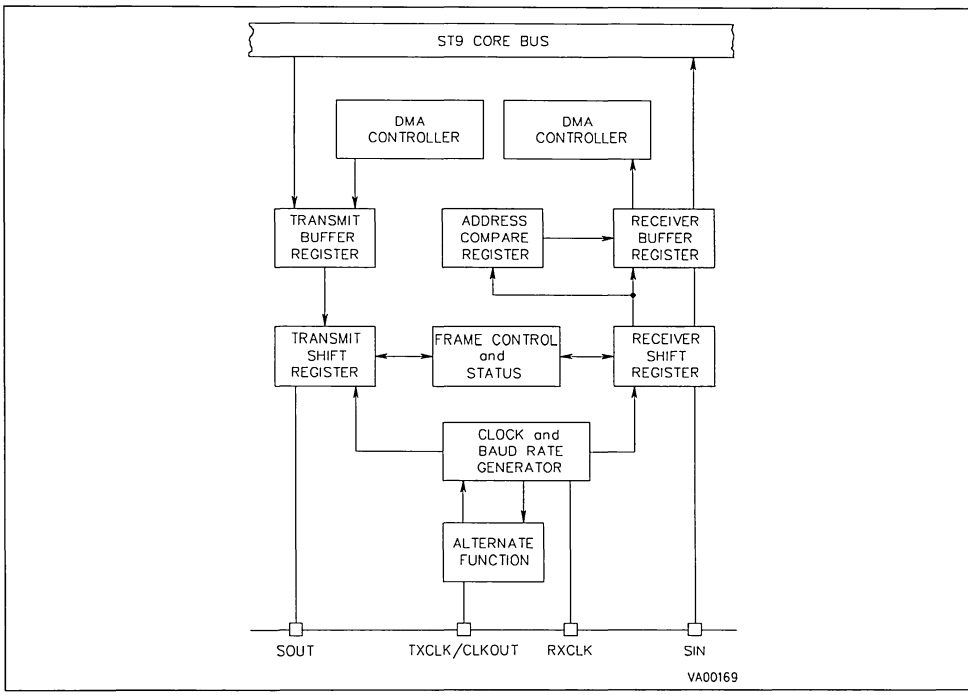
14.1 INTRODUCTION

The ST9 Serial Communications Interface (SCI) offers a means of full-duplex serial data transfer to a wide range of external equipments.

The SCI has the following features:

- Full duplex character-oriented synchronous and asynchronous operation
- Synchronous serial port expansion capability
- Transmit, receive, line status, and device address interrupt generation
- Integral Baud Rate Generator capable of dividing the input clock by any value from 2 to 2^{16}-1 (16 bit word) and generating the internal 16X clock for asynchronous operation or 1X clock for synchronous operation
- Fully programmable serial-interface characteristics:
  - 5, 6, 7, or 8 bit word length
  - Even, odd, or no parity generation and detection
  - 1, 1-1/2, 2, 2-1/2 stop bit generation
  - False start bit detection
  - Complete status reporting capabilities
  - Line break generation and detection
- Programmable address indication bit (wake-up bit) and User invisible compare logic to support network communication of multiple microcomputers. Optional character search function.
- Internal diagnostic capabilities:
  - Local loopback for communications link fault isolation
  - Auto-echo for communications link fault isolation
- Separate interrupt/DMA channels for both transmit and receive

Figure 14-1. SCI Block Diagram



14.2 FUNCTIONAL DESCRIPTION

SCI can run in three operational modes:

- Asynchronous mode
- Synchronous mode
- Serial expansion mode

Each of these three modes output data with the same serial frame format. The differences are derived from the clock rate (1X, 16X) and the sampling clock (for the serial expansion mode).

Asynchronous Mode

In this mode, data and clock can be asynchronous (the emitter and receiver can have their own clock to sample received data), each data bit is sampled 16 times per clock period. Thus the baud rate clock should be set to the ÷16 Mode and the frequency of the clock input (from an external source or the internal baud-rate generator output) set to suit this.

Synchronous Mode

In this mode, data and clock are synchronous, each data bit is sampled once per clock period.

Serial Expansion Mode

This mode is used to access to an external synchronous peripheral.

The transmitter will provide the clock waveform only during the period that data is being transmitted through CLKOUT pin (the Data Envelope). The data is latched on the rising edge of this clock.

Whenever the SCI is to receive data in the serial port expansion mode, the clock waveform must be supplied externally, synchronous with the data to the ST9. The SCI will latch the incoming data on the rising edge of the receiver I/O expansion clock. The clock input is supplied on pin RXCLK.

14.2.1 Serial Frame Format

Figure 14-3. Asynchronous mode

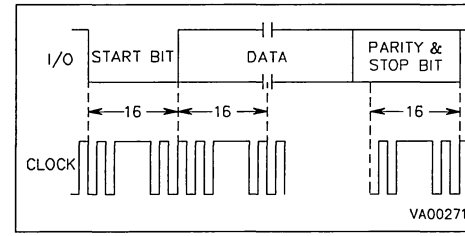


Figure 14-4. Synchronous Mode

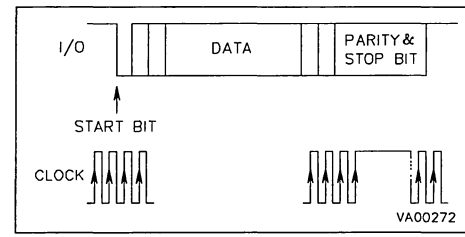


Figure 14-5. Serial Expansion Mode

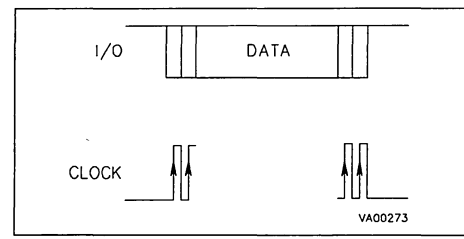
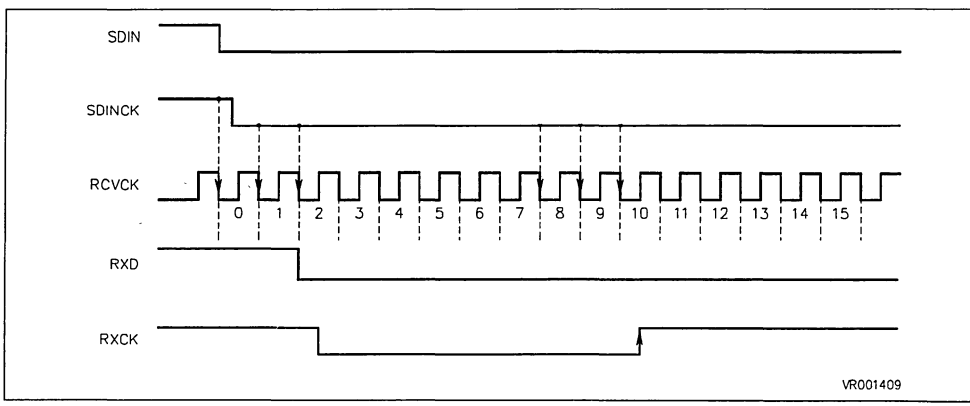


Figure 14-2. Sampling Times in Asynchronous Format



FUNCTIONAL DESCRIPTION (Continued)

Every character sent (or received) by the SCI has the following format:

This format is used by the SCI in all modes:

START: the start bit indicates the beginning of a data frame in the asynchronous mode. START bit is detected as a high to low transition

DATA: the DATA word is programmable to be from 5 to 8 bits long for both synchronous and asynchronous modes

PARITY: The Parity Bit is optional, and can be used with any length of word. It is used for error checking and resets in a resultant state (odd or even) depending on number of "1"s in DATA.

ADDRESS/9TH: The Address/9th Bit is optional and may be added to any word format. It is used in both synchronous or asynchronous mode to indicate that the data is an address (bit = "1").
The ADDRESS/9TH bit is useful when several microcontrollers are exchanging data on the same serial bus. Individual microcontrollers can stay idle on the serial bus, waiting for a transmitted address. When a microcontroller recognizes its own address, it can begin Data Reception, likewise, on the transmit side, the microcontroller can transmit another address to begin communication with a different microcontroller.

The ADDRESS/9TH bit can be used as an additional data bit or to mark control words (9th bit).

STOP: Indicates the end of a data frame for both asynchronous and synchronous modes. The stop bit is programmed to be 1, 1.5, 2, or 2.5 bits long. It returns the SCI to the quiescent marking state (i.e., a constant high-state condition) which lasts until a new start bit indicates an incoming word.

Data transfer

Data to be transmitted by the SCI is first loaded by the program into the Transmitter Buffer Register. The SCI will transfer the data into the Transmitter Shift Register when the Shift Register becomes available (empty). The Transmitter Shift Register converts the parallel data into the serial format for transmission through the SCI Alternate Function output, Serial Data Out. After the completion of the transfer, the transmitter buffer register interrupt pending bit will be updated.
If the selected word format is less than 8 bits, the unused most significant bits are "don't care".

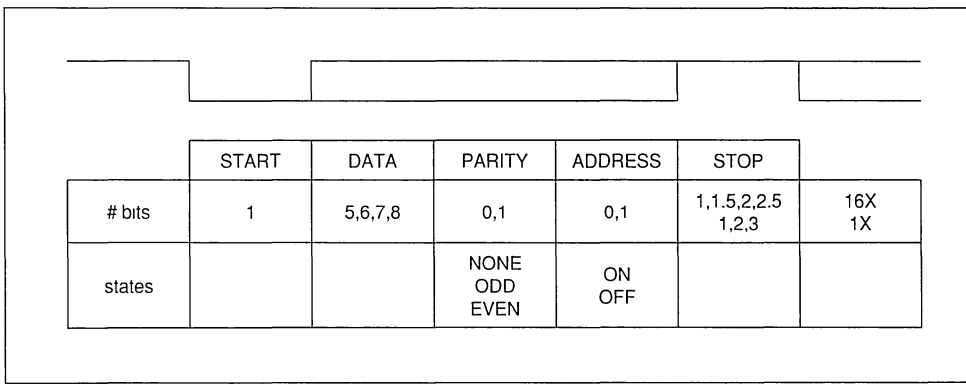
Incoming serial data from the Serial Data Input pin is converted into parallel data for reception in the Receiver Shift Register. At the end of the input, the data portion of the received word is transferred from the Receiver Shift Register into the Receiver Buffer Register. All Receiver interrupt conditions are updated at the time of transfer.
If the selected character format is less than 8 bits, the unused most significant bits will have the value "1".

The Frame Control and Status block creates and checks the character configuration (Data length and stop bits), and the source for the transmitter/receiver clock.

The integral Baud Rate Generator contains a programmable divide by "N" counter which can be used to generate the clocks for the transmitter and/or receiver. The baud rate generator can use INTCLK or the Receiver clock input RXCLK.

The Address bit/D9 is optional and may be added to any word format. It is commonly used in network or machine control applications. When enabled (AB, CHCR.4 = "1"), an address or ninth data bit can be added to a transmitted word by setting the

Figure 14-6. SCI Character Format



	START	DATA	PARITY	ADDRESS	STOP	
# bits	1	5,6,7,8	0,1	0,1	1,1.5,2,2.5 1,2,3	16X 1X
states			NONE ODD EVEN	ON OFF		

FUNCTIONAL DESCRIPTION (Continued)

Set Address bit (SA, IDPR.5). This is then appended to the next word entered into the (empty) Transmitter Buffer Register and then cleared by hardware.
On character input an Address Bit set can indicate that the data preceding the bit is an address which may be compared in hardware with the value in the Address Compare Register (ACR) to generate an Address Match interrupt when equal.

The Address bit and Address Comparison Register can also be combined to generate an Address Interrupt in 4 modes to suit different protocols, based upon the status of the Address Mode Enable bit (AMEN, IDPR.7) and the Address Mode bit (AM, CHCR.7).

Table 14-1. Address Interrupt Modes

If 9th Data Bit = 1
If Character Match
If Character Match and 9th Data Bit = 1
If Character Match on Word Immediately Following Break

The character match Address Interrupt mode may be used as a powerful character search mode, giving an interrupt on reception of a predetermined character e.g. Carriage Return or End of Block codes.

The Line Break condition is fully supported for both transmission and detection. Line Break is sent by setting the SET_BREAK bit (SB, IDPR.6). This causes the transmitter output to be held low (after all buffered data has been transmitted) for a minimum of one complete word length and until the SB bit is Reset.

Testing of the communications channel may be performed using the facilities of the SCI. Auto Echo mode (SCI SOUT disconnected, SIN pin internally connected to SOUT pin) and Loopback mode (SCI transmitter and receiver sections disconnected from SOUT and SIN pins and directly connected internally) may be used individually or together.

Figure 14-7. Auto Echo Configuration

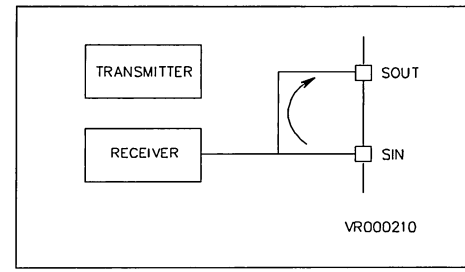


Figure 14-8. Loop Back Configuration

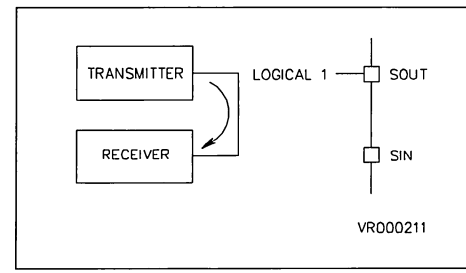
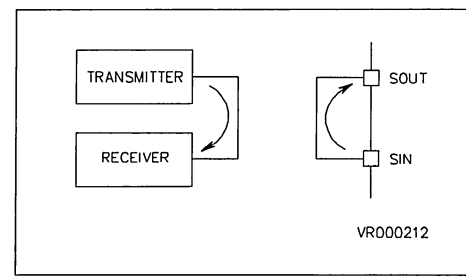


Figure 14-9. Auto Echo and Loop Back Config.



FUNCTIONAL DESCRIPTION (Continued)

14.2.2 Clocks And Serial Transmission Rates

The communication bit frequency of the SCI transmitter and receiver sections can be provided from the integral Baud Rate Generator (allowing a maximum asynchronous bit rate of 350k Baud) or from external sources (maximum bit rate 175k Baud). This clock is divided by 16 for asynchronous mode (CD, CCR.3, = "0"), or divided by 1 for synchronous modes (CD = "1").

External Clock Sources. The External Clock input pin TXCLK may be programmed by bits TXCLK (CCR.7) and OCLK (CCR.6) to be: the transmit clock input (respecting the ÷16 and ÷1 timing requirements), to act as the output of the Baud Rate Generator (allowing an external divider circuit to provide the receive clock for split rate transmit and receive e.g. 1200/75 baud), or to be CLKOUT, the clock output for the synchronous mode.
The Receive clock input via RXCLK input function is enabled by the XRX bit CCR.5, this input should be set according to the setting of the CD bit.

Baud Rate Generator. The integral Baud Rate Generator is a 16-bit programmable divide by "N" counter which can be used to generate the clocks for the transmitter and/or receiver. The minimum baud rate divisor is 2 and the maximum divisor is 2^{16}-1. After initialization of the baud rate generator, the divisor value is immediately loaded into the counter. This prevents potentially long random counts on the initial load.

Baud Rate generator frequency = Input Clock frequency/Divisor

WARNING. *Programming the baud rate division to 0 or 1 will stop the divisor.*

The output of the baud generator has an exact 50% duty cycle. The output can provide either the 16X clock for asynchronous operation or a 1X clock for synchronous and serial port expansion modes for the receiver and the transmitter. An additional divide by 16 may be appropriate to compute the SCI data rate if in this normal operating mode.

The Baud Rate generator can use INTCLK for the input clock source. In this case, INTCLK should be chosen to provide a suitable frequency for division by the Baud Rate Generator to give the required transmit and receive bit rates.
Suitable INTCLK frequencies and the divider values for standard Baud rates are shown in Table 14-2.

Notes:

1) Writing to a Baud Rate Generator Register immediately disables and resets both the SCI baud rate generator, the transmitter and receiver circuitry. After writing to the remaining Baud Rate Generator Register, the transmitter and receiver circuits are enabled. The Baud Rate generator will load the new value and start counting.

Thus to initialize the SCI, the user should first initialize one Baud Rate Generator Divisor Register. This will reset all SCI circuitry. Initialize all other SCI registers for the desired operating mode, and then, to enable the SCI, initialize the remaining Baud Rate Generator Register.

2) For synchronous receive operation, the data and receive clock must not have significant skew between clock and data. The received data and clock are internally synchronized to INTCLK clock.

For synchronous transmit operation, a general purpose I/O port pin must be programmed to output the CLKOUT signal from the baud rate generator. If the SCI is provided with an external transmission clock source, there will be a skew equivalent to two INTCLK periods between clock and data.

The synchronous data will be transmitted on the fall of the transmit clock. The synchronous received data will be latched into the SCI on the rising edge of the provided receive clock.

The maximum data transfer rate is in synchronous mode (1x mode):

- Maximum bit rate = INTCLK/8 = 12MHz/8 = 1.5 Mbit/s
- Maximum byte rate = 1.5 Mbit/10 = 150 Kbytes/s

(one byte = 8 bits of data + 1 stop bit + 1 start bit = 10 bits)

FUNCTIONAL DESCRIPTION (Continued)

Table 14-2. SCI Baud Rate Generator Divider Values

INTCLK: 7680.000 KHz							
Baud Rate	Clock Factor	Desired Freq (kHz)	Divisor		Actual Baud Rate	Actual Freq (kHz)	Deviation
			Dec	Hex			
50.00	16 X	0.80000	9600	2580	50.00	0.80000	0.0000%
75.00	16 X	1.20000	6400	1900	75.00	1.20000	0.0000%
110 00	16 X	1.76000	4364	110C	109.99	1.75985	0.0083%
300 00	16 X	4.80000	1600	0640	300 00	4 80000	0.0000%
600 00	16 X	9.60000	800	0320	600.00	9 60000	0 0000%
1200.00	16 X	19.20000	400	0190	1200.00	19.20000	0.0000%
2400.00	16 X	38.40000	200	00C8	2400.00	38.40000	0.0000%
4800.00	16 X	76.80000	100	0064	4800.00	76.80000	0.0000%
9600.00	16 X	153.60000	50	0032	9600.00	153.60000	0.0000%
19200.00	16 X	307.20000	25	0019	19200.00	307 20000	0.0000%
38400.00	16 X	614.40000	13	000D	36923.08	590 76923	3.8462%
76800.00	16 X	1228.80000	6	0006	80000.00	1280 00000	4.1667%
INTCLK: 11059.20 kHz							
Baud Rate	Clock Factor	Desired Freq (kHz)	Divisor		Actual Baud Rate	Actual Freq (kHz)	Deviation
			Dec	Hex			
50.00	16 X	0.80000	13824	3600	50 00	0.80000	0.0000%
75.00	16 X	1.20000	9216	2400	75.00	1 20000	0.0000%
110.00	16 X	1.76000	6284	188C	109.99	1.75990	0.0058%
300.00	16 X	4.80000	2304	0900	300.00	4 80000	0.0000%
600 00	16 X	9.60000	1152	0480	600.00	9.60000	0 0000%
1200.00	16 X	19.20000	576	0240	1200 00	19.20000	0 0000%
2400.00	16 X	38 40000	288	0120	2400 00	38 40000	0.0000%
4800.00	16 X	76.80000	144	0090	4800.00	76.80000	0.0000%
9600.00	16 X	153.60000	72	0048	9600.00	153.60000	0.0000%
19200.00	16 X	307.20000	36	0024	19200.00	307 20000	0.0000%
38400.00	16 X	614.40000	18	0012	38400.00	614.40000	0 0000%
76800 00	16 X	1228 80000	9	0009	76800.00	1228.80000	0 0000%

FUNCTIONAL DESCRIPTION (Continued)

14.2.3 Input Signals

SIN: Serial Data Input. This pin is the serial data input to the SCI receiver shift register.

TXCLK: External Transmitter Clock Input. This pin is the external input clock driving the SCI transmitter. The TXCLK frequency must be greater than or equal to 16 times the transmitter data rate (depending on the selection of X16 or X1 clock operating mode). The use of the TXCLK pin is optional.

RXCLK: External Receiver Clock Input. This input is the clock to the SCI receiver when using an external clock source to the SCI baud rate generator. INTCLK is normally the clock source. A 50/50 duty cycle is not required for this input, however, the short period must last more than two INTCLK periods. The use of the RXCLK pin is optional.

14.2.4 Output Signals

SOUT: Serial Data Output. This Alternate Function output signal is the serial data output from the SCI transmitter shift register.

CLKOUT: Clock Output. The Alternate Function of this pin outputs either the data clock from the transmitter to an external shift register in the serial expansion mode or the clock output from the Baud rate generator. In serial expansion mode it will clock only the data portion of the frame. The data is valid on the rising edge of the clock. The CLKOUT idle state is low.

14.3 INTERRUPTS AND DMA

14.3.1 Interrupts

The SCI is able to generate interrupts from multiple sources. Receive interrupts include data pending, receive errors (overrun, framing and parity), address or break pending. Transmit interrupts are software selectable for either the Transmit Holding Register Empty (HSN, IMR.7 = "1") or for the Transmit Shift Register Empty (HSN = "0").

Typical Usage of the Interrupts provided by the SCI is shown in Figure 14-10.

The SCI is able to generate interrupt requests on 10 events. Several of these events share the same interrupt vector, so it is necessary to poll ISR, the Interrupt Status Register, to determine the active trigger. These bits should be reset by the programmer during the Interrupt Service routine.

The four major levels of interrupt are encoded in hardware to provide two bits of the interrupt vector register, allowing the position of the block of pointer vectors to be resolved to a block size of 8 bytes.

Table 14-3. SCI Interrupt Vector

Interrupt Source	Vector Address
Transmitter Buffer or Shift Register Empty Transmit DMA end of Block	xxx x110
Received Ready Receive DMA end of Block	xxxx x100
Break Detector Address Word Match	xxxx x010
Receiver Error	xxxx x000

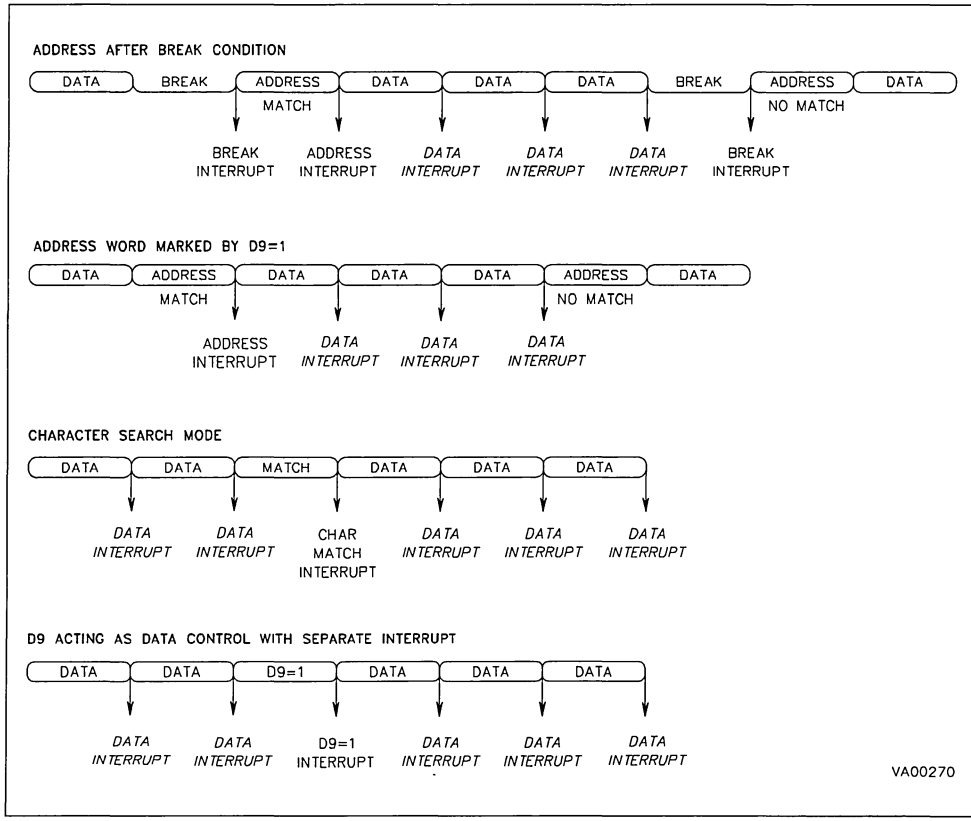
The SCI interrupts have an internal priority structure in order to resolve simultaneous events.

Table 14-4. SCI Interrupt Internal Priority

Receive DMA Request	Highest Priority
Transmit DMA Request	
Receive Interrupt	
Transmit Interrupt	Lower Priority

INTERRUPTS AND DMA (Continued)

Figure 14-10. SCI Interrupt Typical Usage



INTERRUPTS AND DMA (Continued)

14.3.2 DMA

Two DMA channels are associated with the SCI, for transmit and for receive. These follow the register scheme as described in DMA chapter. It should be noted that, after initializing the DMA counter and pointer registers and enabling DMA, data transmission is triggered by a character written into the Transmit Holding register.

When DMA is active the Receive Data Pending bit (RXDP, ISR.2), and the Transmit status bit interrupt sources are replaced by the DMA End Of Block Interrupt sources for transmit and receive, respectively.

The last DMA data word of a block of data will cause a DMA cycle followed by a transmit interrupt. This sequence will signal to the ST9 core to reinitialize the transmit DMA block counter. The Transmit End of Block status bit (TXEOB) should be reset by software in order to avoid undesired interrupt routines, especially in the initialisation routine (after reset) and after entering the End Of Block interrrupt routine.

Similarly the last DMA data word of a block of data will cause a DMA cycle followed by a receiver data ready interrupt. This sequence will signal to the ST9 core to reinitialize the receiver DMA block counter. The Received End of Block status bit (RXEOB) should be reset by software in order to avoid undesired interrupt routines, especially in the initialisation routine (after reset) and after entering the End Of Block interrupt routine.

Remark: If properly initialized, the DMA controller starts a data transfer after and only if the running program has loaded the Transmitter Buffer Register with a value. In order to execute properly a DMA transmission, the End Of Block interrupt routine must include the following actions:

- Load the Transmitter Buffer Register (TXBR) with the first byte to transmit.
- Restore the DMA counter (TDCPR)
- Restore the DMA pointer (TDAPR)
- Reset the transmitter end of block bit TXEOB (IMR.5)
- Reset the transmitter holding empty bit TXHEM (ISR.1)
- Enable DMA

14.4 CONTROL REGISTERS

The relative pages of the SCI in the ST9 are:

- SCI number 1: page 24 (18h)
- SCI number 2: page 25 (19h) (when available)

The SCI is controlled by the following registers:

Address	Register
R240 (F0h)	Receiver DMA Transaction Counter Pointer Register
R241 (F1h)	Receiver DMA Source Address Pointer Register
R242 (F2h)	Transmitter DMA Transaction Counter Pointer Register
R243 (F3h)	Transmitter DMA Destination Address Pointer Register
R244 (F4h)	Interrupt Vector Register
R245 (F5h)	Address Compare Register
R246 (F6h)	Interrupt Mask Register
R247 (F7h)	Interrupt Status Register
R248 (F8h)	Receive Buffer Register same Address as Transmitter Buffer Register (Read Only)
R248 (F8h)	Transmitter Buffer Register same Address as Receive Buffer Register (Write only)
R249 (F9h)	Interrupt/DMA Priority Register
R250 (FAh)	Character Configuration Register
R251 (FBh)	Clock Configuration Register
R252 (FCh)	Baud Rate Generator Register
R253 (FDh)	Baud Rate Generator Register
R254 (FEh)	Reserved
R255 (FFh)	Reserved

CONTROL REGISTERS (Continued)

RDCPR R240 (F0h) Read/Write
Receiver DMA Transaction Counter Pointer

Reset value: undefined

7							0
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RR/M

b7-b1 = **RC7-RC1:** *Receive DMA Counter Pointer.* RDCPR contains the address of the pointer (in the Register File) of the DMA receiver transaction counter.

b0 = **RR/M:** *Receiver Register File/Memory Selector.* If this bit = "1" the Register File will be selected as Destination, if this bit = "0" the Memory space will be selected.

RDAPR R241 (F1h) Read/Write
Receiver DMA Source Address Pointer

Reset value: undefined

7							0
RA7	RA6	RA5	RA4	RA3	RA2	RA1	RD/P

b7-b1 = **RA7-RA1:** *Receive DMA Address Pointer.* RDAPR contains the address of the pointer (in the Register File) of the receiver DMA data source.

b0 = **RD/P:** *Receive DMA Data/Program Memory Selector.* If memory (RR/M = "0") has been selected for DMA transfers, when this bit = "1" receiver DMA transfers will go to Data Memory. If this bit = "0" receiver DMA transfers will go to Program Memory.

TDCPR R242 (F2h) Read/Write
Transmitter DMA Transaction Counter Pointer

Reset value: undefined

7							0
TC7	TC6	TC5	TC4	TC3	TC2	TC1	TR/M

b7-b1 = **TC7-TC1:** *Transmitter DMA Counter Pointer.* TDCPR contains the address of the pointer (in the Register File) of the DMA transmitter transaction counter.

b0 = **TR/M:** *Transmitter Register File/Memory Selector.* If this bit = "1" the Register File will be selected as Source, if this bit = "0" the Memory space will be selected.

TDAPR R243 (F3h) Read/Write
Transmitter DMA Destination Address Pointer

Reset value: undefined

7							0
TA7	TA6	TA5	TA4	TA3	TA2	TA1	TD/P

b7-b1 = **TA7-TA1:** *Transmitter DMA Address Pointer.* TDAPR contains the address of the pointer (in the Register File) of the transmitter DMA data source.

b0 = **TD/P:** *Transmitter DMA Data/Program Memory Selector.* If memory (TR/M = "0") has been selected for DMA transfers, when this bit = "1" transmitter DMA transfers come from Data Memory. If this bit = "0" transmitter DMA transfers come from Program Memory

IVR R244 (F4h)Read/Write
Interrupt Vector Register

Reset value: undefined

7							0
V7	V6	V5	V4	V3	EV2	EV1	0

b7-b3 = **V7-V3:** *SCI Interrupt Vector Base Address.* User programmable interrupt vector bits for transmitter and receiver

b2-b1 = **EV2-EV1:** *Encoded Interrupt Source (Read only).* EV2 and EV1 are set by hardware according to the interrupt source.

EV2	EV1	Interrupt source
0	0	Receiver Error (Overrun, Framing, Parity)
0	1	Break detect or address match
1	0	Receiver data ready/receiver DMA End of Block
1	1	Transmitter buffer or shift register empty transmitter DMA End of Block

b0 = **D0:** This bit is fixed by hardware. It always returns the value "0" when read.

CONTROL REGISTERS (Continued)

ACR R245 (F5h) Read/Write
Address/Data Compare Register

Reset value: undefined

7							0
AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0

b7-b0 = **AC7-AC0:** *Address/Compare Character.* With either 9th bit address mode, address after break mode, or character search, the received address will be compared to the value stored in this register. When a valid address matches this register content, the Receive Address Pending bit is set. After the RXAP bit is set in an addressed mode all received data words will be transferred to the Receiver Buffer Register.

IMR R246 (F6h) Read/Write
Interrupt Mask Register

Reset value: 0xx0 0000b

7							0
HSN	RXEOB	TXEOB	RXE	RXA	RXB	RXDI	TXDI

b7 = **HSN:** *Holding or shift register empty interrupt.* This bit selects the source of interrupt/DMA as the transmitter register empty event. If this bit is set to "1", a holding register empty will generate a transmitter register empty interrupt.
If this bit has a "0" value, a shift register empty will generate a transmitter register empty interrupt.

b6 = **RXEOB:** *Received End of Block.* This bit is set after a receiver DMA cycle to mark the end of a block of data. The last DMA data word will cause a DMA cycle followed by a receiver data ready interrupt. This sequence will signal to the ST9 core to reinitialize the receiver DMA block counter. RXEOB should be reset by software in order to avoid undesired interrupt routines, especially in the initialisation routine (after reset) and after entering the End Of Block interrupt routine.
Writing "0" in this bit will cancel the interrupt request.

Note. RXEOB can only be written with a "0" (RXEOB = set only by the ST9 core).

b5 = **TXEOB:** *Transmitter End of Block.* This bit is set in a transmitter DMA cycle to mark the end of a block of data. The last DMA data word will cause a DMA cycle followed by a transmitter interrupt. This sequence will signal to the ST9 core to reinitialize the transmitter DMA block counter. TXEOB should be reset by software in order to avoid undesired interrupt routines, especially in initialisation routine (after reset) and after entering the End Of Block interrrupt routine.
Writing "0" in this bit will cancel the interrupt request.

Note. TXEOB can only be written with a 0 (TXEOB is set only by the ST9 core)

b4 = **RXE:** *Receiver Error Mask.* When this bit is set to "0", the receiver error bits: Overrun Error (OE), Parity Error (PE), and Framing Error (FE), cannot generate an interrupt.

b3 = **RXA:** *Receiver Address Mask.* When this bit is set to "0", the Receiver Address Pending (RXAP) bit cannot generate an interrupt.

b2 = **RXB:** *Receiver Break Mask.* When this bit is set to "0", the Receiver Break Pending (RBP) bit cannot generate an interrupt.

b1 = **RXDI:** *Receiver Data Interrupt Mask.* When this bit is set to "0", the Receiver Data Pending (RDP) bit and the Receiver End of Block (RXEOB) bit cannot generate an interrupt. RXDI has no effect on DMA transfers.

b0 = **TXDI:** *Transmitter Data Interrupt Mask.* When this bit is set to "0", neither the Transmitter Holding or Shift Register Empty (TXHEM) bit or the Transmitter End of Block (TXEOB) bit can generate an interrupt. TXDI has no effect on DMA transfers.

CONTROL REGISTERS (Continued)

ISR R247 (F7h) Read/Write
Interrupt Status Register

Reset value: undefined

7							0
OE	FE	PE	RXAP	RXBP	RXDP	TXHEM	TXSEM

b7 = **OE:** *Overrun Error Pending.* This bit is set to a logic "1" if the data in the Receiver Buffer Register was not read by the CPU before the next character was transferred into the Receiver Buffer Register (the previous data is lost). It is cleared by writing a zero into OE.

b6 = **FE:** *Framing Error Pending bit.* This bit is set to a logic "1" if the received data word did not have a valid stop bit. It is cleared by writing a zero to the bit. In the case where a framing error occurs when the SCI is programmed in an address mode, and is monitoring for an address, this interrupt is asserted and the corrupted data element is transferred to the Receiver Buffer Register.

b5 = **PE:** *Parity Error Pending.* This bit is set to a logic "1" if the received word did not have the correct even or odd parity bit. It is cleared by writing a zero into PE.

b4 = **RXAP:** *Receiver Address Pending.* RXAP is set to "1" after an interrupt acknowledged in the address mode. The source of this interrupt is given by the couple of bits (AMEN, AM) as detailed in the "Interrupt/DMA Priority Register" description. RXAP is cleared by software.

b3 = **RXBP:** *Receiver Break Pending bit.* This bit is set to a logic "1" if the received data input is held low for the full word transmission time (start bit, data bits, parity bit, stop bit). It is cleared by writing a zero into RXBP.

b2 = **RXDP:** *Receiver Data Pending bit.* This bit is set to a logic "1" when data is loaded into the Receiver Holding Register. It is cleared by writing a zero into RXDP.

b1 = **TXHEM:** *Transmitter buffer register Empty.* This bit is set to a logic "1" if the Holding Register is empty. It is cleared by writing a zero into TXHEM.

b0 = **TXSEM:** *Transmitter Shift Register Empty.* This bit is set to a logic "1" if the Shift Register has completed the transmission of the available data. It is cleared by writing a "0" into TXSEM.

Note.
The Interrupt Status Register bits can be reset by writing a "0" but it is not possible to write a "1" into any bit in this register. It is mandatory to clear the interrupt source by writing a "0" in the pending bit when executing the interrupt service routine.

When servicing an interrupt routine, the User should reset ONLY the pending bit relative to the serviced interrupt routine (and not reset the other pending bits).

RXBR R248 (F8h) Read only
Receive Buffer Register

Reset value: undefined

7							0
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0

b7-b0 = **RD7-RD0:** *Received Data.* This register stores the data portion of the received word. The data will be transferred from the Receiver Shift Register into the Receiver Buffer Register at the end of the word. All receiver interrupt conditions will be updated at the time of transfer. If the selected character format is less than 8 bits, unused most significant bits will forced to "1".

TXBR R248 (F8h) Write only
Transmitter Buffer Register

Reset value: undefined

7							0
TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

b7-b0 = **TD7-TD0:** *Transmit Data.* The ST9 core will load the data for transmission into this register. The SCI will transfer the data from the buffer into the Shift Register when available. At the transfer, the Transmitter Buffer Register interrupt is updated. If the selected word format is less than 8 bits, the unused most significant bits are not significant.

CONTROL REGISTERS (Continued)

IDPR R249 (F9h) Read/Write
Interrupt/DMA Priority Register

Reset value: undefined

7							0
AMEN	SB	SA	RXD	TXD	PRL2	PRL1	PRL0

b7 = **AMEN:** *Address Mode Enable.* This bit, with AM (R250), decodes the desired addressing/9th data bit/character match operation.

AMEN	AM	
0	0	Address interrupt if 9th data bit =1
0	1	Address interrupt if character match
1	0	Address interrupt if character match and 9th data bit =1
1	1	Address interrupt if character match with word immediately following Break

In an addressed mode the SCI will monitor the input serial data until its address is detected.

Upon reception of address, the RXAP bit (in the Interrupt Status Register) is set and an interrupt cycle can begin. The address character will not be transferred into the Receiver Buffer Register but, all data following the matched SCI address and preceeding the next address word will be transferred to the Receiver Buffer Register and the proper interrupts updated. If the address does not match, all data following this unmatched address will not be transferred to the Receiver Buffer Register.

In any of the cases the RXAP bit must be reset by software before the next word is transferred into the Buffer Register.

When AMEN = "0" and AM = "1", a useful character search function is performed. This allows the SCI to generate an interrupt whenever a specific character is encountered (e.g. Carriage Return).

b6 = **SB:** *Set Break.* If this bit is set, a break will be transmitted following the transmission of all data in the Transmitter Shift Register and the Buffer Register. The break will be a "0" value on the transmitter data output for at least one complete word format. If software does not reset SB before the minimum break length has finished, the break condition will continue until software resets SB. The SCI terminates the break condition with a "1" on the transmitter data output for one transmission clock period.

b5 = **SA:** *Set Address.* If an address/9th data bit mode is selected, SA value will be loaded for transmission. Setting this bit indicates an address word. SA will be cleared by hardware after it is loaded into the Shift Register. Proper procedure would be, when the Transmitter Buffer Register is empty, to load the value of SA and then load the data into the Transmitter Buffer Register.

b4 = **RXD:** *Receiver DMA Mask.* If this bit is "0", no receiver DMA request will be generated, and the RXDP bit in the Interrupt Status Register can request an interrupt. If RXD is set to "1", the RXDP bit can request a DMA transfer. This bit is reset by hardware when the transaction counter value decrements to zero. At that time a receiver "end of block" interrupt can occur.

b3 = **TXD:** *Transmitter DMA Mask.* If this bit is "0" no transmitter DMA request will be generated and the TXHEM (or TXSEM) bit in the Interrupt Status Register can request an interrupt. If TXD is set, the TXHEM (or TXSEM) bit can request a DMA transfer. This bit is reset by hardware when the transaction counter value decrements to zero. At that time a transmitter End Of Block interrupt can occur.

b2-b0 = **PRL2, PRL2, PRL0:** *SCI Interrupt/DMA Priority bits.* The priority for the SCI is encoded with (PRL2,PRL1,PRL0). A priority value of 0 has the highest priority, a value of 7 has no priority.

When user has defined a priority level for the SCI, priorities inside the SCI are hardware defined. These SCI internal priorities are:

receiver DMA request	higher priority
transmitter DMA request	
receiver interrupt	
transmitter interrupt	lower priority

CONTROL REGISTERS (Continued)

CHCR R250 (FAh) Read/Write
Character Configuration Register

Reset value: undefined

7							0
AM	EP	PEN	AB	SB1	SB0	WL1	WL0

b7 = **AM:** *Address Mode.* decodes the desired addressing/9th data bit/character match operation in conjunction with AMEN (IDPR.7, R249).

b6 = **EP:** *Even Parity.* When parity is enabled, this bit selects between even or odd parity. If this bit is equal to "0", odd parity will be selected. If this bit is equal to "1", even parity will be selected.

b5 = **PEN:** *Parity Enable.* When this bit is equal to "1", a parity bit is generated (transmit data) or checked (received data) between the last word bit and the stop bits. If the address/9th bit is enabled, the parity bit will precede the address/9th bit (The parity bit is used to produce an even or odd number of 1's when the parity bit and all data bits are summed. The 9th bit is never included in the parity calculation).

b4 = **AB:** *Address/9th Bit.* If this bit equals "1" the transmit and receive character format will include a bit between the parity bit and the first stop bit. This bit can be used to address the SCI or as a ninth data bit.

b3-b2 = **SB1-SB2:** *Stop Bits.* This bit field specifies the number of stop bits to be included in the data format

SB2	SB1	Number of stop bits	
		in 16X mode	in 1X mode
0	0	1	1
0	1	1.5	2
1	0	2	2
1	1	2.5	3

b1-b0 = **WL1, WL0:** These two bits specify the number of data bits in each transmitted or received character. The following table shows the coding of WL.

WL1	WL0	Data Length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

CCR R251 (FBh) Read/Write
Clock Configuration Register

Reset value: 0000 0000 (00h)

7							0
XTCLK	OCLK	XRX	XBRG	CD	AEN	LBEN	STPEN

b7 = **XTCLK:**

b6 = **OCLK:** These two bits select the source for the transmitter clock. The following table shows the coding of XTCLK and OCLK.

XTCLK	OCLK	Pin Function
0	0	Pin is used as a general I/O
0	1	Pin = TXCLK (used as an input)
1	0	Pin = CLKOUT (outputs the Baud Rate Generator clock)
1	1	Pin = CLKOUT (outputs the Serial exp. mode clock)

b5 = **XRX:** *External Receiver Clock Source.* If this bit is "1", the receiver will use the external receiver clock pin for its clock source. The external clock must be equal to 16 times the data rate or equal to the data rate depending on the bit CD.

b4 = **XBRG:** *Baud Rate Generator Clock Source.* If this bit is "1", the baud rate generator will use the external receiver clock pin for its clock source. If this bit is "0", the baud rate generator will use the ST9 system clock (INTCLK).

b3 = **CD:** *Clock Divisor.* If CD = "1", both the receiver and the transmitter will be in 1X clock mode. In 1X clock mode, the transmitter will transmit data at one data bit per clock period. If this bit is "0", both the receiver and the transmitter will be in 16X mode. In 16X mode each data bit period will be 16 clock periods long.

The CD value will determine the synchronous/asynchronous SCI configuration mode.

b2 = **AEN:** *Auto Echo Enable.* If AEN = "1", the SCI is in auto echo mode. In this mode the SCI transmitter is disconnected from the transmitter data-out pin (SOUT). The transmitter data-out pin (SOUT) is driven directly by the receiver data-in pin (SIN). The receiver remains connected to the receiver data-in pin (SIN) and is operational, unless loopback mode is also selected.

CONTROL REGISTERS (Continued)

b1 = **LBEN:** *Loopback Enable.* If this bit is set to "1", the loopback mode is enabled. In this mode the transmitter output is set to "1", the receiver input is disconnected, and the output of the Transmitter Shift Register is looped back into the Receiver Shift Register input. All interrupt sources for both the transmitter and the receiver are operational.

b0 = **STPEN:** *Stick Parity Enable.* If this bit is set to "1", the transmitter and the receiver will use the opposite parity type selected by the even parity bit (EP).

EP	SPEN	Parity (Transmitter & Receiver)
0 (odd)	0	Odd
1 (even)	0	Even
0 (odd)	1	Even
1 (even)	1	Odd

BRGHR R252 (FCh) Read/Write
Baud Rate Generator Register, High byte.

Reset value: undefined

15							8
BG15	BG14	BG13	BG12	BG11	BG10	BG9	BG8

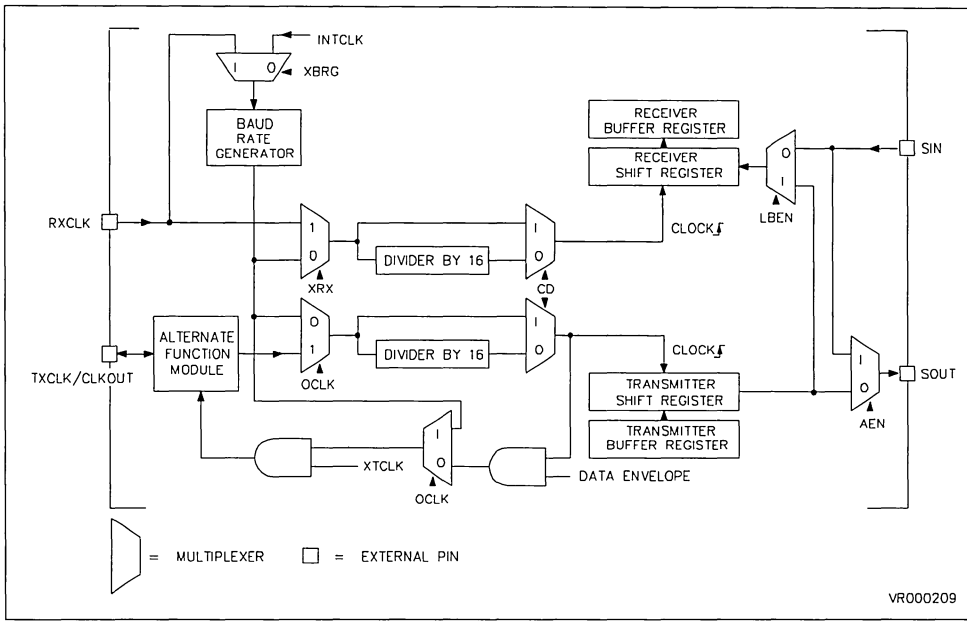
BRGLR R253 (FDh) Read/Write
Baud Rate Generator Register, Low byte.

Reset value: undefined

7							0
BG7	BG6	BG5	BG4	BG3	BG2	BG1	BG0

b15-b0: *The Baud Rate generator* is a programmable divide by "N" counter which can be used to generate the clocks for the transmitter and/or receiver. This counter divides the clock input by the value in the Baud Rate Generator Register. The minimum baud rate divisor is 2 and the maximum divisor is 2^{16}-1. After initialization of the baud rate generator, the divisor value is immediately loaded into the counter. This prevents potentially long random counts on the initial load.
If set to 0 or 1, the Baud Rate Generator is stopped.

Figure 14-11. SCI Functional Scheme



15 A/D CONVERTER

15.1 INTRODUCTION

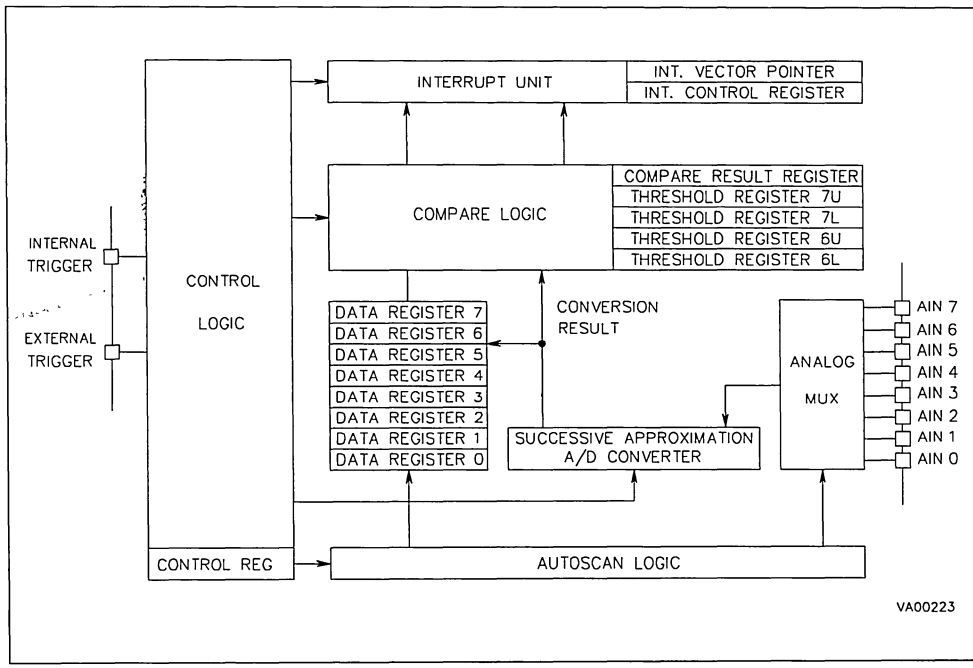
The Analog to Digital Converter (A/D) is comprised of an 8 channel multiplexed input selector and a Successive Approximation converter. The conversion time is thus a function of the INTCLK frequency; for the maximum 12MHz clock rate, conversion of the selected channel requires 11µs. This time also includes the 3µs of the integral Sample and Hold circuitry, which minimizes the need for external components and allows quick sampling of the signal for the minimum warping effect and Integral conversion error.
The resolution of the converted channel is 8 bits, with ±1/2 LSB maximum DNL error between the Analog V_{SS} and V_{DD} references.

The converter uses a fully differential analog input configuration for the best noise immunity and precision performance, along with two separate supply references, allowing the best supply noise rejection and possible analog references lower than the Digital V_{CC}. In fact, the converted digital value, is referred to the Analog V_{CC} (AV_{CC}) as the full scale value, so that using, for example a value of 4 Volt for this supply, all conversions will accordingly refer to this 4 Volt full scale value ($AV_{SS} = V_{SS}$ mandatory).

Up to 8 multiplexed Analog Inputs are available. A group of signals can be converted sequentially by simply programming the starting address of the first analog channel to be converted and using the AUTOSCAN feature.

Two Analog Watchdogs are provided, on analog input channels 6 and 7, allowing a continuous hardware monitoring of these two inputs. An alarm Interrupt request will be generated whenever the converted value of either of these two analog inputs exceed one of the two programmed threshold

Figure 15-1. Block Diagram



INTRODUCTION (Continued)

values (Upper and Lower) for each channel. The comparison result is stored in a dedicated register.

Single, continuous, or externally triggered conversion modes are available, internal clock sample synchronization is also available through the "On chip Event" synchronization logic of a Multifunction Timer Unit,

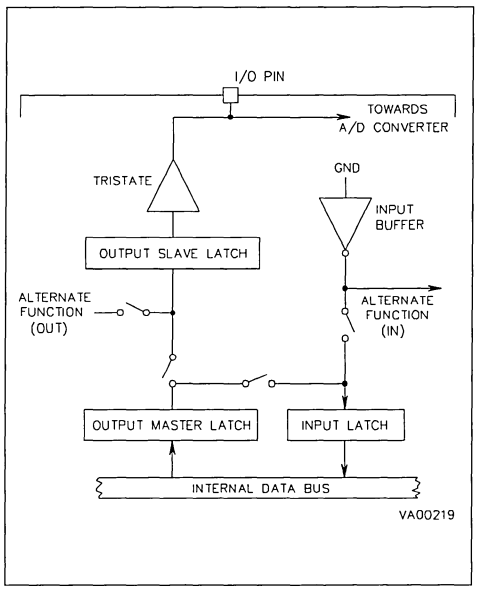
A Power-Down programmable bit allows to set the A/D converter to a minimum consumption idle status.

The ST9 A/D Interrupt Unit provides two maskable channels (Analog Watchdog and End of Conversion) with hardware fixed priority, and up to 7 programmable priority levels.

WARNING: A/D INPUT PIN DECLARATION

The input Analog channel is selected by using the Alternate Function setting (PXC2, PXC1, PXC0 = 1,1,1) as shown in the I/O ports section. The I/O bit structure of the port connected to the A/D converter is modified as shown in Figure 11-2 to prevent the Analog voltage present at the I/O pin from causing high power dissipation across the input buffer. Un-selected analog channels should also be maintained in the Alternate function mode for this reason.

Figure 15-2. A/D Input Configuration Status



15.2 FUNCTIONAL DESCRIPTION

15.2.1 Operational Modes

Two main operational modes are available: Continuous Mode and Single Mode. To enter one of these modes it is necessary to program the CONT bit of the Control Logic Register, the Continuous Mode is selected when CONT = "1", while CONT = "0" enables the Single Mode.

Both modes operate in the AUTOSCAN configuration, allowing a sequential conversion flow of the input channels. It is possible to choose by software the number of analog inputs to be converted by writing into the Control Register (SC2, SC1, SC0 bits) the number of the first channel to be converted. Subsequentially, after each conversion is completed, the channel number is automatically incremented, up to channel 7. For example, if (SC2, SC1, SC0) = 0,1,1 the conversion flow runs from channel 3 up to channel 7. If (SC2, SC1, SC0) = 1,1,1 only channel 7 is converted.

When the ST bit of the Control Logic Register is written to "1" by software or hardware, the analog inputs are sequentially converted (from the first selected channel up to channel 7) and the results are stored in the relevant Data Registers.

In **Single Mode** (CONT = "0"), the ST bit is reset by hardware at the end of conversion of channel 7, an End of Conversion (ECV) interrupt request is issued, and the A/D waits for a new start event.

In **Continuous Mode** (CONT = "1"), a continuous conversion flow is entered by the start event. After the conversion of channel 7 ends, the conversion of channel 's' starts (where 's' is specified in the (SC2, SC1, SC0) bits), this will continue until the ST bit is reset by software. In all cases, an ECV interrupt is issued each time the channel 7 conversion ends.

When channel 'i' is converted ('s' < 'i' < 7), the Data Register is reloaded with the new conversion result and the previous value is lost. The ECV interrupt routine can be used to save the current values before a new conversion sequence (so as to create signal sample tables within Register File or Memory).

15.2.2 Synchronisation

Conversion start synchronisation for all modes may be internal or external. The external (ADTRG, as an Alternate Function input of an I/O port) or the internal (INTRG, produced by a Multifunction Timer peripheral) can be used to synchronise the conversion start with a trigger pulse. Both external

FUNCTIONAL DESCRIPTION (Continued)

and internal events can be seperately masked by the programming of the EXTG/INTG bits of the Control Logic Register. The events are internally OR'ed, thus avoiding potential hardware conflicts, however the correct procedure is to always enable only one alternate synchronisation input at any time.

The effect of the alternate synchronisation is to set the ST bit by hardware. This bit is reset, only in Single Mode, at the end of each group of conversions. In Continuous Mode all trigger pulses, following the first, are ignored.

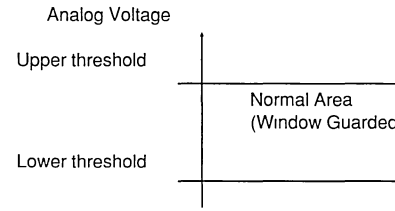
The two synchronisation sources must have a clock cycle minimum length of 83ns (at INTCLK = 12MHz), and a period greater (in Single Mode) than the total time of a group of conversions (11.5µs x the number of channels scanned (at INTCLK = 12MHz)). If a trigger occurs when the ST bit is still "1" (conversion is still in progress), it is ignored.

15.2.3 Analog Watchdog

Two internal Analog Watchdogs are available, allowing great flexibility in automatic threshold monitoring in those applications experiencing a maximum range of fluctuations. Analog channels 6 and 7 define a voltage window for the allowed values of the converted analog input. The range of values of the external voltage applied to input 6 and 7 are accepted as normal whenever *below* the Upper threshold and *above or equal* to the Lower threshold.

When the external voltage is greater or equal to the upper, or is less than the lower programmed voltage limits, a maskable interrupt request is generated and the Compare Results Register is updated to inform which threshold (Upper or Lower) of which channel (6 or 7) has been exceeded. The 4 threshold voltages are user programmable in 4 dedicated registers (8 up to B) of the A/D register page, storing their 8 bit binary code. Only the 4 MSB of the Compare Results Register are used (the 4 LSB always return "1" if read), each bit for each threshold possible overflow or underflow status.

After an hardware reset, these bit values are "0". During the normal A/D operation, the CRR bits are set to "1" to flag an over-range and are automatically reset by hardware after a software reset of the analog Watchdog request flag in the ECR Register.



15.2.4 Power down Mode

Before enabling any A/D conversion, it is mandatory to set the POW bit of the Control Logic Register to "1" at least 60µs before the first conversion start. This is in order to correctly bias the analog section of the converter, if this is not done, then functionality of the converter will be locked.

Setting POW to "0" is useful when the A/D is not required in order to reduce the total power consumption. This is the reset configuration, and is also entered automatically when the ST9 is in Halt Mode (following the execution of the `halt` instruction).

Figure 15-3. A/D Trigger Source

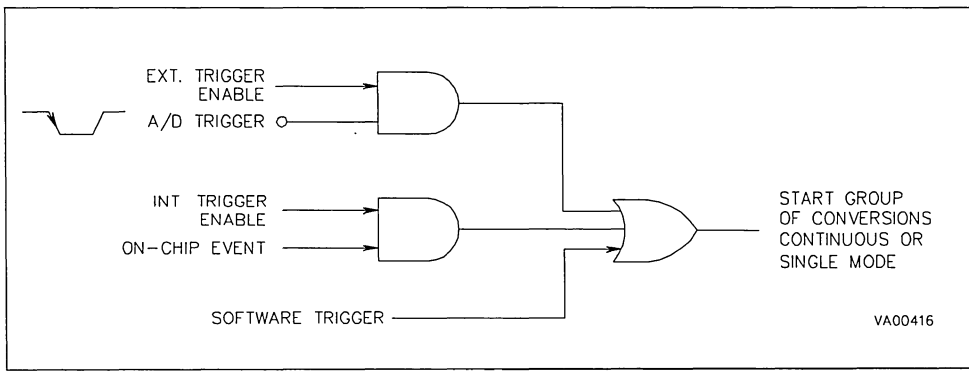
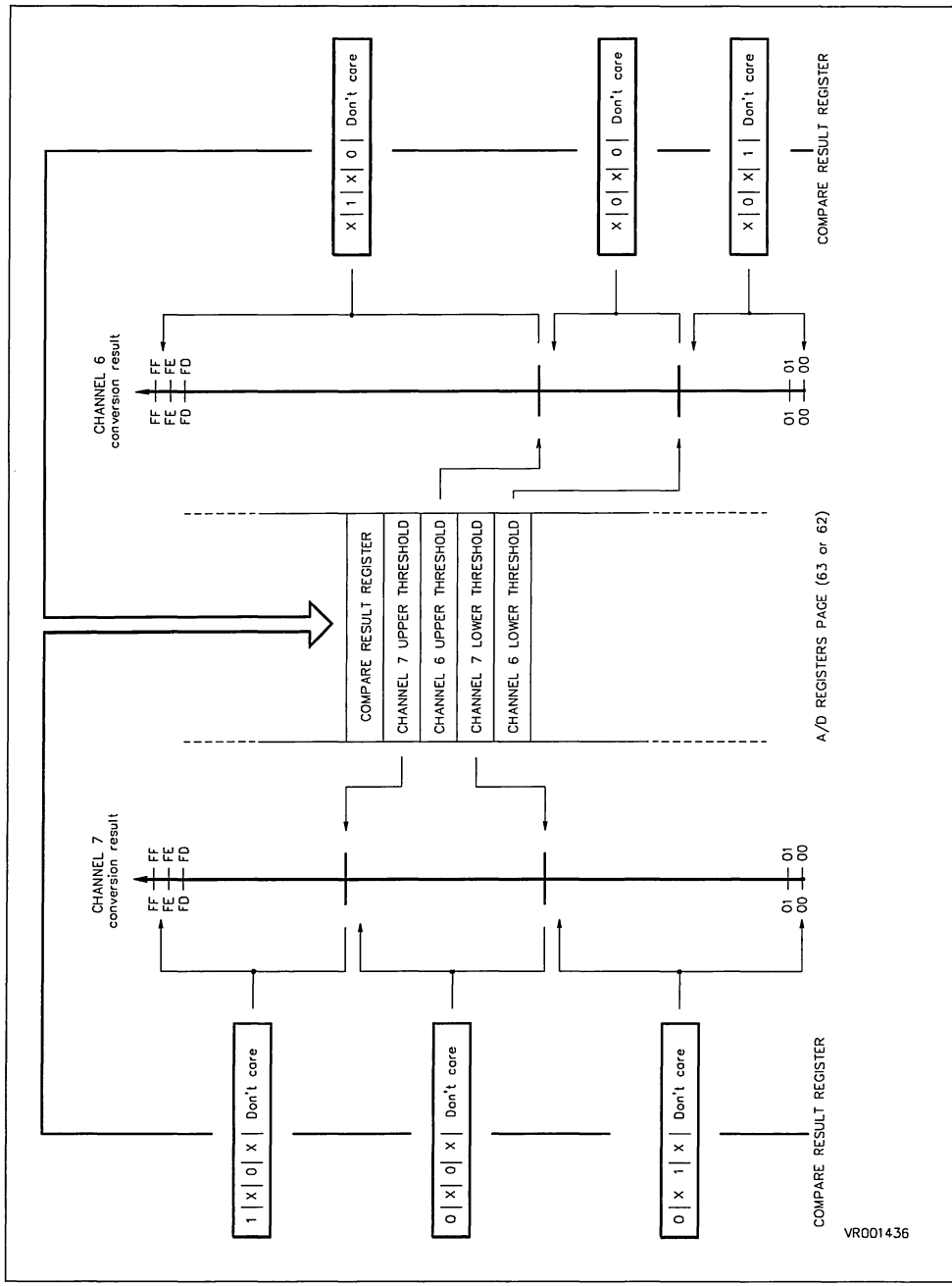
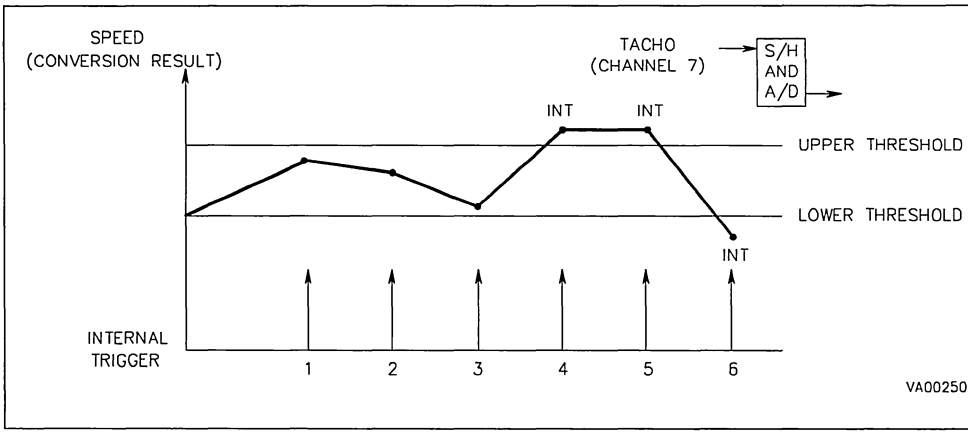


Figure 15-4. Functional Diagram



FUNCTIONAL DESCRIPTION (Continued)

Figure 15-5. Analog Watchdog used in Motorspeed Control



15.3 INTERRUPT

The A/D converter provides two interrupt sources, End of Conversion and an Analog Watchdog Request. The A/D Interrupt Vector Register (IVR) provides 1 bit generated in hardware to follow the interrupt source, allowing the automatic addressing of the relevant A/D Interrupt Service routine for the two sources.

	7						0	0	
ANALOG WATCHOG REQUEST	X	X	X	X	X	X	0	0	Lower Word Address

	7						1	0	
END OF CONV. REQUEST	X	X	X	X	X	X	1	0	Upper Word Address

The A/D Interrupt vector should be programmed by the User to point to the first memory location in the Interrupt Vector table containing the base address of the four byte area of the interrupt vector table in which to store the address of the A/D interrupt service routines.

The Analog Watchdog Interrupt Pending bit (AWD, ICR.6), is automatically hardware set whenever any of the two guarded analog inputs goes out of bounds. The Compare Result Register (CRR) keeps track of the analog inputs exceeding the thresholds.

When 2 requests occur simultaneously the Analog watchdog request has priority over the End of Conversion request which is held pending, to be served after the current routine.

The Analog Watchdog Request requires the user to poll within the Compare Result Register (CRR) to determine which of the four thresholds has been exceeded. The threshold status bits are set to "1" to flag an over-range and are automatically reset by hardware after a software reset of the analog Watchdog request flag in the ECR Register.
The interrupt pending flags, ECV (End of Conversion, ICR.7) and AWD should also be reset by the User in the Interrupt service routine before the return. Setting either of these two bits by software will cause a software interrupt request to be generated.

15.4 REGISTERS

15.4.1 Register Mapping

A/D registers are mapped page 63.

15.4.2 Data Registers (DiR)

The result of the conversions of the 8 available channels are loaded in the 8 DiR (channel0→D0R ....channel7→D7R); every Data Register is reloaded with a new value at the end of the conversion of the correspondent analog input.

D0R R240 (F0h) Page 63 Read/Write
Channel 0 Data Register

Reset Value: Undefined

7							0
D0.7	D0.6	D0.5	D0.4	D0.3	D0.2	D0.1	D0.0

b7-b0 = **D0.7-D0.0**: Channel 0 Data

D1R R241 (F1h) Page 63 Read/Write
Channel 1 Data Register

Reset Value: Undefined

7							0
D1.7	D1.6	D1.5	D1.4	D1.3	D1.2	D1.1	D1.0

b7-b0 = **D1.7-D1.0**: Channel 1 Data

D2R R242 (F2h) Page 63 Read/Write
Channel 2 Data Register

Reset Value: Undefined

7							0
D2.7	D2.6	D2.5	D2.4	D2.3	D2.2	D2.1	D2.0

b7-b0 = **D2.7-D2.0**: Channel 2 Data

D3R R243 (F3h) Page 63 Read/Write
Channel 3 Data Register

Reset Value: Undefined

7							0
D3.7	D3.6	D3.5	D3.4	D3.3	D3.2	D3.1	D3.0

b7-b0 = **D3.7-D3.0**: Channel 3 Data

D4R R244 (F4h) Page 63 Read/Write
Channel 4 Data Register

Reset Value: Undefined

7							0
D4.7	D4.6	D4.5	D4.4	D4.3	D4.2	D4.1	D4.0

b7-b0 = **D4.7-D4.0**: Channel 4 Data

D5R R245 (F5h) Page 63 Read/Write
Channel 5 Data Register

Reset Value: Undefined

7							0
D5.7	D5.6	D5.5	D5.4	D5.3	D5.2	D5.1	D5 0

b7-b0 = **D5.7-D5.0**: Channel 5 Data

D6R R246 (F6h) Page 63 Read/Write
Channel 6 Data Register

Reset Value: Undefined

7							0
D6.7	D6.6	D6.5	D6.4	D6.3	D6.2	D6.1	D6 0

b7-b0 = **D6.7-D6.0**: Channel 6 Data

D7R R247 (F7h) Page 63 Read/Write
Channel 7 Data Register

Reset Value: Undefined

7							0
D7 7	D7.6	D7.5	D7 4	D7 3	D7.2	D7 1	D7 0

b7-b0 = **D7.7-D7.0**: Channel 7 Data

REGISTERS (Continued)

15.4.3 Lower Threshold Registers (LTiR)

The 2 lower threshold registers are used to store the 2 user programmable lower threshold voltages (i.e. their 8 bit binary code) to be compared with the present channel 6 or 7 conversion result. They fix the lower voltage window limit.

LT6R R248 (F8h) Page 63 Read/Write
Channel 6 Lower Threshold Register

Reset Value: Undefined

7							0
LT6.7	LT6.6	LT6.5	LT6.4	LT6.3	LT6.2	LT6.1	LT6.0

b7-b0 = **LT6.7-LT6.0**: *Channel 6 Lower Threshold*

LT7R R249 (F9h) Page 63 Read/Write
Channel 7 Lower Threshold Register

Reset Value: Undefined

7							0
LT7.7	LT7.6	LT7.5	LT7.4	LT7.3	LT7.2	LT7.1	LT7.0

b7-b0 = **LT7.7-LT7.0**: *Channel 7 Lower Threshold*

11.4.6 Upper Threshold Registers (UTiR)

The 2 upper threshold registers are used to store the 2 user programmable upper threshold voltages (i.e. their 8 bit binary code) to be compared with the present channel 6 or 7 conversion result. They fix the upper voltage window limit.

UT6R R250 (FAh) Page 63 Read/Write
Channel 6 Upper Threshold Register

Reset Value: Undefined

7							0
UT6.7	UT6.6	UT6.5	UT6.4	UT6.3	UT6.2	UT6.1	UT6.0

b7-b0 = **UT6.7-UT6.0**: *Channel 6 Upper Threshold*

UT7R R251 (FBh) Page 63 Read/Write
Channel 7 Upper Threshold Register

Reset Value: Undefined

7							0
UT7.7	UT7.6	UT7.5	UT7.4	UT7.3	UT7.2	UT7.1	UT7.0

b7-b0 = **UT7.7-UT7.0**: *Channel 7 Upper Threshold*

15.4.4 Compare Result Register (CRR)

The result of comparison between the current value of data registers 6 and 7 and the threshold registers is stored in this 4 bit register.

CRR R252 (FCh) Page 63 Read/Write
Compare Result Register

Reset Value: 0000 1111 (0Fh)

7							0
C7U	C6U	C7L	C6L	X	X	X	X

b7 = **C7U:** *Compare Reg 7 Upper threshold* Set to "1" when converted data is greater than or equal to the threshold value. Not affected otherwise

b6 = **C6U:** *Compare Reg 6 Upper threshold* Set to "1" when converted data is greater than or equal to the threshold value. Not affected otherwise

b5 = **C7L:** *Compare Reg 7 Lower threshold* Set to "1" when converted data is less than the threshold value. Not affected otherwise.

b4 = **C6L:** *Compare Reg 6 Lower threshold* Set to "1" when converted data is less than the threshold value. Not affected otherwise.

These bits should be Software reset at the end of the 'Out of Bounds' Interrupt routine.

b3-b0 = undefined, return "1" when read.

Note. any Software request reset in the ICR, will cause also all the Compare status bits to be hardware forced to zero, to prevent possible overwriting if an Interrupt request occurs between the Software reset and the Interrupt Request Software reset.

REGISTERS (Continued)

15.4.5 Control Logic Register (CLR)

This register manages the A/D logic operations. Writing into this register will cause the current conversion to be aborted and the autoscan logic to be re-initialized to the starting configuration. CLR is programmable as following:

CLR R253 (FDh) Page 63 Read/Write
Control Logic Register

Reset Value: 0000 0000 (00h)

7							0
SC2	SC1	SC0	EXTG	INTG	POW	CONT	ST

b7-b5 = **SC2, SC1, SC0:** *Start Conversion Address.* These 3 bits define the starting analog input in Autoscan mode. The first channel addressed by SC2-SC0 is converted, then the address is incremented for the successive conversion, until channel 7 (111) is converted. The (SC2, SC1, SC0) bits define the group of channels to be scanned. When setting SC2=1 SC1=1 SC0=1 only channel 7 is converted.

b4 = **EXTG:** *External Trigger.* When set to a logical "1", this bit allows to start a group of conversion synchronized on the following edge of the external signal applied on pin ADTRG (when enabled for Alternate Function)..

b3 = **INTG:** *Internal Trigger.* When set to a logical level "1", this bit enables the start of a group of conversion, synchronized with an internal signal (On chip Event signal) from a Multifunction Timer Unit.

Both External and Internal Trigger inputs are internally OR'ed, thus avoiding Hardware conflicts, however the correct procedure is to enable only one alternate synchronization input at time.

b2 = **POW:** *Power Up/Power Down* A logical "1" enables the A/D logic and analog circuitry.
A logical "0" disables all power consuming logic, thus allowing a low power idle status.

b1 = **CONT:** *Continuous/Single.* When this bit is set to "1" (Continuous Mode), the first group of conversions are started either by software (setting to "1" the ST bit) or by hardware (on an Internal or external trigger, depending on the INTG and EXTG bits status), and a continuous conversion flow is then processed.

When this bit is set to "0" (single mode), only a single group of conversions (1 up to 8) are started whenever any External (or Internal) trigger occurs, or the ST bit is set to "1" by software.

The effect of the alternate synchronization is to hardware set the START/STOP bit which is hardware reset when in SINGLE mode, at the end of each group of conversions.

Requirements:

The External Synchronisation Input must receive a pulse (low level) wider than an INTCLK period (83ns) minimum and for both External and On chip Event synchronisation, a period greater than the time required for a group of conversion (number of channels times x 11µs).

b0 = **ST:** *Start/Stop* A logical "1" level enables the starting of a group of conversions; a logical level "0" stops the conversion. When the A/D is running in the Single Mode, this bit is hardware reset at the end of a group of conversions.

REGISTERS (Continued)

b0 = **ST:** *Start/Stop* A logical "1" level enables the starting of a group of conversions; a logical level "0" stops the conversion. When the A/D is running in the Single Mode, this bit is hardware reset at the end of a group of conversions.

15.4.6 Interrupt Control Register (ICR)

This register contains the three priority level bits, the two sources flags, and their bit mask:

ICR R254 (FEh) Page 63 Read/Write
Interrupt Control Register

Reset Value: 0000 1111 (0Fh)

7							0
ECV	AWD	ECI	AWDI	X	PL2	PL1	PL0

b7 = **ECV:** *End of Conversion.* ECV is automatically set by hardware after a group of conversions is completed.

b6 = **AWD:** *Analog Watchdog.* AWD is automatically hardware set whenever any of the two guarded analog inputs goes out of bounds. The threshold values are stored in registers F8h and FAh for channel 6, and in registers F9h and FBh for channel 7 respectively. The Compare Result Register (CRR) keeps track of the analog inputs exceeding the thresholds.

AWD and ECV must be reset by the user, before returning from the Interrupt service routine. Setting either of them by software will cause a software interrupt request to be generated.

b5 = **ECI:** *End of Conversion Interrupt Enable* This bit masks the End of Conversion interrupt request. A logical level "1" enables the request, a logical level "0" masks the request.

b4 = **AWDI:** *Analog Watchdog Interrupt Enable.* This bit masks the Analog Watchdog interrupt request. A logical level "1" enables the request, a logical level "0" masks the request.

b3 = **D3:** *Undefined*

b2-b0 = **PL2, PL1, PL0:** *A/D Interrupt Priority Level.* With these three bits it is possible to select the Interrupt priority level of the A/D Converter.

15.4.7 Interrupt Vector Register (IVR)

IVR R255 (FFh) Page 63 Read/Write
Interrupt Vector Register

Reset Value: xxxx xx10 (x2h)

7							0
V7	V6	V5	V4	V3	V2	W1	D0

b7-b2 = **V7-V2:** *A/D Interrupt Vector.* This vector should be programmed by the User to point to the first memory location in the Interrupt Vector table containing the starting addresses of the A/D interrupt service routines.

b1 = **W1:** *Word Select.* This bit is set by hardware, according to the A/D interrupt source. It is set to "0" if the source is the Analog Watchdog, pointing to the lower word of the A/D interrupt service block (defined by V7-V2). It is set to "1" if the source is the End of Conversion interrupt, thus pointing to the upper word.

When 2 requests occur simultaneously the Analog watchdog request has priority over the End of Conversion request which is held pending, to be served after the current routine.

b0 = **D0:** This bit is fixed by hardware. It always returns the value "0" if read.

16 SOFTWARE DESCRIPTION

16.1 ADDRESSING MODES

The ST9 offers a wide variety of established and new addressing modes and combinations to facilitate full and rapid access to the address spaces while reducing program length. The available addressing modes are shown in Table 16-1:

Single operand arithmetic, logic and shift byte instructions have direct register and indirect register addressing modes. For a full list of the possible combinations for each instruction type, please refer to the ST9 Programming Manual.

Table 1-1. Addressing Modes

Operand is In	Addressing Mode	Destination Location	Notation
Instruction	Immediate	Byte Word	#N #NN
Register File	Direct	Byte Word	r rr
	Indirect	Byte/Word	(r)
	Indexed	Byte/Word	N(r)
	Indirect Post-Increment	Byte	(r)+
Program or Data Memory	Direct	Byte/Word	NN
	Indirect	Byte/Word	(rr)
	Indirect Post-Increment	Byte/Word	(rr)+
	Indirect Pre-Decrement	Byte/Word	-(rr)
	Short Indexed	Byte/Word	N(rr)
	Long Indexed	Byte/Word	NN(rr)
	Register Indexed	Byte/Word	rr(rr)
Any bit of any working register	Direct	Bit	r.b
Any bit in program or data memory	Indirect	Bit	(rr).b

ADDRESSING MODES (Continued)

Two Operands Arithmetic and Logic Instructions	
Destination	**Source**
Register Direct	Register Direct
Register Direct	Register Indirect
Register Direct	Memory Indirect
Register Direct	Memory Indexed
Register Direct	Memory Indirect with Post-Increment
Register Direct	Memory Indirect with Pre-Decrement
Register Direct	Memory Direct
Register Indirect	Register Direct
Memory Indirect	Register Direct
Memory Indexed	Register Direct
Memory Indirect with Post-Increment	Register Direct
Memory Indirect with Pre-Decrement	Register Direct
Memory Direct	Register Direct
Register Direct	Immediate
Memory Direct	Immediate
Memory Indirect	Immediate

Two Operands Arithmetic, Logic and Load Instructions	
Destination	**Source**
Memory Indirect	Memory Direct

ADDRESSING MODES (Continued)

Two Operands Load Instructions	
Destination	**Source**
Register Direct	Register Direct
Register Direct	Register Indirect
Register Direct	Register Indexed
Register Direct	Memory Indirect
Register Direct	Memory Indexed
Register Direct	Memory Indirect with Post-Increment
Register Direct	Memory Indirect with Pre-Decrement
Register Direct	Memory Direct
Register Indirect	Register Direct
Register Indexed	Register Direct
Memory Indirect	Register Direct
Memory Indexed	Register Direct
Memory Indirect with Post-Increment	Register Direct
Memory Indirect with Pre-Decrement	Register Direct
Memory Direct	Register Direct
Register Direct	Immediate
Memory Direct	Immediate
Memory Indirect	Immediate
Long Indexed Memory (1)	Immediate

Two Operands Load Instructions (2)	
Destination	**Source**
Register Indirect with Post-Increment	Memory Indirect with Post-Increment
Memory Indirect with Post-Increment	Register Indirect with Post-Increment
Memory Indirect with Post-Increment	Memory Indirect with Post-Increment

Notes:
1 Word Instructions Only
2 Load Byte Only

ADDRESSING MODES (Continued)

16.1.1 Register Addressing Modes

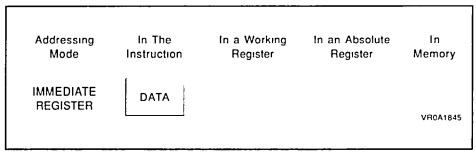
Immediate Addressing Mode

In the Immediate addressing mode, the data is found in the instruction. When using immediate data, a hash-mark (#) is used to distinguish it from an absolute address in memory.

Example: `ldw RR42,#65536`
loads the immediate value 65536 into the register pair R42 & R43. While the example shows decimal data, hexadecimal and binary values may also be used.

Example: `ldw RR42,#0FFFFh.`

Figure 16-1. Immediate Register

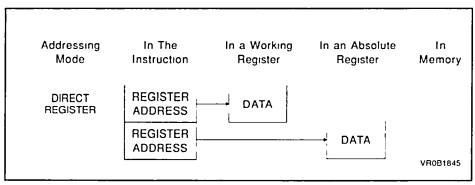


Direct Addressing Mode

In the direct addressing mode, a register can be addressed by using its absolute address in the Register File (in decimal, hexadecimal or binary form). Alternatively a register can be addressed directly as a working register;

Example: `xch R0A2h,r4`
exchanges the values in the register RA2h and working register number 4.

Figure 16-2. Direct Register



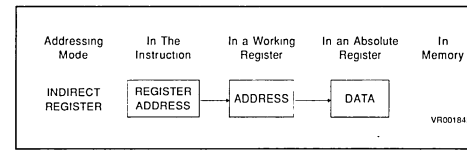
Indirect Addressing Mode

In the Indirect Register Addressing mode, the address of the data does not appear in the instruction but is located in a working register. The address of this register is given in the instruction. The indirect addressing mode is indicated by the use of parentheses.

Example:
If register 200 contains 178 and working register 11 contains 86 then the instruction `ld (r11),R200` loads the value 178 into register 86.

Note: the indirect address can only be contained in a working register

Figure 16-3. Indirect Register



Indexed Addressing Mode

To address a register using the Indexed mode, an offset value is used to add to an index value (which acts as a base or starting value). The offset value is the Immediate value given in the instruction while the index value is given by the contents of the working register.

Example: if working register 10 contains 55 then the instruction

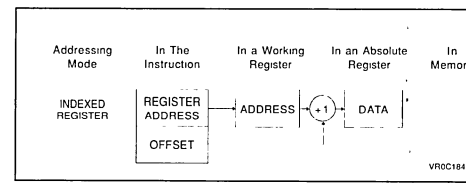
`ld 40(r10),r18`

loads register 95 (i.e.55+40) with the contents of working register 18.

The Register File never needs an absolute value requiring more than one byte and therefore only requires a short offset and a single register to contain the index.

Note: The index value can only be contained in a working register

Figure 16-4. Indexed Register



ADDRESSING MODES (Continued)

Indirect Register Post-increment Addressing Mode

In this addressing mode, both destination and source addresses are given by the contents of working registers which are then post-incremented. The address of the memory location is contained in a working register pair, and the address of the register is contained into a single working register. Only working registers may be used to contain the addresses, this mode being indicated by both source and destination using parentheses followed by plus sign.

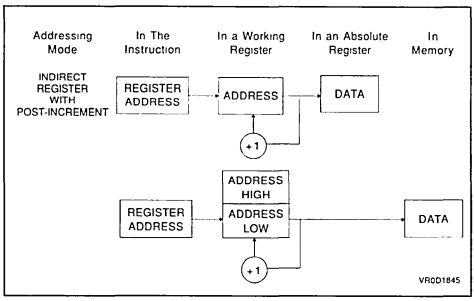
Example: if working register 8 contains the value 44, working register pair rr2 contains the value 2000, and register 44 contains the value 56, then by using the instruction

```
ld (rr2)+,(r8)+
```

the memory location 2000 will be loaded with the value 56. Immediately following this, the contents of r8 is incremented to 45 and the contents of rr2 is incremented to 2001.

This addressing mode is useful for moving blocks of data either from Register File to Memory or from Memory to Register File.

Figure 16-5. Register Indirect Post-Increment



Direct Bit Addressing Mode

In the direct bit addressing mode, any bit in any working register can be addressed

Examples: `bset r7.3`

This instruction sets the bit 3 of the working register 7.

```
bld r7.3, r12.6
```

This instruction loads the bit 6 of the working register 12 in bit 3 of working register 7

16.1.2 Memory Addressing Modes

The memory addressing modes described in this section are available to data and program memory. Thus before addressing the memory, it is necessary to indicate by use of the Set Program/Data Memory instructions, `spm` and `sdm`, in which memory the instructions are working. Since each memory space is 64K byte long, a word address is necessary to specify memory locations.

Direct Addressing Mode

The Memory Direct addressing mode requires the specific location within the memory. This only needs the absolute offset value which can be given in decimal, hex or binary form.

Thus the instruction

```
ld 12345,r9
```

loads working register 9 data into memory location 12345

In the memory direct mode, it is possible to use an immediate addressing mode for the source operand.

Examples:

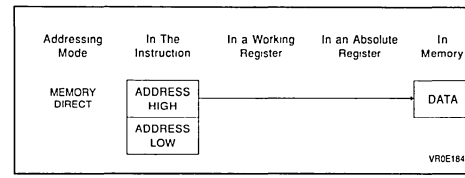
```
ld 12354,#34
```

will load the value 34 into the memory location 12354.

```
ldw 12354,#3457
```

will load the location pair 12354 and 12355 with the value 3457.

Figure 16-6. Memory Direct



ADDRESSING MODES (Continued)

Indirect Addressing Mode

When using the indirect addressing mode to access memory, the address is contained in a pair of working registers.

Example: if the working register pair r8 and r9 contains the value 2000 then the instruction

```
ld (rr8),#34
```

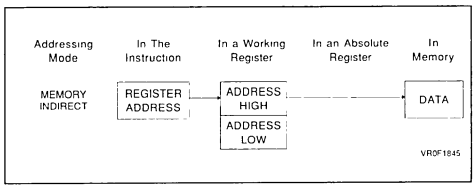
loads the value 34 into memory location 2000.

If the data to be stored is a word then the instruction `ldw` will automatically interpret the address as a pair of memory locations. So if rr8 contains 2000 then the instruction

```
ldw (rr8),#3467
```

loads the memory locations 2000 and 2001 with the value 3467.

Figure 16-7. Memory Indirect



Indirect With Post-increment Adressing Mode

The indirect with post-increment addressing mode is similar to the memory indirect addressing mode but, in addition, after accessing the data in the currently pointed address, the value in the pointing working register pair is incremented. This mode is indicated by a plus sign following a working register pair in parentheses, e.g. `(rr4)+`.

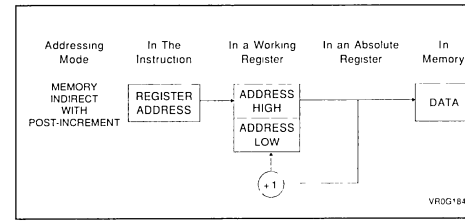
Example:
If the working register pair rr4 (working registers r4 and r5) contains the value 3000 and memory location 3000 contains the value 88, then the instruction

```
ld R50,(rr4)+
```

loads register 50 with the value 88 and then the value in rr4 to be incremented to 3001.

This mode uses only working registers to contain the address. Thus the Indirect with Post-Increment addressing mode is most useful in repeated situations when a number of adjacent items of data are required in succession. The use of this addressing mode saves both time and program memory space since it cuts the usual increment instruction.

Figure 16-8. Memory Indirect Post-Increment



Indirect With Pre-decrement Addressing Mode

This indirect memory addressing mode has an automatic pre-decrement. The address can only be contained in working registers and the mode is indicated by a minus sign in front of the working registers which are in parentheses, e.g. `-(rr6)`.

Thus if the working register pair rr6 contains the value 1111 and location 1110 contains the value 40 then the instruction

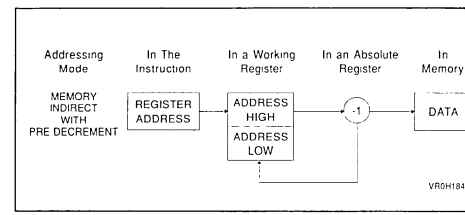
```
ld R56,-(rr6)
```

decrements the value in rr6 to 1110 and then loads the value 40 into register 56.

This addressing mode allows the ST9 to deal in the reverse order with data previously managed using the indirect post-increment mode without resetting the pointing registers (of the last post-increment).

The pre-decrement mode has the same benefits of time and program memory saving as the post-increment mode.

Figure 16-9. Memory Indirect Pre-Decrement



ADDRESSING MODES (Continued)

Indexed Addressing Modes

There are three indexed addressing modes, each using an indirect address plus offset format. The index address is given as an indirect address contained in a working register pair, while the offset can be long or short (a word or a byte). The address of the data required is given by the value of the working register pair indicated (the index), plus the value of the given offset. The specification of this offset which differentiates the three modes, is as follows:

- Indexed with an Immediate Short and Long Offset

In these indexed modes the offset is a fixed and Immediate value included in the instruction. It may be either a short or long index as required, this immediate value being added to the address given by the working register pair.

Example: if the working register pair, rr6, contains the value 8000 and memory location 8034 contains the value 254 then the instruction

```
ld R55,34(rr6)
```

loads the value 254 into register 55.

Or, as another example, if the working register pair rr2 contains the value 2000 and register 78 contains the value 34 then the instruction.

```
ld 322(rr2),r78
```

loaded the value 34 into memory location 2322.

- Indexed with a Register Offset

In this addressing mode, the index is supplied by one pair of working registers and the offset is supplied by a second pair of working registers. The format is rrx(rry), x and y being in the range 0,2,4...12,14.

Example
If working register pair rr0 contains the value 2222 and working register pair rr4 contains 3333 while register 45 contains the value 78 then the instruction

```
ld rr4(rr0),R45
```

loads the value 78 into memory location 5555.

Figure 16-10. Memory Indexed with Immediate Short Offset

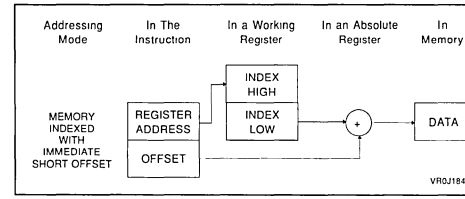


Figure 16-11. Memory Indexed with Immediate Long Offset

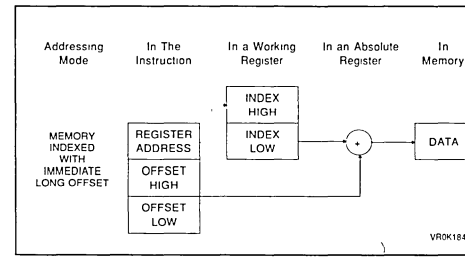
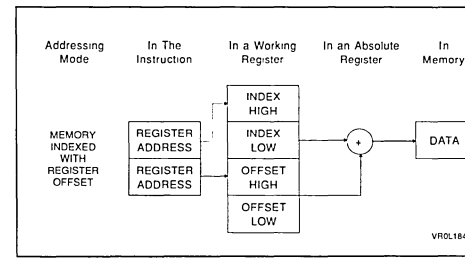


Figure 16-12. Memory Indexed with Register Offset



Indirect Memory Bit Addressing Mode

In the indirect memory bit addressing mode, any bit of Program/Data memory location can be addressed with the `btset` (Bit Test and SET) instruction.

Example

```
btset (rr8).3
```

This instruction sets bit 3 of the memory location addressed by the working registers r8, r9 contents.

16.2 INSTRUCTION SET

The ST9 instruction set consists of 87 instruction types which can be divided into eight groups:

- Load (two operands)
- Arithmetic & logic (two operands)
- Arithmetic Logic and Shift (one operand)
- Stack (one operand)
- Multiply & Divide (two operands)
- Boolean (one or two operands)
- Program Control (zero to three operands)
- Miscellaneous (zero to two operands)

The wide range of instructions eases use of the register file and address spaces, reducing operation times, while the register pointers mechanism allows an unmatched code efficiency and ultrafast context switching. A particularly notable feature is the comprehensive "Any Bit, Any Register" (ABAR) addressing capability of the Boolean instructions.

The ST9 can operate with a wide range of data lengths from single bits, 4-bit nibbles which can be in the form of Binary Coded Decimal (BCD) digits, 8-bit bytes, and 16-bit words.

The following summary shows the instructions belonging to each group and the number of operands required for each instruction. The source operand is “src”, “dst” is the destination operand, and “cc” is a condition code.

INSTRUCTION SET (Continued)

Load Instructions (Two Operands)

Mnemonic	Operands	Instruction
LD LDW	dst,src dst,src	Load Load Word
LDPP LDPD LDDP LDDD	dst,src dst,src dst,src dst,src	Load Program Memory -> Program Memory Load Data Memory -> Program Memory Load Program Memory -> Data Memory Load Data Memory -> Data Memory

Arithmetic and Logic Instructions (Two Operands)

Mnemonic	Operands	Instruction
ADD ADDW	dst,src dst,src	Add Add Word
ADC ADCW	dst,src dst,src	Add With carry Add Word With Carry
SUB SUBW	dst,src dst,src	Substract Substract Word
SBC SBCW	dst,src dst,src	Substract With Carry Substract Word With Carry
AND ANDW	dst,src dst,src	Logical AND Logical Word AND
OR ORW	dst,src dst,src	Logical OR Logical Word OR
XOR XORW	dst,src dst,src	Logical Exclusive OR Logical Word Exclusive OR
CP CPW	dst,src dst,src	Compare Compare Word
TM TMW	dst,src dst,src	Test Under Mask Test Word Under Mask
TCM TCMW	dst,src dst,src	Test Complement Under Mask Test Word Complement Under Mask

INSTRUCTION SET (Continued)

Arithmetic Logic and Shift Instructions (One Operand)

Mnemonic	Operands	Instruction
INC INCW	dst dst	Increment Increment Word
DEC DECW	dst dst	Decrement Decrement Word
SLA SLAW	dst dst	Shift Left Arithmetic Shift Word Left Arithmetic
SRA SRAW	dst dst	Shift Right Arithmetic Shift Word Right Arithmetic
RRC RRCW	dst dst	Rotate Right Through Carry Rotate Word Right Through Carry
RLC RLCW	dst dst	Rotate Left Through Carry Rotate Word Left Through Carry
ROR	dst	Rotate Right
ROL	dst	Rotate Left
CLR	dst	Clear Register
CPL	dst	Complement Register
SWAP	dst	Swap Nibbles
DA	dst	Decimal Adjust

Stack Instructions (One Operand)

Mnemonic	Operands	Instruction
PUSH PUSHW PEA	src src src	Push on System Stack Push Word on System Stack Push Effective Address on System Stack
POP POPW	dst dst	Pop From System Stack Pop Word from System Stack
PUSHU PUSHUW PEAU	src src src	Push on User Stack Push Word on User Stack Push Effective Address on User Stack
POPU POPUW	dst dst	Pop From User Stack Pop Word From User Stack

INSTRUCTION SET (Continued)

Multiply and Divide Instructions (Two Operands)

Mnemonic	Operands	Instruction
MUL	dst,src	Multiply 8x8
DIV DIVWS	dst,src dst,src	Divide 16/8 Divide Word Stepped 32/16

Boolean Instructions (One or Two Operands)

Mnemonic	Operands	Instruction
BSET	dst	Bit Set
BRES	dst	Bit Reset
BCPL	dst	Bit Complement
BTSET	dst	Bit Test and Set
BLD	dst,src	Bit Load
BAND	dst,src	Bit AND
BOR	dst,src	Bit OR
BXOR	dst,src	Bit XOR

INSTRUCTION SET (Continued)

Program Control Instructions (One, Two or Three Operands)

Mnemonic	Operands	Instruction
RET		Return from Subroutine
IRET		Return from Interrupt
WFI		Stop Program Execution and Wait for the next Enabled Interrupt. If a DMA request is present, the CPU executes the DMA service routine and then automatically returns to the WFI
HALT		Stop Program Execution Until Next System Reset
JR	cc,dst	Jump Relative If Condition is Met
JP	cc,dst	Jump if Condition is Met
JP	dst	Unconditional Jump
CALL	dst	Unconditional Call
BTJF	dst,N	Bit Test and Jump if False
BTJT	dst,N	Bit Test and Jump if True
DJNZ	dst,N	Decrement a Working Register and Jump if Non Zero
DWJNZ	dst,N	Decrement a Register Pair and Jump if Non Zero
CPJFI	dst,N	Compare and Jump on False. Otherwise Post Increment
CPJTI	dst,N	Compare and Jump on True. Otherwise Post Increment

INSTRUCTION SET (Continued)

Miscellaneous (None, One or Two Operands)

Mnemonic	Operands	Instruction
XCH	dst,src	Exchange Registers
SRP	src	Set Register Pointer Long (16 working registers)
SRP0	src	Set Register Pointer 0 (8 LSB working register)
SRP1	src	Set Register Pointer 1 (8 MSB working register)
SPP	src	Set Page Pointer
EXT	dst	Sign Extend
EI		Enable Interrupts
DI		Disable Interrupts
SCF		Set Carry Flag
RCF		Reset Carry Flag
CCF		Complement Carry Flag
SPM		Select Program Memory
SDM		Select Data Memory
NOP		No Operation

INSTRUCTION SET (Continued)

16.2.1 ST9 Processor Flags

An important feature of a single chip microcomputer is the ability to test data and make the appropriate action based on the results. In order to provide this facility, FLAGR (register 231) in the register file is used as a flag register. Six bits of this register are used as the following flags:

C - Carry

Z - Zero

S - Sign

V - Overflow

D - Decimal Adjust

H - Half Carry

Bit 1 is available to the user. Bit 0 is the Program/Data Memory selector bit.

The Flag Register is further described in the Architecture Chapter.

16.2.2 Condition Codes

Flags C, Z, S, and OV control the operation of the "conditional" Jump instructions. The next table shows the condition codes and the flag settings.

Note : Some of the Status flags are used to indicate more than one condition e.g . Zero and Equal. In such cases the condition code is the same for both conditions.

Table 16-2. Condition Codes Table

Mnemonic code	Meaning	Flag setting	Hex. value	Binary value
F	Always False	____	0	0000
T	Always True	____	8	1000
C	Carry	C=1	7	0111
NC	Not carry	C=0	F	1111
Z	Zero	Z=1	6	0011
NZ	Not Zero	Z=0	E	1110
PL	Plus	S=0	D	1101
MI	Minus	S=1	5	0101
OV	Overflow	V=1	4	0100
NOV	No Overflow	V=0	C	1100
EQ	Equal	Z=1	6	0110
NE	Not Equal	Z=0	E	1110
GE	Greater Than or Equal	(S xor V)=0	9	1001
LT	Less Than	(S xor V)=1	1	0001
GT	Greater Than	(Z or(S xor V))=0	A	1010
LE	Less Than or Equal	(Z or(S xor V))=1	2	0010
UG	Unsigned Greater Than or Equal	C=0	F	1111
UL	Unsigned Less Than	C=1	7	0111
UGT	Unsigned Greater Than	(C=0 and Z=0)=1	B	1011
ULE	Unsigned Less Than or Equal	(C or Z)=1	3	0011

INSTRUCTION SET (Continued)

16.2.3 Notation

Operands and status flags are represented by a notational shorthand in the detailed instruction description (see programming manual).

The notation for operands (condition codes and address modes) and the actual operands they represent are as follows:

Table 16-3. Notation (Part 1)

Notation	Significance	Actual Operand/Range	
cc	Condition Code		
#N #NN	Immediate Byte Immediate Word	# data # data	where data is a byte expression where data is a word expression
r	Direct Working Register	rn	where n=0-15
R	Direct Register	Rn	where n=0-255
rr	Direct Working Register Pair	rrn	where n is an even number in the range 0-15. (n=0,2,4,6....14)
RR	Direct Register Pair	RRn	where n is an even number in the range 0-254. (n=0,2,4,6....254)
(r)	Indirect Working Register	(rn)	where n=0-15
(R)	Indirect register	(Rn)	where n=0-255
(r)+	Indirect working register post increment	(rn)+	where n=0-15
N(rx)	Indexed register	N(rx)	where x=0-15; N=0-255 (one byte)
N	Memory relative Short Address		Program label or expression in the range +127/-128 starting from the address of the next instruction
NN	Direct Memory Long Address		Program label or expression in the range 0-65535 in memory area
(rr)	Indirect Pair of Working Register Pointers	(rrn)	Where n is an even number in the range 0-15.(n=0,2,4,6....14)
(rr)+	Indirect Pair of Working Register Pointers with Post Increment	(rrn)+	where n is an even number in the range 0-15.(n=0,2,4,6....14)
-(rr)	Indirect Pair of Working Register Pointers with Pre Decrement	-(rrn)	where n is an even number in the range 0-15.(n=0,2,4,6....14)

INSTRUCTION SET (Continued)

Table 16-4. Notation (Part 2)

Notation	Significance	Actual Operand/Range	
N(rrx)	Indexed Pair of Working Register Pointers with Short Offset	N(rrx)	where x is an even number in the range 0-15.(n=0,2,4,6....14) and N is a signed one byte expression between +127/-128
NN(rrx)	Indexed Pair of Working Register Pointers with Long Offset	NN(rrx)	where x is an even number in the range 0-15.(n=0,2,4,6....14) and NN is word expression in the range between 0 and 65535
N(RRx)	Indexed Pair of Register Pointers with Short Offset	N(RRx)	where x is an even number in the range 0-255.(n=0,2,4,6...254) and N is a one byte signed expression in the range +127/-128
NN(RRx)	Indexed Pair of Register Pointers with Long Offset	NN(RRx)	where x is an even number in the range 0-255.(n=0,2,4,6....14) and NN is word expression in the range between 0 and 65535
rr(rrx)	Indexed Pair of Working Registers with a Pair of Working Registers used as Offset	rrn(rrx)	where n and x are two even numbers in the range 0-15. (n,x=0,2,4,6....14)
r.b	Bit pointer in a direct working register	rn.b	n=0.15 and b is a number between 0-7;0 LSB 7 MSB
(rr).b	Bit pointer in a Memory Location using a Pair of Indirect Working Registers as Address Pointer	(rrn).b	where n is an even number in the range 0-15.(n=0,2,4,6....14) and be is a number between 0-7 0 LSB 7 MSB
(RR)	Indirect pair of Register Pointer	(RRn)	where n is an even number in the range 0-255.(n=0,2,4,6....254)

16.3 INSTRUCTION SUMMARY

The following tables summarize the operation for each of the instructions which are listed with their corresponding mnemonic codes, addressing modes, byte counts, timing information, and affected flags.

GENERAL NOTES:

FLAGS STATUS:

- ^ : affected
- - : not affected
- 0 : reset to zero
- 1 : set to one
- ? : undefined

Note: for detailed information on the instruction set refer to the ST9 programming manual.

- dst: destination operand
- src: source operand
- SSP: system stack pointer
- USP: user stack pointer
- PC: program counter
- cc: condition code
- C: carry flag
- Z: zero flag
- S: sign flag
- V: overflow flag
- D: decimal adjust flag
- CIC: central interrupt control register
- DP : data/program memory flag

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADC : Addition of 2 bytes with carry						
ADC	r	r	2	6	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	R	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	R	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	r	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	(r)	2	6	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	(r)	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	(rr)	3	12	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	(rr)	3	12	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	NN	4	18	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	N(rrx)	4	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	N(rrx)	4	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	NN(rrx)	5	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	NN(rrx)	5	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	rr(rrx)	3	22	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	(rr)+	3	16	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	R	(rr)+	3	16	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	r	-(rr)	3	16	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	-(rr)	3	16	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(r)	r	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(r)	R	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	r	3	18	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	R	3	18	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)+	r	3	22	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	(rr)+	R	3	22	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	NN	r	4	20	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	N(rrx)	r	4	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	N(rrx)	R	4	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	NN(rrx)	r	5	28	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	NN(rrx)	R	5	28	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	rr(rrx)	r	3	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	-(rr)	r	3	24	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	-(rr)	R	3	22	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	#N	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	#N	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	#N	3	16	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	NN	#N	5	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	(rr)	3	20	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(RR)	(rr)	3	20	dst<-dst+src+C	^ ^ ^ ^ 0 ^

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADCW : Add word with carry						
ADCW	rr	rr	2	10	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	RR	3	12	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	RR	3	12	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	rr	3	12	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	(r)	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	(r)	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	(rr)	2	16	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	(rr)	3	18	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	NN	4	22	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	N(rrx)	4	28	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	N(rrx)	4	28	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	NN(rrx)	5	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	NN(rrx)	5	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	rr(rrx)	3	26	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	(rr)+	3	22	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	RR	(rr)+	3	22	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(r)	rr	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(r)	RR	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	rr	2	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	RR	3	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)+	rr	3	32	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	(rr)+	RR	3	32	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	NN	rr	4	32	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	N(rrx)	rr	4	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	N(rrx)	RR	4	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN(rrx)	rr	5	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN(rrx)	RR	5	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr(rrx)	rr	3	34	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	-(rr)	rr	3	34	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	#NN	4	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	#NN	4	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	#NN	4	32	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN	#NN	6	36	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	N(rrx)	#NN	5	36	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN(rrx)	#NN	6	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	(rr)	2	32	dst<-dst+src+C	^ ^ ^ ^ ? ?

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADD : Addition of 2 bytes without carry						
ADD	r	r	2	6	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	R	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	R	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	r	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	(r)	2	6	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	(r)	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	(rr)	3	12	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	(rr)	3	12	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	NN	4	18	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	N(rrx)	4	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	N(rrx)	4	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	NN(rrx)	5	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	NN(rrx)	5	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	rr(rrx)	3	22	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	(rr)+	3	16	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	R	(rr)+	3	16	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	r	-(rr)	3	16	rr<-rr-1 dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	-(rr)	3	16	rr<-rr-1 dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(r)	r	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(r)	R	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	r	3	18	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	R	3	18	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)+	r	3	22	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	(rr)+	R	3	22	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	NN	r	4	20	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	N(rrx)	r	4	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	N(rrx)	R	4	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	NN(rrx)	r	5	28	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	NN(rrx)	R	5	28	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	rr(rrx)	r	3	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	-(rr)	r	3	22	rr<-rr-1 dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	-(rr)	R	3	22	rr<-rr-1 dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	#N	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	#N	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	#N	3	16	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	NN	#N	5	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	(rr)	3	20	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(RR)	(rr)	3	20	dst<-dst+src	^ ^ ^ ^ 0 ^

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADDW : Add word without carry						
ADDW	rr	rr	2	10	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	RR	3	12	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	RR	3	12	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	rr	3	12	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	(r)	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	(r)	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	(rr)	2	16	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	(rr)	3	18	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	NN	4	22	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	N(rrx)	4	28	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	N(rrx)	4	28	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	NN(rrx)	5	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	NN(rrx)	5	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	rr(rrx)	3	26	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	(rr)+	3	22	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	RR	(rr)+	3	22	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(r)	rr	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(r)	RR	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	rr	2	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	RR	3	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)+	rr	3	32	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	(rr)+	RR	3	32	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	NN	rr	4	32	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	N(rrx)	rr	4	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	N(rrx)	RR	4	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN(rrx)	rr	5	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN(rrx)	RR	5	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr(rrx)	rr	3	34	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	#NN	4	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	#NN	4	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	#NN	4	32	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN	#NN	6	36	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	N(rrx)	#NN	5	36	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN(rrx)	#NN	6	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	(rr)	2	32	dst<-dst+src	^ ^ ^ ^ ? ?

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
AND : Logical AND between 2 bytes						
AND	r	r	2	6	dst<-dst AND src	- ^ ^ 0 - -
AND	R	R	3	10	dst<-dst AND src	- ^ ^ 0 - -
AND	r	R	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	R	r	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	r	(r)	2	6	dst<-ds AND src	- ^ ^ 0 - -
AND	R	(r)	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	r	(rr)	3	12	dst<-ds AND src	- ^ ^ 0 - -
AND	R	(rr)	3	12	dst<-ds AND src	- ^ ^ 0 - -
AND	r	NN	4	18	dst<-ds AND src	- ^ ^ 0 - -
AND	r	N(rrx)	4	24	dst<-ds AND src	- ^ ^ 0 - -
AND	R	N(rrx)	4	24	dst<-ds AND src	- ^ ^ 0 - -
AND	r	NN(rrx)	5	26	dst<-ds AND src	- ^ ^ 0 - -
AND	R	NN(rrx)	5	26	dst<-ds AND src	- ^ ^ 0 - -
AND	r	rr(rrx)	3	22	dst<-ds AND src	- ^ ^ 0 - -
AND	r	(rr)+	3	16	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	R	(rr)+	3	16	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	r	-(rr)	3	16	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	R	-(rr)	3	16	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	(r)	r	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	(r)	R	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	r	3	18	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	R	3	18	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)+	r	3	22	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	(rr)+	R	3	22	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	NN	r	4	20	dst<-ds AND src	- ^ ^ 0 - -
AND	N(rrx)	r	4	26	dst<-ds AND src	- ^ ^ 0 - -
AND	N(rrx)	R	4	26	dst<-ds AND src	- ^ ^ 0 - -
AND	NN(rrx)	r	5	28	dst<-ds AND src	- ^ ^ 0 - -
AND	NN(rrx)	R	5	28	dst<-ds AND src	- ^ ^ 0 - -
AND	rr(rrx)	r	3	24	dst<-ds AND src	- ^ ^ 0 - -
AND	-(rr)	r	3	22	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	-(rr)	R	3	22	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	r	#N	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	R	#N	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	#N	3	16	dst<-ds AND src	- ^ ^ 0 - -
AND	NN	#N	5	24	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	(rr)	3	20	dst<-ds AND src	- ^ ^ 0 - -
AND	(RR)	(rr)	3	20	dst<-ds AND src	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ANDW : Logical AND between two words						
ANDW	rr	rr	2	10	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	RR	3	12	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	RR	3	12	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	rr	3	12	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	(r)	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	(r)	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	(rr)	2	16	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	(rr)	3	18	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	NN	4	22	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	N(rrx)	4	28	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	N(rrx)	4	28	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	NN(rrx)	5	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	NN(rrx)	5	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	rr(rrx)	3	26	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	(rr)+	3	22	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	RR	(rr)+	3	22	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	(r)	rr	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(r)	RR	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	rr	2	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	RR	3	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)+	rr	3	32	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	(rr)+	RR	3	32	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	NN	rr	4	32	dst<-dst AND src	- ^ ^ 0 - -
ANDW	N(rrx)	rr	4	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	N(rrx)	RR	4	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN(rrx)	rr	5	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN(rrx)	RR	5	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr(rrx)	rr	3	34	dst<-dst AND src	- ^ ^ 0 - -
ANDW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	#NN	4	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	#NN	4	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	#NN	4	32	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN	#NN	6	36	dst<-dst AND src	- ^ ^ 0 - -
ANDW	N(rrx)	#NN	5	36	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN(rrx)	#NN	6	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	(rr)	2	32	dst<-dst AND src	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
BAND : Bit AND						
BAND	r.b	r.b	3	14	dst bit<-dst bit AND src bit	- - - - - -
BAND	r.b	r.!b	3	14	dst bit<-dst bit AND complemented src bit	- - - - - -
BCPL : Bit Complement						
BCPL	r.b		2	6	dst bit<-dst bit complemented	- - - - - -
BLD : Bit Load						
BLD	r.b	r.b	3	14	dst bit<-src bit	- - - - - -
BLD	r.b	r.!b	3	14	dst bit<-src bit complemented	- - - - - -
BOR : Bit OR						
BOR	r.b	r.b	3	14	dst bit<-dst bit OR src bit	- - - - - -
BOR	r.b	r.!b	3	14	dst bit<-dst bit OR complemented src bit	- - - - - -
BRES : Bit Reset						
BRES	r.b		2	6	dst bit<- 0	- - - - - -
BSET : Bit Set						
BSET	r.b		2	6	dst bit<- 1	- - - - - -
BTJF, BTJT : Bit test and jump						
BTJF	r.b	N	3	14/16	If test bit is 0, PC<-PC+N	- - - - - -
BTJT	r.b	N	3	14/16	If test bit is 1, PC<-PC+N	- - - - - -
BXOR : Bit Exclusive OR						
BXOR	r.b	r.b	3	14	dst bit<-dst bit XOR src bit	- - - - - -
BXOR	r.b	r.!b	3	14	dst bit<-dst bit XOR complemented src bit	- - - - - -
BTSET : Bit Test and Set						
BTSET	r.b		2	8	If test bit = 0, test bit <-1,Z<-1	- ^ ^ 0 - -
BTSET	(rr}.b		2	20	If test bit = 0, test bit <-1,Z<-1	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
CALL : Call a subroutine						
CALL	NN		3	18	SSP<-SSP-2,(SP)< PC, PC<-dst	- - - - - -
CALL	(rr)		2	16	" "	- - - - - -
CALL	(RR)		2	16	" "	- - - - - -
CCF : Complement Carry Flag						
CCF			1	6	C <- C complemented	- - - - - -
CLR : Clear register						
CLR	r		2	6	dst<-0	- - - - - -
CLR	R		2	6	dst<-0	- - - - - -
CLR	(r)		2	6	dst<-0	- - - - - -
CLR	(R)		2	6	dst<-0	- - - - - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
CP : Compare bytes						
CP	r	r	2	6	dst-src	^ ^ ^ ^ - -
CP	R	R	3	10	dst-src	^ ^ ^ ^ - -
CP	r	R	3	10	dst-src	^ ^ ^ ^ - -
CP	R	r	3	10	dst-src	^ ^ ^ ^ - -
CP	r	(r)	2	6	dst-src	^ ^ ^ ^ - -
CP	R	(r)	3	10	dst-src	^ ^ ^ ^ - -
CP	r	(rr)	3	12	dst-src	^ ^ ^ ^ - -
CP	R	(rr)	3	12	dst-src	^ ^ ^ ^ - -
CP	r	NN	4	18	dst-src	^ ^ ^ ^ - -
CP	r	N(rrx)	4	24	dst-src	^ ^ ^ ^ - -
CP	R	N(rrx)	4	24	dst-src	^ ^ ^ ^ - -
CP	r	NN(rrx)	5	26	dst-src	^ ^ ^ ^ - -
CP	R	NN(rrx)	5	26	dst-src	^ ^ ^ ^ - -
CP	r	rr(rrx)	3	22	dst-src	^ ^ ^ ^ - -
CP	r	(rr)+	3	16	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	R	(rr)+	3	16	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	r	-(rr)	3	16	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	R	-(rr)	3	16	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	(r)	r	3	10	dst-src	^ ^ ^ ^ - -
CP	(r)	R	3	10	dst-src	^ ^ ^ ^ - -
CP	(rr)	r	3	18	dst-src	^ ^ ^ ^ - -
CP	(rr)	R	3	18	dst-src	^ ^ ^ ^ - -
CP	(rr)+	r	3	22	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	(rr)+	R	3	22	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	NN	r	4	20	dst-src	^ ^ ^ ^ - -
CP	N(rrx)	r	4	26	dst-src	^ ^ ^ ^ - -
CP	N(rrx)	R	4	26	dst-src	^ ^ ^ ^ - -
CP	NN(rrx)	r	5	28	dst-src	^ ^ ^ ^ - -
CP	NN(rrx)	R	5	28	dst-src	^ ^ ^ ^ - -
CP	rr(rrx)	r	3	24	dst-src	^ ^ ^ ^ - -
CP	-(rr)	r	3	22	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	-(rr)	R	3	22	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	r	#N	3	10	dst-src	^ ^ ^ ^ - -
CP	R	#N	3	10	dst-src	^ ^ ^ ^ - -
CP	(rr)	#N	3	16	dst-src	^ ^ ^ ^ - -
CP	NN	#N	5	22	dst-src	^ ^ ^ ^ - -
CP	(rr)	(rr)	3	18	dst-src	^ ^ ^ ^ - -
CP	(RR)	(rr)	3	18	dst-src	^ ^ ^ ^ - -
CPL : Complement register						
CPL	r		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPL	R		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPL	(r)		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPL	(R)		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPJFI, CPJTI : Compare with post-increment						
CPJFI	(rr)	r,N	3	22/24	If compare not verified jump otherwise post-increment	- - - - - -
CPJTI	(rr)	r,N	3	22/24	If compare verified jump otherwise post-increment	- - - - - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
CPW : Compare word						
CPW	rr	rr	2	10	dst-src	^ ^ ^ ^ - -
CPW	RR	RR	3	12	dst-src	^ ^ ^ ^ - -
CPW	rr	RR	3	12	dst-src	^ ^ ^ ^ - -
CPW	RR	rr	3	12	dst-src	^ ^ ^ ^ - -
CPW	rr	(r)	3	14	dst-src	^ ^ ^ ^ - -
CPW	RR	(r)	3	14	dst-src	^ ^ ^ ^ - -
CPW	rr	(rr)	2	16	dst-src	^ ^ ^ ^ - -
CPW	RR	(rr)	3	18	dst-src	^ ^ ^ ^ - -
CPW	rr	NN	4	22	dst-src	^ ^ ^ ^ - -
CPW	rr	N(rrx)	4	28	dst-src	^ ^ ^ ^ - -
CPW	RR	N(rrx)	4	28	dst-src	^ ^ ^ ^ - -
CPW	rr	NN(rrx)	5	30	dst-src	^ ^ ^ ^ - -
CPW	RR	NN(rrx)	5	30	dst-src	^ ^ ^ ^ - -
CPW	rr	rr(rrx)	3	26	dst-src	^ ^ ^ ^ - -
CPW	rr	(rr)+	3	22	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	RR	(rr)+	3	22	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	rr	-(rr)	3	24	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	RR	-(rr)	3	24	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	(r)	rr	3	14	dst-src	^ ^ ^ ^ - -
CPW	(r)	RR	3	14	dst-src	^ ^ ^ ^ - -
CPW	(rr)	rr	2	26	dst-src	^ ^ ^ ^ - -
CPW	(rr)	RR	3	28	dst-src	^ ^ ^ ^ - -
CPW	(rr)+	rr	3	30	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	(rr)+	RR	3	30	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	NN	rr	4	30	dst-src	^ ^ ^ ^ - -
CPW	N(rrx)	rr	4	36	dst-src	^ ^ ^ ^ - -
CPW	N(rrx)	RR	4	36	dst-src	^ ^ ^ ^ - -
CPW	NN(rrx)	rr	5	36	dst-src	^ ^ ^ ^ - -
CPW	NN(rrx)	RR	5	36	dst-src	^ ^ ^ ^ - -
CPW	rr(rrx)	rr	3	32	dst-src	^ ^ ^ ^ - -
CPW	-(rr)	rr	3	30	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	-(rr)	RR	3	30	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	rr	#NN	4	14	dst-src	^ ^ ^ ^ - -
CPW	RR	#NN	4	14	dst-src	^ ^ ^ ^ - -
CPW	(rr)	#NN	4	30	dst-src	^ ^ ^ ^ - -
CPW	NN	#NN	6	34	dst-src	^ ^ ^ ^ - -
CPW	N(rrx)	#NN	5	34	dst-src	^ ^ ^ ^ - -
CPW	NN(rrx)	#NN	6	36	dst-src	^ ^ ^ ^ - -
CPW	(rr)	(rr)	2	32	dst-src	^ ^ ^ ^ - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
DA : Decimal adjust						
DA	r		2	6	dst<- DA dst	^ ^ ^ ? - -
DA	R		2	6	dst<- DA dst	^ ^ ^ ? - -
DA	(r)		2	6	dst<- DA dst	^ ^ ^ ? - -
DA	(R)		2	6	dst<- DA dst	^ ^ ^ ? - -
DEC : Decrement						
DEC	r		2	6	dst<- dst-1	- ^ ^ ^ - -
DEC	R		2	6	dst<- dst-1	- ^ ^ ^ - -
DEC	(r)		2	6	dst<- dst-1	- ^ ^ ^ - -
DEC	(R)		2	6	dst<- dst-1	- ^ ^ ^ - -
DECW : Decrement Word						
DECW	rr		2	8	dst<-dst-1	- ^ ^ ^ - -
DECW	RR		2	8	dst<-dst-1	- ^ ^ ^ - -
DI : Disable Interrupts						
DI			1	6	Bit 4 of the CIC Register is set to 0	- - - - - -
DIV : Divide 16 by 8						
DIV	rr	r	2	28/20	dst / src <- dst high=remainder 16/8 <- dst low=result	note 1
DIVWS : Divide Word Stepped 32 by 16						
DIVWS	rrhigh rrlow	rr	3	28	32/16	note 1
DJNZ : Decrement a working register and Jump if Non Zero						
DJNZ	r	N	2	10/12	r <- r-1, If r=0 then PC<-PC+N	note 2
DWJNZ : Decrement a register pair and Jump if Non Zero						
DWJNZ	rr	N	3	12/16	rr<-rr-1, If rr=0 then PC<-PC+N	note 2
DWJNZ	RR	N	3	12/16	RR<-RR-1,If RR=0 then PC<-PC+N	

Notes :

1. Refer to the ST9 Programming Manual for detailed information.
2. Working registers in groups D, E and F are not allowed.

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
EI : Enable Interrupts						
EI			1	6	Bit 4 of the CICR register is set to 1	- - - - - -
EXT : Sign extend						
EXT EXT	rr RR		2 2	10 10	r(7) --> r(n) n=8-15 R(7) --> R(n) n=8-15	- - - - - - - - - - - -
HALT						
HALT			2	6	Stops all internal clocks until next system reset if not in Watchdog Mode	- - - - - -
INC : Increment						
INC INC INC INC	r R (r) (R)		2 2 2 2	6 6 6 6	dst<- dst+1 dst<- dst+1 dst<- dst+1 dst<- dst+1	- ^ ^ ^ - - - ^ ^ ^ - - - ^ ^ ^ - - - ^ ^ ^ - -
INCW : Increment Word						
INCW INCW	rr RR		2 2	8 8	dst<-dst+1 dst<-dst+1	- ^ ^ ^ - - - ^ ^ ^ - -
IRET : Return from Interrupt Routine						
IRET			1	16	FLAGS<-(SSP),SSP<-SSP+1, PC<-(SSP), SSP<-SPP+2, CIC(4)<-1	note 1
JP : Jump to a Routine						
JP	NN		3	10	PC<-dst	- - - - - -
JP	(rr)		2	8	PC<-dst	- - - - - -
JP	(RR)		2	8	PC<-dst	- - - - - -
JPcc	NN		3	10	IF cc(condition code) is true, PC<-dst	- - - - - -
JRcc : Conditional Relative Jump to a Routine						
JRcc	N		2	10/12	IF cc(condition code)is true, PC<-PC+dst	- - - - - -

Note 1 : All flags are restored to original setting (before interrupt occured).

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
LD : Load byte instructions						
LD	r	r	2	6	dst<-src	- - - - - -
LD	R	R	3	10	dst<-src	- - - - - -
LD	r	R	2	6	dst<-src	- - - - - -
LD	R	r	2	6	dst<-src	- - - - - -
LD	r	(r)	2	6	dst<-src	- - - - - -
LD	R	(r)	3	10	dst<-src	- - - - - -
LD	r	(rr)	2	10	dst<-src	- - - - - -
LD	R	(rr)	3	12	dst<-src	- - - - - -
LD	r	NN	4	18	dst<-src	- - - - - -
LD	r	N(rx)	3	10	dst<-src	- - - - - -
LD	r	N(rrx)	4	24	dst<-src	- - - - - -
LD	R	N(rrx)	4	24	dst<-src	- - - - - -
LD	r	NN(rrx)	5	26	dst<-src	- - - - - -
LD	R	NN(rrx)	5	26	dst<-src	- - - - - -
LD	r	rr(rrx)	3	22	dst<-src	- - - - - -
LD	r	(rr)+	3	16	dst<-src,rr<-rr+1	- - - - - -
LD	R	(rr)+	3	16	dst<-src,rr<-rr+1	- - - - - -
LD	r	-(rr)	3	16	rr<-rr-1,dst<-src	- - - - - -
LD	R	-(rr)	3	16	rr<-rr-1,dst<-src	- - - - - -
LD	(r)	r	2	6	dst<-src	- - - - - -
LD	(r)	R	3	10	dst<-src	- - - - - -
LD	(rr)	r	2	10	dst<-src	- - - - - -
LD	(rr)	R	3	14	dst<-src	- - - - - -
LD	(rr)+	r	3	18	dst<-src,rr<-rr+1	- - - - - -
LD	(rr)+	R	3	18	dst<-src,rr<-rr+1	- - - - - -
LD	NN	r	4	18	dst-src	- - - - - -
LD	N(rx)	r	3	10	dst-src	- - - - - -
LD	N(rrx)	r	4	24	dst-src	- - - - - -
LD	N(rrx)	R	4	24	dst-src	- - - - - -
LD	NN(rrx)	r	5	26	dst-src	- - - - - -
LD	NN(rrx)	R	5	26	dst-src	- - - - - -
LD	rr(rrx)	r	3	22	dst-src	- - - - - -
LD	-(rr)	r	3	18	rr<-rr-1,dst<-src	- - - - - -
LD	-(rr)	R	3	18	rr<-rr-1,dst<-src	- - - - - -
LD	r	#N	2	6	dst<-src	- - - - - -
LD	R	#N	3	10	dst<-src	- - - - - -
LD	(rr)	#N	3	12	dst<-src	- - - - - -
LD	NN	#N	5	20	dst<-src	- - - - - -
LD	(rr)	(rr)	3	16	dst<-src	- - - - - -
LD	(RR)	(rr)	3	16	dst<-src,	- - - - - -
LD	(r)+	(rr)+	2	14	rr<-rr+1,r<-r+1	- - - - - -
LD	(rr)+	(r)+	2	18	rr<-rr+1,r<-r+1	- - - - - -
LDPP,LDDP,LDPD, LDDD : Load from / to program / data memory						
LDPP	(rr)+	(rr)+	2	16	dst<-src (1),rr<-rr+1	- - - - - -
LDDP	(rr)+	(rr)+	2	16	dst<-src (2),rr<-rr+1	- - - - - -
LDPD	(rr)+	(rr)+	2	16	dst<-src (3),rr<-rr+1	- - - - - -
LDDD	(rr)+	(rr)+	2	16	dst<-src (4),rr<-rr+1	- - - - - -

Notes:

1 dst in Program Memory, src in Program Memory

2. dst in Data Memory, src in Program Memory

3 dst in Program Memory, src in Data Memory

4 dst in Data Memory, src in Data Memory

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
LDW : Load word instructions						
LDW	rr	rr	2	10	dst<-src	- - - - - -
LDW	RR	RR	3	10	dst<-src	- - - - - -
LDW	rr	RR	3	10	dst<-src	- - - - - -
LDW	RR	rr	3	10	dst<-src	- - - - - -
LDW	rr	(r)	3	10	dst<-src	- - - - - -
LDW	RR	(r)	3	10	dst<-src	- - - - - -
LDW	rr	(rr)	2	16	dst<-src	- - - - - -
LDW	RR	(rr)	3	18	dst<-src	- - - - - -
LDW	rr	NN	4	22	dst<-src	- - - - - -
LDW	rr	N(rx)	3	16	dst<-src	- - - - - -
LDW	rr	N(rrx)	4	28	dst<-ssc	- - - - - -
LDW	RR	N(rrx)	4	28	dst<-src	- - - - - -
LDW	rr	NN(rrx)	5	30	dst<-src	- - - - - -
LDW	RR	NN(rrx)	5	30	dst<-src	- - - - - -
LDW	rr	rr(rrx)	3	24	dst<-src	- - - - - -
LDW	rr	(rr)+	3	20	dst<-src,rr<-rr+2	- - - - - -
LDW	RR	(rr)+	3	20	dst<-src,rr<-rr+2	- - - - - -
LDW	rr	-(rr)	3	22	rr<-rr-2,dst<-src	- - - - - -
LDW	RR	-(rr)	3	22	rr<-rr-2,dst<-src	- - - - - -
LDW	(r)	rr	3	10	dst<-src	- - - - - -
LDW	(r)	RR	3	10	dst<-src	- - - - - -
LDW	(rr)	rr	2	18	dst<-src	- - - - - -
LDW	(rr)	RR	3	20	dst<-src	- - - - - -
LDW	(rr)+	rr	3	24	rr<-rr+2,dst<-src	- - - - - -
LDW	(rr)+	RR	3	24	rr<-rr+2,dst<-src	- - - - - -
LDW	NN	rr	4	22	dst<-src	- - - - - -
LDW	N(rx)	rr	3	14	dst<-src	- - - - - -
LDW	N(rrx)	RR	4	26	dst<-src	- - - - - -
LDW	N(rrx)	rr	4	26	dst<-src	- - - - - -
LDW	NN(rrx)	RR	5	28	dst<-src	- - - - - -
LDW	NN(rrx)	rr	5	28	dst<-src	- - - - - -
LDW	rr(rrx)	rr	3	24	dst<-src	- - - - - -
LDW	-(rr)	rr	3	26	rr<-rr-2,dst<-src	- - - - - -
LDW	-(rr)	RR	3	26	rr<-rr-2,dst<-src	- - - - - -
LDW	rr	#NN	4	12	dst<-src	- - - - - -
LDW	RR	#NN	4	12	dst<-src	- - - - - -
LDW	(rr)	#NN	4	22	dst<-src	- - - - - -
LDW	N(rrx)	#NN	5	28	dst<-src	- - - - - -
LDW	NN(rrx)	#NN	6	30	dst<-src	- - - - - -
LDW	NN	#NN	6	26	dst<-src	- - - - - -
LDW	(rr)	(rr)	2	22	dst<-src	- - - - - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
MUL : Multiply						
MUL	rr	r	2	22	dst <- dst x src, 8 x 8 multiply	note 1
NOP . No operation						
NOP			1	6	No Operation	- - - - - -
OR : Logical OR between 2 bytes						
OR	r	r	2	6	dst<-dst OR src	- ^ ^ 0 - -
OR	R	R	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	r	R	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	R	r	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	r	(r)	2	6	dst<-dst OR src	- ^ ^ 0 - -
OR	R	(r)	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	r	(rr)	3	12	dst<-dst OR src	- ^ ^ 0 - -
OR	R	(rr)	3	12	dst<-dst OR src	- ^ ^ 0 - -
OR	r	NN	4	18	dst<-dst OR src	- ^ ^ 0 - -
OR	r	N(rrx)	4	24	dst<-dst OR src	- ^ ^ 0 - -
OR	R	N(rrx)	4	24	dst<-dst OR src	- ^ ^ 0 - -
OR	r	NN(rrx)	5	26	dst<-dst OR src	- ^ ^ 0 - -
OR	R	NN(rrx)	5	26	dst<-dst OR src	- ^ ^ 0 - -
OR	r	rr(rrx)	3	22	dst<-dst OR src	- ^ ^ 0 - -
OR	r	(rr)+	3	16	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	R	(rr)+	3	16	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	r	-(rr)	3	16	rr<-rr-1 dst<-dst OR src	- ^ ^ 0 - -
OR	R	-(rr)	3	16	rr<-rr-1 dst<-dst OR src	- ^ ^ 0 - -
OR	(r)	r	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	(r)	R	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	r	3	18	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	R	3	18	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)+	r	3	22	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	(rr)+	R	3	22	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	NN	r	4	20	dst<-dst OR src	- ^ ^ 0 - -
OR	N(rrx)	r	4	26	dst<-dst OR src	- ^ ^ 0 - -
OR	N(rrx)	R	4	26	dst<-dst OR src	- ^ ^ 0 - -
OR	NN(rrx)	r	5	28	dst<-dst OR src	- ^ ^ 0 - -
OR	NN(rrx)	R	5	28	dst<-dst OR src	- ^ ^ 0 - -
OR	rr(rrx)	r	3	24	dst<-dst OR src	- ^ ^ 0 - -
OR	-(rr)	r	3	22	dst<-dst OR src rr<-rr-1	- ^ ^ 0 - -
OR	-(rr)	R	3	22	dst<-dst OR src rr<-rr-1	- ^ ^ 0 - -
OR	r	#N	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	R	#N	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	#N	3	16	dst<-dst OR src	- ^ ^ 0 - -
OR	NN	#N	5	24	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	(rr)	3	20	dst<-dst OR src	- ^ ^ 0 - -
OR	(RR)	(rr)	3	20	dst<-dst OR src	- ^ ^ 0 - -

Note 1. Refer to ST9 programming manual for detailed information

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ORW : Logical OR between two words						
ORW	rr	rr	2	10	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	RR	3	12	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	RR	3	12	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	rr	3	12	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	(r)	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	(r)	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	(rr)	2	16	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	(rr)	3	18	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	NN	4	22	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	N(rrx)	4	28	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	N(rrx)	4	28	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	NN(rrx)	5	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	NN(rrx)	5	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	rr(rrx)	3	26	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	(rr)+	3	22	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	RR	(rr)+	3	22	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	(r)	rr	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	(r)	RR	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	rr	2	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	RR	3	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)+	rr	3	32	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	(rr)+	RR	3	32	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	NN	rr	4	32	dst<-dst OR src	- ^ ^ 0 - -
ORW	N(rrx)	rr	4	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	N(rrx)	RR	4	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN(rrx)	rr	5	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN(rrx)	RR	5	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr(rrx)	rr	3	34	dst<-dst OR src	- ^ ^ 0 - -
ORW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	#NN	4	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	#NN	4	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	#NN	4	32	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN	#NN	6	36	dst<-dst OR src	- ^ ^ 0 - -
ORW	N(rrx)	#NN	5	36	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN(rrx)	#NN	6	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	(rr)	2	32	dst<-dst OR src	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
PEA : Push effective address on system stack						
PEA		N(rrx)	4	20	SSP<-USP-2, (SSP)<-rrx+N	- - - - - -
PEA		NN(rrx)	5	26	SSP<-USP-2, (SSP)<-rrx+N	- - - - - -
PEA		N(RRx)	4	20	SSP<-USP-2, (SSP)<-RRx+N	- - - - - -
PEA		NN(RRx)	5	26	SSP<-USP-2, (SSP)<-RRx+N	- - - - - -
PEAU : Push effective address on user stack						
PEAU		N(rrx)	4	20	USP<-USP-2, (USP)<-rrx+N	- - - - - -
PEAU		NN(rrx)	5	26	USP<-USP-2, (USP)<-rrx+N	- - - - - -
PEAU		N(RRx)	4	20	USP<-USP-2, (USP)<-RRx+N	- - - - - -
PEAU		NN(RRx)	5	26	USP<-USP-2, (USP)<-RRx+N	- - - - - -
POP : Pop system stack						
POP	r		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POP	R		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POP	(r)		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POP	(R)		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POPU : Pop user stack						
POPU	r		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPU	R		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPU	(r)		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPU	(R)		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPUW : Pop word from user stack						
POPUW	rr		2	14	dst<-(USP), USP<-USP+2	- - - - - -
POPUW	RR		2	14	dst<-(USP), USP<-USP+2	- - - - - -
POPW : Pop word from system stack						
POPW	rr		2	14	dst<-(SSP), SSP<-SSP+2	- - - - - -
POPW	RR		2	14	dst<-(SSP), SSP<-SSP+2	- - - - - -
PUSH : Push system stack						
PUSH		r	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		R	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		(r)	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		(R)	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		#N	3	16	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSHU : Push user stack						
PUSHU		r	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		R	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		(r)	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		(R)	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		#N	3	16	USP<-USP-1, (USP)<-src	- - - - - -
PUSHUW : Push word on user stack						
PUSHUW		rr	2	12	USP<-USP-2, (USP)<-src	- - - - - -
PUSHUW		RR	2	12	USP<-USP-2, (USP)<-src	- - - - - -
PUSHUW		#NN	4	20	USP<-USP-2, (USP)<-src	- - - - - -
PUSHW : Push Word on System Stack						
PUSHW		rr	2	12	SSP<-SSP-2, (SSP)<-src	- - - - - -
PUSHW		RR	2	12	SSP<-SSP-2, (SSP)<-src	- - - - - -
PUSHW		#NN	4	20	SSP<-SSP-2, (SSP)<-src	- - - - - -

INSTRUCTION SUMMARY (Continued)

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
RCF : Reset carry flag						
RCF			1	6	C <- 0	0 - - - - -
RET : Return from subroutine						
RET			1	12	PC <- (SSP), SSP <- SPP+2	- - - - - -
RLC : Rotate left through carry						
RLC	r		2	6	dst(0)<-C, C<-dst(7) dst(n+1)<-dst(n) n=0-6	^ ^ ^ ^ - -
RLC	R		2	6	" "	^ ^ ^ ^ - -
RLC	(r)		2	6	" "	^ ^ ^ ^ - -
RLC	(R)		2	6	" "	^ ^ ^ ^ - -
RLCW : Rotate word left through carry						
RLCW	rr		2	8	dst(0)<-C, C<-dst(15) dst(n+1)<-dst(n) n=0-14	
RLCW	RR		2	8	" "	
ROL : Rotate left						
ROL	r		2	6	C<-dst(7), dst(0)<-dst(7) dst(n+1)<-dst(n) n=0-6	^ ^ ^ ^ - -
ROL	R		2	6	" "	^ ^ ^ ^ - -
ROL	(r)		2	6	" "	^ ^ ^ ^ - -
ROL	(R)		2	6	" "	^ ^ ^ ^ - -
ROR : Rotate right						
ROR	r		2	6	C<-dst(0), dst(7)<-dst(0) dst(n)<-dst(n+1) n=0-6	^ ^ ^ ^ - -
ROR	R		2	6	" "	^ ^ ^ ^ - -
ROR	(r)		2	6	" "	^ ^ ^ ^ - -
ROR	(R)		2	6	" "	^ ^ ^ ^ - -
RRC : Rotate right through carry						
RRC	r		2	6	dst(7)<-C, C<-dst(0) dst(n)<-dst(n+1) n=0-6	^ ^ ^ ^ - -
RRC	R		2	6	" "	^ ^ ^ ^ - -
RRC	(r)		2	6	" "	^ ^ ^ ^ - -
RRC	(R)		2	6	" "	^ ^ ^ ^ - -
RRCW : Rotate word right through carry						
RRCW	rr		2	8	dst(15)<-C, C<-dst(0) dst(n)<-dst(n+1) n=0-14	^ ^ ^ ^ - -
RRCW	RR		2	8	" "	^ ^ ^ ^ - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SBC : Subtraction of 2 bytes with carry						
SBC	r	r	2	6	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	R	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	R	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	r	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	(r)	2	6	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	(r)	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	(rr)	3	12	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	(rr)	3	12	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	NN	4	18	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	N(rrx)	4	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	N(rrx)	4	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	NN(rrx)	5	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	NN(rrx)	5	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	rr(rrx)	3	22	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	(rr)+	3	16	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	R	(rr)+	3	16	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	r	-(rr)	3	16	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	-(rr)	3	16	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(r)	r	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(r)	R	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	r	3	18	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	R	3	18	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)+	r	3	22	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	(rr)+	R	3	22	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	NN	r	4	20	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	N(rrx)	r	4	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	N(rrx)	R	4	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	NN(rrx)	r	5	28	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	NN(rrx)	R	5	28	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	rr(rrx)	r	3	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	-(rr)	r	3	22	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	-(rr)	R	3	22	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	#N	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	#N	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	#N	3	16	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	NN	#N	5	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	(rr)	3	20	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(RR)	(rr)	3	20	dst<-dst-src-C	^ ^ ^ ^ 1 ^

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SBCW : Subtract word with carry						
SBCW	rr	rr	2	10	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	RR	3	12	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	RR	3	12	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	rr	3	12	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	(r)	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	(r)	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	(rr)	2	16	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	(rr)	3	18	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	NN	4	22	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	N(rrx)	4	28	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	N(rrx)	4	28	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	NN(rrx)	5	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	NN(rrx)	5	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	rr(rrx)	3	26	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	(rr)+	3	22	dst<-dst-src-C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	RR	(rr)+	3	22	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(r)	rr	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(r)	RR	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	rr	2	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	RR	3	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)+	rr	3	32	dst<-dst-src-C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	(rr)+	RR	3	32	dst<-dst-src-C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	NN	rr	4	32	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	N(rrx)	rr	4	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	N(rrx)	RR	4	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN(rrx)	rr	5	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN(rrx)	RR	5	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr(rrx)	rr	3	34	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	#NN	4	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	#NN	4	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	#NN	4	32	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN	#NN	6	36	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	N(rrx)	#NN	5	36	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN(rrx)	#NN	6	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	(rr)	2	32	dst<-dst-src-C	^ ^ ^ ^ ? ?

INSTRUCTION SUMMARY (Continued)

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SCF : Set carry flag						
SCF			1	6	C <- 1	1 - - - - -
SDM : Set data memory						
SDM			1	6	Set Data Memory DP<-1 Note 1	- - - - - -
SLA : Shift left arithmetic						
SLA	r R (rr)		2 3 3	6 10 20	dst C<-dst(7), dst (0)<-0 dst(n+1)<-dst(n)n=0-6 " " " "	^ ^ ^ ^ 0 - ^ ^ ^ ^ 0 - ^ ^ ^ ^ 0 -
SLAW : Shift word left arithmetic						
SLAW	rr RR (rr)		2 3 2	10 12 32	C<-dst(15), dst (0)<-0 dst(n+1)<-dst(n)n=1-14 " " " "	^ ^ ^ ^ - - ^ ^ ^ ^ - - ^ ^ ^ ^ - -
SPM : Set program memory						
SPM			1	6	Set Program Memory DP<-0 Note 2	- - - - - -
SPP : Set page pointer						
SPP		#N	2	6	Set Page Pointer	- - - - - -
SRA : Shift right arithmetic						
SRA SRA SRA SRA	r R (r) (R)		2 2 2 2	6 6 6 6	dst(7)<-dst(7), C<-dst(0) dst(n)<-dst(n+1)n=0-6 " " " " " "	^ ^ ^ ^ 0 ^ ^ ^ ^ ^ 0 ^ ^ ^ ^ ^ 0 ^ ^ ^ ^ ^ 0 ^
SRAW : Shift word right arithmetic						
SRAW SRAW	rr RR		2 2	6 8	dst(15)<-dst(15), C<-dst(0) dst(n)<-dst(n+1)n=0-14 " "	^ ^ ^ 0 - - ^ ^ ^ 0 - -

Notes:

1 Following this instruction, all addressing modes referring to address spaces will refer to the Data Space

2 Following this instruction, all addressing modes referring to address spaces will refer to the Program Space, except for the following instructions which operate with Dataspace independently of the setting of the DP flag .

PUSH(W)/PUSHU(W), POP(W)/POPU(W), PEA/PEAU, and CALL, RET, IRET and interrupt execution (assuming the Stack Pointers are not pointing to the Register File.

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SRP : Set register pointer						
SRP		#N	2	6	Set Register Pointer	- - - - - -
SRP0 : Set register pointer 0						
SRP0		#N	2	6	Set Register Pointer 0	- - - - - -
SRP1 : Set register pointer 1						
SRP1		#N	2	6	Set Register Pointer 1	- - - - - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SUB : Subtraction of 2 bytes without carry						
SUB	r	r	2	6	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	R	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	R	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	r	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	(r)	2	6	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	(r)	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	(rr)	3	12	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	(rr)	3	12	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	NN	4	18	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	N(rrx)	4	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	N(rrx)	4	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	NN(rrx)	5	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	NN(rrx)	5	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	rr(rrx)	3	22	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	(rr)+	3	16	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	R	(rr)+	3	16	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	r	-(rr)	3	16	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	-(rr)	3	16	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(r)	r	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(r)	R	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	r	3	18	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	R	3	18	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)+	r	3	22	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	(rr)+	R	3	22	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	NN	r	4	20	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	N(rrx)	r	4	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	N(rrx)	R	4	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	NN(rrx)	r	5	28	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	NN(rrx)	R	5	28	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	rr(rrx)	r	3	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	-(rr)	r	3	22	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	-(rr)	R	3	22	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	#N	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	#N	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	#N	3	16	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	NN	#N	5	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	(rr)	3	20	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(RR)	(rr)	3	20	dst<-dst-src	^ ^ ^ ^ 1 ^

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SUBW : Subtract words						
SUBW	rr	rr	2	10	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	RR	3	12	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	RR	3	12	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	rr	3	12	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	(r)	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	(r)	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	(rr)	2	16	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	(rr)	3	18	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	NN	4	22	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	N(rrx)	4	28	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	N(rrx)	4	28	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	NN(rrx)	5	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	NN(rrx)	5	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	rr(rrx)	3	26	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	(rr)+	3	22	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	RR	(rr)+	3	22	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(r)	rr	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(r)	RR	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	rr	2	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	RR	3	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)+	rr	3	32	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	(rr)+	RR	3	32	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	NN	rr	4	32	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	N(rrx)	rr	4	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	N(rrx)	RR	4	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN(rrx)	rr	5	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN(rrx)	RR	5	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr(rrx)	rr	3	34	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	#NN	4	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	#NN	4	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	#NN	4	32	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN	#NN	6	36	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	N(rrx)	#NN	5	36	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN(rrx)	#NN	6	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	(rr)	2	32	dst<-dst-src	^ ^ ^ ^ ? ?
SWAP : Swap nibbles						
SWAP	r		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -
SWAP	R		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -
SWAP	(r)		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -
SWAP	(R)		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TCM : Test and complement byte under mask						
TCM	r	r	2	6	NOT dst AND src	- ^ ^ 0 - -
TCM	R	R	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	r	R	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	R	r	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	r	(r)	2	6	NOT dst AND src	- ^ ^ 0 - -
TCM	R	(r)	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	r	(rr)	3	12	NOT dst AND src	- ^ ^ 0 - -
TCM	R	(rr)	3	12	NOT dst AND src	- ^ ^ 0 - -
TCM	r	NN	4	18	NOT dst AND src	- ^ ^ 0 - -
TCM	r	N(rrx)	4	24	NOT dst AND src	- ^ ^ 0 - -
TCM	R	N(rrx)	4	24	NOT dst AND src	- ^ ^ 0 - -
TCM	r	NN(rrx)	5	26	NOT dst AND src	- ^ ^ 0 - -
TCM	R	NN(rrx)	5	26	NOT dst AND src	- ^ ^ 0 - -
TCM	r	rr(rrx)	3	22	NOT dst AND src	- ^ ^ 0 - -
TCM	r	(rr)+	3	16	NOT dst AND src rr<-rr+1	- ^ ^ 0 - -
TCM	R	(rr)+	3	16	NOT dst AND src rr<-rr+1	- ^ ^ 0 - -
TCM	r	-(rr)	3	16	rr<-rr-1 NOT dst AND src	- ^ ^ 0 - -
TCM	R	-(rr)	3	16	rr<-rr-1 NOT dst AND src	- ^ ^ 0 - -
TCM	(r)	r	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	(r)	R	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	r	3	18	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	R	3	18	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)+	r	3	22	NOT dst AND src rr<-rr+1	- ^ ^ 0 - -
TCM	(rr)+	R	3	22	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
TCM	NN	r	4	20	NOT dst AND src	- ^ ^ 0 - -
TCM	N(rrx)	r	4	26	NOT dst AND src	- ^ ^ 0 - -
TCM	N(rrx)	R	4	26	NOT dst AND src	- ^ ^ 0 - -
TCM	NN(rrx)	r	5	28	NOT dst AND src	- ^ ^ 0 - -
TCM	NN(rrx)	R	5	28	NOT dst AND src	- ^ ^ 0 - -
TCM	rr(rrx)	r	3	24	NOT dst AND src	- ^ ^ 0 - -
TCM	-(rr)	r	3	22	NOT dst AND src rr<-rr-1	- ^ ^ 0 - -
TCM	-(rr)	R	3	22	NOT dst AND src rr<-rr-1	- ^ ^ 0 - -
TCM	r	#N	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	R	#N	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	#N	3	16	NOT dst AND src	- ^ ^ 0 - -
TCM	NN	#N	5	22	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	(rr)	3	18	NOT dst AND src	- ^ ^ 0 - -
TCM	(RR)	(rr)	3	18	NOT dst AND src	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TCMW : Test and complement word under mask						
TCMW	rr	rr	2	10	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	RR	3	12	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	RR	3	12	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	rr	3	12	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	(r)	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	(r)	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	(rr)	2	16	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	(rr)	3	18	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	NN	4	22	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	N(rrx)	4	28	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	N(rrx)	4	28	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	NN(rrx)	5	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	NN(rrx)	5	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	rr(rrx)	3	26	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	(rr)+	3	22	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	RR	(rr)+	3	22	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	rr	-(rr)	3	24	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	-(rr)	3	24	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	(r)	rr	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	(r)	RR	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	rr	2	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	RR	3	28	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)+	rr	3	30	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	(rr)+	RR	3	30	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	NN	rr	4	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	N(rrx)	rr	4	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	N(rrx)	RR	4	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN(rrx)	rr	5	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN(rrx)	RR	5	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr(rrx)	rr	3	32	NOT dst AND src	- ^ ^ 0 - -
TCMW	-(rr)	rr	3	30	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	-(rr)	RR	3	30	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	#NN	4	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	#NN	4	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	#NN	4	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN	#NN	6	34	NOT dst AND src	- ^ ^ 0 - -
TCMW	N(rrx)	#NN	5	34	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN(rrx)	#NN	6	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	(rr)	2	32	NOT dst AND src	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TM : Test byte under mask						
TM	r	r	2	6	dst AND src	- ^ ^ 0 - -
TM	R	R	3	10	dst AND src	- ^ ^ 0 - -
TM	r	R	3	10	dst AND src	- ^ ^ 0 - -
TM	R	r	3	10	dst AND src	- ^ ^ 0 - -
TM	r	(r)	2	6	dst AND src	- ^ ^ 0 - -
TM	R	(r)	3	10	dst AND src	- ^ ^ 0 - -
TM	r	(rr)	3	12	dst AND src	- ^ ^ 0 - -
TM	R	(rr)	3	12	dst AND src	- ^ ^ 0 - -
TM	r	NN	4	18	dst AND src	- ^ ^ 0 - -
TM	r	N(rrx)	4	24	dst AND src	- ^ ^ 0 - -
TM	R	N(rrx)	4	24	dst AND src	- ^ ^ 0 - -
TM	r	NN(rrx)	5	26	dst AND src	- ^ ^ 0 - -
TM	R	NN(rrx)	5	26	dst AND src	- ^ ^ 0 - -
TM	r	rr(rrx)	3	22	dst AND src	- ^ ^ 0 - -
TM	r	(rr)+	3	16	dst AND src rr<-rr+1	- ^ ^ 0 - -
TM	R	(rr)+	3	16	dst AND -src rr<-rr+1	- ^ ^ 0 - -
TM	r	-(rr)	3	16	rr<-rr-1 dst AND src	- ^ ^ 0 - -
TM	R	-(rr)	3	16	rr<-rr-1 dst AND src	- ^ ^ 0 - -
TM	(r)	r	3	10	dst AND src	- ^ ^ 0 - -
TM	(r)	R	3	10	dst AND src	- ^ ^ 0 - -
TM	(rr)	r	3	18	dst AND src	- ^ ^ 0 - -
TM	(rr)	R	3	18	dst AND src	- ^ ^ 0 - -
TM	(rr)+	r	3	22	dst AND src rr<-rr+1	- ^ ^ 0 - -
TM	(rr)+	R	3	22	dst AND src rr<-rr+1	- ^ ^ 0 - -
TM	NN	r	4	20	dst AND src	- ^ ^ 0 - -
TM	N(rrx)	r	4	26	dst AND src	- ^ ^ 0 - -
TM	N(rrx)	R	4	26	dst AND src	- ^ ^ 0 - -
TM	NN(rrx)	r	5	28	dst AND src	- ^ ^ 0 - -
TM	NN(rrx)	R	5	28	dst AND src	- ^ ^ 0 - -
TM	rr(rrx)	r	3	24	dst AND src	- ^ ^ 0 - -
TM	-(rr)	r	3	22	rr->rr-1 dst AND src	- ^ ^ 0 - -
TM	-(rr)	R	3	22	rr->rr-1 dst AND src	- ^ ^ 0 - -
TM	r	#N	3	10	dst AND src	- ^ ^ 0 - -
TM	R	#N	3	10	dst AND src	- ^ ^ 0 - -
TM	(rr)	#N	3	16	dst AND src	- ^ ^ 0 - -
TM	NN	#N	5	22	dst AND src	- ^ ^ 0 - -
TM	(rr)	(rr)	3	18	dst AND src	- ^ ^ 0 - -
TM	(RR)	(rr)	3	18	dst AND src	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TMW : Test word under mask						
TMW	rr	rr	2	10	dst AND src	- ^ ^ 0 - -
TMW	RR	RR	3	12	dst AND src	- ^ ^ 0 - -
TMW	rr	RR	3	12	dst AND src	- ^ ^ 0 - -
TMW	RR	rr	3	12	dst AND src	- ^ ^ 0 - -
TMW	rr	(r)	3	14	dst AND src	- ^ ^ 0 - -
TMW	RR	(r)	3	14	dst AND src	- ^ ^ 0 - -
TMW	rr	(rr)	2	16	dst AND src	- ^ ^ 0 - -
TMW	RR	(rr)	3	18	dst AND src	- ^ ^ 0 - -
TMW	rr	NN	4	22	dst AND src	- ^ ^ 0 - -
TMW	rr	N(rrx)	4	28	dst AND src	- ^ ^ 0 - -
TMW	RR	N(rrx)	4	28	dst AND src	- ^ ^ 0 - -
TMW	rr	NN(rrx)	5	30	dst AND src	- ^ ^ 0 - -
TMW	RR	NN(rrx)	5	30	dst AND src	- ^ ^ 0 - -
TMW	rr	rr(rrx)	3	26	dst AND src	- ^ ^ 0 - -
TMW	rr	(rr)+	3	22	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	RR	(rr)+	3	22	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	rr	-(rr)	3	24	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	RR	-(rr)	3	24	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	(r)	rr	3	14	dst AND src	- ^ ^ 0 - -
TMW	(r)	RR	3	14	dst AND src	- ^ ^ 0 - -
TMW	(rr)	rr	2	28	dst AND src	- ^ ^ 0 - -
TMW	(rr)	RR	3	28	dst AND src	- ^ ^ 0 - -
TMW	(rr)+	rr	3	30	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	(rr)+	RR	3	30	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	NN	rr	4	30	dst AND src	- ^ ^ 0 - -
TMW	N(rrx)	rr	4	36	dst AND src	- ^ ^ 0 - -
TMW	N(rrx)	RR	4	36	dst AND src	- ^ ^ 0 - -
TMW	NN(rrx)	rr	5	36	dst AND src	- ^ ^ 0 - -
TMW	NN(rrx)	RR	5	36	dst AND src	- ^ ^ 0 - -
TMW	rr(rrx)	rr	3	32	dst AND src	- ^ ^ 0 - -
TMW	-(rr)	rr	3	30	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	-(rr)	RR	3	30	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	rr	#NN	4	14	dst AND src	- ^ ^ 0 - -
TMW	RR	#NN	4	14	dst AND src	- ^ ^ 0 - -
TMW	(rr)	#NN	4	30	dst AND src	- ^ ^ 0 - -
TMW	NN	#NN	6	34	dst AND src	- ^ ^ 0 - -
TMW	N(rrx)	#NN	5	34	dst AND src	- ^ ^ 0 - -
TMW	NN(rrx)	#NN	6	36	dst AND src	- ^ ^ 0 - -
TMW	(rr)	(rr)	2	32	dst AND src	- ^ ^ 0 - -
WFI : Wait for Interrupt						
WFI			2	18	wait for interrupt	- - - - - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
XCH : Exchange Register						
XCH	r	r	3	12	dst <-> src	- - - - - -
XCH	R	R	3	12	dst <-> src	- - - - - -
XCH	r	r	3	12	dst <-> src	- - - - - -
XCH	R	R	3	12	dst <-> src	- - - - - -
XOR : Logical exclusive OR						
XOR	r	r	2	6	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	R	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	R	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	r	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	(r)	2	6	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	(r)	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	(rr)	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	(rr)	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	NN	4	18	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	N(rrx)	4	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	N(rrx)	4	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	NN(rrx)	5	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	NN(rrx)	5	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	rr(rrx)	3	22	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	(rr)+	3	16	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	R	(rr)+	3	16	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	r	-(rr)	3	16	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	-(rr)	3	16	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	(r)	r	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(r)	R	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	r	3	18	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	R	3	18	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)+	r	3	22	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	(rr)+	R	3	22	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	NN	r	4	20	dst<-dst XOR src	- ^ ^ 0 - -
XOR	N(rrx)	r	4	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	N(rrx)	R	4	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	NN(rrx)	r	5	28	dst<-dst XOR src	- ^ ^ 0 - -
XOR	NN(rrx)	R	5	28	dst<-dst XOR src	- ^ ^ 0 - -
XOR	rr(rrx)	r	3	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	-(rr)	r	3	22	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	-(rr)	R	3	22	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	#N	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	#N	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	#N	3	16	dst<-dst XOR src	- ^ ^ 0 - -
XOR	NN	#N	5	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	(rr)	3	20	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(RR)	(rr)	3	20	dst<-dst XOR src	- ^ ^ 0 - -

INSTRUCTION SUMMARY (Continued)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
XORW : Logical exclusive OR between words						
XORW	rr	rr	2	10	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	RR	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	RR	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	rr	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	(r)	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	(r)	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	(rr)	2	16	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	(rr)	3	18	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	NN	4	22	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	N(rrx)	4	28	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	N(rrx)	4	28	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	NN(rrx)	5	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	NN(rrx)	5	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	rr(rrx)	3	26	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	(rr)+	3	22	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	RR	(rr)+	3	22	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	(r)	rr	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(r)	RR	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	rr	2	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	RR	3	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)+	rr	3	32	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	(rr)+	RR	3	32	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	NN	rr	4	32	dst<-dst XOR src	- ^ ^ 0 - -
XORW	N(rrx)	rr	4	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	N(rrx)	RR	4	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN(rrx)	rr	5	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN(rrx)	RR	5	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr(rrx)	rr	3	34	dst<-dst XOR src	- ^ ^ 0 - -
XORW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	#NN	4	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	#NN	4	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	#NN	4	32	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN	#NN	6	36	dst<-dst XOR src	- ^ ^ 0 - -
XORW	N(rrx)	#NN	5	36	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN(rrx)	#NN	6	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	(rr)	2	32	dst<-dst XOR src	- ^ ^ 0 - -

REGISTER MAP

1 INTRODUCTION						Page
2 CORE ARCHITECTURE						
CICR	**R230**	(E6h)	System	Read/Write	Central Interrupt Control Register	13
FLAGR	**R231**	(E7h)	System	Read/Write	Flag Register	14
RP0	**R232**	(E8h)	System	Read/Write	Register Pointer 0	15
RP1	**R233**	(E9h)	System	Read/Write	Register Pointer 1	15
PPR	**R234**	(EAh)	System	Read/Write	Page Pointer Register	17
MODER	**R235**	(EBh)	System	Read/Write	Mode Register	17
3 MEMORY						
EECR	**R241**	(F1h)	Page 0	Read/Write	EEPROM Control Register	24
4 INTERRUPTS						
CICR	**R230**	(E6h)	System	Read/Write	Central Interrupt Control Register	40
EITR	**R242**	(F2h)	Page 0	Read/Write	External Interrupt Trigger Event Register	40
IDPR	**R243**	(F3h)	Page 0	Read/Write	External Interrupt Pending Register	40
EIMR	**R244**	(F4h)	Page 0	Read/Write	External Interrupt Mask-bit Register	41
EIPLR	**R245**	(F5h)	Page 0	Read/Write	External Interrupt Priority Level Register	41
EIVR	**R246**	(F6h)	Page 0	Read/Write	External Interrupt Vector Register	41
NICR	**R247**	(F7h)	Page 0	Read/Write	Nested Interrupt Control Register	41
5 ON-CHIP DMA						
DCPR	Address set by Peripheral			Read/Write	DMA Counter Pointer Register	48
IDCR	Address set by Peripheral			Read/Write	Generic Peripheral Interrupt and DMA Control	48
DAPR	Address set by Peripheral			Read/Write	DMA Address Pointer Register	48
6 CLOCK						
MODER	**R235**	(EBh)	System	Read/Write	Mode Register	50
8 EXTERNAL MEMORY INTERFACE						
WCR	**R252**	(FCh)	Page 0	Read/Write	Wait Control Register	66
10 HANDSHAKE/DMA CONTROLLER						
HDCTLx				Read/Write	Handshake/DMA Control Register	84
11 SERIAL PERIPHERAL INTERFACE						
SPIDR	**R253**	(FDh)	Page 0	Read/Write	SPI Data Register (R253)	90
SPICR	**R255**	(FEh)	Page 0	Read/Write	SPI Control Register (R254)	90

12 TIMER/WATCHDOG

WDTHR	**R248**	(F8h)	Page 0	Read/Write	Timer/Watchdog Counter Register, High byte	101
WDTLR	**R249**	(F9h)	Page 0	Read/Write	Timer/Watchdog Counter Register, Low byte.	101
WDTPR	**R250**	(FAh)	Page 0	Read/Write	Timer/Watchdog Prescaler Register	101
WDTCR	**R251**	(FBh)	Page 0	Read/Write	Timer/Watchdog Control Register	101

13 MULTIFUNCTION TIMER

REG0HR	**R240**	(F0h)		Read/Write	Capture Load Register 0 (High)	117
REG0LR	**R241**	(F1h)		Read/Write	Capture Load Register 0 (Low)	117
REG1HR	**R242**	(F2h)		Read/Write	Capture Load Register 1 (High)	117
REG1LR	**R243**	(F3h)		Read/Write	Capture Load Register 1 (Low)	117
CMP0HR	**R244**	(F4h)		Read/Write	Compare 0 Register (High)	117
CMP0LR	**R245**	(F5h)		Read/Write	Compare 0 Register (Low)	117
CMP1HR	**R246**	(F6h)		Read/Write	Compare 1 Register (High)	117
CMP1LR	**R247**	(F7h)		Read/Write	Compare 1 Register (Low)	117
TCR	**R248**	(F8h)		Read/Write	Timer Control Register	118
TMR	**R249**	(F9h)		Read/Write	Timer Mode Register	118
ICR	**R250**	(FAh)		Read/Write	External Input Control Register	119
PRSR	**R251**	(FBh)		Read/Write	Prescaler Register	120
OACR	**R252**	(FCh)		Read/Write	Output A Control Register	120
OBCR	**R253**	(FDh)		Read/Write	Output B Control Register	121
FLAGR	**R254**	(FEh)		Read/Write	Flags Register	121
IDMR	**R255**	(FFh)		Read/Write	Interrupt/DMA Mask Register	122
DCPR	**R240**	(F0h)	[R244 (F4h)]	Read/Write	DMA Counter Pointer Register	122
DAPR	**R241**	(F1h)	[R245 (F5h)]	Read/Write	DMA Address Pointer Register	123
IVR	**R242**	(F2h)	[R246 (F6h)]	Read/Write	Interrupt Vector Register	123
IDCR	**R243**	(F3h)	[R247 (F7h)]	Read/Write	Interrupt/DMA Control Register	124
IOCR	**R248**	(F8h)		Read/Write	I/O Connection Register	124

14 SERIAL COMMUNICATIONS INTERFACE

RDCPR	**R240**	(F0h)	Read/Write	Receiver DMA Transaction Counter Pointer	134
RDAPR	**R241**	(F1h)	Read/Write	Receiver DMA Source Address Pointer	134
TDCPR	**R242**	(F2h)	Read/Write	Transmitter DMA Transaction Counter Pointer	134
TDAPR	**R243**	(F3h)	Read/Write	Transmitter DMA Destination Address Pointer	134
IVR	**R244**	(F4h)	Read/Write	Interrupt Vector Register	134
ACR	**R245**	(F5h)	Read/Write	Address/Data Compare Register	135
IMR	**R246**	(F6h)	Read/Write	Interrupt Mask Register	135
ISR	**R247**	(F7h)	Read/Write	Interrupt Status Register	136
RXBR	**R248**	(F8h)	Read only	Receive Buffer Register	136
TXBR	**R248**	(F8h)	Write only	Transmitter Buffer Register	136
IDPR	**R249**	(F9h)	Read/Write	Interrupt/DMA Priority Register	137

CHCR	**R250**	(FAh)		Read/Write	Character Configuration Register	138
CCR	**R251**	(FBh)		Read/Write	Clock Configuration Register	138
BRGHR	**R252**	(FCh)		Read/Write	Baud Rate Generator Register, High byte.	139
BRGLR	**R253**	(FDh)		Read/Write	Baud Rate Generator Register, Low byte.	139

15 A/D CONVERTER

D0R	**R240**	(F0h)	Page 63	Read/Write	Channel 0 Data Register	146
D1R	**R241**	(F1h)	Page 63	Read/Write	Channel 1 Data Register	146
D2R	**R242**	(F2h)	Page 63	Read/Write	Channel 2 Data Register	146
D3R	**R243**	(F3h)	Page 63	Read/Write	Channel 3 Data Register	146
D4R	**R244**	(F4h)	Page 63	Read/Write	Channel 4 Data Register	146
D5R	**R245**	(F5h)	Page 63	Read/Write	Channel 5 Data Register	146
D6R	**R246**	(F6h)	Page 63	Read/Write	Channel 6 Data Register	146
D7R	**R247**	(F7h)	Page 63	Read/Write	Channel 7 Data Register	146
LT6R	**R248**	(F8h)	Page 63	Read/Write	Channel 6 Lower Threshold Register	147
LT7R	**R249**	(F9h)	Page 63	Read/Write	Channel 7 Lower Threshold Register	147
UT6R	**R250**	(FAh)	Page 63	Read/Write	Channel 6 Upper Threshold Register	147
UT7R	**R251**	(FBh)	Page 63	Read/Write	Channel 7 Upper Threshold Register	147
CRR	**R252**	(FCh)	Page 63	Read/Write	Compare Result Register	147
CLR	**R253**	(FDh)	Page63	Read/Write	Control Logic Register	148
ICR	**R254**	(FEh)	Page 63	Read/Write	Interrupt Control Register	149
IVR	**R255**	(FFh)	Page 63	Read/Write	Interrupt Vector Register	149

17 ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{DD}	Supply Voltage	– 0.3 to 7.0	V
AV_{DD}, AV_{SS}	Analog Supply Voltage	$V_{SS} = AV_{SS} < AV_{DD} \leq V_{DD}$	V
V_I	Input Voltage	– 0.3 to V_{DD} +0.3	V
V_O	Output Voltage	– 0.3 to V_{DD} +0.3	V
T_{STG}	Storage Temperature	– 55 to + 150	°C
I_{INJ}	Pin Injection Current Digital Input	-5 to +5	mA
I_{INJ}	Pin Injection Current Analog Input	-5 to +5	mA
	Maximum Accumulated Pin injection Current in the device	-50 to +50	mA

Note Stresses above those listed as "absolute maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability. All voltages are referenced to VSS

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Value		Unit
		Min.	Max.	
T_A	Operating Temperature	– 40	85	°C
V_{DD}	Operating Supply Voltage	4.5	5.5	V
f_{OSCE}	External Oscillator Frequency		24	MHz
f_{OSCI}	Internal Clock Frequency (INTCLK)		12	MHz

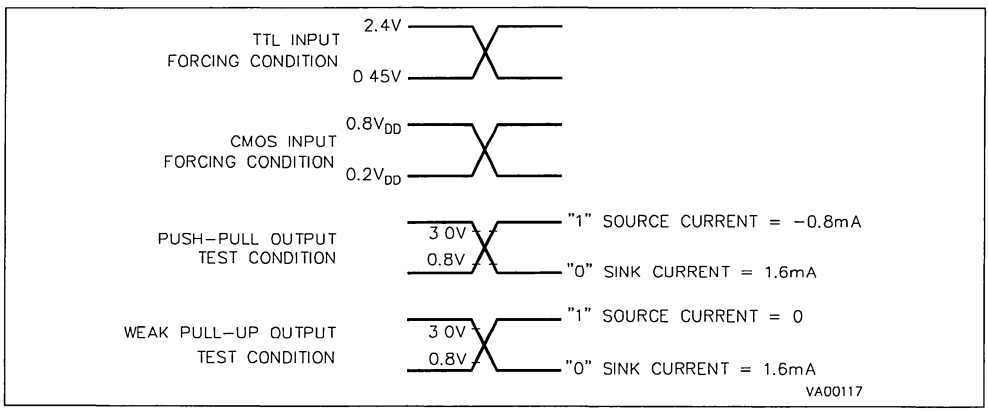
DC ELECTRICAL CHARACTERISTICS

(V_{DD} = 5V ± 10% T_A = – 40 °C to + 85°C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Value			Unit
			Min.	Typ.	Max.	
V_{IHCK}	Clock Input High Level	External Clock	0.7 V_{DD}		V_{DD} + 0.3	V
V_{ILCK}	Clock Input Low Level	External Clock	– 0.3		0.3 V_{DD}	V
V_{IH}	Input High Level	TTL	2.0		V_{DD} + 0.3	V
		CMOS	0.7 V_{DD}		VDD + 0.3	V
V_{IL}	Input Low Level	TTL	– 0.3		0.8	V
		CMOS	– 0.3		0.3 V_{DD}	V
V_{IHRS}	$\overline{\text{RESET}}$ Input High Level		0.7 V_{DD}		V_{DD} + 0.3	V
V_{ILRS}	$\overline{\text{RESET}}$ Input Low Level		–0.3		0.3 V_{DD}	V
V_{HYRS}	$\overline{\text{RESET}}$ Input Hysteresis		0.3		1.5	V
V_{OH}	Output High Level	Push Pull, Iload = – 0.8mA	V_{DD} – 0.8			V
V_{OL}	Output Low Level	Push Pull or Open Drain, Iload = 1.6mA			0.4	V
I_{WPU}	Weak Pull-up Current	Bidirectional Weak Pull-up, V_{OL} = 0V	– 50	– 200	– 420	µA
I_{APU}	Active Pull-up Current, for INT0 and INT7 only	V_{IN} < 0.8V, under Reset	– 80	– 200	– 420	µA
I_{LKIO}	I/O Pin Input Leakage	Input/Tri-State, 0V < V_{IN} < V_{DD}	– 10		+ 10	µA
I_{LKRS}	$\overline{\text{Reset}}$ Pin Input Leakage	0V < V_{IN} < V_{DD}	– 30		+ 30	µA
I_{LKAD}	A/D Pin Input Leakage	Alternate Function, Open Drain, 0V < V_{IN} < V_{DD}	– 13		+1 3	µA
I_{LKAP}	Active Pull-up Input Leakage	0V < V_{IN} < 0.8V	– 10		+ 10	µA
I_{LKOS}	OSCIN Pin Input Leakage	0V < V_{IN} < V_{DD}	– 10		+ 10	µA

Note All I/O Ports are configured in Bidirectional Weak Pull-up Mode with no DC load, External Clock pin (OSCIN) is driven by square wave external clock No peripheral working

DC TEST CONDITIONS

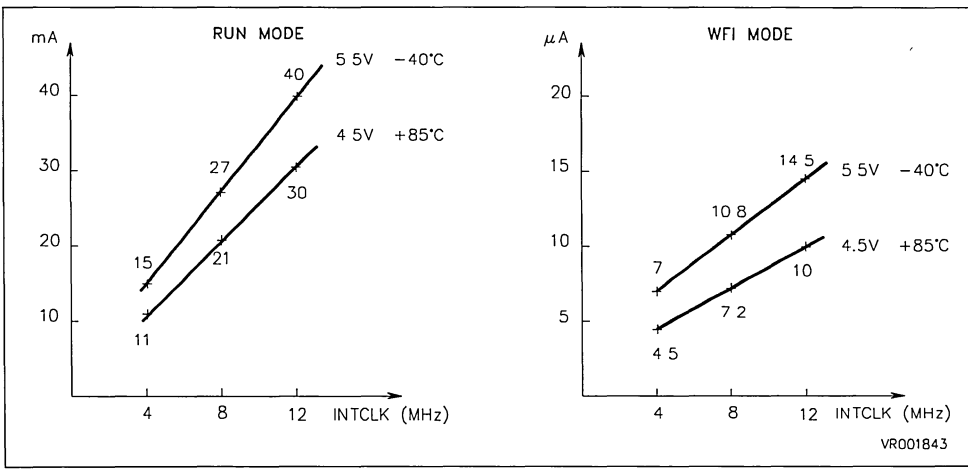


AC ELECTRICAL CHARACTERISTICS

V_{DD} = 5V ± 10% T_A = – 40 °C to + 85°C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Value			Unit
			Min.	Typ.	Max.	
I_{DD}	Run Mode Current no CPUCLK prescale, Clock divide by 2	24MHz, Note 1		40	70	mA
I_{DP2}	Run Mode Current Prescale by 2 Clock divide by 2	24MHz, Note 1		19	40	mA
I_{WFI}	WFI Mode Current no CPUCLK prescale, Clock divide by 2	24MHz, Note 1		15	20	mA
I_{HALT}	HALT Mode Current	24MHz, Note 1		50	100	µA

Typical Current Versus Frequency of Operation (f_{OSC})



CLOCK TIMING TABLE

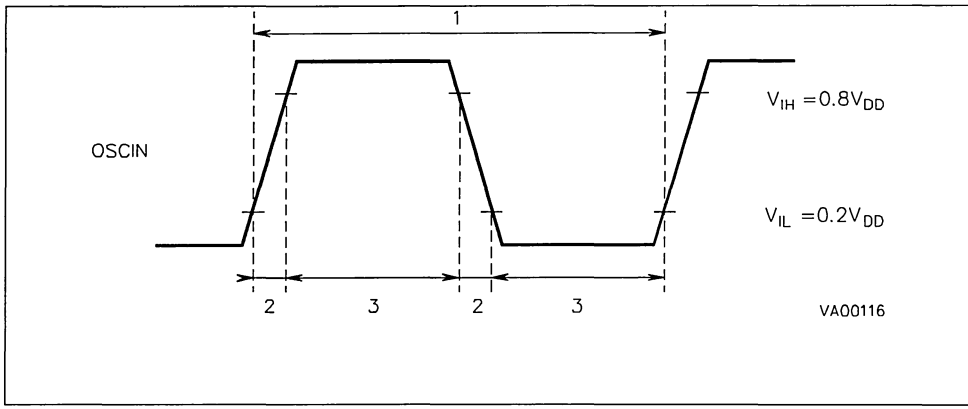
(V_{DD} = 5V ± 10%, T_A = − 40°C to + 85°C, INTCLK = 12MHz, unless otherwise specified

N°	Symbol	Parameter	Value		Unit	Note
			Min.	Max.		
1	TpC	OSCIN Clock Period	41.5		ns	1
			83		ns	2
2	TrC, TfC	OSCIN Rise and Fall Time		12	ns	
3	TwCL, TwCH	OSCIN Low and High Width	17	25	ns	1
			38		ns	2

Notes:

1. Clock divided by 2 internally (MODER.DIV2=1)
2. Clock not divided by 2 internally (MODER DIV2=0)

CLOCK TIMING



EXTERNAL BUS TIMING TABLE (V_{DD} = 5V ± 10%, T_A = – 40 °C to + 85 °C, Cload = 50pF, CPUCLK = 12MHz, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TsA (AS)	Address Set-up Time before $\overline{AS}\uparrow$	TpC (2P+1) –22	TWCH+PTpC –18	20		ns
2	ThAS (A)	Address Hold Time after $\overline{AS}\uparrow$	TpC –17	TwCL –13	25		ns
3	TdAS (DR)	$\overline{AS}\uparrow$ to Data Available (read)	TpC (4P+2W+4) –52	TpC (2P+W+2) –51		115	ns
4	TwAS	$\overline{AS}$ Low Pulse Width	TpC (2P+1) –7	TwCH+PTpC –3	35		ns
5	TdAz (DS)	$\overline{DS}\downarrow$ to Address Float			12		ns
6	TwDSR	$\overline{DS}$ Low Pulse Width (read)	TpC (4P+2W+3) –20	TwCH+TpC (2P+W+1) –16	105		ns
7	TwDSW	$\overline{DS}$ Low Pulse Width (write)	TpC (2P+2W+2) –13	TpC (P+W+1) –13	70		ns
8	TdDSR (DR)	$\overline{DS}\downarrow$ to Data Valid Delay (read)	TpC (4P+2W-3) –50	TwCH+TpC(2P+W+1) –46		75	ns
9	ThDR (DS)	Data to $\overline{DS}\uparrow$ Hold Time (read)	0	0	0		ns
10	TdDS (A)	$\overline{DS}\uparrow$ to Address Active Delay	TpC –7	TwCL –3	35		ns
11	TdDS (AS)	$\overline{DS}\uparrow$ to $\overline{AS}\downarrow$ Delay	TpC –18	TwCL –14	24		ns
12	TsR/W (AS)	R/$\overline{W}$ Set-up Time before $\overline{AS}\uparrow$	TpC (2P+1) –22	TwCH+PTpC –18	20		ns
13	TdDSR (R/W)	$\overline{DS}\uparrow$ to R/$\overline{W}$ and Address Not Valid Delay	TpC –9	TwCL –5	33		ns
14	TdDW (DSW)	Write Data Valid to $\overline{DS}\downarrow$ Delay (write)	TpC (2P+1) –32	TwCH+PTpC –28	10		ns
15	ThDS (DW)	Data Hold Time after $\overline{DS}\uparrow$ (write)	TpC –9	TwCL –5	33		ns
16	TdA (DR)	Address Valid to Data Valid Delay (read)	TpC (6P+2W+5) –68	TwCH+TpC (3P+W+2) –64		140	ns
17	TdAs (DS)	$\overline{AS}\uparrow$ to $\overline{DS}\downarrow$ Delay	TpC –18	TwCL –14	24		ns

EXTERNAL WAIT TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

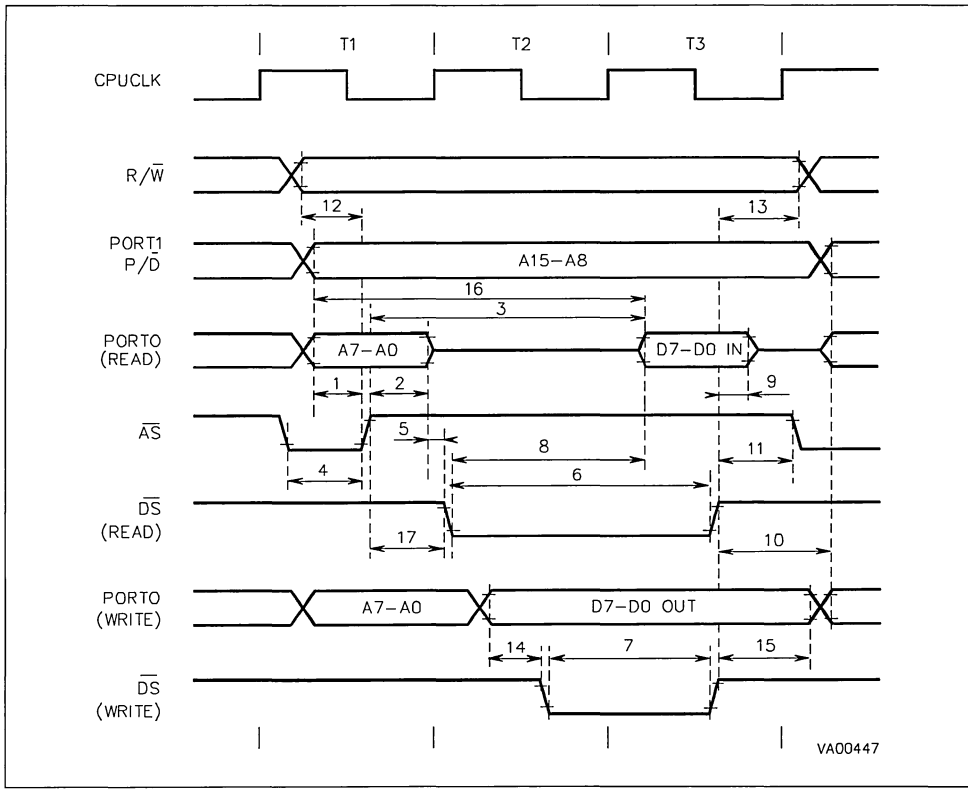
N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TdAs (WAIT)	$\overline{AS}\uparrow$ to $\overline{WAIT}\downarrow$ Delay	2(P+1)TpC –29	2(P+1)TpC –29		40	ns
2	TdAs (WAIT)	$\overline{AS}\uparrow$ to $\overline{WAIT}\downarrow$ Min. Delay	2(P+W+1)TpC –4	2(P+W+1)TpC –4	80		ns
3	TdAs (WAIT)	$\overline{AS}\uparrow$ to $\overline{WAIT}\downarrow$ Max. Delay	2(P+W+1)TpC –29	2(P+W+1)TpC –29		33W+ 40	ns

Note: (for both table) The value in the left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value in the right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescaler value of zero and zero wait status

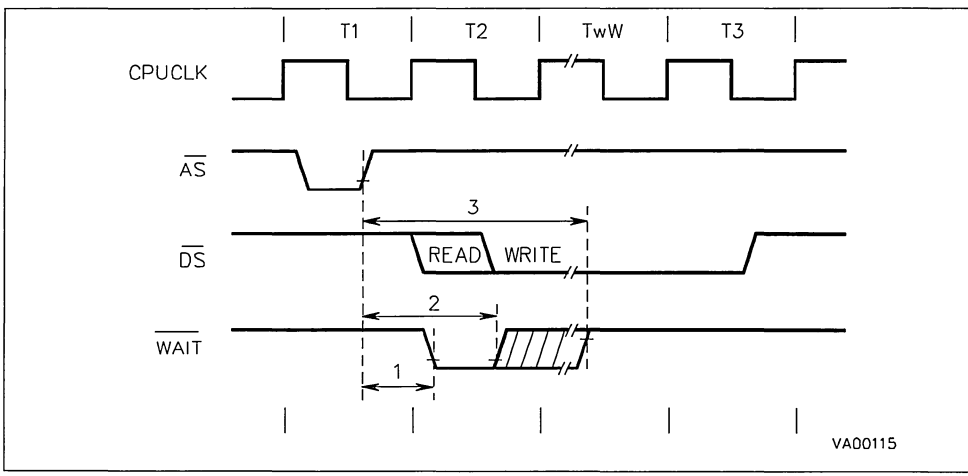
Legend:

P = Clock Prescaling Value
W = Wait Cycles
TpC =OSCIN Period
TwCH =High Level OSCIN half period
TwCL =Low Level OSCIN half period

EXTERNAL BUS TIMING



EXTERNAL WAIT TIMING



HANDSHAKE TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Min.	Max.	Unit
			OSCIN Divided By 2		OSCIN Not Divided By 2				
			Min.	Max.	Min.	Max.			
1	TwRDY	RDRDY, WRRDY Pulse Width in One Line Handshake	2TpC (P+W+1) –18		TpC (P+W+1) – 18		65		ns
2	TwSTB	$\overline{RDSTB}$, $\overline{WRSTB}$ Pulse Width	2TpC+12		TpC+12		95		ns
3	TdST (RDY)	$\overline{RDSTB}$, or $\overline{WRSTB}$ ↑ to RDRDY or WRRDY ↓		TpC+45		(TpC-TwCL) +45		87	ns
4	TsPD (RDY)	Port Data to RDRDY ↑ Set-up Time	(2P+2W+1) TpC –25		TwCH+ (W+P) TpC –25		16		ns
5	TsPD (RDY)	Port Data to WRRDY ↓ Set-up Time in One Line Handshake	43		43		43		ns
6	ThPD (RDY)	Port Data to WRRDY ↓ Hold Time in One Line Handshake	0		0		0		ns
7	TsPD (STB)	Port Data to $\overline{WRSTB}$ ↑ Set-up Time	10		10		10		ns
8	ThPD (STB)	Port Data to WRSTB ↑ Hold Time	25		25		25		ns
9	TdSTB (PD)	$\overline{RDSTBD}$ ↑ to Port Data Delay Time in Bidirectional Handshake		35		35		35	ns
10	TdSTB (PHZ)	$\overline{RDSTB}$ ↑ to Port High-Z Delay Time in Bidirectional Handshake		25		25		25	ns

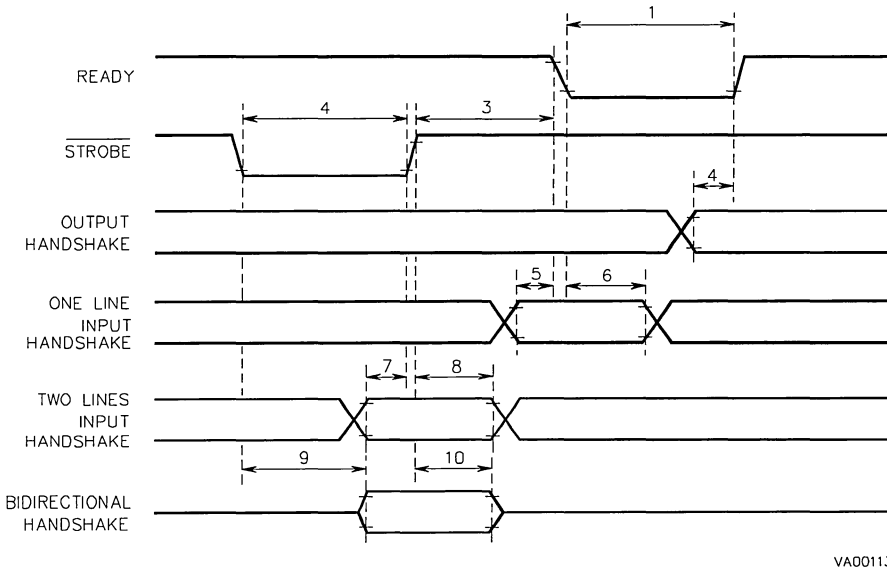
Note: The value in the left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value in the right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescaler value of zero and zero wait status.

Legend:

P = Clock Prescaling Value (R235 4,3,2)

W = Programmable Wait Cycles (R252.2 1 0/5,4,3) + External Wait Cycles

HANDSHAKE TIMING

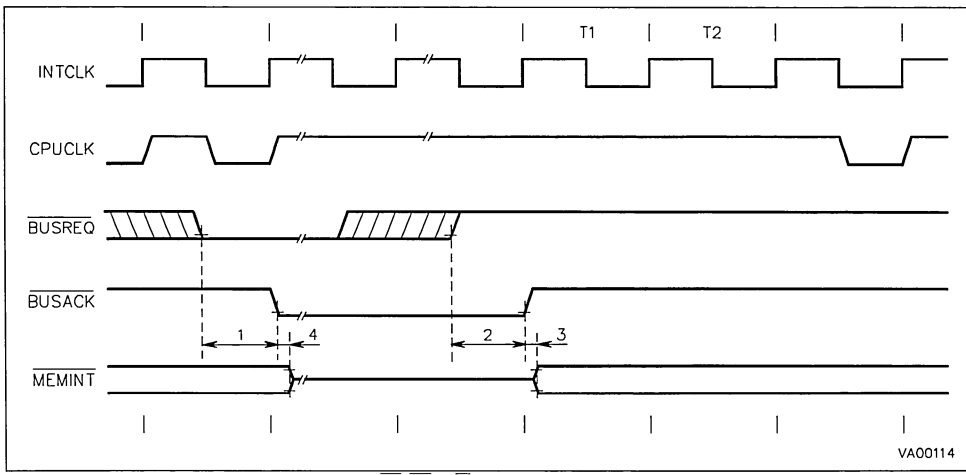


BUS REQUEST/ACKNOWLEDGE TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TdBR (BACK)	$\overline{BREQ}\downarrow$ to $\overline{BUSACK}\downarrow$	TpC+8	TwCL+12	50		ns
			TpC(6P+2W+7)+65	TpC(3P+W+3)+TwCL+65		360	ns
2	TdBR (BACK)	$\overline{BREQ}\uparrow$ to $\overline{BUSACK}\uparrow$	3TpC+60	TpC+TwCL+60		185	ns
3	TdBACK (BREL)	$\overline{BUSACK}\downarrow$ to Bus Release	20	20		20	ns
4	TdBACK (BACT)	$\overline{BUSACK}\uparrow$ to Bus Active	20	20		20	ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value right hand two columns show the timing minimum and maximum for an external clock at 24MHz divided by 2, prescale value of zero and zero wait status

BUS REQUEST/ACKNOWLEDGE TIMING



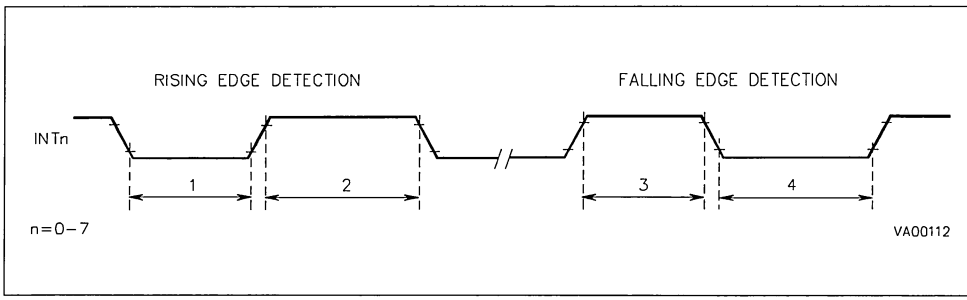
Note : MEMINT = Group of memory interface signals $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, P00-P07, P10-P17

EXTERNAL INTERRUPT TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2 Min.	OSCIN Not Divided By 2 Min.	Min.	Max.	
1	TwLR	Low Level Minimum Pulse Width in Rising Edge Mode	2TpC+12	TpC+12	95		ns
2	TwHR	High Level Minimum Pulse Width in Rising Edge Mode	2TpC+12	TpC+12	95		ns
3	TwHF	High Level Minimum Pulse Width in Falling Edge Mode	2TpC+12	TpC+12	95		ns
4	TwLF	Low Level Minimum Pulse Width in Falling Edge Mode	2TpC+12	TpC+12	95		ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescale value of zero and zero wait status

EXTERNAL INTERRUPT TIMING

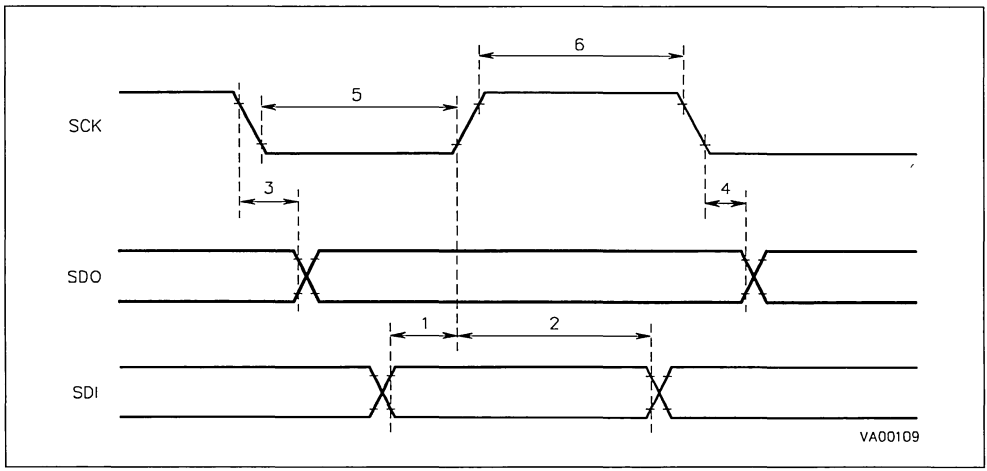


SPI TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Output Alternate Function set as Push-pull)

N°	Symbol	Parameter	Value		Unit
			Min.	Max.	
1	TsDI	Input Data Set-up Time	100		ns
2	ThDI (1)	Input Data Hold Time	1/2 TpC+100		ns
3	TdOV	SCK to Output Data Valid		100	ns
4	ThDO	Output Data Hold Time	-20		ns
5	TwSKL	SCK Low Pulse Width	300		ns
6	TwSKH	SCK High Pulse Width	300		ns

Note: TpC is the OSCIN Clock period

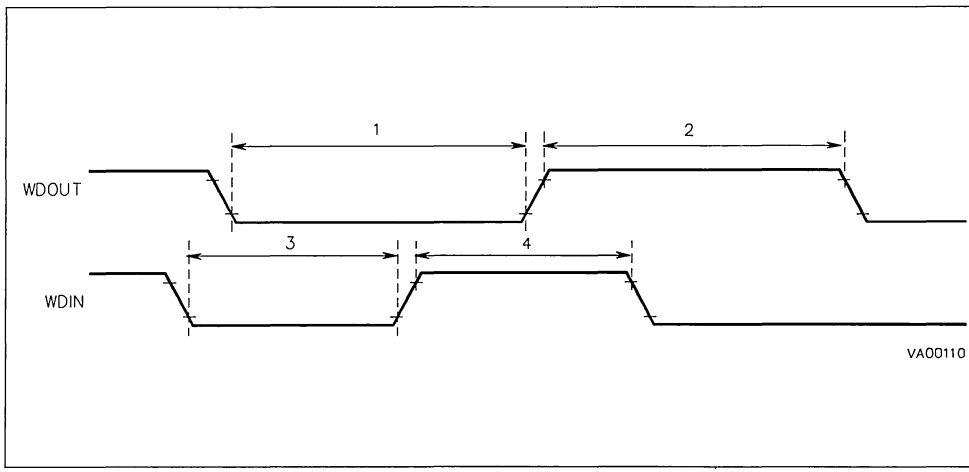
SPI TIMING



WATCHDOG TIMING TABLE(V_{DD} = 5V ± 10%, T_A = – 40 °C to +85°C, Cload = 50pF, CPUCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Values		Unit
			Min.	Max.	
1	TwWDOL	WDOUT Low Pulse Width	620		ns
2	TwWDOH	WDOUT High Pulse Width	620		ns
3	TwWDIL	WDIN High Pulse Width	350		ns
4	TwWDIH	WDIN Low Pulse Width	350		ns

WATCHDOG TIMING



A/D CONVERTER

EXTERNAL TRIGGER TIMING (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF)

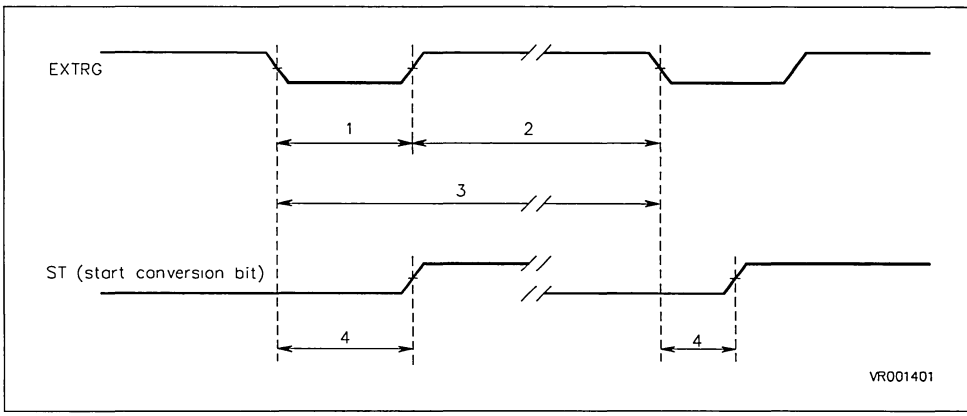
N°	Symbol	Parameter	Oscin divided by 2 (1)		Oscin not divided (1)		Value(2)		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	
1	T_{LOW}	External Trigger pulse width	$2xT_{PC}$		T_{PC}		83		ns
2	T_{HIGH}	External Trigger pulse	$2xT_{PC}$		T_{PC}		83		ns
3	T_{EXT}	External trigger active edges distance	$138xT_{PC}$		$69xT_{PC}$		5.75		µs
4	T_{STR}	Internal delay between EXTRG falling edge and first conversion start	T_{PC}	$3xT_{PC}$	0.5xTPC	$1.5xT_{PC}$	41.5	125	ns

Notes:

1 Variable clock (T_{PC}=OSCIN clock period)

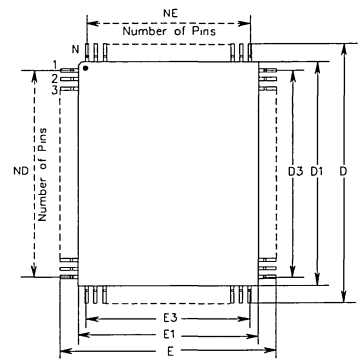
2. INTCLK=12MHz

A/D External Trigger Timing



PACKAGE MECHANICAL DATA

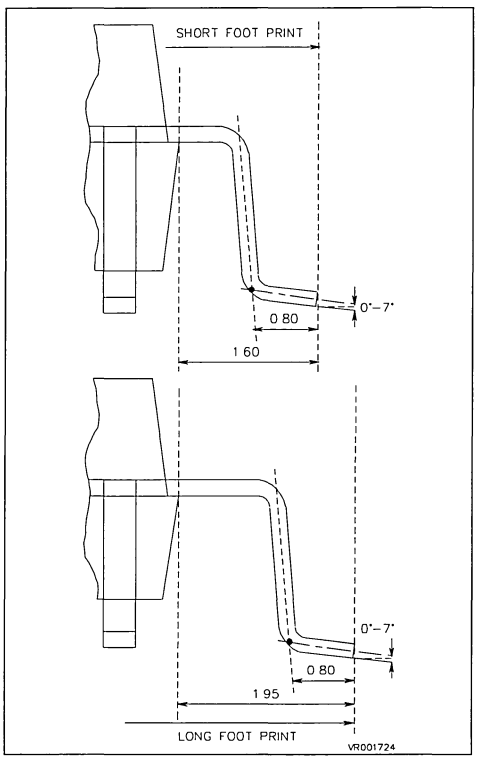
80-Pin Plastic Quad Flat Package



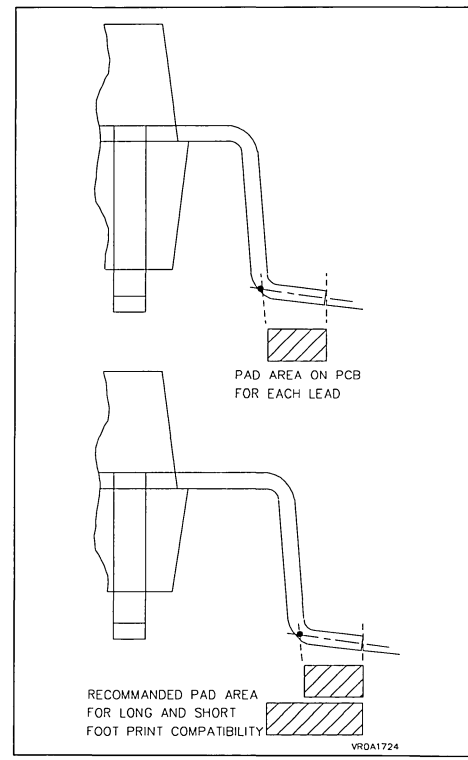
Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A			3.30			0.130
A2	2.55	2.80	3.05	0.100	0.110	0.120
D*	23.65	23.90	24.15	0.931	0.941	0.951
D1	19.90	20.00	20.10	0.783	0.787	0.791
D3		18.40			0.724	
E*	17.65	17.90	18.15	0.695	0.705	0.715
E1	13.90	14.00	14.10	0.547	0.551	0.555
E3		12.00			0.472	
e		0.80			0.032	
Number of Pins						
N	80					
ND	24					
NE	16					

* Subject to change,
Typ D will change to 23.20mm
Typ E will change to 17.20mm

Short/Long Footprint Measurement

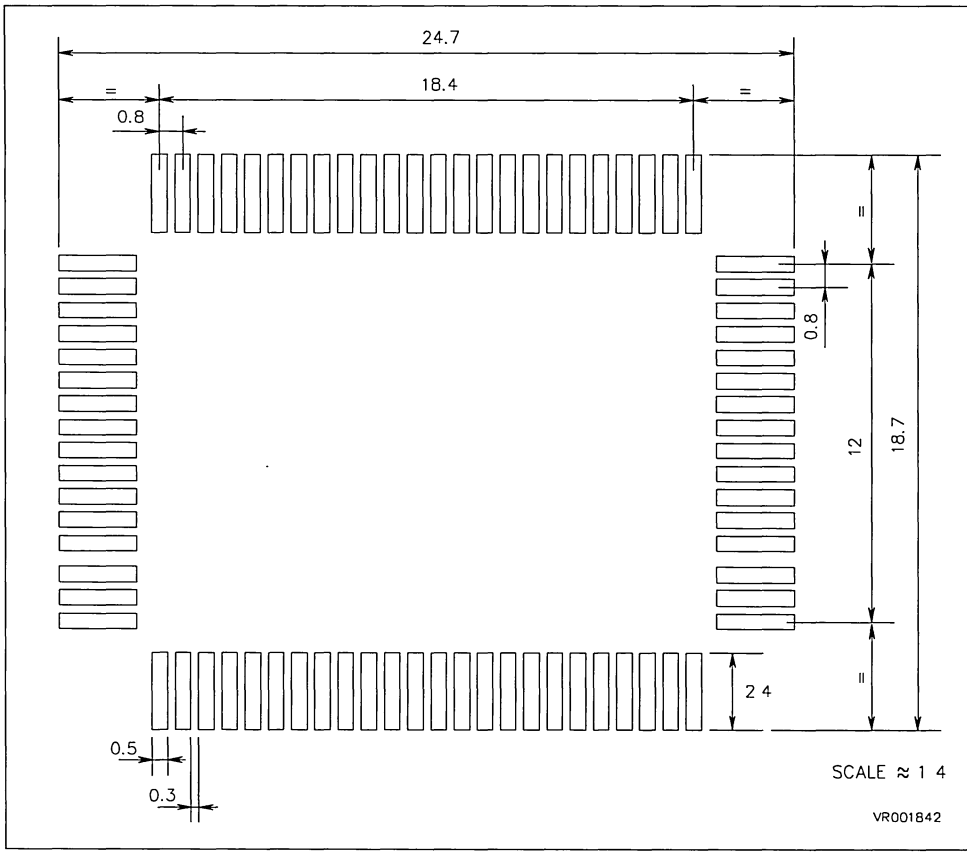


Short/Long Footprint recommended Padding

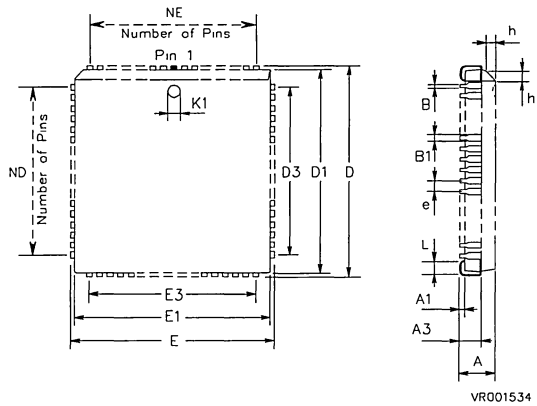


PACKAGE MECHANICAL DATA (Continued)

Solder Pad Footprint For QFP80



68-Pin Plastic Leadless Chip Carrier



Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A	4.20		5.08	0.165		0.200
A1	0.51			0.020		
A3	2.29		3.30	0.090		0.130
B	-	-	-	-	-	-
B1	-	-	-	-	-	-
D	25.02		25.27	0.985		0.995
D1	24.13		24.33	0.950		0.958
D3		20.32			0.800	
E	25.02		25.27	0.985		0.995
E1	24.13		24.33	0.950		0.958
E3		20.32			0.800	
K1	-	-	-	-	-	-
h						
e		1.27			0.050	
Number of Pins						
N	64					
ND	16					
NE	16					

ORDERING INFORMATION

Sales Type	Frequency	Temperature Range	Package
ST9040Q1/XX	24MHz	0°C to + 70°C	PQFP80
ST9040C1/XX			PLCC68
ST9040C6/XX		-40°C to + 85°C	PLCC68

Note: "XX" is the ROM code identifier that is allocated by SGS-THOMSON after receipt of all required options and the related ROM file

ST9040 STANDARD OPTION LIST

Please copy this page (enlarge if possible) and complete ALL sections.
Send the form, with the ROM code image required, to your local SGS-THOMSON sales office.

Customer Company : [. . . . .]
Company Address : [. . . . .]
[. . . . .]
Telephone : [. . . . .]
FAX : [. . . . .]
Contact : [. . . . .] Telephone (Direct) : [. . . . .]

Please confirm characteristics of device :

Device ST9040

Package [ ] PQFP80 [ ] PLCC68

Temperature Range [ ] -40°C to +85°C [ ] 0°C to +70°C

Special Marking [ ] No
[ ] Yes 14 characters [| | | | | | | | | | | | |]
Authorized characters are letters, digits, '.', '-', '/' and spaces only.
Please consult your local SGS-THOMSON sales office for other marking details if required.
Notes :

ROMless Option (Consult text)
[ ] No
[ ] Yes Port Bit [ ] P7.1 [ ] P2.0

Code : [ ] EPROM (27128, 27256)
[ ] HEX format files on IBM-PC® compatible disk
filename : [. . . . .]

Confirmation : [ ] Code checked with EPROM device in application

Yearly Quantity forecast : [. . . . .] k units
- for a period of : [. . . . .] years

Preferred Production start dates : [. . . . .] (YY/MM/DD)

Customer Signature :
Date :

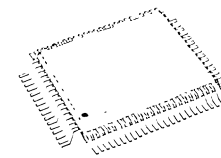
SGS-THOMSON MICROELECTRONICS

ST90E40
ST90T40

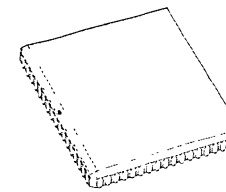
16K EPROM HCMOS MCU WITH EEPROM, RAM AND A/D CONVERTER

PRELIMINARY DATA

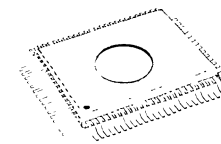
- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time: 500ns (12MHz internal)
- Internal Memory :
  EPROM 16K bytes
  RAM 256 bytes
  EEPROM 512 bytes
  224 general purpose registers available as RAM, accumulators or index pointers (Register File)
- 80-pin Plastic Quad Flat Pack package for ST90T40Q
- 68-lead Plastic Leaded Chip Carrier package for ST90T40C
- 80-pin Windowed Ceramic Quad Flat Pack package for ST90E40G
- 68-lead Windowed Ceramic Leaded Chip Carrier package for ST90E40L
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 56 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile Development tools, including assembler, linker, C-compiler, archiver, graphic oriented debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9040 16K ROM device



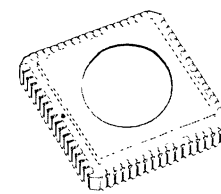
PQFP80



PLCC68



CQFP80W



CLCC68W

(Ordering Information at the end of the Datasheet)

Figure 1. 80 Pin QFP Package

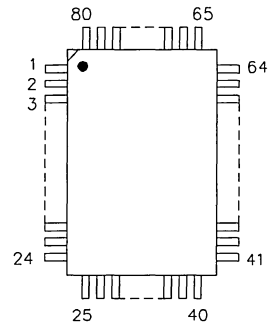


Table 1. ST90E40G-ST90T40Q Pin Description

Pin	Name
1	AV_{SS}
2	AV_{SS}
3	NC
4	P44/Ain4
5	P57
6	P56
7	P55
8	P54
9	INT7
10	INT0
11	P53
12	NC
13	P52
14	P51
15	P50
16	OSCOUT
17	V_{SS}
18	V_{SS}
19	NC
20	OSCIN
21	$\overline{RESET}/V_{PP}$
22	P37/T1OUTB
23	P36/T1INB
24	P35/T1OUTA

Pin	Name
25	P34/T1INA
26	P33/T0OUTB
27	P32/T0INB
28	P31/T0OUTA
29	P30/P/$\overline{D}$/T0INA
30	P17/A15
31	P16/A14
32	NC
33	P15/A13
34	P14/A12
35	P13/A11
36	P12/A10
37	P11/A9
38	P10/A8
39	P00/A0/D0
40	P01/A1/D1

Pin	Name
64	P20/NMI
63	NC
62	V_{SS}
61	P70/SIN
60	P71/SOUT
59	P72/INT4/TXCLK /CLKOUT
58	P73/INT5 /RXCLK/ADTRG
57	P74/P/$\overline{D}$/INT6
56	P75/$\overline{WAIT}$
55	P76/WDOUT /$\overline{BUSREQ}$
54	P77/WDIN /$\overline{BUSACK}$
53	R/$\overline{W}$
52	NC
51	$\overline{DS}$
50	$\overline{AS}$
49	NC
48	V_{DD}
47	V_{DD}
46	P07/A7/D7
45	P06/A6/D6
44	P05/A5/D5
43	P04/A4/D4
42	P03/A3/D3
41	P02/A2/D2

Pin	Name
80	AV_{DD}
79	NC
78	P47/Ain7
77	P46/Ain6
76	P45/Ain5
75	P43/Ain3
74	P42/Ain2
73	P41/Ain1
72	P40/Ain0
71	P27/RRDY5
70	P26/INT3 /$\overline{RDSTB5}$/P/$\overline{D}$
69	P25/WRRDY5
68	P24/INT1 /$\overline{WRSTB5}$
67	P23/SDO
66	P22/INT2/SCK
65	P21/SDI/P/$\overline{D}$

Figure 2. 68 Pin LCC Package

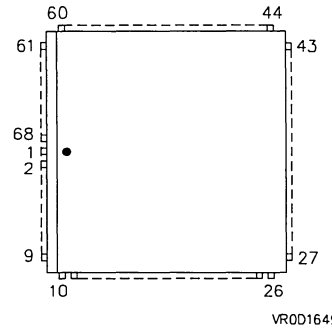


Table 2. ST90E40L-ST90T40C

Pin	Name
61	P44/Ain4
62	P57
63	P56
64	P55
65	P54
66	INT7
67	INT0
68	P53
λ 1	P52
2	P51
3	P50
4	OSCOUT
5	V_{SS}
6	OSCIN
7	$\overline{RESET}/V_{PP}$
8	P37/T1OUTB
9	P36/T1INB

Pin	Name
10	P35/T1OUTA
11	P34/T1INA
12	P33/T0OUTB
13	P32/T0INB
14	P31/T0OUTA
15	P30/P/$\overline{D}$/T0INA
16	P17/A15
17	P16/A14
18	P15/A13
19	P14/A12
20	P13/A11
21	P12/A10
22	P11/A9
23	P10/A8
24	P00/A0/D0
25	P01/A1/D1
26	P02/A2/D2

Pin	Name
43	P70/SIN
42	P71/SOUT
41	P72/CLKOUT /TXCLK/INT4
40	P73/ADTRG /RXCLK/INT5
39	P74/P/$\overline{D}$/INT6
38	$\overline{P75/WAIT}$
37	P76/WDOUT /$\overline{BUSREQ}$
36	P77/WDIN /$\overline{BUSACK}$
35	R/$\overline{W}$
34	DS
33	$\overline{AS}$
32	V_{DD}
31	P07/A7/D7
30	P06/A6/D6
29	P05/A5/D5
28	P04/A4/D4
27	P03/A3/D3

Pin	Name
60	AV_{SS}
59	AV_{DD}
58	P47/Ain7
57	P46/Ain6
56	P45/Ain5
55	P43/Ain3
54	P42/Ain2
53	P41/Ain1
52	P40/Ain0
51	P27/RRDY5
50	P26/INT3 /RDSTB5/P/$\overline{D}$
49	P25/WRRDY5
48	P24/INT1 /WRSTB5
47	P23/SDO
46	P22/INT2/SCK
45	P21/SDI/P/$\overline{D}$
44	P20/NMI

1.1 GENERAL DESCRIPTION

The ST90E40 and ST90T40 (following mentioned as ST90E40)are EPROM members with EEPROM of the ST9 family of microcontrollers, in windowed ceramic (E) and plastic OTP (T) packages respectively, completely developed and produced by SGS-THOMSON Microelectronics using a n-well proprietary HCMOS process.

The EPROM parts are fully compatible with their ROM versions and this datasheet will thus provide only information specific to the EPROM based devices.

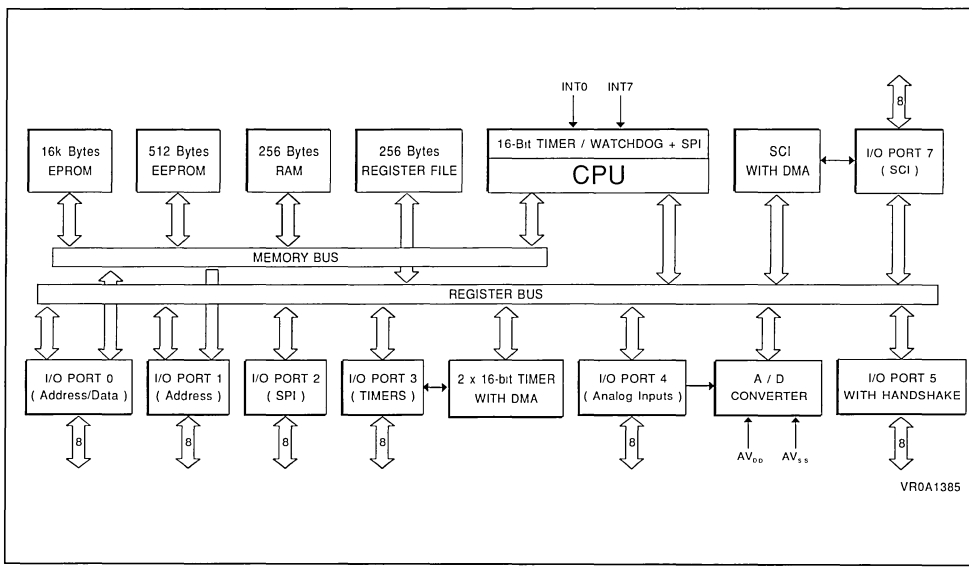
THE READER IS ASKED TO REFER TO THE DATASHEET OF THE ST9040 ROM-BASED DEVICE FOR FURTHER DETAILS.

The EPROM ST90E40 may be used for the prototyping and pre-production phases of development, and can be configured as: a standalone microcontroller with 16K bytes of on-chip ROM, a microcontroller able to manage external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

The nucleus of the ST90E40 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-bus, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90E40 with up to 56 I/O lines dedicated to digital Input/Output. These lines are grouped into up to seven 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing, status signals, an address/data bus for interfacing external memory, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Figure 3. ST90E40 Block Diagram



GENERAL DESCRIPTION (Continued)

Three basic memory spaces are available to support this wide range of configurations: Program Memory (internal and external), Data Memory (external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other systemmsiming functions by the usage of the two associated DMA channels for each timer.

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375,000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

1.2 PIN DESCRIPTION

$\overline{AS}$. *Address Strobe (output, active low, 3-state).* Address Strobe is pulsed low once at the beginning of each memory cycle. The rising edge of $\overline{AS}$ indicates that address, Read/Write (R/$\overline{W}$), and Data Memory signals are valid for program or data memory transfers. Under program control, $\overline{AS}$ can be placed in a high-impedance state along with Port 0 and Port 1, Data Strobe ($\overline{DS}$) and R/$\overline{W}$.

$\overline{DS}$. *Data Strobe (output, active low, 3-state).* Data Strobe provides the timing for data movement to or from Port 0 for each memory transfer. During a write cycle, data out is valid at the leading edge of $\overline{DS}$. During a read cycle, Data In must be valid prior to the trailing edge of $\overline{DS}$. When the ST9040 accesses on-chip memory, $\overline{DS}$ is held high during the whole memory cycle. It can be placed in a high impedance state along with Port 0, Port 1, $\overline{AS}$ and R/$\overline{W}$.

R/$\overline{W}$. *Read/Write (output, 3-state).* Read/Write determines the direction of data transfer for external memory transactions. R/$\overline{W}$ is low when writing to external program or data memory, and high for all other transactions. It can be placed in a high impedance state along with Port 0, Port 1, $\overline{AS}$ and $\overline{DS}$.

$\overline{RESET}$/V_{PP}. *Reset (input, active low) or V_{PP} (input).* The ST9 is initialised by the Reset signal. With the deactivation of $\overline{RESET}$, program execution begins from the Program memory location pointed to by the vector contained in program memory locations 00h and 01h. In the EPROM programming Mode, this pin acts as the programming voltage input VPP.

OSCIN, OSCOUT. *Oscillator (input and output).* These pins connect a parallel-resonant crystal (24MHz maximum), or an external source to the on-chip clock oscillator and buffer. OSCIN is the input of the oscillator inverter and internal clock generator; OSCOUT is the output of the oscillator inverter.

AV_{DD}. Analog V_{DD} of the Analog to Digital Converter.

AV_{SS}. Analog V_{SS} of the Analog to Digital Converter. *Must be tied to V_{SS}.*

V_{DD}. Main Power Supply Voltage (5V ± 10%)

V_{SS}. Digital Circuit Ground.

P0.0-P0.7, P1.0-P1.7, P2.0-P2.7 P3.0-P3.7, P4.0-P4.7, P5.0-P5.7, P7.0-P7.7 *I/O Port Lines (Input/Output, TTL or CMOS compatible).* 56 lines grouped into I/O ports of 8 bits, bit programmable under program control as general purpose I/O or as alternate functions.

1.3 I/O PORT ALTERNATE FUNCTIONS

Each pin of the I/O ports of the ST90E40/T40 may assume software programmable Alternative Functions as shown in the Pin Configuration Tables. Due to Bonding options for the packages, some functions may not be present, Table 3 shows the Functions allocated to each I/O Port pin and a summary of packages for which they are available.

PIN DESCRIPTION (Continued)

Table 3. ST90E40, T40 I/O Port Alternate Function Summary

I/O PORT	Name	Function	Alternate Function	Pin Assignment	
Port. bit				PLCC	PQFP
P0.0	A0/D0	I/O	Address/Data bit 0 mux	24	39
P0.1	A1/D1	I/O	Address/Data bit 1 mux	25	40
P0.2	A2/D2	I/O	Address/Data bit 2 mux	26	41
P0.3	A3/D3	I/O	Address/Data bit 3 mux	27	42
P0.4	A4/D4	I/O	Address/Data bit 4 mux	28	43
P0.5	A5/D5	I/O	Address/Data bit 5 mux	29	44
P0.6	A6/D6	I/O	Address/Data bit 6 mux	30	45
P0.7	A7/D7	I/O	Address/Data bit 7 mux	31	46
P1.0	A8	O	Address bit 8	23	38
P1.1	A9	O	Address bit 9	22	37
P1.2	A10	O	Address bit 10	21	36
P1.3	A11	O	Address bit 11	20	35
P1.4	A12	O	Address bit 12	19	34
P1.5	A13	O	Address bit 13	18	33
P1.6	A14	O	Address bit 14	17	31
P1.7	A15	O	Address bit 15	16	30
P2.0	NMI	I	Non-Maskable Interrupt	44	64
P2.0	ROMless	I	ROMless Select (Mask option)	44	64
P2.1	P/$\overline{D}$	O	Program/Data Space Select	45	65
P2.1	SDI	I	SPI Serial Data Out	45	65
P2.2	INT2	I	External Interrupt 2	46	66
P2.2	SCK	O	SPI Serial Clock	46	66
P2.3	SDO	O	SPI Serial Data In	47	67
P2.4	INT1	I	External Interrupt 1	48	68
P2.4	$\overline{WRSTB5}$	I	Handshake Write Strobe P5	48	68
P2.5	WRRDY5	O	Handshake Write Ready P5	49	69
P2.6	INT3	I	External Interrupt 3	50	70
P2.6	$\overline{RDSTB5}$	I	Handshake Read Strobe P5	50	70
P2.6	P/$\overline{D}$	O	Program/Data Space Select	50	70
P2.7	RDRDY5	O	Handshake Read Ready P5	51	71
P3.0	T0INA	I	MF Timer 0 Input A	15	29
P3.0	P/$\overline{D}$	O	Program/Data Space Select	15	29
P3.1	T0OUTA	O	MF Timer 0 Output A	14	28
P3.2	T0INB	I	MF Timer 0 Input B	13	27
P3.3	T0OUTB	O	MF Timer 0 Output B	12	26
P3.4	T1INA	I	MF Timer 1 Input A	11	25

PIN DESCRIPTION (Continued)

Table 4. ST90E40, T40 I/O Port Alternate Function Summary

I/O PORT	Name	Function	Alternate Function	Pin Assignment	
Port. bit				PLCC	PQFP
P3.5	T1OUTA	O	MF Timer 1 Output A	10	24
P3.6	T1INB	I	MF Timer 1 Input B	9	23
P3.7	T1OUTB	O	MF Timer 1 Output B	8	22
P4.0	Ain0	I	A/D Analog Input 0	52	72
P4.1	Ain1	I	A/D Analog Input 1	53	73
P4.2	Ain2	I	A/D Analog Input 2	54	74
P4.3	Ain3	I	A/D Analog Input 3	55	75
P4.4	Ain4	I	A/D Analog Input 4	61	4
P4.5	Ain5	I	A/D Analog Input 5	56	76
P4.6	Ain6	I	A/D Analog Input 6	57	77
P4.7	Ain7	I	A/D Analog Input 7	58	78
P5.0		I/O	I/O Handshake Port 5	3	15
P5.1		I/O	I/O Handshake Port 5	2	14
P5.2		I/O	I/O Handshake Port 5	1	13
P5.3		I/O	I/O Handshake Port 5	68	11
P5.4		I/O	I/O Handshake Port 5	65	8
P5.5		I/O	I/O Handshake Port 5	64	7
P5.6		I/O	I/O Handshake Port 5	63	6
P5.7		I/O	I/O Handshake Port 5	62	5
P7.0	SIN	I	SCI Serial Input	43	61
P7.1	SOUT	O	SCI Serial Output	42	60
P7.1	ROMless	I	ROMless Select (Mask option)	42	60
P7.2	INT4	I	External Interrupt 4	41	59
P7.2	TXCLK	I	SCI Transmit Clock Input	41	59
P7.2	CLKOUT	O	SCI Byte Sync Clock Output	41	59
P7.3	INT5	I	External Interrupt 5	40	58
P7.3	RXCLK	I	SCI Receive Clock Input	40	58
P7.3	ADTRG	I	A/D Conversion Trigger	40	58
P7.4	INT6	I	External Interrupt 6	39	57
P7.4	P/$\overline{D}$	O	Program/Data Space Select	39	57
P7.5	$\overline{WAIT}$	I	External Wait Input	38	56
P7.6	WDOUT	O	T/WD Output	37	55
P7.6	$\overline{BUSREQ}$	I	External Bus Request	37	55
P7.7	WDIN	I	T/WD Input	36	54
P7.7	$\overline{BUSACK}$	O	External Bus Acknowledge	36	54

1.4 MEMORY

The memory of the ST90E40 is functionally divided into two areas, the Register File and Memory. The Memory is divided into two spaces, each having a maximum of 65,536 bytes. The two memory spaces are separated by function, one space for Program code, the other for Data. The ST90E40 16K bytes of on-chip EPROM memory are selected at memory addresses 0 through 3FFFh (hexadecimal) in the PROGRAM space, while the ST90T40 OTP version has the top 64 bytes of the EPROM reserved by SGS-THOMSON for testing purposes. The DATA space includes the 512 bytes of on-chip EEPROM at addresses 0 through 1FFh and the 256 bytes of on-chip RAM memory at memory addresses 200h through 2FFh.

WARNING. *The ST90T40 has its 64 upper bytes in the internal EPROM reserved for testing purpose.*

External memory may be addressed using the multiplexed address and data buses (Alternate Functions of Ports 0 and 1). At addresses greater than the first 16K of program space, the ST90E40 executes external memory cycles for instruction fetches. Additional Data Memory may be decoded externally by using the P/$\overline{D}$ Alternate Function output. The on-chip general purpose (GP) Registers may also be used as RAM memory for minimum chip count systems.

1.5 EPROM PROGRAMMING

The 16384 bytes of EPROM memory of the ST90E40 (16320 for the ST90T40) may be programmed by using the EPROM Programming Boards (EPB) available from SGS-THOMSON.

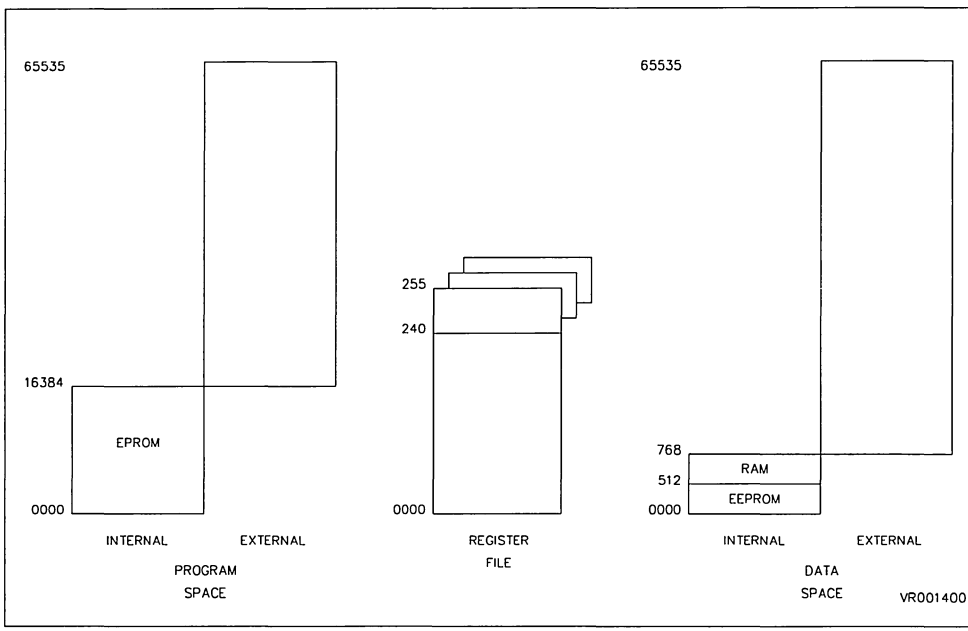
1.5.1 Eprom Erasing

The EPROM of the windowed package of the ST90E40 may be erased by exposure to Ultra-Violet light.

The erasure characteristic of the ST90E40 is such that erasure begins when the memory is exposed to light with a wave lengths shorter than approximately 4000Å. It should be noted that sunlight and some types of fluorescent lamps have wavelengths in the range 3000-4000Å. It is thus recommended that the window of the ST90E40 packages be covered by an opaque label to prevent unintentional erasure problems when testing the application in such an environment.

The recommended erasure procedure of the EPROM is the exposure to short wave ultraviolet light which have a wave-length 2537Å. The integrated dose (i.e. U.V. intensity x exposure time) for erasure should be a minimum of 15W-sec/cm2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with 12000µW/cm^2 power rating. The ST90E40 should be placed within 2.5cm (1Inch) of the lamp tubes during erasure.

Figure 4. Memory Spaces



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{DD}	Supply Voltage	– 0.3 to 7.0	V
AV_{DD}, AV_{SS}	Analog Supply Voltage	$V_{SS} = AV_{SS} < AV_{DD} \leq V_{DD}$	V
V_I	Input Voltage	– 0.3 to V_{DD} +0.3	V
V_O	Output Voltage	– 0.3 to V_{DD} +0.3	V
V_{PP}	Input Voltage on V_{PP} Pin	-0.3 to 13.5	V
T_{STG}	Storage Temperature	– 55 to + 150	°C
I_{INJ}	Pin Injection Current Digital	-5 to 5	mA
I_{INJ}	Pin Injection Current Analog	-5 to 5	mA
	Maximum accumulated pin injection Current in the device	-50 to 50	mA

Note Stresses above those listed as "absolute maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability. All voltages are referenced to V_{SS}

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Value		Unit
		Min.	Max.	
T_A	Operating Temperature	– 40	85	°C
V_{DD}	Operating Supply Voltage	4.5	5.5	V
f_{OSCE}	External Oscillator Frequency		24	MHz
f_{OSCI}	Internal Clock Frequency (INTCLK)		12	MHz

DC ELECTRICAL CHARACTERISTICS

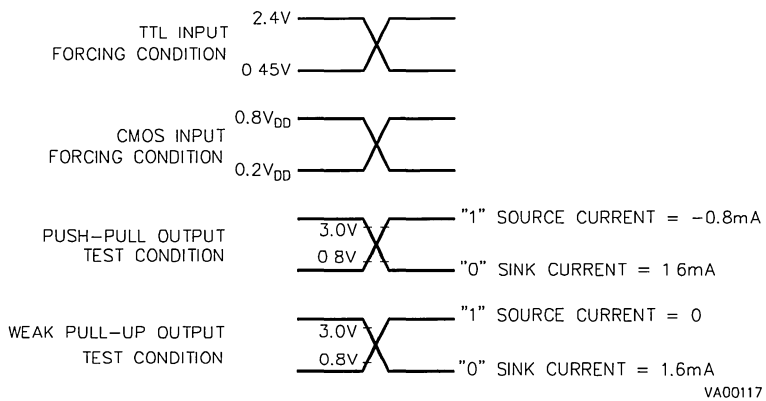
(V_{DD} = 5V ± 10% T_A = – 40°C to + 85°C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Value			Unit
			Min.	Typ.	Max.	
V_{IHCK}	Clock Input High Level	External Clock	0.7 V_{DD}		V_{DD} + 0.3	V
V_{ILCK}	Clock Input Low Level	External Clock	– 0.3		0.3 V_{DD}	V
V_{IH}	Input High Level	TTL	2.0		V_{DD} + 0.3	V
		CMOS	0.7 V_{DD}		VDD + 0.3	V
V_{IL}	Input Low Level	TTL	– 0.3		0.8	V
		CMOS	– 0.3		0.3 V_{DD}	V
V_{IHRS}	$\overline{\text{RESET}}$ Input High Level		0.7 V_{DD}		V_{DD} + 0.3	V
V_{ILRS}	$\overline{\text{RESET}}$ Input Low Level		–0.3		0.3 V_{DD}	V
V_{HYRS}	$\overline{\text{RESET}}$ Input Hysteresis		0.3		1.5	V
V_{OH}	Output High Level	Push Pull, Iload = – 0.8mA	V_{DD} – 0.8			V
V_{OL}	Output Low Level	Push Pull or Open Drain, Iload = 1.6mA			0.4	V

DC ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Value			Unit
			Min.	Typ.	Max.	
I_{WPU}	Weak Pull-up Current	Bidirectional Weak Pull-up, V_{OL} = 0V	– 50	– 200	– 420	µA
I_{APU}	Active Pull-up Current, for INT0 and INT7 only	V_{IN} < 0.8V, under Reset	– 80	– 200	– 420	µA
I_{LKIO}	I/O Pin Input Leakage	Input/Tri-State, 0V < V_{IN} < V_{DD}	– 10		+ 10	µA
I_{LKRS}	$\overline{\text{Reset}}$ Pin Input Leakage	0V < V_{IN} < V_{DD}	– 30		+ 30	µA
I_{LKAD}	A/D Pin Input Leakage	Alternate Function, Open Drain, 0V < V_{IN} < V_{DD}	– 13		+13	µA
I_{LKAP}	Active Pull-up Input Leakage	0V < V_{IN} < 0.8V	– 10		+ 10	µA
I_{LKOS}	OSCIN Pin Input Leakage	0V < V_{IN} < V_{DD}	– 10		+ 10	µA
V_{PP}	EPROM Programming Voltage		12.2	12.5	12.8	V
I_{PP}	EPROM Programming Current				30	mA

DC TEST CONDITIONS

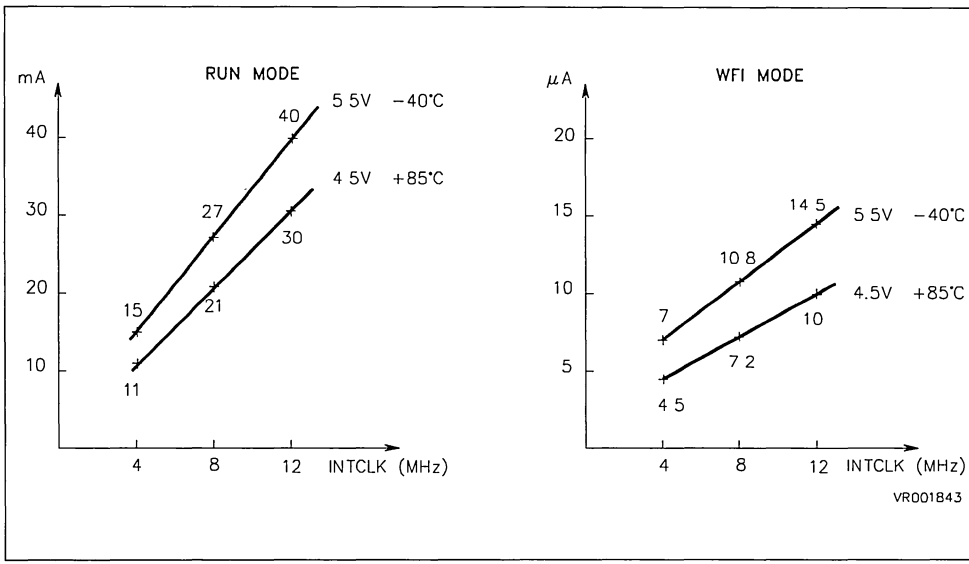


AC ELECTRICAL CHARACTERISTICS

(V_{DD} = 5V ± 10% T_A = – 40°C to + 85°C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Value			Unit
			Min.	Typ.	Max.	
I_{DD}	Run Mode Current no CPUCLK prescale, Clock divide by 2	24MHz		40	70	mA
I_{DP2}	Run Mode Current Prescale by 2 Clock divide by 2	24MHz		19	40	mA
I_{WFI}	WFI Mode Current no CPUCLK prescale, Clock divide by 2	24MHz		15	20	mA
I_{HALT}	HALT Mode Current	24MHz		50	100	µA

Typical Current Versus Frequency of Operation (f_{OSC})



CLOCK TIMING TABLE

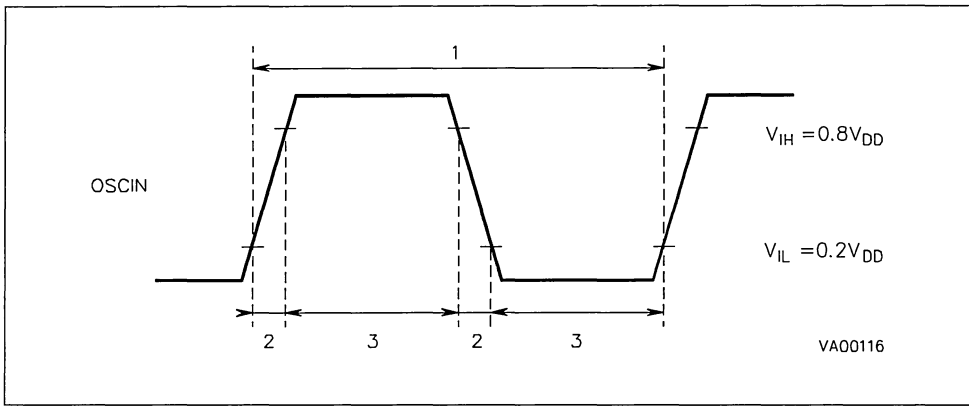
(V_{DD} = 5V ± 10%, T_A = − 40°C to + 85°C, INTCLK = 12MHz, unless otherwise specified

N°	Symbol	Parameter	Value		Unit	Note
			Min.	Max.		
1	TpC	OSCIN Clock Period	41.5		ns	1
			83		ns	2
2	TrC, TfC	OSCIN Rise and Fall Time		12	ns	
3	TwCL, TwCH	OSCIN Low and High Width	17	25	ns	1
			38		ns	2

Notes:

1. Clock divided by 2 internally (MODER DIV2=1)
2. Clock not divided by 2 internally (MODER.DIV2=0)

CLOCK TIMING



EXTERNAL BUS TIMING TABLE (V_{DD} = 5V ± 10%, T_A = – 40°C to + 85°C, Cload = 50pF, CPUCLK = 12MHz, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TsA (AS)	Address Set-up Time before $\overline{AS}$ ↑	TpC (2P+1) –22	TWCH+PTpC –18	20		ns
2	ThAS (A)	Address Hold Time after $\overline{AS}$ ↑	TpC –17	TwCL –13	25		ns
3	TdAS (DR)	$\overline{AS}$ ↑ to Data Available (read)	TpC (4P+2W+4) –52	TpC (2P+W+2) –51		115	ns
4	TwAS	$\overline{AS}$ Low Pulse Width	TpC (2P+1) –7	TwCH+PTpC –3	35		ns
5	TdAz (DS)	$\overline{DS}$ ↓ to Address Float			12		ns
6	TwDSR	$\overline{DS}$ Low Pulse Width (read)	TpC (4P+2W+3) –20	TwCH+TpC (2P+W+1) –16	105		ns
7	TwDSW	$\overline{DS}$ Low Pulse Width (write)	TpC (2P+2W+2) –13	TpC (P+W+1) –13	70		ns
8	TdDSR (DR)	$\overline{DS}$ ↓ to Data Valid Delay (read)	TpC (4P+2W-3) –50	TwCH+TpC(2P+W+1) –46		75	ns
9	ThDR (DS)	Data to $\overline{DS}$ ↑ Hold Time (read)	0	0	0		ns
10	TdDS (A)	$\overline{DS}$ ↑ to Address Active Delay	TpC –7	TwCL –3	35		ns
11	TdDS (AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	TpC –18	TwCL –14	24		ns
12	TsR/W (AS)	R/$\overline{W}$ Set-up Time before $\overline{AS}$ ↑	TpC (2P+1) –22	TwCH+PTpC –18	20		ns
13	TdDSR (R/W)	$\overline{DS}$ ↑ to R/$\overline{W}$ and Address Not Valid Delay	TpC –9	TwCL –5	33		ns
14	TdDW (DSW)	Write Data Valid to $\overline{DS}$ ↓ Delay (write)	TpC (2P+1) –32	TwCH+PTpC –28	10		ns
15	ThDS (DW)	Data Hold Time after $\overline{DS}$ ↑ (write)	TpC –9	TwCL –5	33		ns
16	TdA (DR)	Address Valid to Data Valid Delay (read)	TpC (6P+2W+5) –68	TwCH+TpC (3P+W+2) –64		140	ns
17	TdAs (DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	TpC –18	TwCL –14	24		ns

EXTERNAL WAIT TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

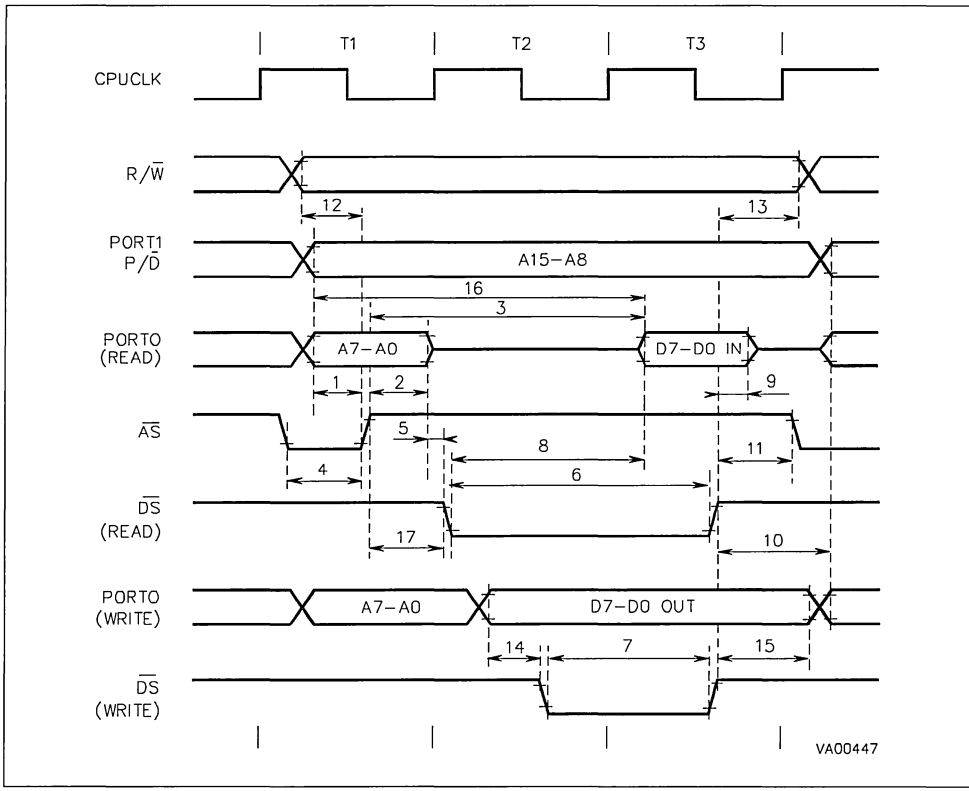
N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TdAs (WAIT)	$\overline{AS}$ ↑ to $\overline{WAIT}$ ↓ Delay	2(P+1)TpC –29	2(P+1)TpC –29		40	ns
2	TdAs (WAIT)	$\overline{AS}$ ↑ to $\overline{WAIT}$ ↓ Min. Delay	2(P+W+1)TpC –4	2(P+W+1)TpC –4	80		ns
3	TdAs (WAIT)	$\overline{AS}$ ↑ to $\overline{WAIT}$ ↓ Max. Delay	2(P+W+1)TpC –29	2(P+W+1)TpC –29		33W+ 40	ns

Note: (for both table) The value in the left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value in the right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescaler value of zero and zero wait status

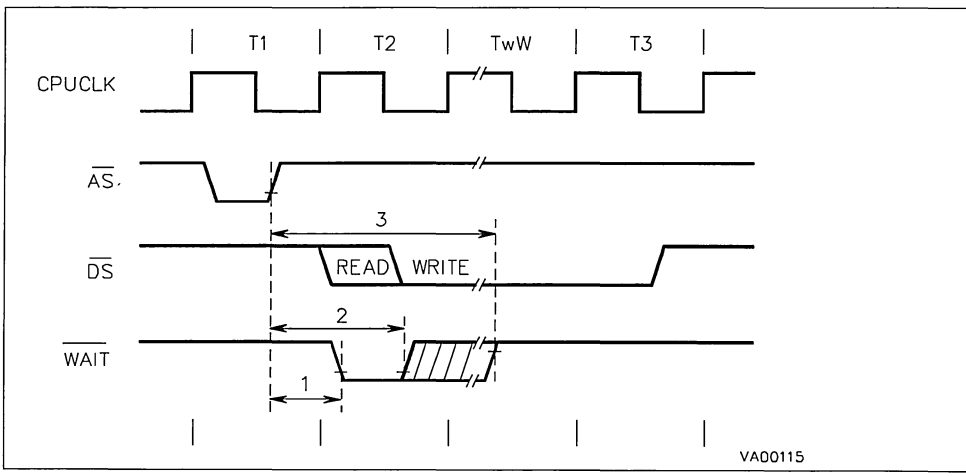
Legend:

P = Clock Prescaling Value
W = Wait Cycles
TpC =OSCIN Period
TwCH =High Level OSCIN half period
TwCL =Low Level OSCIN half period

EXTERNAL BUS TIMING



EXTERNAL WAIT TIMING



HANDSHAKE TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Min.	Max.	Unit
			OSCIN Divided By 2		OSCIN Not Divided By 2				
			Min.	Max.	Min.	Max.			
1	TwRDY	RDRDY, WRRDY Pulse Width in One Line Handshake	2TpC (P+W+1) –18		TpC (P+W+1) – 18		65		ns
2	TwSTB	$\overline{RDSTB}$, $\overline{WRSTB}$ Pulse Width	2TpC+12		TpC+12		95		ns
3	TdST (RDY)	$\overline{RDSTB}$, or $\overline{WRSTB}$ ↑ to RDRDY or WRRDY ↓		TpC+45		(TpC-TwCL) +45		87	ns
4	TsPD (RDY)	Port Data to RDRDY ↑ Set-up Time	(2P+2W+1) TpC –25		TwCH+ (W+P) TpC –25		16		ns
5	TsPD (RDY)	Port Data to WRRDY ↓ Set-up Time in One Line Handshake	43		43		43		ns
6	ThPD (RDY)	Port Data to WRRDY ↓ Hold Time in One Line Handshake	0		0		0		ns
7	TsPD (STB)	Port Data to $\overline{WRSTB}$ ↑ Set-up Time	10		10		10		ns
8	ThPD (STB)	Port Data to WRSTB ↑ Hold Time	25		25		25		ns
9	TdSTB (PD)	$\overline{RDSTBD}$ ↑ to Port Data Delay Time in Bidirectional Handshake		35		35		35	ns
10	TdSTB (PHZ)	$\overline{RDSTB}$ ↑ to Port High-Z Delay Time in Bidirectional Handshake		25		25		25	ns

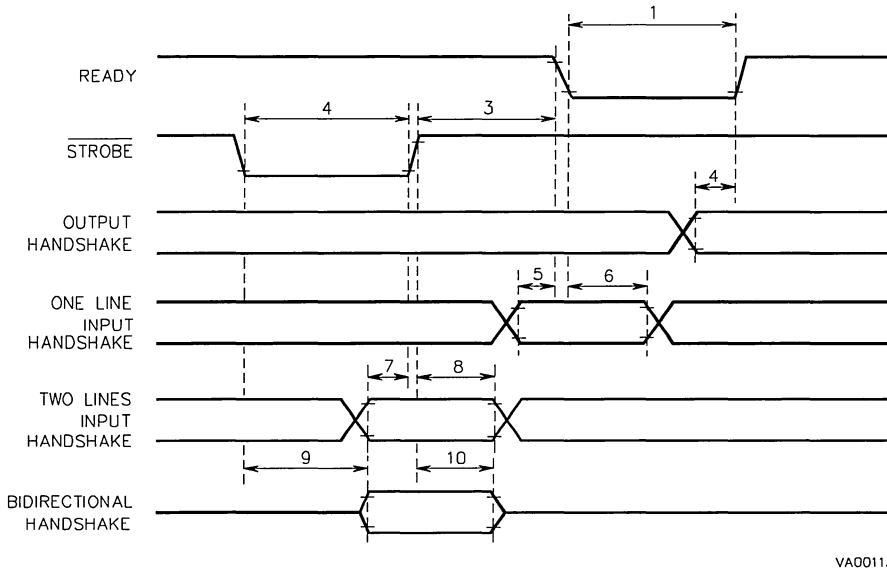
Note: The value in the left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value in the right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescaler value of zero and zero wait status

Legend:

P = Clock Prescaling Value (R235 4,3,2)

W = Programmable Wait Cycles (R252 2.1.0/5,4,3) + External Wait Cycles

HANDSHAKE TIMING

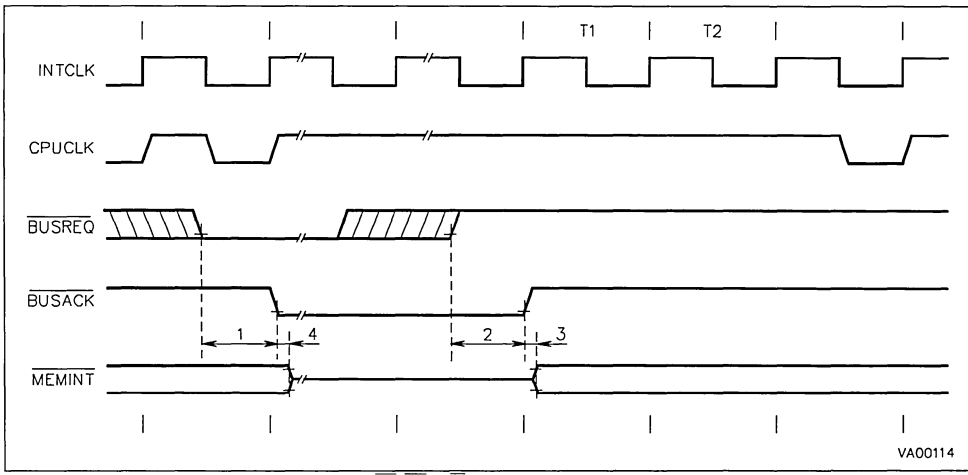


BUS REQUEST/ACKNOWLEDGE TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TdBR (BACK)	$\overline{BREQ}$ ↓ to $\overline{BUSACK}$ ↓	TpC+8	TwCL+12	50		ns
			TpC(6P+2W+7)+65	TpC(3P+W+3)+TwCL+65		360	ns
2	TdBR (BACK)	$\overline{BREQ}$ ↑ to $\overline{BUSACK}$ ↑	3TpC+60	TpC+TwCL+60		185	ns
3	TdBACK (BREL)	$\overline{BUSACK}$ ↓ to Bus Release	20	20		20	ns
4	TdBACK (BACT)	$\overline{BUSACK}$ ↑ to Bus Active	20	20		20	ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted.
The value right hand two columns show the timing minimum and maximum for an external clock at 24MHz divided by 2, prescale value of zero and zero wait status.

BUS REQUEST/ACKNOWLEDGE TIMING



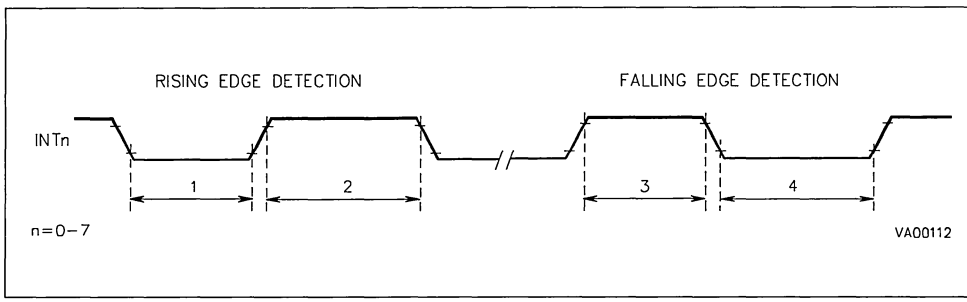
Note : MEMINT = Group of memory interface signals. AS, DS, R/W, P00-P07, P10-P17

EXTERNAL INTERRUPT TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2 Min.	OSCIN Not Divided By 2 Min.	Min.	Max.	
1	TwLR	Low Level Minimum Pulse Width in Rising Edge Mode	2TpC+12	TpC+12	95		ns
2	TwHR	High Level Minimum Pulse Width in Rising Edge Mode	2TpC+12	TpC+12	95		ns
3	TwHF	High Level Minimum Pulse Width in Falling Edge Mode	2TpC+12	TpC+12	95		ns
4	TwLF	Low Level Minimum Pulse Width in Falling Edge Mode	2TpC+12	TpC+12	95		ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescale value of zero and zero wait status.

EXTERNAL INTERRUPT TIMING

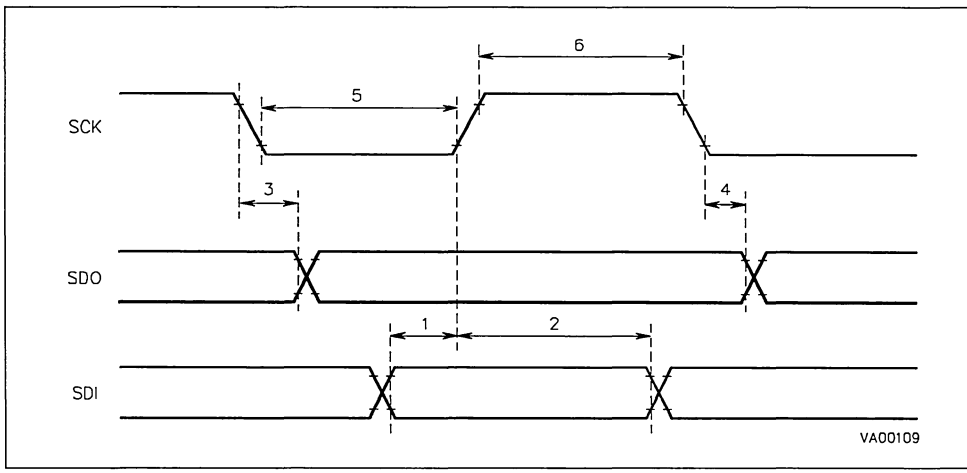


SPI TIMING TABLE (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF, INTCLK = 12MHz, Output Alternate Function set as Push-pull)

N°	Symbol	Parameter	Value		Unit
			Min.	Max.	
1	TsDI	Input Data Set-up Time	100		ns
2	ThDI (1)	Input Data Hold Time	1/2 TpC+100		ns
3	TdOV	SCK to Output Data Valid		100	ns
4	ThDO	Output Data Hold Time	-20		ns
5	TwSKL	SCK Low Pulse Width	300		ns
6	TwSKH	SCK High Pulse Width	300		ns

Note: 1. TpC is the OSCIN Clock period.

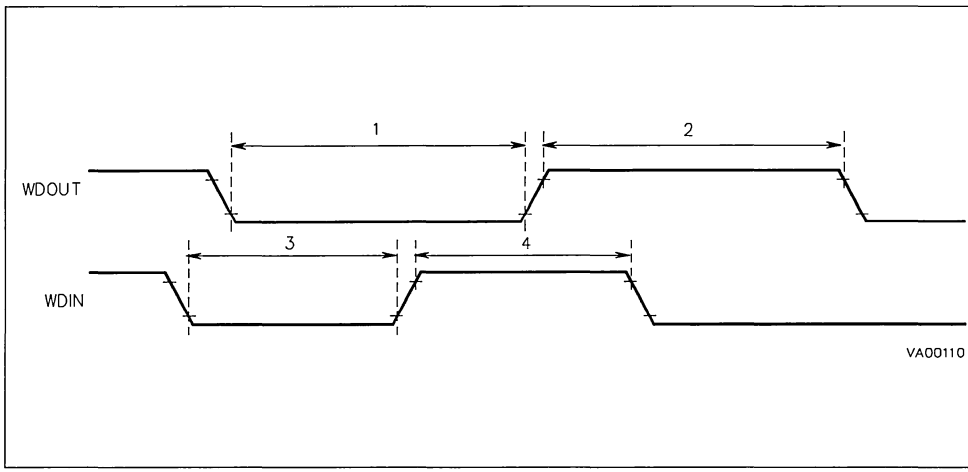
SPI TIMING



WATCHDOG TIMING TABLE(V_{DD} = 5V ± 10%, T_A = – 40°C to +85°C, Cload = 50pF, CPUCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Values		Unit
			Min.	Max.	
1	TwWDOL	WDOUT Low Pulse Width	620		ns
2	TwWDOH	WDOUT High Pulse Width	620		ns
3	TwWDIL	WDIN High Pulse Width	350		ns
4	TwWDIH	WDIN Low Pulse Width	350		ns

WATCHDOG TIMING



A/D CONVERTER

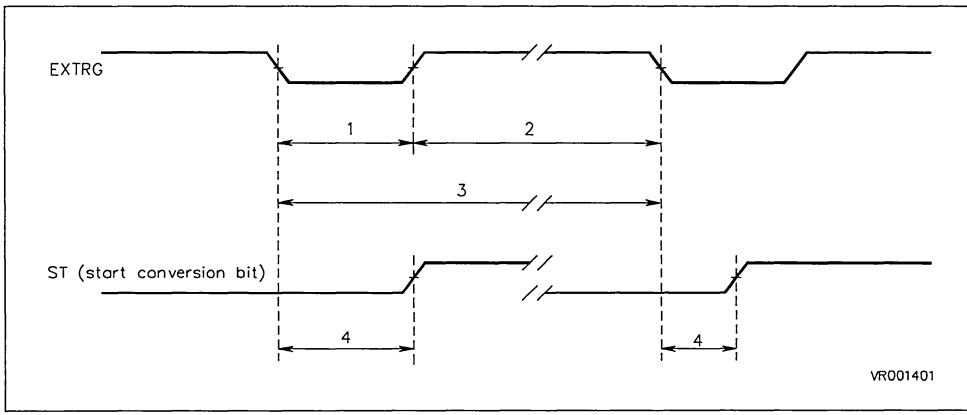
EXTERNAL TRIGGER TIMING (V_{DD} = 5V ± 10%, T_A = –40°C to +85°C, Cload = 50pF)

N°	Symbol	Parameter	Oscin divided by 2 (1)		Oscin not divided (1)		Value(2)		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	
1	T_{LOW}	External Trigger pulse width	$2xT_{PC}$		T_{PC}		83		ns
2	T_{HIGH}	External Trigger pulse	$2xT_{PC}$		T_{PC}		83		ns
3	T_{EXT}	External trigger active edges distance	$138xT_{PC}$		$69xT_{PC}$		5.75		µs
4	T_{STR}	Internal delay between EXTRG falling edge and first conversion start	T_{PC}	$3xT_{PC}$	0.5xTPC	$1.5xT_{PC}$	41.5	125	ns

Notes:

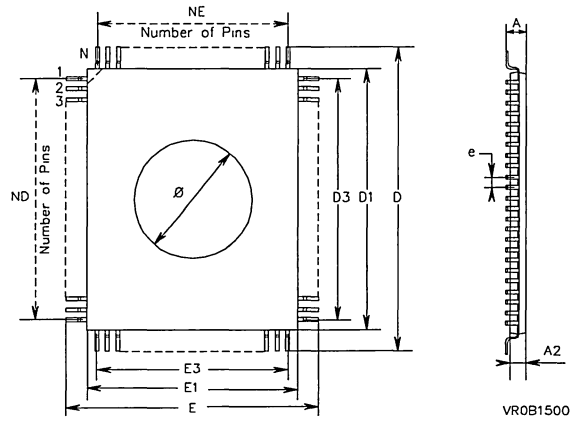
1. Variable clock (T_{PC}=OSCIN clock period)
2. INTCLK=12MHz

A/D External Trigger Timing



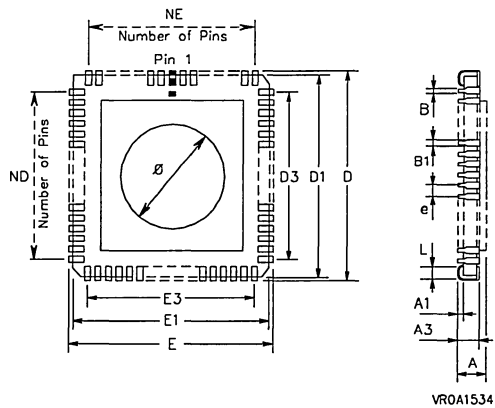
PACKAGE MECHANICAL DATA

80-Pin Ceramic Quad Flat Package with Window



Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A		3.55			0.14	
A2		3.40			0.133	
D		23.90			0.941	
D1		20.00			0.787	
D3		18.40			0.724	
E		17.90			0.705	
E1		14.00			0.551	
E3		12.00			0.472	
Ø		7.62			0.3	
e		0.80			0.032	
Number of Pins						
N	80					
ND	24					
NE	16					

68-Pin Ceramic Leadless Chip Carrier with Window



Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A		4.47			0.176	
A1		0.89			0.035	
A3		-			-	
B		0.48			0.019	
B1		-			-	
D		25.1			0.990	
D1		23.6			0.930	
D3		20.3			0.800	
E		25.1			0.990	
E1		23.6			0.930	
E3		20.3			0.800	
Ø		8			0.32	
e		1.27			0.050	
Number of Pins						
N	64					
ND	16					
NE	16					

ORDERING INFORMATION

Sales Type	Frequency	Temperature Range	Package
ST90E40L1/ES[1]	24MHz	0°C to + 70°C	CLCC68W
ST90E40G1/ES[1]		0°C to + 70°C	CQFP80W
ST90T40C6	24MHz	-40°C to + 85°C	PLCC68
ST90T40Q1		0°C to + 70°C	PQFP80

Note . EPROM parts are tested at 25°C only

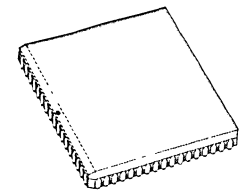
SGS-THOMSON MICROELECTRONICS

ST90R40

ROMLESS HCMOS MCU WITH EEPROM, RAM AND A/D CONVERTER

PRELIMINARY DATA

- Register oriented 8/16 bit CORE with RUN, WFI and HALT modes
- Minimum instruction cycle time:500ns (12MHz internal)
- ROMless to allow maximum external memory flexibility
- Internal Memory :
  RAM 256 bytes
  EEPROM 512 bytes
  224 general purpose registers available as RAM, accumulators or index pointers (register file)
- 68-lead Plastic Leaded Chip Carrier package
- DMA controller, Interrupt handler and Serial Peripheral Interface as standard features
- 40 fully programmable I/O pins
- Up to 8 external plus 1 non-maskable interrupts
- 16 bit Timer with 8 bit Prescaler, able to be used as a Watchdog Timer
- Two 16 bit Multifunction Timers, each with an 8 bit prescaler and 13 operating modes
- 8 channel 8 bit Analog to Digital Converter, with Analog Watchdogs and external references
- Serial Communications Interface with asynchronous and synchronous capability
- Rich Instruction Set and 14 Addressing modes
- Division-by-Zero trap generation
- Versatile development tools, including assembler, linker, C-compiler, archiver, graphic orinted debugger and hardware emulators
- Real Time Operating System
- Compatible with ST9040 16K ROM device (also available in windowed and One Time Programmable EPROMpackages)



PLCC68

(Ordering Information at the end of the Datasheet)

Figure 1. 68 Pin PLCC Package)

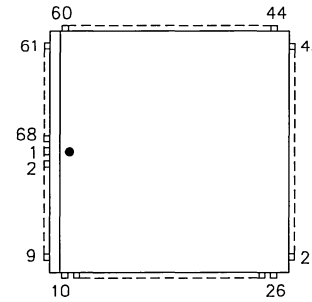


Table 1. ST9040C Pin Description

Pin	Name
61	P44/Ain4
62	P57
63	P56
64	P55
65	P54
66	INT7
67	INT0
68	P53
λ 1	P52
2	P51
3	P50
4	OSCOUT
5	V_{SS}
6	OSCIN
7	$\overline{RESET}$
8	P37/T1OUTB
9	P36/T1INB

Pin	Name
10	P35/T1OUTA
11	P34/T1INA
12	P33/T0OUTB
13	P32/T0INB
14	P31/T0OUTA
15	P30/P/$\overline{D}$/T0INA
16	A15
17	A14
18	A13
19	A12
20	A11
21	A10
22	A9
23	A8
24	A0/D0
25	A1/D1
26	A2/D2

Pin	Name
43	P70/SIN
42	P71/SOUT
41	P72/CLKOUT /TXCLK/INT4
40	P73/ADTRG /RXCLK/INT5
39	P74/P/$\overline{D}$/INT6
38	$\overline{P75/WAIT}$
37	P76/WDOUT /BUSREQ
36	P77/WDIN /BUSACK
35	R/W
34	DS
33	$\overline{AS}$
32	V_{DD}
31	A7/D7
30	A6/D6
29	A5/D5
28	A4/D4
27	A3/D3

Pin	Name
60	AV_{SS}
59	AV_{DD}
58	P47/Ain7
57	P46/Ain6
56	P45/Ain5
55	P43/Ain3
54	P42/Ain2
53	P41/Ain1
52	P40/Ain0
51	P27/RRDY5
50	P26/INT3 /RDSTB5/P/D
49	P25/WRRDY5
48	P24/INT1 /WRSTB5
47	P23/SDO
46	P22/INT2/SCK
45	P21/SDI/P/$\overline{D}$
44	P20/NMI

1.1 GENERAL DESCRIPTION

The ST90R40 is a ROMLESS member of the ST9 family of microcontrollers, completely developed and produced by SGS-THOMSON Microelectronics using a proprietary n-well HCMOS process.

The ROMLESS part may be used for the prototyping and pre-production phases of development, and offers the maximum in program flexibility in production systems.

The ST90R40 is fully compatible with the ST9040 ROM version and this datasheet will thus provide only information specific to the ROMLESS device.

THE READER IS ASKED TO REFER TO THE DATASHEET OF THE ST9040 ROM-BASED DEVICE.

The ROMLESS ST90R40 can be configured as a microcontroller able to manage external memory, or as a parallel processing element in a system with other processors and peripheral controllers.

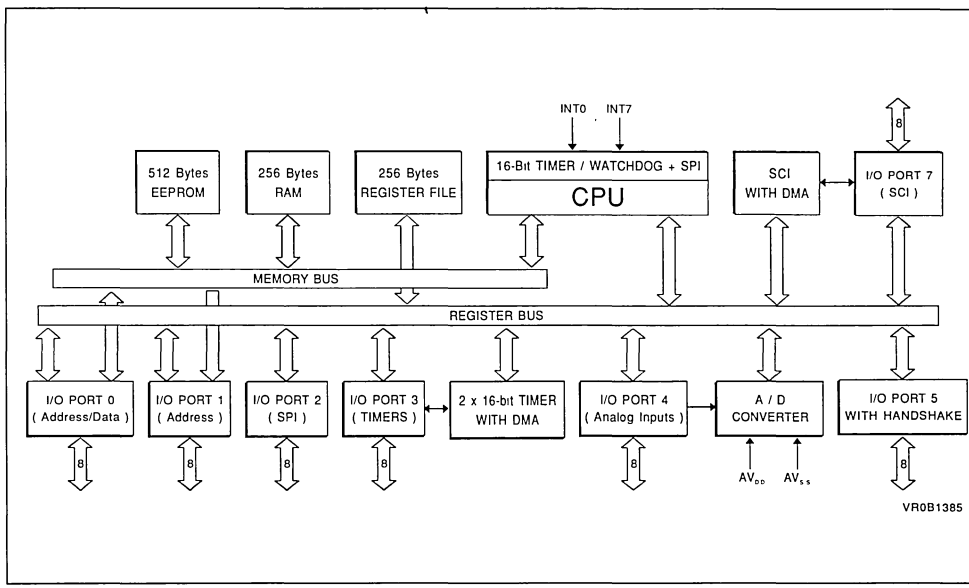
The nucleus of the ST90R40 is the advanced Core which includes the Central Processing Unit (CPU), the Register File, a 16 bit Timer/Watchdog with 8 bit Prescaler, a Serial Peripheral Interface supporting S-BUS, I^2C-bus and IM-bus Interface, plus two 8 bit I/O ports. The Core has independent memory and register buses allowing a high degree of pipelining to add to the efficiency of the code execution speed of the extensive instruction set.

The powerful I/O capabilities demanded by microcontroller applications are fulfilled by the ST90R40 with up to 48 I/O lines dedicated to digital Input/Output. These lines are grouped into up to six 8 bit I/O Ports and can be configured on a bit basis under software control to provide timing and status signals, address lines, timer inputs and outputs, analog inputs, external interrupts and serial or parallel I/O with or without handshake.

Three memory spaces are available: Program Memory (external), Data Memory (internal and external) and the Register File, which includes the control and status registers of the on-chip peripherals.

Two 16 bit MultiFunction Timers, each with an 8 bit Prescaler and 13 operating modes allow simple use for complex waveform generation and measurement, PWM functions and many other system timing functions by the usage of the two associated DMA channels for each timer.

Figure 2. Block Diagram



GENERAL DESCRIPTION (Continued)

In addition there is an 8 channel Analog to Digital Converter with integral sample and hold, fast 11µs conversion time and 8 bit resolution. An Analog Watchdog feature is included for two input channels.

Completing the device is a full duplex Serial Communications Interface with an integral 110 to 375000 baud rate generator, asynchronous and 1.5Mbyte/s synchronous capability (fully programmable format) and associated address/wake-up option, plus two DMA channels.

1.2 PIN DESCRIPTION

$\overline{AS}$. *Address Strobe (output, active low, 3-state).* Address Strobe is pulsed low once at the beginning of each memory cycle. The rising edge of $\overline{AS}$ indicates that address, Read/Write (R/$\overline{W}$), and Data Memory signals are valid for program or data memory transfers. Under program control, $\overline{AS}$ can be placed in a high-impedance state along with Port 0 and Port 1, Data Strobe ($\overline{DS}$) and R/$\overline{W}$.

$\overline{DS}$. *Data Strobe (output, active low, 3-state).* Data Strobe provides the timing for data movement to or from Port 0 for each memory transfer. During a write cycle, data out is valid at the leading edge of $\overline{DS}$. During a read cycle, Data In must be valid prior to the trailing edge of $\overline{DS}$. When the ST90R40 accesses on-chip Data memory, $\overline{DS}$ is held high during the whole memory cycle. It can be placed in a high impedance state along with Port 0, Port 1, $\overline{AS}$ and R/$\overline{W}$.

R/$\overline{W}$. *Read/Write (output, 3-state).* Read/Write determines the direction of data transfer for memory transactions. R/$\overline{W}$ is low when writing to program or data memory, and high for all other transactions. It can be placed in a high impedance state along with Port 0, Port 1, $\overline{AS}$ and $\overline{DS}$.

$\overline{RESET}$. *Reset (input, active low).* The ST9 is initialised by the Reset signal. With the deactivation of $\overline{RESET}$, program execution begins from the Program memory location pointed to by the vector contained in program memory locations 00h and 01h.

OSCIN, OSCOUT. *Oscillator (input and output).* These pins connect a parallel-resonant crystal (24MHz maximum), or an external source to the on-chip clock oscillator and buffer. OSCIN is the input of the oscillator inverter and internal clock generator; OSCOUT is the output of the oscillator inverter.

AV_{DD}. Analog V_{DD} of the Analog to Digital Converter.

AV_{SS}. Analog V_{SS} of the Analog to Digital Converter. *Must be tied to V_{SS}.*

V_{DD}. Main Power Supply Voltage (5V±10%)

V_{SS}. Digital Circuit Ground.

AD0-AD7, (P0.0-P0.7) *Address/Data Lines (Input/Output, TTL or CMOS compatible).* 8 lines providing a multiplexed address and data bus, under control of the $\overline{AS}$ and $\overline{DS}$ timing signals.

A8-A15 *Address Lines (Output, TTL or CMOS compatible).* 8 lines providing non-multiplexing address bus, under control of the $\overline{AS}$ and $\overline{DS}$ timing signals.

P2.0-P2.7 P3.0-P3.7, P4.0-P4.7, P5.0-P5.7, P7.0-P7.7 *I/O Port Lines (Input/Output, TTL or CMOS compatible).* 40 lines grouped into I/O ports of 8 bits, bit programmable under program control as general purpose I/O or as Alternate functions (see next section).

1.3 I/O PORT ALTERNATE FUNCTIONS

Each pin of the I/O ports of the ST90R40 may assume software programmable Alternative Functions as shown in the Pin Configuration Drawings. Table 2 shows the Functions allocated to each I/O Port pins.

PIN DESCRIPTION (Continued)

Table 2. I/O Port Alternate Function Summary

I/O PORT Port.bit	Name	Function IN/OUT	Alternate Function	Pin Number
P0.0	A0/D0	I/O	Address/Data bit 0 mux	24
P0.1	A1/D1	I/O	Address/Data bit 1 mux	25
P0.2	A2/D2	I/O	Address/Data bit 2 mux	26
P0.3	A3/D3	I/O	Address/Data bit 3 mux	27
P0.4	A4/D4	I/O	Address/Data bit 4 mux	28
P0.5	A5/D5	I/O	Address/Data bit 5 mux	29
P0.6	A6/D6	I/O	Address/Data bit 6 mux	30
P0.7	A7/D7	I/O	Address/Data bit 7 mux	31
P1.0	A8	O	Address bit 8	23
P1.1	A9	O	Address bit 9	22
P1.2	A10	O	Address bit 10	21
P1.3	A11	O	Address bit 11	20
P1.4	A12	O	Address bit 12	19
P1.5	A13	O	Address bit 13	18
P1.6	A14	O	Address bit 14	17
P1.7	A15	O	Address bit 15	16
P2.0	NMI	I	Non-Maskable Interrupt	44
P2.1	$P/\overline{D}$	O	Program/Data Space Select	45
P2.1	SDI	I	SPI Serial Data Out	45
P2.2	INT2	I	External Interrupt 2	46
P2.2	SCK	O	SPI Serial Clock	46
P2.3	SDO	O	SPI Serial Data In	47
P2.4	INT1	I	External Interrupt 1	48
P2.4	$\overline{WRSTB5}$	O	Handshake Write Strobe P5	48
P2.5	WRRDY5	I	Handshake Write Ready P5	49
P2.6	INT3	I	External Interrupt 3	50
P2.6	$\overline{RDSTB5}$	I	Handshake Read Strobe P5	50
P2.6	$P/\overline{D}$	O	Program/Data Space Select	50
P2.7	RDRDY5	O	Handshake Read Ready P5	51
P3.0	T0INA	I	MF Timer 0 Input A	15
P3.0	$P/\overline{D}$	O	Program/Data Space Select	15
P3.1	T0OUTA	O	MF Timer 0 Output A	14
P3.2	T0INB	I	MF Timer 0 Input B	13
P3.3	T0OUTB	O	MF Timer 0 Output B	12
P3.4	T1INA	I	MF Timer 1 Input A	11

PIN DESCRIPTION (Continued)

Table 2. I/O Port Alternate Function Summary (Continued)

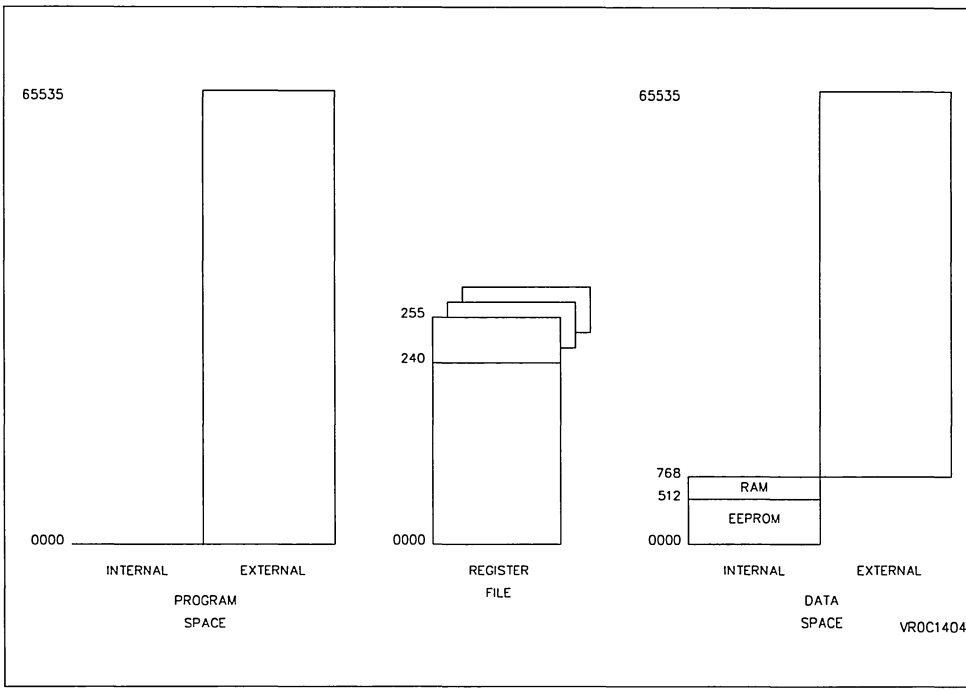
I/O PORT Port.bit	Name	Function IN/OUT	Alternate Function	Pin Number
P3.5	T1OUTA	O	MF Timer 1 Output A	10
P3.6	T1INB	I	MF Timer 1 Input B	9
P3.7	T1OUTB	O	MF Timer 1 Output B	8
P4.0	Ain0	I	A/D Analog Input 0	52
P4.1	Ain1	I	A/D Analog Input 1	53
P4.2	Ain2	I	A/D Analog Input 2	54
P4.3	Ain3	I	A/D Analog Input 3	55
P4.4	Ain4	I	A/D Analog Input 4	61
P4.5	Ain5	I	A/D Analog Input 5	56
P4.6	Ain6	I	A/D Analog Input 6	57
P4.7	Ain7	I	A/D Analog Input 7	58
P5.0		I/O	I/O Handshake Port 5	3
P5.1		I/O	I/O Handshake Port 5	2
P5.2		I/O	I/O Handshake Port 5	1
P5.3		I/O	I/O Handshake Port 5	68
P5.4		I/O	I/O Handshake Port 5	65
P5.5		I/O	I/O Handshake Port 5	64
P5.6		I/O	I/O Handshake Port 5	63
P5.7		I/O	I/O Handshake Port 5	62
P7.0	SIN	I	SCI Serial Input	43
P7.1	SOUT	O	SCI Serial Output	42
P7.2	INT4	I	External Interrupt 4	41
P7.2	TXCLK	I	SCI Transmit Clock Input	41
P7.2	CLKOUT	O	SCI Byte Sync Clock Output	41
P7.3	INT5	I	External Interrupt 5	40
P7.3	RXCLK	I	SCI Receive Clock Input	40
P7.3	ADTRG	I	A/D Conversion Trigger	40
P7.4	INT6	I	External Interrupt 6	39
P7.4	P/$\overline{D}$	O	Program/Data Space Select	39
P7.5	$\overline{WAIT}$	I	External Wait Input	38
P7.6	WDOUT	O	T/WD Output	37
P7.6	$\overline{BUSREQ}$	I	External Bus Request	37
P7.7	WDIN	I	T/WD Input	36
P7.7	$\overline{BUSACK}$	O	External Bus Acknowledge	36

1.4 MEMORY

The memory of the ST90R40 is functionally divided into two areas, the Register File and Memory. The Memory may optionally be divided into two spaces, each having a maximum of 65,536 bytes. The two memory spaces are separated by function, one space for Program code, the other for Data. The ST90R40 addresses all program memory in the external PROGRAM space. The DATA space includes the 512 bytes of on-chip EEPROM at addresses 0 through 1FFh and the 256 bytes of on-chip RAM memory at memory addresses 200h through 2FFh.

The External Memory spaces are addressed using the multiplexed address and data buses on Ports 0 and 1. Additional Data Memory may be decoded externally by using the P/$\overline{D}$ Alternate Function output. The on-chip general purpose (GP) Registers may be used as RAM memory.

Figure 3. Memory Spaces



ORDERING INFORMATION

Sales Type	Frequency	Temperature Range	Package
ST90R40C6	24MHz	-40°C to + 85°C	PLCC68

APPLICATION NOTES

SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

SYMBOLS.INC ST9 REGISTER ADDRESS AND CONTENT NAMES

Pierre Guillemin

INTRODUCTION

This document has been written in order to provide, to the ST9 software programmer, a suggested guide and a clear notation of ST9 register and bit names for standardisation across software modules.

The **SYMBOLS.INC** files give a symbolic definition for:

- each group within the Register File
- each peripheral page
- each ST9 register pair
- each ST9 peripheral or core register
- each ST9 system and peripheral control bit with its associated mask.

This document assumes a previous knowledge of the ST9 architecture and software tools. Please refer to the ST9 Technical Manual, ST9 Programming Manual and ST9 Software Tools manuals for an understanding of the terms used.

INVOCATION

The **SYMBOLS.INC** file or a part it (depending on the application and the peripherals used) must be assembled with each ST9 software module in order to use the symbolic names.

The ST9 Macro Assembler (AST9) provides two methods for using include files:

1) directly in the invocation line of **AST9** (by giving a list of all include files). In this case, an example of the invocation line of **AST9**, implemented within an MS-DOS batch file and with other options, could be the following:

```
AST9 -v -g -r -o %1.obj -l %1.lst c:\ST9\INC\SYMBOLS.INC %1.ST9
```

where the source file name **%1 (.ST9)** is passed as the first batch parameter, and the **SYMBOLS.INC** file is located in the sub-directory **c:\ST9\INC**. Listing **(%1.lst)** and object code **(%1.obj)** files are produced.

2) inside each ST9 module by using the .include pseudo-instruction. In this case the syntax is:

```
.include "c:\ST9\INC\SYMBOLS.INC"
```

This method of using include files from within the ST9 software module has been expanded to allow the use only of the symbols applicable to the target software module or ST9 device. In this case, the **SYMBOLS.INC** file has been split into several include files related to each peripheral (e.g. A/D converter, MFTimer), or to specific features (e.g. the Bank Switch registers for ST905x family or the security register for ST904x family).

These peripheral include files are also associated with four include files named **ST90xx.inc** (ST902x, ST903x, ST904x, ST905x). These list all the related files applicable to the ST9 family member.

example:

```
; ST904x family description:
; Include file for the definition of the registers and bits for the
; ST904X family
 .include    "c:\st9\inc\system.inc"        ; System register
 .include    "c:\st9\inc\page_0.inc"        ; Page 0 register
 .include    "c:\st9\inc\eeprom.inc"        ; EEPROM register
 .include    "c:\st9\inc\sec_reg.inc"       ; Security register
 .include    "c:\st9\inc\io_port.inc"       ; I/O port register
 .include    "c:\st9\inc\mftimer.inc"       ; MF Timer register
 .include    "c:\st9\inc\ad_c.inc"          ; A/D converter register
 .include    "c:\st9\inc\sci.inc"           ; SCI register
```

The ST9 software programmer may use these include files in the two following methods:

1) use the include file corresponding to the target device within each ST9 software module.

2) directly use the include files corresponding to the ST9 peripheral programmed or used in a specific module: for example, inside a module using I/O ports and MFTimer, only the two include files related to MFTimer and I/O port could be used.

DEFINITIONS AND USAGE

Registers

A name is given for each ST9 register pair or register in upper case letters, corresponding to an "absolute" addressing mode inside the Register File, and in lower case letters, corresponding to an addressing mode inside a working register group (defined by the working register pointer pair **RP0** and **RP1**). This choice follows the notation used in the ST9 Macro Assembler (ie R or RR indicate any 8 or 16 bit register within the Register File, r or rr indicate a register within the working register group.):

```
FCW     :=      RR230     ; Flags and control word: Absolute address
fcw      =      rr6       ; Working Register address, Group E
CICR    :=      R230      ; Central interrupt control register
cicr     =      r6        ; Working Register address, Group E
```

Bits

A name for each bit and its associated mask for each control register has been defined:

1) the bit name is defined using the .defstr pseudo-instruction and the bit location within the working register. This name can be used directly with the boolean instructions:

```
.defstr  gcen  "cicr.7"     ; Global counter enable bit definition
```

2) the mask name is defined by a "one" shifted left by the bit location value inside the associated register. This allows the setting or masking of named bits within a working register:

```
gcenm    :=    (1 <- 7 )    ; Global counter enable masl
```

Bit symbol names are given in lower case letters.
Further examples of the use of these definitions follow.

Two other kinds of names are provided in this file:

- names for register groups within the Register File: one name for 8-working register groups, used with the `SRP0` and `SRP1` instructions, and one name for 16-working register groups (for system registers and page registers), used with the SRP instruction:

```
srp      #BK_F              ; select working register group F
```

- names for peripheral pages (Group F):

```
spp  #SCI1_PG           ; select SCI1 register page
```

Example 1: SCI initialization using working register addressing mode

The following example, extracted from an initialization routine of the Serial Communications Interface (SCI), shows how to use the **SYMBOLS.INC** file with the working register addressing mode. In the case of initialization of several peripherals (e.g. SCI and TIMER), this method allows the user to save bytes and execution time, due to the shorter instructions and execution times of the working register addressing modes.

```
sppSCI1_PG                  ; select SCI1 register page
srp#BK_F                    ; select working register group F
ld s_brghr,#0               ; initialize SCI baud rate generator
                            ; register in group F.
ld   s_chcr,#(wl8 | pen | ep | 20)
                            ; initialize character
                            ; configuration register using mask
                            ; bit definitions OR'd together
```

Example 2: using mask and complemented mask

This example shows how to use the mask and complemented mask. In this example, the enable receiver error interrupt and type of parity are set.

```
srp#BK_0                    ; working register in group 0
spp#SCI1_PG                 ; select SCI1 register page
or   S_IMR,#rxe             ; enable Rx error INTERRUPT using absolute
                            ; addressing in register file (R)
ld data,S_RXBR              ; load data with SCI receiver reg. (R)
if   [data == #0] {         ; if data is 00h...
   and S_CHCR,#~ep          ; ... enable odd parity detection
}                           ; (~ indicates 1's complement)
```

Example 3: bit manipulation

The SYMBOLS.INC file shows definition of the bits in a working register using the AST9 .defstr pseudo-instruction.

```
.defstr S_txdi "s_imr.0" ; transmitter data interrupt

btjz  S_txdi, checkrx    ; poll on tx data interrupt
bres  S_txdi             ; reset if active
```

An alternative method of defininition and usage of bits consists of giving a name to a bit location and associating this name to a register.

```
STATUS_SCI      =       R3      ; SCI status register (Absolute Reg.)
status_sci      =       r3      ; SCI status register (working Reg.)
P_er            =       0       ; parity error detected
Oe_er           =       1       ; overrun error detected
Fe_er           =       2       ; framing error detected
Tx_go           =       3       ; Tx on going
Tx_err          =       4       ; Tx error detected

bset    status_sci.Tx_go        ; set Tx ongoing bit
bres    status_sci.Tx_err       ; reset Tx error detection flag
```

SYMBOL TABLE OVERLOAD PROTECTION

SYMBOLS.INC contains more than 500 symbols and should be included with each ST9 software module which uses the symbols. In order not to overload the symbol table produced by the ST9 linker **LST9**, an **AST9** option **(-r)**, is available, which does not preserve register symbols in its output. When assembling several ST9 modules (each one containing **SYMBOLS.INC**), the first module (usually the main module) must be assembled without the **-r** option, all other modules must be assembled with the -r option. (Refer to the **AST9** User Manual for further information).

INCLUDE FILES

The name of the Include files related to the ST9 Core and Peripherals are listed below. These files are extracted from the SYMBOLS.INC file which is shown in Appendix B, and thus are not shown independently.

AD_C.INC	Analog to Digital Converter Registers
BS_REG.INC	Bankswitch Registers (ST905X only)
EEPROM.INC	EEPROM control Registers (ST904X only)
IO_PORT.INC	I/O Ports Registers (All ST9 devices)
MFTIMER.INC	Multifunction Timer Registers
PAGE_0.INC	Page 0 Registers (All ST9 devices)
RW_REG.INC	R/W Control Registers (ST905X only)
SCI.INC	Serial Communications Interface
SEC_REG.INC	Security Register (ST904X only)
SYSTEM.INC	System Registers (Group E, All ST9 devices)
ST902X.INC	ST902X Include File
ST903X.INC	ST903X Include File
ST904X.INC	ST904X Include File
ST905X.INC	ST905X Include File
SYMBOL36.INC	ALL Registers Include File

APPENDIX A: ST90XX Family Definition include files

ST902X family Description **(ST902X.INC)**:

```
; Include file for the definition of the registers and bits for the
; ST902x family
     .include "     c:\st9\inc\system.inc"       ; System register
     .include "     c:\st9\inc\page_0.inc"       ; Page 0 register
     .include "     c:\st9\inc\io_port.inc"      ; I/O port register
     .include "     c:\st9\inc\mftimer.inc"      ; MF Timer register
     .include "     c:\st9\inc\sci.inc"          ; SCI register
```

ST903x family description **(ST903X.INC)**:

```
; Include file for the definition of the registers and bits for the
; ST903x family
     .include       "c:\st9\inc\system.inc"      ; System register
     .include       "c:\st9\inc\page_0.inc"      ; Page 0 register
     .include       "c:\st9\inc\io_port.inc"     ; I/O port register
     .include       "c:\st9\inc\mftimer.inc"     ; MF Timer register
     .include       "c:\st9\inc\ad_c.inc"        ; A/D converter register
     .include       "c:\st9\inc\sci.inc"         ; SCI register
```

ST904x family description **(ST904X.INC)**:

```
; Include file for the definition of the registers and bits for the
; ST904x family
     .include       "c:\st9\inc\system.inc"      ; System register
     .include       "c:\st9\inc\page_0.inc"      ; Page 0 register
     .include       "c:\st9\inc\eeprom.inc"      ; EEPROM register
     .include       "c:\st9\inc\sec_reg.inc"     ; Security register
     .include       "c:\st9\inc\io_port.inc"     ; I/O port register
     .include       "c:\st9\inc\mftimer.inc"     ; MF Timer register
     .include       "c:\st9\inc\ad_c.inc"        ; A/D converter register
     .include       "c:\st9\inc\sci.inc"         ; SCI register
```

ST905x family description **(ST905X.INC)**:

```
; Include file for the definition of the registers and bits for the
; ST905x family
     .include       "c:\st9\inc\system.inc"      ; System register
     .include       "c:\st9\inc\page_0.inc"      ; Page 0 register
     .include       "c:\st9\inc\rw_reg.inc"      ; R/W signal register
     .include       "c:\st9\inc\io_port.inc"     ; I/O port register
     .include       "c:\st9\inc\bs_reg.inc"      ; Bank switching register
     .include       "c:\st9\inc\mftimer.inc"     ; MF Timer register
     .include       "c:\st9\inc\ad_c.inc"        ; A/D converter register
     .include       "c:\st9\inc\sci.inc"         ; SCI register
```

APPENDIX B: Symbols.inc listing

```
.sbttl" ST9 family registers and register-bits "
        .pl     66              ; Number of lines per page
;       .list
;       .list   me              ; Enable macro expansion control
;       .list   bex             ; Enable continuation of code on next
                                ; Line
;       .nlist  line            ; Disable source line number control
;       .nlist  loc             ; Disable current location counter
                                ; control
;       .nlist  code            ; Disable binary code control
;       .nlist  src             ; Disable source line control
;       .nlist  com             ; Disable comment control
;       .nlist  md              ; Disable macro definition control
;       .nlist  mc              ; Disable macro call control
        .nlist

;*****************************************************************************
;               ****************************************
;               *   Revision 3.6       MARCH, 04th 1991    *
;               ****************************************

;               ST9 family registers addresses and contents.
;     This file contains the symbolic definitions for the ST9 CPU
;     and Peripherals registers and bits.
;     There is a symbol for each register and for each flag used in
;     an ST9 family device.
;  - Lowercase letters refer to addressing using working registers ( r )
;  - Uppercase letters refer to addressing using direct registers  ( R )

;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************
;
;  ST9 family:  Core, Timer Watch-dog, SPI and EEPROM Control Register
;
;*************************************************************************

;*********************************
;*REGISTER FILE GROUPS DEFINITION*
;*********************************
BK00     =     0          ; r0 to r7        in group 0
BK01     =     1          ; r8 to r15       in group 0
BK10     =     2          ; r0 to r7        in group 1
BK11     =     3          ; r8 to r15       in group 1
BK20     =     4          ; r0 to r7        in group 2
BK21     =     5          ; r8 to r15       in group 2
BK30     =     6          ; r0 to r7        in group 3
BK31     =     7          ; r8 to r15       in group 3
BX40     =     8          ; r0 to r7        in group 4
BK41     =     9          ; r8 to r15       in group 4
BK50     =     10         ; r0 to r7        in group 5
BK51     =     11         ; r8 to r15       in group 5
BK60     =     12         ; r0 to r7        in group 6
BK61     =     13         ; r8 to r15       in group 6
BK70     =     14         ; r0 to r7        in group 7
BK71     =     15         ; r8 to r15       in group 7
BK80     =     16         ; r0 to r7        in group 8
BK81     =     17         ; r8 to r15       in group 8
BK90     =     18         ; r0 to r7        in group 9
BK91     =     19         ; r8 to r15       in group 9
BKA0     =     20         ; r0 to r7        in group A
BKA1     =     21         ; r8 to r15       in group A
BKB0     =     22         ; r0 to r7        in group B
BKB1     =     23         ; r8 to r15       in group B
BKC0     =     24         ; r0 to r7        in group C
BKC1     =     25         ; r8 to r15       in group C
BKD0     =     26         ; r0 to r7        in group D
BKD1     =     27         ; r8 to r15       in group D
BKE0     =     28         ; r0 to r7        in group E
BKE1     =     29         ; r8 to r15       in group E
BKF0     =     30         ; r0 to r7        in group F
BKF1     =     31         ; r8 to r15       in group F
BK_SYS   =     BKE0       ; Group system definition
BK_F     =     BKF0       ; page register definition
```

Appendix B: Symbols.inc listing

```
;******************
;*SYSTEM REGISTERS*
;******************
FCW       := RR230                        ; Flags and control word.
fcw        = rr6
CICR      := R230                         ; Central interrupt control register.
cicr       = r6
          .defstr gcen       "cicr.7"    ; Global counter enable.
          .defstr tlip       "cicr.6"    ; Top level interrupt pending bit
          .defstr tli        "cicr.5"    ; Top level interrupt bit.
          .defstr ien        "cicr.4"    ; Interrupt enable flag.
          .defstr iam        "cicr.3"    ; Interrupt arbitration mode.
          .defstr cpl2       "cicr.2"    ; Current priority level bit 2.
          .defstr cpl1       "cicr.1"    ; Current priority level bit 1.
          .defstr cpl0       "cicr.0"    ; Current priority level bit 0.
gcenm     := ( 1 <- 7 )                   ; Global counter enable bit mask
tlipm     := ( 1 <- 6 )                   ; Top level interrupt pending mask.
tlim      := ( 1 <- 5 )                   ; Top level interrupt mask.
ienm      := ( 1 <- 4 )                   ; Interrupt enabla flag mask.
iamm      := ( 1 <- 3 )                   ; Interrupt arbitration mode mask.
cpl2m     := ( 1 <- 2 )                   ; Current priority level bit 2 mask.
cpl1m     := ( 1 <- 1 )                   ; Current priority level bit 1 mask.
cpl0m     := ( 1 <- 0 )                   ; Current priority level bit 0 mask.
cplm      := ( cpl2m|cpl1m|cpl0 )         ; Current priority level

FLAGR     := R231                         ; Flags register.
flagr      = r7
          .defstr c          "flagr.7"   ; Carry flag.
          .defstr z          "flagr.6"   ; Zero flag.
          .defstr s          "flagr.5"   ; Sign flag.
          .defstr v          "flagr.4"   ; Overflow flag.
          .defstr d          "flagr.3"   ; Decimal adjust flag.
          .defstr h          "flagr.2"   ; Half carry flag.
          .defstr uf         "flagr.1"   ; User flag 1.
          .defstr dp         "flagr.0"   ; Data/program memory flag.
cm        := ( 1 <- 7 )                   ; Carry flag mask.
zm        := ( 1 <- 6 )                   ; Zero flag mask.
sm        := ( 1 <- 5 )                   ; Sign flag mask.
vm        := ( 1 <- 4 )                   ; Overflow flag mask.
dm        := ( 1 <- 3 )                   ; Decimal adjust flag mask.
hm        := ( 1 <- 2 )                   ; Half carry flag mask.
ufm       := ( 1 <- 1 )                   ; User flag 1 mask.
dpm       := ( 1 <- 0 )                   ; Data/program memory mask.
RPP       := RR232                        ; Register pointer pair.
rpp        = rr8
RP0R      := R232                         ; Register pointer # 0.
rp0r       = r8
          .defstr rp0s       "rp0r.2"    ; Register pointer selector
rp0sm     := ( 1 <- 2 )                   ; Register pointer selector mask
```

APPENDIX B: Symbols.inc listing

```
RP1R     := R233                        ; Register pointer # 1.
 rp1r     = r9
         .defstr rp1s       "rp1r.2"    ; Register pointer selector
rp1sm    := ( 1 <- 2 )                  ; Register pointer selector mask
PPR      := R234                        ; Page pointer register.
ppr       = r10
MODER    := R235                        ; Mode register.
moder     = r11
         .defstr ssp        "moder.7"   ; System stack pointer flag (Int/Ext).
         .defstr usp        "moder.6"   ; User stack pointer flag (Int/Ext).
         .defstr div2       "moder.5"   ; External clock divided by 2.
         .defstr prs2       "moder.4"   ; Internal clock prescaling bit 2.
         .defstr prs1       "moder.3"   ; Internal clock prescaling bit 1.
         .defstr prs0       "moder.2"   ; Internal clock prescaling bit 0.
         .defstr brqen      "moder.1"   ; Bus request enable.
         .defstr himp       "moder.0"   ; High impedance enable.
sspm     := ( 1 <- 7 )                  ; System stack pointer mask (Int/Ext).
uspm     := ( 1 <- 6 )                  ; User stack pointer mask (Int/Ext).
div2m    := ( 1 <- 5 )                  ; External clock divided by 2 mask.
prs2m    := ( 1 <- 4 )                  ; Internal clock prescaling bit 2 mask.
prs1m    := ( 1 <- 3 )                  ; Internal clock prescaling bit 1 mask.
prs0m    := ( 1 <- 2 )                  ; Internal clock prescaling bit 0 mask.
prsm     := ( prs2m|prs1m|prs0m )       ; Internal clock prescaler
brqenm   := ( 1 <- 1 )                  ; Bus request enable mask.
himpm    := ( 1 <- 0 )                  ; High impedence enable mask.
USPR     := RR236                       ; User stack pointer.
uspr      = rr12
USPHR    := R236                        ; User stack pointer, msb.
usphr     = r12
USPLR    := R237                        ; User stack pointer, lsb.
usplr     = r13
SSPR     := RR238                       ; System stack pointer.
sspr      = rr14
SSPHR    := R238                        ; System stack pointer, msb.
ssphr     = r14
SSPLR    := R239                        ; System stack pointer, lsb.
ssplr     = r15
;.page
```

APPENDIX B: Symbols.inc listing

```
;****************
;*PAGE REGISTERS*
;****************
EEP_PG   := 0                          ; EEPROM register page
EECR     := R241                       ; EEprom control register
eecr      = r1
         .defstr verify    "eecr.6"   ; EEPROM verify mode
         .defstr EEstby    "eecr.5"   ; EEPROM stand-by
         .defstr EEien     "eecr.4"   ; EEPROM interrupt enable
         .defstr pllst     "eecr.3"   ; Parallel write start
         .defstr pllen     "eecr.2"   ; Parallel write enable
         .defstr EEbusy    "eecr.1"   ; EEPROM busy
         .defstr EEwen     "eecr.0"   ; EEPROM write enable
verify         := ( 1 <- 6 )           ; EEPROM verify mode mask
eestby         := ( 1 <- 5 )           ; EEPROM stand-by mask
eeien := ( 1 <- 4 )                    ; EEPROM interrupt enable mask
pllst    := ( 1 <- 3 )                 ; Parallel write start mask
pllen    := ( 1 <- 2 )                 ; Parallel write enable mask
eebusy   := ( 1 <- 1 )                 ; EEPROM busy mask
eewen    := ( 1 <- 0 )                 ; EEPROM Write enable mask
EXINT_PG := 0                          ; EXTERNAL interrupt register page
EITR     := R242                       ; External interrupt trigger level register
eitr      = r2
         .defstr tea0      "eitr.0"   ; Trigger Event A0 bit
         .defstr tea1      "eitr.1"   ; Trigger Event A1 bit
         .defstr teb0      "eitr.2"   ; Trigger Event B0 bit
         .defstr teb1      "eitr.3"   ; Trigger Event B1 bit
         .defstr tec0      "eitr.4"   ; Trigger Event C0 bit
         .defstr tec1      "eitr.5"   ; Trigger Event C1 bit
         .defstr ted0      "eitr.6"   ; Trigger Event D0 bit
         .defstr ted1      "eitr.7"   ; Trigger Event D1 bit
tea0m    := ( 1 <- 0 )                 ; Trigger Event A0 mask
tea1m    := ( 1 <- 1 )                 ; Trigger Event A1 mask
teb0m    := ( 1 <- 2 )                 ; Trigger Event B0 mask
teb1m    := ( 1 <- 3 )                 ; Trigger Event B1 mask
tec0m    := ( 1 <- 4 )                 ; Trigger Event C0 mask
tec1m    := ( 1 <- 5 )                 ; Trigger Event C1 mask
ted0m    := ( 1 <- 6 )                 ; Trigger Event D0 mask
ted1m    := ( 1 <- 7 )                 ; Trigger Event D1 mask
EIPR     := R243                       ; External interrupt pending register
eipr      = r3
         .defstr ipa0      "eipr.0"   ; Interrupt Pending bit Channel A0
         .defstr ipa1      "eipr.1"   ; Interrupt Pending bit      "   A1
         .defstr ipb0      "eipr.2"   ; Interrupt Pending bit      "   B0
         .defstr ipb1      "eipr.3"   ; Interrupt Pending bit      "   B1
         .defstr ipc0      "eipr.4"   ; Interrupt Pending bit      "   C0
         .defstr ipc1      "eipr.5"   ; Interrupt Pending bit      "   C1
         .defstr ipd0      "eipr.6"   ; Interrupt Pending bit      "   D0
         .defstr ipd1      "eipr.7"   ; Interrupt Pending bit      "   D1
```

APPENDIX B: Symbols.inc listing

```
ipa0m    := ( 1 <- 0 )                ; Interrupt Pending A0 mask
ipa1m    := ( 1 <- 1 )                ; Interrupt Pending A1 mask
ipb0m    := ( 1 <- 2 )                ; Interrupt Pending B0 mask
ipb1m    := ( 1 <- 3 )                ; Interrupt Pending B1 mask
ipc0m    := ( 1 <- 4 )                ; Interrupt Pending C0 mask
ipc1m    := ( 1 <- 5 )                ; Interrupt Pending C1 mask
ipd0m    := ( 1 <- 6 )                ; Interrupt Pending D0 mask
ipd1m    := ( 1 <- 7 )                ; Interrupt Pending D1 mask

EIMR     := R244                      ; External interrupt mask register
eimr      = r4

         .defstr ima0 "eimr.0"        ; Int. A0 bit
         .defstr ima1 "eimr.1"        ; Int. A1 bit
         .defstr imb0 "eimr.2"        ; Int. B0 bit
         .defstr imb1 "eimr.3"        ; Int. B1 bit
         .defstr imc0 "eimr.4"        ; Int. C0 bit
         .defstr imc1 "eimr.5"        ; Int. C1 bit
         .defstr imd0 "eimr.6"        ; Int. D0 bit
         .defstr imd1 "eimr.7"        ; Int. D1 bit

ia0m     := ( 1 <- 0 )                ; Int. A0 mask
ia1m     := ( 1 <- 1 )                ; Int. A1 mask
ib0m     := ( 1 <- 2 )                ; Int. B0 mask
ib1m     := ( 1 <- 3 )                ; Int. B1 mask
ic0m     := ( 1 <- 4 )                ; Int. C0 mask
ic1m     := ( 1 <- 5 )                ; Int. C1 mask
id0m     := ( 1 <- 6 )                ; Int. D0 mask
id1m     := ( 1 <- 7 )                ; Int. D1 mask

EIPLR    := R245                      ; Ext. interrupt priority level register
eiplr     = r5

EIVR     := R246                      ; External interrupt vector register
eivr      = r6

         .defstr ewen  "eivr.0"       ; External wait enable
         .defstr ia0s  "eivr.1"       ; Interrupt A0 selection
         .defstr tlis  "eivr.2"       ; Top level input selection
         .defstr tltev "eivr.3"       ; Top level trigger event

ewenm    := ( 1 <- 0 )                ; External wait enable mask
iaosm    := ( 1 <- 1 )                ; Interrupt A0 selection mask
tlism    := ( 1 <- 2 )                ; Top level Input selection mask
tltevm   := ( 1 <- 3 )                ; Top level trigger event mask


NICR     := R247                      ; Nested interrupt control register
nicr      = r7

         .defstr tlnm      "nicr.7"   ; Top level not maskable

tlnmm    := ( 1 <- 7 )                ; Top level not maskable mask
```

APPENDIX B: Symbols.inc listing

```
WDT_PG    := 0                              ; Timer Watchdog page
WDTR      := RR248                          ; TWD timer constant register.
wdtr       = rr8
WDTHR     := R248                           ; TWD timer high constant register
wdthr      = r8
WDTLR     := R249                           ; TWD timer low constant register
wdtlr      = r9
WDTPR     := R250                           ; TWD timer prescaler constant register
wdtpr      = rl0
WDTCR     := R251                           ; TWD timer control register
wdtcr      = r11
          .defstr WD_stsp    "wdtcr.7"     ; TWD start stop.
          .defstr WD_sc      "wdtcr.6"     ; TWD single continuous mode.
          .defstr WD_inmd1   "wdtcr.5"     ; Input mode 1
          .defstr WD_inmd2   "wdtcr.4"     ; Input mode 2
          .defstr WD_inen    "wdtcr.3"     ; TWD input section enable/disable.
          .defstr WD_outmd   "wdtcr.2"     ; TWD output mode.
          .defstr WD_wrout   "wdtcr.1"     ; TWD output bit.
          .defstr WD_outen   "wdtcr.0"     ; TWD output enable.
stsp      := ( 1 <- 7 )                    ; TWD start stop mask
sc        := ( 1 <- 6 )                    ; TWD single continuous mode mask
inen      := ( 1 <- 3 )                    ; TWD input section enable/disable mask
outmd     := ( 1 <- 2 )                    ; TWD output mode mask
wrout     := ( 1 <- 1 )                    ; TWD output bit mask
outen     := ( 1 <- 0 )                    ; TWD output enable mask
inm_evc   := 0                             ; TWD input mode event counter.
inm_g     := 010h                          ; TWD input mode gated.
inm_t     := 020h                          ; TWD input mode triggerable.
inm_r     := 030h                          ; TWD input mode retriggerable.
```

APPENDIX B: Symbols.inc listing

```
WCR       :=    R252                    ; Wait control register
wcr        =    r12
          .defstr WD_wden   "wcr.6"     ; TWD timer enable.
wdgen     := ( 1 <- 6 )                 ; TWD timer enable mask
wdm2      := ( 1 <- 5 )                 ; Data Memory Wait Cycle
wdm1      := ( 1 <- 4 )
wdm0      := ( 1 <- 3 )
wpm2      := ( 1 <- 2 )                 ; Program Memory Wait Cycle
wpm1      := ( 1 <- 1 )
wpm0      := ( 1 <- 0 )
dmwc1     := wdm0                       ; 1 wait cycle  on Data M.
dmwc2     := wdm1                       ; 2 wait cycles on Data M.
dmwc3     := ( wdm1 | wdm0 )            ; 3 wait cycles on Data M.
dmwc4     := wdm2                       ; 4 wait cycles on Data M.
dmwc5     := ( wdm2 | wdm0 )            ; 5 wait cycles on Data M.
dmwc6     := ( wdm2 | wdm1 )            ; 6 wait cycles on Data M.
dmwc7     := (wdm2 | wdm1 | wdm0)       ; 7 wait cycles on Data M.
pmwc1     := wpm0                       ; 1 wait cycle  on Prog M.
pmwc2     := wpm1                       ; 2 wait cycles on Prog M.
pmwc3     := ( wpm1 | wpm0 )            ; 3 wait cycles on Prog M.
pmwc4     := wpm2                       ; 4 wait cycles on Prog M.
pmwc5     := ( wpm2 | wpm0 )            ; 5 wait cycles on Prog M.
pmwc6     := ( wpm2 | wpm1 )            ; 6 wait cycles on Prog M.
pmwc7     := (wpm2 | wpm1 | wpm0 )      ; 7 wait cycles on Prog M.
```

APPENDIX B: Symbols.inc listing

```
SPI_PG    := 0                          ; SPI register page
SPIDR     := R253                       ; SPI Data register
spidr      = r13
SPICR     := R254                       ; SPI Control register
spicr      = r14
          .defstr SP_spen    "spicr.7"  ; Serial Peripheral Enable.
          .defstr SP_bms     "spicr.6"  ; SBUS/I2C bus Mode Selector.
          .defstr SP_arb     "spicr.5"  ; Arbitration flag bit.
          .defstr SP_busy    "spicr.4"  ; SPI busy flag.
          .defstr SP_cpol    "spicr.3"  ; SPI transmission clock polarity
          .defstr SP_cpha    "spicr.2"  ; SPI transmission clock phase
          .defstr SP_spr1    "spicr.1"  ; SPI rate bit 1
          .defstr SP_spr0    "spicr.0"  ; SPI rate bit 0
spen      := ( 1 <- 7 )                 ; Serial Peripheral Enable mask
bms       := ( 1 <- 6 )                 ; SBUS/I2C bus selector mask
arb       := ( 1 <- 5 )                 ; Arbitration mask
sp_busy   := ( 1 <- 4 )                 ; SPI busy mask
cpol      := ( 1 <- 3 )                 ; SPI transmission clock polarity mask
cpha      := ( 1 <- 2 )                 ; SPI transmission clock phase
SP_8      := 0                          ; SPI clock divider   8 = 1500 kHz  (12MHz)
SP_16     := 1                          ; SPI clock divider  16 =  750 kHz  (12MHz)
SP_128    := 2                          ; SPI clock divider 128 = 93.75 kHz (12MHz)
SP_256    := 3                          ; SPI clock divider 256 = 46.87 kHz (12MHz)
RW_PG     := 0                          ; R/W signal programming page
RWR       := R255                       ; R/W signal programming register
rwr        = r15
          .defstr RW_rw      "rwr.0"    ; R/W bit
          .defstr RW_bs      "rwr.1"    ; Bank switch port timing
rw        := ( 1 <- 0 )                 ; R/W mask
bs        := ( 1 <- 1 )                 ; Bank Switch mask
;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;              ST9 FAMILY I/O PORTS REGISTER ADDRESSES.

;*************************************************************************

; P0DR, P1DR, P2DR, P3DR, P4DR, P5DR are mapped in the system registers
; BS_DSR, BS_PSR are mapped in the system registers
P0C_PG   := 2                          ; Port 0 control registers page
P0DR     := R224                       ; Port 0 data register
P0C0R    := R240                       ; Port 0 control register 0
P0C1R    := R241                       ; Port 0 control register 1
P0C2R    := R242                       ; Port 0 control register 2
p0dr      = r0
p0c0r     = r0
p0c1r     = r1
p0c2r     = r2
P1C_PG   := 2                          ; Port 1 control registers page
P1DR     := R225                       ; Port 1 data register
P1C0R    := R244                       ; Port 1 control register 0
P1C1R    := R245                       ; Port 1 control register 1
P1C2R    := R246                       ; Port 1 control register 2
p1dr      = r1
p1c0r     = r4
p1c1r     = r5
p1c2r     = r6
P2C_PG   := 2                          ; Port 2 control registers page
P2DR     := R226                       ; Port 2 data register
BS_DSR   := R226                       ; Bank Switch data segment register
P2C0R    := R248                       ; Port 2 control register 0
BS_DDSR  := R248                       ; Bank Switch Data DMA segment register
P2C1R    := R249                       ; Port 2 control register 1
BS_PDSR  := R249                       ; Bank Switch Program DMA segment Register
P2C2R    := R250                       ; Port 2 control register 2
p2dr      = r2
bs_dsr    = r2
p2c0r     = r8
bs_ddsr   = r8
p2c1r     = r9
bs_pdsr   = r9
p2c2r     = r10
```

APPENDIX B: Symbols.inc listing

```
P3C_PG   := 2                          ; Port 3 control registers page
P3DR     := R227                       ; Port 3 data register
BS_PSR   := R227                       ; Bank Switch Program Segment Register
P3C0R    := R252                       ; Port 3 control register 0
P3C1R    := R253                       ; Port 3 control register 1
P3C2R    := R254                       ; Port 3 control register 2

p3dr      = r3
bs_psr    = r3
p3c0r     = r12
p3c1r     = r13
p3c2r     = r14

P4C_PG   := 3                          ; Port 4 control registers page

P4DR     := R228                       ; Port 4 data register
P4C0R    := R240                       ; Port 4 control register 0
P4C1R    := R241                       ; Port 4 control register 1
P4C2R    := R242                       ; Port 4 control register 2

p4dr      = r4
p4c0r     = r0
p4c1r     = r1
p4c2r     = r2

P5C_PG   := 3                          ; Port 5 control registers page

P5DR     := R229                       ; Port 5 data register
P5C0R    := R244                       ; Port 5 control register 0
P5C1R    := R245                       ; Port 5 control register 1
P5C2R    := R246                       ; Port 5 control register 2

p5dr      = r5
p5c0r     = r4
p5c1r     = r5
p5c2r     = r6

P6C_PG   := 3                          ; Port 6 control registers page
P6D_PG   := 3                          ; Port 6 data register page

P6DR     := R251                       ; Port 6 data register
P6C0R    := R248                       ; Port 6 control register 0
P6C1R    := R249                       ; Port 6 control register 1
P6C2R    := R250                       ; Port 6 control register 2

p6dr      = r11
p6c0r     = r8
p6c1r     = r9
p6c2r     = r10
```

APPENDIX B: Symbols.inc listing

```
P7C_PG    := 3                          ; Port 7 control registers page
P7D_PG    := 3                          ; Port 7 data register page
P7DR      := R255                       ; Port 7 data register
P7C0R     := R252                       ; Port 7 control register 0
P7C1R     := R253                       ; Port 7 control register 1
P7C2R     := R254                       ; Port 7 control register 2

p7dr       = r15
p7c0r      = r12
p7c1r      = r13
p7c2r      = r14

P8C_PG    := 43                         ; Port 8 control registers page
P8D_PG    := 43                         ; Port 8 data register page

P8DR      := R251                       ; Port 8 data register
P8C0R     := R248                       ; Port 8 control register 0
P8C1R     := R249                       ; Port 8 control register 1
P8C2R     := R250                       ; Port 8 control register 2

p8dr       = r11
p8c0r      = r8
p8c1r      = r9
p8c2r      = r10

P9C_PG    := 43                         ; Port 9 control registers page
P9D_PG    := 43                         ; Port 9 data register page

P9DR      := R255                       ; Port 9 data register
P9C0R     := R252                       ; Port 9 control register 0
P9C1R     := R253                       ; Port 9 control register 1
P9C2R     := R254                       ; Port 9 control register 2

p9dr       = r15
p9c0r      = r12
p9c1r      = r13
p9c2r      = r14
```

APPENDIX B: Symbols.inc listing

```
HDCTL2R  := R251                        ; Port 2 handshake DMA control register
HDCTL3R  := R255                        ; Port 3 handshake DMA control register
HDCTL4R  := R243                        ; Port 4 handshake DMA control register
HDCTL5R  := R247                        ; Port 5 handshake DMA control register

hdctl2r   = r11
hdctl3r   = r15
hdctl4r   = r3
hdctl5r   = r7


;Handshake DMA control register configuration.

hsdis    := 0E0h                        ; Handshake disabled mask
hso2     := 0C0h                        ; Handshake output 2 lines mask
hso1     := 040h                        ; Handshake output 1 line mask
hsi2     := 0A0h                        ; Handshake input 2 lines mask
hsi1     := 020h                        ; Handshake input 1 line mask
hsb      := 000h                        ; Handshake bidirectional mask
den      := 000h                        ; DMA enable mask
ddi      := 010h                        ; DMA disable mask
ddw      := 000h                        ; Data direction output mask (write)
ddr      := 008h                        ; Data direction input mask (read)
dst      := 004h                        ; DMA strobe on chip event mask
dcp0     := 000h                        ; DMA channel capture0 mask
dcm0     := 002h                        ; DMA channel compare0 mask

;.page
```

APPENDIX B: Symbols.inc listing

```
;*****************************************************************************

;     ST9 FAMILY MULTI-FUNCTION TIMER DESCRIPTION.

;*****************************************************************************

T0D_PG    := 10                         ; MFTimer 0 data registers page
T0C_PG    := 9                          ; MFTimer 0 control registers page
T1D_PG    := 8                          ; MFTimer 1 data registers page
T1C_PG    := 9                          ; MFTimer 1 control registers page
T2D_PG    := 14                         ; MFTimer 2 data registers page
T2C_PG    := 13                         ; MFTimer 2 control registers page
T3D_PG    := 12                         ; MFTimer 3 data registers page
T3C_PG    := 13                         ; MFTimer 3 control registers page


T_REG0R   := RR240                      ; MFTimer REG0 load and capture register.
t_reg0r    = rr0
T_REG0HR  := R240                       ; Register 0 high register
t_reg0hr   = r0
T_REG0LR  := R241                       ; Register 0 low register
t_reg0lr   = r1
T_REG1R   := RR242                      ; MFTimer REG1 load constant
t_reg1r    = rr2                              ; and capture register.
T_REG1HR  := R242                       ; Register 1 high register
t_reg1hr   = r2
T_REG1LR  := R243                       ; Register 1 low register
t_reg1lr   = r3
T_CMP0R   := RR244                      ; MFTimer CMP0 store compare constant.
t_cmp0r    = rr4
T_CMP0HR  := R244                       ; Compare 0 high register
t_cmp0hr   = r4
T_CMP0LR  := R245                       ; Compare 0 low register
t_cmp0lr   = r5
T_CMP1R   := RR246                      ; MFTimer CMP1 store compare constant.
t_cmp1r    = rr6
T_CMP1HR  := R246                       ; Compare 1 high register
t_cmp1hr   = r6
T_CMP1LR  := R247                       ; Compare 1 low register
t_cmp1lr   = r7
```

APPENDIX B: Symbols.inc listing

```
T_TCR     := R248                          ; MFTimer Control Register.
t_tcr      = r8
          .defstr T_cs       "t_tcr.0"   ; Counter status
          .defstr T_of0      "t_tcr.1"   ; over/underflow on CAP on REG0
          .defstr T_udcs     "t_tcr.2"   ; up/down count status
          .defstr T_udc      "t_tcr.3"   ; up/down count
          .defstr T_ccl      "t_tcr.4"   ; Counter clear
          .defstr T_ccmp0    "t_tcr.5"   ; Clear on compare 0
          .defstr T_ccp0     "t_tcr.6"   ; Clear on capture
          .defstr T_cen      "t_tcr.7"   ; Counter enable
cs        := ( 1 <- 0 )                   ; Counter status mask
of0       := ( 1 <- 1 )                   ; over/underflow mask on CAP on REG0
udcs      := ( 1 <- 2 )                   ; up/down count status mask
udc       := ( 1 <- 3 )                   ; up/down count mask
ccl       := ( 1 <- 4 )                   ; Counter clear mask
ccmp0     := ( 1 <- 5 )                   ; Clear on compare mask
ccp0      := ( 1 <- 6 )                   ; Clear on capture mask
cen       := ( 1 <- 7 )                   ; Counter enable mask

T_TMR     := R249                          ; MFTimer Mode Register.
t_tmr      = r9
    .defstr T_co           "t_tmr.0"     ; Continuous/one shot bit
    .defstr T_ren          "t_tmr.1"     ; retrigger enable bit
    .defstr T_eck          "t_tmr.2"     ; Enable clocking mode bit
    .defstr T_rm0          "t_tmr.3"     ; register 0 mode bit
    .defstr T_rm1          "t_tmr.4"     ; register 1 mode bit
    .defstr T_bm           "t_tmr.5"     ; bivalue mode bit
    .defstr T_oe0          "t_tmr.6"     ; output 0 enable bit
    .defstr T_oe1          "t_tmr.7"     ; output 1 enable bit
co        := ( 1 <- 0 )                   ; Continuous/one shot mask
ren       := ( 1 <- 1 )                   ; retrigger enable mask
eck       := ( 1 <- 2 )                   ; Enable clocking mode mask
rm0       := ( 1 <- 3 )                   ; register 0 mode mask
rm1       := ( 1 <- 4 )                   ; register 1 mode mask
bm        := ( 1 <- 5 )                   ; bivalue mode mask
oe0       := ( 1 <- 6 )                   ; output 0 enable mask
oe1       := ( 1 <- 7 )                   ; output 1 enable mask
```

APPENDIX B: Symbols.inc listing

```
T_ICR     := R250                    ; MFTimer External Input Control Register.
t_icr      = r10
exb_f     := 01h                     ; External B falling edge sensitive mask
exb_r     := 02h                     ; External B rising edge sensitive mask
exb_rf    := 03h                     ; External B falling and rising edge mask
exa_f     := 04h                     ; External A falling edge sensitive mask
exa_r     := 08h                     ; External A rising edge sensitive mask
exa_rf    := 0Ch                     ; External A falling and rising edge mask
ab_ii     := 00h                     ; A I/O B I/O mask
ab_it     := 10h                     ; A I/O B trigger mask
ab_gi     := 20h                     ; A gate B I/O mask
ab_gt     := 30h                     ; A gate B trigger mask
ab_ie     := 40h                     ; A I/O B external clock mask
ab_ti     := 50h                     ; A trigger B I/O mask
ab_ge     := 60h                     ; A gate B external clock mask
ab_tt     := 70h                     ; A trigger B trigger mask
ab_cucd   := 80h                     ; A clock up B clock down mask
ab_ue     := 90h                     ; A clock up/down B external clock mask
ab_tutd   := 0A0h                    ; A trigger up B trigger down mask
ab_ui     := 0B0h                    ; A up/down clock B I/O mask
ab_aa     := 0C0h                    ; A autodiscr. B autodiscr. mask
ab_te     := 0D0h                    ; A trigger B external clock mask
ab_et     := 0E0h                    ; A external clock B trigger mask
ab_tg     := 0F0h                    ; A trigger B gate mask
T_PRSR    := R251                    ; MFTimer prescaler register
t_prsr     = r11
T_OACR    := R252                    ; MFTimer Output A Control Register.
t_oacr     = r12
cev       := 02h                     ; on chip event bit on COMPARE 0 mask
T_OBCR    := R253                    ; MFTimer Output B Control Register.
t_obcr     = r13
op        := 01h                     ; output preset bit mask
oev       := 02h                     ; on chip event bit on OVF/UDF mask
ou_set    := 00h                     ; overflow underflow set mask
ou_tog    := 04h                     ; overflow underflow toggle mask
ou_res    := 08h                     ; overflow underflow reset mask
ou_nop    := 0Ch                     ; overflow underflow nop mask
c1_set    := 00h                     ; Compare 1 set mask
c1_tog    := 10h                     ; Compare 1 toggle mask
c1_res    := 20h                     ; Compare 1 reset mask
c1_nop    := 30h                     ; Compare 1 nop mask
c0_set    := 00h                     ; Compare 0 set mask
c0_tog    := 40h                     ; Compare 0 toggle mask
c0_res    := 80h                     ; Compare 0 reset mask
c0_nop    := 0C0h                    ; Compare 0 nop mask
```

APPENDIX B: Symbols.inc listing

```
T_FLAGR  := R254                        ; MFTimer Flags Register.
t_flagr   = r14

         .defstr T_ao       "t_flagr.0"; and/or on capture interrupt
         .defstr T_ocm0     "t_flagr.1"; overrun compare 0
         .defstr T_ocp0     "t_flagr.2"; overrun capture 0
         .defstr T_ouf      "t_flagr.3"; overflow underflow flag
         .defstr T_cm1      "t_flagr.4"; successful compare 1
         .defstr T_cm0      "t_flagr.5"; successful compare 0
         .defstr T_cp1      "t_flagr.6"; successful capture 1
         .defstr T_cp0      "t_flagr.7"; successful capture 0

ao       := ( 1 <- 0 )                  ; and/or on capture interrupt mask
ocm0     := ( 1 <- 1 )                  ; overrun compare 0 mask
ocp0     := ( 1 <- 2 )                  ; overrun capture 0 mask
ouf      := ( 1 <- 3 )                  ; overflow underflow flag mask
cm1      := ( 1 <- 4 )                  ; successful compare 1 mask
cm0      := ( 1 <- 5 )                  ; successful compare 0 mask
cp1      := ( 1 <- 6 )                  ; successful capture 1 mask
cp0      := ( 1 <- 7 )                  ; successful capture 0 mask

T_IDMR   := R255                        ; MFTimer Interrupt DMA Mask Register.
t_idmr    = r15

         .defstr T_oui      "t_idmr.0" ; overflow underflow interrupt
         .defstr T_cm1i     "t_idmr.1" ; Compare 1 interrupt
         .defstr T_cm0i     "t_idmr.2" ; Compare 0 interrupt
         .defstr T_cm0d     "t_idmr.3" ; Compare 0 DMA
         .defstr T_cp1i     "t_idmr.4" ; Capture 1 interrupt
         .defstr T_cp0i     "t_idmr.5" ; Capture 0 interrupt
         .defstr T_cp0d     "t_idmr.6" ; Capture 0 DMA
         .defstr T_gtien    "t_idmr.7" ; global timer interrupt enable

oui      := ( 1 <- 0 )                  ; overflow underflow interrupt mask
cm1i     := ( 1 <- 1 )                  ; Compare 1 interrupt mask
cm0i     := ( 1 <- 2 )                  ; Compare 0 interrupt mask
cm0d     := ( 1 <- 3 )                  ; Compare 0 DMA mask
cp1i     := ( 1 <- 4 )                  ; Capture 1 interrupt mask
cp0i     := ( 1 <- 5 )                  ; Capture 0 interrupt mask
cp0d     := ( 1 <- 6 )                  ; Capture 0 DMA mask
gtien    := ( 1 <- 7 )                  ; global timer interrupt enable mask

T0_DCPR  := R240                        ; MFTimer 0 DMA Counter Pointer Register.
t0_dcpr   = r0

T1_DCPR  := R244                        ; MFTimer 1 DMA Counter Pointer Register.
t1_dcpr   = r4

T0_DAPR  := R241                        ; MFTimer 0 DMA Address Pointer Register.
t0_dapr   = r1

T1_DAPR  := R245                        ; MFTimer 1 DMA Address Pointer Register.
t1_dapr   = r5

T0_IVR   := R242                        ; MFTimer 0 Interrupt Vector Register.
t0_ivr    = r2

T1_IVR   := R246                        ; MFTimer 1 Interrupt Vector Register.
t1_ivr    = r6
```

APPENDIX B: Symbols.inc listing

```
T0_IDCR  := R243          ; MFTimer 0 Interrupt/DMA Control Register.
t0_idcr   = r3
T1_IDCR  := R247          ; MFTimer 1 Interrupt/DMA Control Register.
t1_idcr   = r7
T2_DCPR  := R240          ; MFTimer 2 DMA Counter Pointer Register.
t2_dcpr   = r0
T3_DCPR  := R244          ; MFTimer 3 DMA Counter Pointer Register.
t3_dcpr   = r4
T2_DAPR  := R241          ; MFTimer 2 DMA Address Pointer Register.
t2_dapr   = r1
T3_DAPR  := R245          ; MFTimer 3 DMA Address Pointer Register.
t3_dapr   = r5
T2_IVR   := R242          ; MFTimer 2 Interrupt Vector Register.
t2_ivr    = r2
T3_IVR   := R246          ; MFTimer 3 Interrupt Vector Register.
t3_ivr    = r6
T2_IDCR  := R243          ; MFTimer 2 Interrupt/DMA Control Register.
t2_idcr   = r3
T3_IDCR  := R247          ; MFTimer 3 Interrupt/DMA Control Register.
t3_idcr   = r7
plm      := 07h           ; Priority level mask
swen     := 08h           ; Swap function enable mask
dctd     := 10h           ; DMA compare transaction destination mask
dcts     := 20h           ; DMA capture transaction source mask
cme      := 40h           ; Compare 0 end of block mask
cpe      := 80h           ; Capture 0 end of block mask
T_IOCR   := R248          ; MFTimer I/O connection register
t_iocr    = r8
sc0      := 01h           ; TxOUTA and TxINA connection bit
                          ; for even MFTimer
sc1      := 02h           ; TxOUTA and TxINA connection bit
                          ; for odd MFTimer

;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;              ST9 FAMILY A/D CONVERTER REGISTERS.

;*************************************************************************

AD0_PG    := 63                          ; A/D converter registers page
AD1_PG    := 62                          ; second A/D unit
AD_D0R    := R240                        ; Channel 0 data register
ad_d0r     = r0                          ; Channel 0 data register
AD_D1R    := R241                        ; Channel 1 data register
ad_d1r     = r1                          ; Channel 1 data register
AD_D2R    := R242                        ; Channel 2 data register
ad_d2r     = r2                          ; Channel 2 data register
AD_D3R    := R243                        ; Channel 3 data register
ad_d3r     = r3                          ; Channel 3 data register
AD_D4R    := R244                        ; Channel 4 data register
ad_d4r     = r4                          ; Channel 4 data register
AD_D5R    := R245                        ; Channel 5 data register
ad_d5r     = r5                          ; Channel 5 data register
AD_D6R    := R246                        ; Channel 6 data register
ad_d6r     = r6                          ; Channel 6 data register
AD_D7R    := R247                        ; Channel 7 data register
ad_d7r     = r7                          ; Channel 7 data register
AD_LT6R   := R248                        ; Channel 6 lower threshold register
ad_lt6r    = r8                          ; Channel 6 lower threshold register
AD_LT7R   := R249                        ; Channel 7 lower threshold register
ad_lt7r    = r9                          ; Channel 7 lower threshold register
AD_UT6R   := R250                        ; Channel 6 upper threshold register
ad_ut6r    = r10                         ; Channel 6 upper threshold register
AD_UT7R   := R251                        ; Channel 7 upper threshold register
ad_ut7r    = r11                         ; Channel 7 upper threshold register
AD_CRR    := R252                        ; Compare result register
ad_crr     = r12                         ; Compare result register
          .defstr AD_c6l    "ad_crr.4" ; Compare channel 6 lower bit
          .defstr AD_c7l    "ad_crr.5" ; Compare channel 7 lower bit
          .defstr AD_c6u    "ad_crr.6" ; Compare channel 6 upper bit
          .defstr AD_c7u    "ad_crr.7" ; Compare channel 7 upper bit
c6l       := ( 1 <- 4 )                  ; Compare channel 6 lower mask
c7l       := ( 1 <- 5 )                  ; Compare channel 7 lower mask
c6u       := ( 1 <- 6 )                  ; Compare channel 6 upper mask
c7u       := ( 1 <- 7 )                  ; Compare channel 7 upper mask
```

APPENDIX B: Symbols.inc listing

```
AD_CLR    := R253                        ; Control logic register
ad_clr     = r13                         ; Control logic register
          .defstr AD_st      "ad_clr.0" ; start/stop bit
          .defstr AD_cont    "ad_clr.1" ; Continuous mode
          .defstr AD_pow     "ad_clr.2" ; power up/down control
          .defstr AD_intg    "ad_clr.3" ; internal trigger
          .defstr AD_extg    "ad_clr.4" ; External trigger
st        := ( 1 <- 0 )                  ; start/stop bit mask
cont      := ( 1 <- 1 )                  ; Continuous mode mask
pow       := ( 1 <- 2 )                  ; power up/down control mask
intg      := ( 1 <- 3 )                  ; internal trigger mask
extg      := ( 1 <- 4 )                  ; External trigger mask
sch       := 0E0h                        ; scan channel selection mask
AD_ICR    := R254                        ; interrupt control register
ad_icr     = r14                         ; interrupt control register
          .defstr AD_awdi    "ad_icr.4" ; analog watch-dog interrupt
          .defstr AD_eci     "ad_icr.5" ; End of count interrupt
          .defstr AD_awd     "ad_icr.6" ; analog watch-dog pending flag
          .defstr AD_ecv     "ad_icr.7" ; End of conversion pending flag
AD_prl    := 07h                         ; priority level mask
awdi      := ( 1 <- 4 )                  ; analog watch-dog interrupt mask
eci       := ( 1 <- 5 )                  ; End of count interrupt mask
awd       := ( 1 <- 6 )                  ; analog watch-dog pending flag
ecv       := ( 1 <- 7 )                  ; End of conversion pending flag
AD_IVR    := R255                        ; interrupt vector register
ad_ivr     = r15                         ; interrupt vector register
;.page
```

APPENDIX B: Symbols.inc listing

```
;*******************************************************************************

;     ST9 FAMILY SERIAL COMMUNICATION INTERFACE REGISTERS.

;*******************************************************************************

SCI1_PG  := 24                          ; SCI1 control registers page
SCI2_PG  := 25                          ; SCI2 control registers page
SCI3_PG  := 26                          ; SCI3 control registers page
SCI4_PG  := 27                          ; SCI4 control registers page


S_RDCPR  := R240                        ; receive DMA counter pointer register
s_rdcpr   = r0                          ; receive DMA counter pointer register
S_RDAPR  := R241                        ; receive DMA address pointer register
s_rdapr   = r1                          ; receive DMA address pointer register
S_TDCPR  := R242                        ; transmit DMA counter pointer register
s_tdcpr   = r2                          ; transmit DMA counter pointer register
S_TDAPR  := R243                        ; transmit DMA address pointer register
s_tdapr   = r3                          ; transmit DMA address pointer register
S_IVR    := R244                        ; interrupt vector register
s_ivr     = r4                          ; interrupt vector register
S_ACR    := R245                        ; address compare register
s_acr     = r5                          ; address compare register
S_IMR    := R246                        ; interrupt mask register
s_imr    = r6                           ; interrupt mask register
         .defstr S_txdi    "s_imr.0"    ; transmitter data interrupt
         .defstr S_rxdi    "s_imr.1"    ; receiver data interrupt
         .defstr S_rxb     "s_imr.2"    ; receiver break
         .defstr S_rxa     "s_imr.3"    ; receiver address
         .defstr S_rxe     "s_imr.4"    ; receiver error
         .defstr S_txeob   "s_imr.5"    ; transmit end of block
         .defstr S_rxeob   "s_imr.6"    ; receive end of block
         .defstr S_hsn     "s_imr.7"    ; Holding or shift register empty.
txdi     := ( 1 <- 0 )                  ; transmitter data interrupt mask
rxdi     := ( 1 <- 1 )                  ; receiver data interrupt mask
rxb      := ( 1 <- 2 )                  ; receiver break mask
rxa      := ( 1 <- 3 )                  ; receiver address mask
rxe      := ( 1 <- 4 )                  ; receiver error mask
txeob    := ( 1 <- 5 )                  ; transmit end of block mask
rxeob    := ( 1 <- 6 )                  ; receive end of block mask
hsn      := ( 1 <- 7 )                  ; Holding or shift register empty mask.
```

APPENDIX B: Symbols.inc listing

```
S_ISR     := R247                          ; interrupt status register
s_isr      = r7                            ; interrupt status register

          .defstr S_txsem    "s_isr.0"    ; transmit shift register empty
          .defstr S_txhem    "s_isr.1"    ; transmit hold register empty
          .defstr S_rxdp     "s_isr.2"    ; received data pending bit
          .defstr S_rxbp     "s_isr.3"    ; received break pending bit
          .defstr S_rxap     "s_isr.4"    ; received address pending bit
          .defstr S_pe       "s_isr.5"    ; parity error pending bit
          .defstr S_fe       "s_isr.6"    ; framing error pending bit
          .defstr S_oe       "s_isr.7"    ; overrun error pending bit

txsem     := ( 1 <- 0 )                    ; transmit shift register empty mask
txhem     := ( 1 <- 1 )                    ; transmit hold register empty mask
rxdp      := ( 1 <- 2 )                    ; received data pending mask
rxbp      := ( 1 <- 3 )                    ; received break pending mask
rxap      := ( 1 <- 4 )                    ; received address pending mask
pe        := ( 1 <- 5 )                    ; parity error pending mask
fe        := ( 1 <- 6 )                    ; framing error pending mask
oe        := ( 1 <- 7 )                    ; overrun error pending mask

S_RXBR    := R248                          ; receive buffer register
s_rxbr     = r8                            ; receive buffer register

S_TXBR    := R248                          ; transmit buffer register
s_txbr     = r8                            ; transmit buffer register

S_IDPR    := R249                          ; interrupt/DMA priority register
s_idpr     = r9                            ; interrupt/DMA priority register

          .defstr S_txd      "s_idpr.3"   ; transmitter DMA
          .defstr S_rxd      "s_idpr.4"   ; receiver DMA
          .defstr S_sa       "s_idpr.5"   ; set address
          .defstr S_sb       "s_idpr.6"   ; set break
          .defstr S_amen     "s_idpr.7"   ; address mode enable

S_pri     := 07h                           ; interrupt/DMA priority mask
txd       := ( 1 <- 3 )                    ; transmitter DMA mask
rxd       := ( 1 <- 4 )                    ; receiver DMA mask
sa        := ( 1 <- 5 )                    ; set address mask
sb        := ( 1 <- 6 )                    ; set break mask
amen      := ( 1 <- 7 )                    ; address mode enable mask

S_CHCR    := R250                          ; Character configuration register
s_chcr     = r10                           ; Character configuration register

wl5       := 000h                          ; 5 bits data word mask
wl6       := 001h                          ; 6 bits data word mask
wl7       := 002h                          ; 7 bits data word mask
wl8       := 003h                          ; 8 bits data word mask
sb10      := 000h                          ; 1.0 stop bit mask
sb15      := 004h                          ; 1.5 stop bit mask
sb20      := 008h                          ; 2.0 stop bit mask
sb25      := 00Ch                          ; 2.5 stop bit mask
ab        := 010h                          ; address bit insertion mask
pen       := 020h                          ; parity enable mask
ep        := 040h                          ; Even parity mask
oddp      := 000h                          ; odd parity mask
am        := 080h                          ; address mode mask
```

APPENDIX B: Symbols.inc listing

```
S_CCR     := R251                          ; Clock configuration register
s_ccr      = r11                           ; Clock configuration register
          .defstr S_stpen    "s_ccr.0"    ; stick parity enable
          .defstr S_lben     "s_ccr.1"    ; loop back enable
          .defstr S_aen      "s_ccr.2"    ; auto echo enable
          .defstr S_cd       "s_ccr.3"    ; Clock divider
          .defstr S_xbrg     "s_ccr.4"    ; External baud rate generator source
          .defstr S_xrx      "s_ccr.5"    ; External receiver source
          .defstr S_oclk     "s_ccr.6"    ; output clock selection
          .defstr S_txclk    "s_ccr.7"    ; transmit clock selection
stpen     := ( 1 <- 0 )                    ; stick parity enable mask
lben      := ( 1 <- 1 )                    ; loop back enable mask
aen       := ( 1 <- 2 )                    ; auto echo enable mask
cd        := ( 1 <- 3 )                    ; Clock divider mask
xbrg      := ( 1 <- 4 )                    ; External baud rate generator source mask
xrx       := ( 1 <- 5 )                    ; External receiver source mask
oclk      := ( 1 <- 6 )                    ; output clock selection mask
txclk     := ( 1 <- 7 )                    ; transmit clock selection mask
S_BRGR    := RR252                         ; baud rate generator register
s_brgr     = rr12                          ; baud rate generator register
S_BRGHR   := R252                          ; baud rate generator reg. high
s_brghr    = r12                           ; baud rate generator reg. high
S_BRGLR   := R253                          ; baud rate generator reg. low
s_brglr    = r13                           ; baud rate generator reg. low
;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;                          ST9040 SECURITY REGISTER.

;*************************************************************************

SEC_PG    := 59                          ; Security register page
SECR       = R255
secr       = r15
          .defstr tlck       "secr.0"    ; test lock bit
          .defstr wf1        "secr.1"    ; write fuse 1 bit
          .defstr hlck       "secr.2"    ; hardware lock bit
          .defstr wf2        "secr.3"    ; write fuse 2 bit
          .defstr f2tst      "secr.4"    ; select fuse 2 bit
          .defstr slck       "secr.7"    ; software lock bit
tlckm     := ( 1 <- 0 )                  ; test lock bit mask
wf1m      := ( 1 <- 1 )                  ; write fuse 1 bit mask
hlckm     := ( 1 <- 2 )                  ; hardware lock bit mask
wf2m      := ( 1 <- 3 )                  ; write fuse 2 bit mask
f2tstm    := ( 1 <- 4 )                  ; select fuse 2 bit mask
slckm     := ( 1 <- 7 )                  ; software lock bit mask
   .list
```

INITIALIZATION OF THE ST9

Pierre Guillemin and Alan Dunworth

INTRODUCTION

The ST9 family offers the microprocessor designer a wide variety of architectural features configurable to the user's specific application requirements. Central to all these configurations is a multiple register based microcomputer core to which may be added on-chip, powerful peripheral components including A/D Convertors, Serial Communication Interface units (SCI's), and 16-bit Multifunction timers with input capture/output compare capabilities. The availability, on-chip, of these application-specific units obviates the need for external interface design as well as offering high-speed and good reliability.

The particular peripherals incorporated on-chip may themselves be individually configured to offer a wide variety of functional (architectural) alternatives. This configuration is typically implemented by simple software routines included in the power-on- or system- reset routines. The sole difficulty which the user may initially encounter stems, in fact, from the power and versatility of this approach to system design. The large number of available options means that the user must specify a large number of system parameters by initializing control register contents for the specific peripheral units.

The objective of this Application Note is to suggest to the user a programming structure and philosophy to aid in the initial configuration of the system. The approach is illustrated by a number of specific examples selected from the wide range available for the ST9030, ST9040 families, but are applicable to all ST9s.

System Reset

After processor Reset the control and status registers, located on the group F pages (0-63) are forced to preset values which define a default Reset configuration for the ST9 system. By way of example the internal clock frequency (INTCLK) is set to the internal crystal oscillator (or externally applied clock frequency, if supplied) divided by two without prescaling, and the individual pins of Parallel Ports 0,1, and 6 are set to bidirectional Pullup mode (for systems with on-chip ROM). On releasing the external RESET signal the processor PC is loaded with the contents of the Reset Vector stored in address locations 0 and 1. This causes a jump to a Reset routine in which the designer may reconfigure the ST9 system as appropriate to the requirements of his particular application, by loading suitable values into the system registers.

The number of registers to be initialized may be considerable for a representative ST9 system. Additionally, the application-specific interrupt routines will, in general, involve the manipulation of substantial system resources, e.g. read/write of data registers, and test/reset of status, mask, and control registers. The associated programming task may appear daunting in prospect on first acquaintance with the ST9 system. Conceptually, the organization of the associated software is relatively simple and straightforward as may be recognized by grouping under four headings the programming steps involved in the initialization of ST9 peripherals and the organization of interrupt service routines.

a) ST9 Core System Configuration

Certain core system resources are common to all on-chip peripherals and may be specified in a common routine which is invoked at System Reset. Such common resources include clock configuration, system and user stack specification, global interrupt masking, processor priority setting, parallel port bit-by-bit specification, and setting of external memory wait-cycles. The setting up of the interrupt vector table, and certain global masking or enabling operations, may also be included under this heading.

b) Individual On-chip Peripheral Configuration

The configuration of on-chip peripherals, e.g. Multifunction Timers, A/D Converters, etc., involves the loading of suitable bit-patterns into group F page registers. This enables the specification of input and output signals, determination of the peripheral's mode of operation, and the selection of internal or external clock and control signals.

c) Individual On-chip Peripheral Initialization

The initialization of a particular on-chip peripheral may involve the setting or clearing of device-specific enable and masking bits, specification of interrupt priority levels, clearing of status/flag values, and the loading of data and/or limit registers.

d) Organization of Interrupt Service Routines

This will normally include context-saving and restoring of the PC and system status, plus the working-register and page-pointer registers, together with the values of any working registers used in the routine. The routine proper may include testing of status flag bits, and the reading and writing of data registers associated with the particular device. Finally, the interrupt pending bits should be cleared, the context restored, and individual masking and enabling bits restored to the appropriate values.

In practical programming terms there will normally be a single routine invoked on system RESET which carries out the core system configurations listed under heading a) above. For each individual peripheral there will typically be a single routine which carries out the configuration and initialization operations listed under headings b) and c). There will also be one or more interrupt routines associated with each peripheral, e.g. the A/D converter may require in general two interrupt routines, one for End of Conversion, and one for out of range operation (i.e. Analog Watchdog operation) on channels 6 and 7.

An example of a core-system configuration is given in Appendix B, and Appendices C,D,E, and F give configuration/initialization examples, and Interrupt routines for the Timer, A/D Convertor, SCI unit, and Timer/Watchdog respectively.

There is not space in a short note to discuss these programmes in detail on a line by line basis. Instead the approach will be to list, for each device, the resources which need to be taken into consideration when configuring, initializing, and servicing the particular device. An example will then be given of the specific use of each such resource. With this background, the interested user should be able to follow in detail those listings most relevant to his particular application area.

ST9 BASIC SYSTEM CONFIGURATION

Tables A.1 and A.2 in Appendix A lists the registers which should be loaded with specified bit-patterns in order to initialize the ST9 to a basic system configuration. A demonstration routine which carries this out for a representative ST9 system is listed in Appendix B. The main routine, RESET_START, is invoked at system Reset. Also shown in Appendix B are the Assembler Declarations and directives which enable the Interrupt Vector Address Table to be set up in program memory.

The Vector Address Table

The ST9 implements an interrupt vectoring structure that allows the on-chip peripheral to identify the location of the first instruction of the Interrupt Service Routine (ISR). Each interrupt module has a specific Interrupt Vector Register (IVR) mapped on the register file pages. When the interrupt request is acknowledged, the peripheral interrupt module provides, via the IVR, the vector to point to the address of the Interrupt Service Routine in the Vector Table.

The Interrupt Vector table.containing the list of addresses of the Interrupt Service Routine must be located in the first 256 locations of program memory. The first 6 locations of Program memory are reserved as follows:

Address	Content
0	Address high of Power on Reset routine
1	Address low of Power on Reset routine
2	Address high of Divide by Zero Trap **Subroutine**
3	Address low of Divide by Zero Trap **Subroutine**
4	Address high of Top Level ISR
5	Address low of Top Level ISR

Note that since the above locations are fixed by the hardware no associated IVR register is involved. For certain interrupt modules more than one interrupt routine may be required. For example the A/D Convertor has separate interrupts for the End of Conversion and Channel 6/7 analog underflow/overflow conditions. In such cases the IVR register specifies the more significant, and the interrupt module hardware specifies the less significant bits of the Vector Table address.

The following Assembler outline shows how the corresponding Vector table entries may be established.

```
        ADC_IT_VECT:=    30h
        .
        .org    ADC_IT_VECT
        .word ADC_WDG
        .word ADC_EOC
        .
ADC_WDG:
        ; Code for the Analog Watchdog Routine is included here
        ; Note that in the example in Appendix B
        ; the System Reset routine is invoked for out of
        ; range conditions on Channels 6 and 7
        .
        iret
ADC_EOC:
        ; End of A/D conversion interrupt routine included here
        iret
```

PORT INITIALIZATION

The ST9 has up to a maximum of 64 lines dedicated to input/output. These lines, grouped into eight 8-bit ports, can be independently programmed to provide parallel input/outputs with or without handshake or may be used to connect in/out signals to/from the peripherals (e.g. Core, Timers, SCI units, etc.) present on the chip. The functional allocation of the Ports to support system tasks may be summarised as follows:

Port	Functions
0	Usable as I/O Port (without handshake) or as multiplexed low-address and data lines for external memory.
1	Usable as I/O Port (without handshake)or as high-address lines for external memory.
2	Usable as I/O Port (without handshake)or for SPI functions; Also INT1, INT2, and INT3 inputs.
3	Usable as I/O Port (without handshake)or for Timer functions.
4	Usable as I/O Port (with or without handshake)
5	Usable as I/O Port (with or without handshake).
6	Usable as I/O Port (without handshake)
7	Usable as I/O Port (without handshake) or for SCI functions. Also used for INT4, INT5, and INT6 inputs or for Control signals for slow external memory

Ports 0, 1, and 6 are automatically initialized on system Reset to correspond to the installed on-chip memory. Ports 2, 3, 4, 5, 6, and 7 need to be initialized (if available) to satisfy the specific application requirements for external I/O, plus any alternative function assignments of port pins, and internal interconnections. Table A.3, Appendix A, lists the complete set of Port Configuration registers together with their addresses.

Example:

```
C7 0A          spp     P3C_PG
F5 FC 05       ld      P3C0R,#00000101b
F5 FD 0F       ld      P3C1R,#00001111b
F5 FE 05       ld      P3C2R,#00000101b
```

In this example Port 3 pins 4, 5, 6, and 7 are configured as bidirectional pins, with weak pull-up output and TTL inputs. Pins 0 (T0INA) and 2 (T0INB) are configured as TTL inputs, and Pins 1 (T0OUTA) and 3 (T0OUTB) are configured as Alternate Function Push-pull outputs.

MULTIFUNCTION TIMER CONFIGURATION

The ST9 Multifunction Timer is configured by loading suitable control-bit patterns in the groupe F page register TCR, TMR, ICR, OACR, and OBCR (see Table A.4 in Appendix A). Note that registers EIMR and CICR provide global control functions common to all on-chip peripherals and are hence initialized conveniently in the basic system configuration routine.

The **External Input Control Register, ICR**, controls input source selection (internal/external), input mode selection (falling/rising edge sensitive, etc.), counter mode of operation (continuous, one-shot, etc.), and input function (Gate, Trigger, up/down control, etc.).

Example:

```
F5 FA 54    ld    T_ICR,#01010100b
```

This instruction selects the external input A as a falling-edge-sensitive Trigger input, and the B input is a normal Port I/O pin.

The **Multifunction Timer Control Register, TCR**, controls counter clear and prescaler reload operations as well as providing a counter enable control bit and counter status flags.

Example:

```
F5 F8 48    ld    T_TCR,#01001000b
```

This instruction halts the counter operation but provides for subsequent UP counting with counter clear and Prescaler reload on Reg0 or Reg1 capture.

The **Multifunction Timer Mode Register, TMR**, selects the clock source for the counter-prescaler input, enables Retrigger or Continuous mode, and controls register load/capture operations.

Example:

```
98 8C ld    T_TMR,#10001100b
```

This pattern enables output 1 and disables output 0, disables bivalue modes, and selects Reg0 for capture and Reg1 for monitor. Retriggerable continuous mode is selected.

The **Output Control Register, OACR,** links the output T0OUTA to counter overflow/underflow and Compare events, and provides for subsequent Set, Reset, or Toggle of the external output. The on-chip event (OCE) may be linked to a COMP0 event.

Example:

```
F5 F5 1B    ld    T_OACR,#00011011b
```

In this example T0OUTA is preset to 1, and is subsequently set by COMP0, toggled by COMP1, and Reset by OVF. The OCE signal is generated by a successful CMP0 compare event.

The **Output Control Register, OBCR**, links the output T0OUTB to counter overflow/underflow and Compare events, and provides for subsequent Set, Reset, or Toggle of the external output. The on-chip event (OCE) may be linked to a counter overflow/underflow event.

Example:

```
F5 F6 83    ld    T_OBCR,#10000011b
```

In this example T0OUTB is preset to 1, and is subsequently reset by COMP0, and set by OVF and COMP1. The OCE signal is generated by a counter overflow/underflow event.

MULTIFUNCTION TIMER INITIALIZATION

Initialization of the Multifunction Timer requires loading of the Prescaler register and the two Comparison registers. The timer Status register should be cleared, the Vector Table entry should be set, and the Multifunction Timer counter actions enabled. The interrupt/DMA priority levels should be set and the mask bits should be adjusted as appropriate to the application. Further, if DMA operations are specified, DMA address and counter registers will require initialization.

The **Prescaler Register, PRSR,** holds the preset value for the 8-bit prescaler.

Example:

```
BC 00          ld    T_PRSR,#00h
```

This defines a division ratio of 1 and the maximum counter clock is generated (INTCLK/3).

The **Multifunction Timer Flags Register, FLAGR**, contains flags which register successful capture or comparison events together with OVF/UNF and overrun conditions.

Example:

```
15 FE FD     and    T_FLAGR,#~ocm0
```

This example resets the overrun bit for COMP0 operations.

The **Interrupt Vector Register, IVR,** should be loaded with the 5 most significant bits of the Multifunction Timer's interrupt vector address in program memory. The interrupt source (compare, capture, or OVF/UNF) provides the least significant 3 bits to provide the correct vector link.

Example:

```
F5 F2 10     ld     T0_IVR,#T0_IT_VECT
```

In this example IVR is loaded with the start address (10h) of the block of 8 words in the vector table allocated to the 5 different Multifunction Timer interrupts.

MULTIFUNCTION TIMER INITIALIZATION (Continued)

The **Interrupt/DMA Control Register, IDCR,** is used to set the Interrupt and DMA priority levels, and the DMA transfer source and destination. It also enables Swap mode and contains End of Block condition flags.

Example:

```
F5 F3 D6    ld    T0_IDCR,#11000110b
```

In this example the priority level is set at a value of 6, and the Swap mode is disabled. The DMA capture channel source is REG0, and the DMA compare channel source is CMP0.

The **Interrupt/DMA Mask Register, IDMR,** contains a global Multifunction Timer Interrupt enable plus individual DMA and Interrupt enable bits for overflow as well as successful capture and comparison events.

Example:

```
F5 FF 04    ld    T_IDMR,#00000100b
0F FF 80    or    T_IDMR,#gtien
```

The first instruction sets the interrupt enable on CMP0, and the second instruction globally enables all Multifunction Timer interrupts.

The **DMA Counter Pointer Register, DCPR,** defines the DMA area and source, and specifies the location of the DMA length register.

Example:

```
F5 F0 4C    ld    T0_DCPR,#CPT_LG_DMA
```

The DMA length register is 4Ch = rr12 = RR76 and the transfer occurs to/from Program/Data memory.

The **DMA Address Pointer Register, DAPR**, defines the DMA area and source, and specifies the location of the DMA address register.

Example:

```
F5 F1 48    ld    T0_DAPR,#CPT_AD_DMA
```

The DMA address register is 48h = rr8 = RR72. In conjunction with the DPCR value in the above example it specifies Program memory for the buffer.

A/D CONVERTOR CONFIGURATION/INITIALIZATION

Configuration of the A/D convertor requires loading of 4 registers only, CLR, CRR, ICR, and IVR (Table A.6), and initialization of this device involves, apart from global masking, loading of two double (threshold registers). Hence a single routine can be written to cover both the configuration and initialization aspects of A/D Convertor use.

The **Control Logic Register, CLR**, defines the Analog channel conversion start address, selects internal/external triggers, and enables continuous or single conversion and power up/down modes. This register also contains a start/stop status/control bit.

Example:

```
F5 FD 04    ld    AD_CLR,#00000100b
```

In this example, the conversion scan starts with channel 0 when enabled, powers up the A/D convertor, halts conversion, and specifies single conversion scan mode.

Please note that before enabling any A/D conversion, it is mandatory to set the low bit of Control Logic Register at least 60µs before the first conversion start. This is in order to correctly bias the analog section of the converter.

The **Interrupt Vector Register, IVR**, defines the most significant 6 bits of the vector table byte address. It thus points to the first of two word addresses which correspond to the analog watchdog and End of conversion interrupt routines.

Example:

```
F5 FF 32    ld    AD_IVR,#ADC_ITEOC_VECT
```

In this example, an address of 50 (decimal) is loaded into IVR. Hence a subsequent A/D convertor EOC interrupt will cause a Vector Table access at location 50.

The **Interrupt Control Register, ICR**, contains the priority level specification, the two source interrupt flags (Analog Watchdog and EOC) and their individual masking bits.

Example:

```
F5 FE 20    ld    AD_ICR,#00100000b
05 FE 20    or    AD_ICR,#00000110b
```

In this example, the priority level is first set at 0, End of Conversion interrupts are enabled, and the Analog Watchdog interrupt is masked. The second instruction then sets the priority to a level of 6.

If the Analog Watchdog is enabled (bit 6 in ICR) it will be necessary to load the threshold registers for channels 6 and 7. In this case access will be made in the interrupt routine to register CRR.

The **Compare Result Register, CRR,** contains 4 flags showing the results of comparison operations between the current values of data registers 6 and 7, and the upper and lower threshold registers.

SCI UNIT CONFIGURATION

The list of registers to be initialized when configuring the SCI unit is given in Table A.9. The functions of these registers, and some illustrative examples of their use, are as follows:

The **Character Configuration Register, CHCR**, is used to define the serial frame format.

Example:

```
AC E3          ld    S_CHCR,#E3h
```

This example defines a serial frame as follows: 8 data bits, 1 stop bit, even parity, and address input if the character matches the contents of the Address Register.

The **Clock Configuration Register, CLCR**, is used to specify the transmitter, receiver, and Baud Rate clock sources, and the clock divisor ratio. It also enables Auto Echo and Loopback test modes.

Example:

```
BC 80          ld    s_clcr,#txclk
```

In this example, the Transmitter and Receiver clocks are provided by the Baud Rate Generator. Each data bit period will be 16 clock periods (asynchronous mode), and the Auto Loop and Loopback modes are disabled.

The **Baud Rate Generator Register, BRGR,** specifies a 16-bit division ratio.

Example:

```
BF DC 00 4E ldw    s_brgr,#DIV_9600
```

This example specifies a division ratio yielding 9600 Bauds with a 24 Mhz external clock.

Writing to a Baud Rate Generator Register immediately disables and resets both the SCI Baud Rate generator, the transmitter and receiver circuitry. After writing to the remaining Baud Rate Generator Register, the transmitter and receiver circuits are enabled. The Baud Rate Generator will load the new value and start counting.

To initialize the SCI, user should first initialize one Baud Rate Generator Divisor Register. This will reset all SCI circuitry. Initialize all other SCI registers for the desired operating mode. To enable the SCI, initialize the remaining Baud Rate Generator Register.

The **Address Compare Register, ACR,** contains an 8-bit value which may be used as a match against which a received address may be tested to set the Receive Address Pending bit.

Example:

```
5C 0D          ld     s_acr,#RETURN
```

This will cause the Receive Address Pending bit to be set if an End of Command character bit-pattern is received.

SCI UNIT CONFIGURATION (Continued)

The **Interrupt Vector Register, IVR**, defines the most significant 5 bits of the vector table byte address. It thus points to the first of four vector table word address entries.

Example:

```
4C 00x      ld     s_ivr,#SCI_IT
```

In this example, after the external symbol has been linked in, the Vector Table entry address will be loaded into IVR at execution time.

The **Interrupt Mask Register, IMR,** contains five interrupt masking bits and two End of Block DMA status bits. It also selects the shift register or holding register as source of the transmitter register empty interrupt.

Example:

```
6C 05          ld    s_imr,#00000101b
```

In this example the interrupt pending bits are reset, the Transmitter data interrupt is masked, and the Receiver data, data error, and address interrupts are unmasked.

The **Interrupt/DMA Priority Register, IDPR,** specifies the Interrupt/DMA priority, selects one of four Address modes, and controls the emission of Break characters and enables address/9th bit data mode. It also provides mask bits for Receive and Transmit DMA transfers.

Example:

```
9C 04          ld    s_idpr,#04h
```

In this example a priority level of 4 is specified, and Transmitter DMA requests are masked.

SCI UNIT INITIALIZATION

The list of registers to be initialized when initializing the SCI unit is given in Table A.10. The functions of these registers, and some illustrative examples of their use, are as follows:

The **Receiver DMA Transaction Counter Pointer Register, RDCPR**, contains the register file address of the receiver DMA transaction counter. In addition it determines whether the DMA transfers occur in the register file or in memory.

Example:

An example of the use of this register is provided below (see RDAPR example).

The **Receiver DMA Destination Address Pointer Register, RDAPR**, contains the register file address of the receiver DMA data destination. In addition, in conjunction with bit 0 of RDCPR, it determines whether the DMA transfers occur in Program or Data memory.

Example:

```
00 FF          LNG-DMA_SCI        :=     0Fh
00 A0          DEPART_DMA_SCI     :=     0A0h
00 02          NUM_RDAP           :=     2
00 03          NUM_RDCP           :=     3
2C 03          ld    S_rdcpr,#NUM_RDCP
1C 02          ld    S_rdapr,#NUM_RDAP
F5 03 0F       ld    R#NUM_RDCP,#(LNG_DMA_SCI)
F5 02 00       ld    R#NUM_RDAP,#(DEPART_DMA_SCI)
```

In this program sequence the DMA transaction counter and Address Pointer register addresses are defined to be R3 and R2 respectively. These two registers are initialized for a block of size 15 bytes starting at register address A0, i.e. R160.

The **Transmitter DMA Transaction Counter Pointer Register, TDCPR**, contains the register file address of the transmitter DMA transaction counter. In addition it determines whether the DMA transfers occur in the register file or in memory.

Example:

An example of the use of this register is provided below (see TDAPR example).

The **Transmitter DMA Destination Address Pointer Register, TDAPR**, contains the register file address of the transmitter DMA data destination. In addition, in conjunction with bit 0 of TDCPR, it determines whether the DMA transfers occur in Program or Data memory.

Example:

```
00 FF          LNG-DMA_SCI        :=     0Fh
00 A0          DEPART_DMA_SCI     :=     0A0h
00 06          NUM_TDAP           :=     6
00 07          NUM_TDCP           :=     7
2C 07          ld    S_TDCPR,#NUM_TDCP
3C 06          ld    S_TDAPR,#NUM_TDAP
F5 07 0F       ld    R#NUM_TDCP,#(LNG_DMA_SCI)
F5 06 00       ld    R#NUM_TDAP,#(DEPART_DMA_SCI)
```

In this program sequence the DMA transaction counter and Address Pointer register addresses are defined to be R7 and R6 respectively. These two registers are initialized for a block of size 15 bytes starting at register address A0, i.e. R160.

TIMER/WATCHDOG UNIT CONFIGURATION

Configuration of the Timer/Watchdog requires loading of the 6 registers listed in Table A.11, Appendix A.

Timer/Watchdog unit Configuration

The **Timer/Watchdog Control Register, WDTCR**, contains a start/stop bit, and is also used to select input, output, and counter modes, as well as input and output enable bits.

Example:

```
BC 80          ld    wdtcr,#80h
```

In this example the Timer starts counting down in continuous mode, and the input and output sections are disabled.

The **Wait Control Register, WCR**, as well as specifying the number of wait states for access to off-chip program and data memory enables the Watchdog function.

Example:

```
CC 40          ld    wcr,#wden
```

In this example the Watchdog action is disabled, and the number of wait states are set to zero.

The **External Interrupt Vector Register, EIVR**, contains a bit, TLIS, which is used to control the Top Level Interrupt source (Timer/Watchdog EOC or External NMI). A second bit IAOS is used to select the Timer/Watchdog as an interrupt source on channel A0 (INT0). This register is also used to supply the 4 most significant bits of the External Interrupt Vector.

Example:

```
6C 20          ld    eivr,#EXT_IT_VECT
```

In this example the Timer/Watchdog EOC generates an interrupt on channel A0 at each End of Count. The Top Level Interrupt is isolated from the NMI input and may be used for a Software Trap.

The **Timer/Watchdog Prescaler Register, WDTPR,** contains an 8-bit value which is loaded into the Prescaler register.

Example:

```
90 DA          clr   wdtpr
```

The specified Prescaler value of zero leads to a minimum timer count period of 333ns, assuming a system clock running at 12MHz.

The **Timer/Watchdog High Register, WDTHR**, and **Timer/Watchdog Low Register, WDTLR,** together contain a 16-bit value which is loaded into the counter at each End of Count.

Example:

```
BF F8 0B BB ldw    WDTR,#3003
```

The specified count value leads to a count period of about 1 millisecond, (3003 x 333ns).

TIMER/WATCHDOG UNIT INITIALIZATION

The **External Interrupt Priority Level Register, EIPLR**, specifies the priority level of four pairs of external interrupts, a), A1,...D0, D1. It is thus used to set the priority of the Timer/Watchdog EOC interrupt routine, called via channel A0.

Example:

```
5C FE          ld   eiplr,#0FEh
```

In this example priority levels of 4 and 5 are specified for the pair INTA0, INTA1.

The **External Interrupts Pending Bit Register, EIPR,** holds the eight interrupt pending bits for the external interrupts, including, in the present context, the Watchdog/Timer EOC interrupt. These bits are set by hardware action and reset by software during the service routine.

Example:

```
90 D3          clr  eipr
```

In this example all the external interrupt pending bits are cleared.

The **External Interrupts Mask-Bit Register, EIMR,** holds the eight interrupt mask bits for the external interrupts, including, in the present context, the Timer/Watchdog EOC interrupt.

Example:

```
4C 01          ld   eimr,#ia0
```

In this example the Timer/Watchdog End of Count on Channel A0 is unmasked.

INTERRUPT SERVICE ROUTINE ORGANIZATION

When an enabled interrupt is acknowledged the Interrupt machine cycle performs the following actions:

(i) All maskable interrupts are disabled by clearing the EI bit of register CICR.

(ii) The PC (two bytes) and the FLAGS register are saved on the System stack.

(iii) The PC is loaded with the 16-bit vector stored in the Vector Table.

On exit from the Interrupt service, using an IRET instruction the following operations are carried out:

(iv) The FLAGR register is restored from the System stack.

(v) The PC is restored from the System stack.

(vi) The unmasked interrupts are enabled by setting the CICR.EI bit.

In general additional resources must be saved and restored apart from those handled automatically by the system as listed above. In a typical case these additional resources will include the two Register pointer registers, the Page-pointer register, and any working registers used in the Interrupt routine.

An outline for a suitable Interrupt service routine is hence as follows:

```
Label_int:
      work_reg_page0 =      (0Dh*2)
      work_reg_page1 =      (0Dh*2) + 1
      WDT_PG         =      0
      T0c_PG         =      9
      T0d_PG         =      10
      S_PG           =      24
      AD0_PG         =      63
      push   RP0
      push   RP1
      push   PPR
      spp    #T0d_PG
      srp0   #work_reg_page0
      srp1   #work_reg_page1
      push   r0
      push   r1
      push   rA
             ;
             ;
             ;Interrupt Service routine
             ;appears here, including
             ;read/write data registers
             ;test status flags
             ;clear interrupt pending flags
      pop    rA
      pop    r1
      pop    r0
      pop    PPR
      pop    RP1
      pop    RP0
      iret
```

SUMMARY

This Application Note has attempted to formalize and simplify the programming task of configuring and initializing an ST9 system. The resources to be controlled have been listed with brief examples of their use. Complete examples of ST9 configuration, initialization, and Interrupt Service routines are presented in a set of Appendices. These programs have been written for an ST9030 but can be readily adapted where necessary for use with other versions.

REFERENCES

(1) "ST9 Technical Manual", SGS-THOMSON Microelectronics.

(2) Application Note AN411, **SYMBOLS.INC** Standard Definitions of ST9 Registers and Register-bits.

APPENDICES

A. ST9 Core and Peripheral Configuration/Initialization Registers.

- A.1. System Configuration: System Registers.
- A.2. System Configuration: Paged Registers.
- A.3. Port Configuration Registers.
- A.4. Multifunction Timer Configuration/Initialization Registers.
- A.5. Multifunction Timer Data/Status Register.
- A.6. A/D Configuration/Initialization Registers.
- A.7. A/D Channel Registers.
- A.8. A/D Threshold Registers
- A.9. SCI Configuration Registers.
- A.10. SCI Initialization Registers.
- A.11. Watchdog Timer Configuration/Initialization Registers.
- A.12. SPI Initialization.
- A.13. EEPROM Initialization.

B. Examples of ST9 System Configurations.

C. Examples of Multifunction Timer 0 Configurations.

D. Examples of A/D Converter Configurations.

E. Examples of SCI Configurations.

F. Examples of Timer/Watchdog Configurations.

APPENDIX A. ST9 CORE AND PERIPHERAL CONFIGURATION/INITIALIZATION

A.1. System Configuration: System Registers

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Hex)
CICR	Central Interrupt Control Register	R230	E6	-	87
FLAGR	Flags Register	R231	E7	-	XX
RP0R	Register Pointer 0	R232	E8	-	XX
RP1R	Register Pointer 1	R233	E9	-	XX
PPR	Page Pointer Register	R234	EA	-	XX
MODER	Mode Register	R235	EB	-	E0
USPHR	User Stack Pointer (high)	R236	EC	-	XX
USPLR	User Stack Pointer (low)	R237	ED	-	XX
SSPHR	System Stack Pointer (high)	R238	EE	-	XX
SSPLR	System Stack Pointer (low)	R239	EF	-	XX

A.2. System Configuration: Page Registers

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Hex)
EECR	EEPROM Control Register Mask Register	R241	F1	0	87
EITR	External Interrupt Trigger-Event Register	R242	F2	0	XX
EIPR	External Interrupt Pending Register	R243	F3	0	XX
EIMR	External Interrupt Mask Register	R244	F4	0	XX
EIPLR	External Interrupt Priority Level Register	R245	F5	0	XX
EIVR	External Interrupt Vector Register	R246	F6	0	E0
NICR	Nested Interrupt Control Register	R247	F7	0	XX
WCR	Wait Control Register	R252	FC	0	7F

A.3. Port Configuration Registers

Port	Name	Registers	Hex	Pg. (Hex)
0	Data Register Control Registers (PxC0-PxC2)	R224 R240-R242	E0 F0-F2	- 2
1	Data Register Control Registers (PxC0-PxC2)	R225 R244-R246	E1 F4-F6	- 2
2	Data Register Control Registers (PxC0-PxC2) Handshake Control Register	R226 R248-R250 R251	E2 F8-FA FB	- 2 2
3	Data Register Control Registers (PxC0-PxC2) Handshake Control Register	R227 R252-R254 R255	E3 FC-FE FF	- 2 2
4	Data Register Control Registers (PxC0-PxC2) Handshake Control Register	R228 R240-R242 R243	E4 F0-F2 F3	- 3 3
5	Data Register Control Registers (PxC0-PxC2) Handshake Control Register	R229 R244-R246 R247	E5 F4-F6 F7	- 3 3
6	Data Register Control Registers (PxC0-PxC2)	R251 R248-R250	FB F8-FA	3 3
7	Data Register Control Registers (PxC0-PxC2)	R255 R252-R254	FF FC-FE	3 3

RESET Values:
Ports 2, 3, 4, and 5: PcX0: 00000000
PcX1: 00000000
PcX2: 00000000
Handshake Control Registers: 111111111

A.4. Multi-Function Timer Configuration/Initialization Registers (MFT0)

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Binary)
CICR	Central Interrupt Control Register	R230	E6	-	10000111
TCR	Timer Control Register	R248	F8	10	00000XXX
TMR	Timer Mode Register	R249	F9	10	00000000
ICR	External Interrupt Control Register	R250	FA	10	0000XXXX
OACR	Output A Control Register 0	R252	FC	10	XXXXXX0X
OBCR	Output A Control Register 1	R253	FD	10	XXXXXX0X
IDMR	Interrupt/DMA Mask Register	R255	FF	10	00000000
DCPR	DMA Counter Pointer Register	R240	F0	9	XXXXXXXX
DAPR	DMA Address Pointer Register	R241	F1	9	XXXXXXXX
IVR	Interrupt Vector Register	R242	F2	9	XXXXXXXX
IDCR	Interrupt/DMA Control Register	R243	F3	9	11000111

A.5. Timer Data/Status Registers (MFT0)

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Binary)
REG0HR	Capture/Reload Register 0 (High)	R240	F0	10	XXXXXXXX
REG0LR	Capture/Reload Register 0 (Low)	R241	F1	10	XXXXXXXX
REG1HR	Capture/Reload Register 1 (High)	R242	F2	10	XXXXXXXX
REG1LR	Capture/Reload Register 1 (Low)	R243	F3	10	XXXXXXXX
CMP0HR	Compare Register Register 0 (High)	R244	F4	10	XXXXXXXX
CMP0LR	Compare Register Register 0 (Low)	R245	F5	10	XXXXXXXX
CMP1HR	Compare Register Register 1 (High)	R246	F6	10	XXXXXXXX
CMP1LR	Compare Register Register 1 (Low)	R247	F7	10	XXXXXXXX
PRSR	Prescaler Register	R251	FB	10	00000000
FLAGR	Timer Flags Register	R254	FE	10	00000000

A.6. A/D Configuration/Initialization Registers

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Binary)
CRR	Compare Result Register	R252	FC	63	00001111
CLR	Control Logic Register	R253	FD	63	00000000
ICR	Interrupt Control Register	R254	FE	63	00001111
IVR	Interrupt Vector Register	R255	FF	63	XXXXXX10

A.7. A/D Channel Registers

Mnem.	Name	Reg.	Hex	Pg.
AD_D0R	Channel 0 Data Register	R240	F0	63
AD_D1R	Channel 1 Data Register	R241	F1	63
AD_D2R	Channel 2 Data Register	R242	F2	63
AD_D3R	Channel 3 Data Register	R243	F3	63
AD_D4R	Channel 4 Data Register	R244	F4	63
AD_D5R	Channel 5 Data Register	R245	F5	63
AD_D6R	Channel 6 Data Register	R246	F6	63
AD_D7R	Channel 7 Data Register	R247	F7	63

A.8. A/D Threshold Registers

Mnem.	Name	Reg.	Hex	Pg.
AD_LT6R	Channel 6 Lower Threshold Register	R248	F8	63
AD_UT6R	Channel 6 Upper Threshold Register	R249	F9	63
AD_LT7R	Channel 7 Lower Threshold Register	R250	FA	63
AD_UT7R	Channel 7 Upper Threshold Register	R251	FB	63

A.9. SCI Configuration Registers

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Binary)
IVR	Interrupt Vector Register	R244	F4	24	XXXXXXXX
IMR	Interrupt Mask Register	R246	F6	24	0XX00000
ISR	Interrupt Status Register	R247	F7	24	XXXXXXXX
IDPR	Interrupt/DMA Priority Register	R249	F9	24	XXXXXXXX
CHCR	Character Recognition Register	R250	FA	24	XXXXXXXX
CCR	Clock Configuration Register	R251	FB	24	00000000
BRGHR	Baud Rate Generator Divisor Register (High)	R252	FC	24	XXXXXXXX
BRGLR	Baud Rate Generator Divisor Register (Low)	R253	FD	24	XXXXXXXX

A.10. SCI Initialization

Mnem.	Name	Reg.	Hex	Pg.	Reset Value
RDCPR	Receiver DMA Transaction Counter Register	R240	F0	24	XXXXXXXX
RDAPR	Receiver DMA Address Pointer Register	R241	F1	24	XXXXXXXX
TDCPR	Transmit DMA Transaction Counter Register	R242	F2	24	XXXXXXXX
TDAPR	Transmit DMA Address Pointer Register	R243	F3	24	XXXXXXXX
ACR	Address Compare Register	R245	F5	24	XXXXXXXX
RXBR	Receive Buffer Register (Read only)	R248	F8	24	XXXXXXXX
TXBR	Transmitter Buffer Register (Write only)	R248	F8	24	XXXXXXXX

A.11. Watchdog Timer Configuration/Initialization

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Binary)
EIPR	External Interrupt Pending Register	R243	F3	0	00000000
EIMR	External Interrupt Masking Register	R244	F4	0	00000000
EIPLR	External Interrupt Priority Register	R245	F5	0	11111111
EIVR	External Interrupt Vector Register	R246	F6	0	XXXX0010
WDTLR	Watchdog Timer Low Register	R248	F8	0	XXXXXXXX
WDTHR	Watchdog Timer High Register	R249	F9	0	XXXXXXXX
WDTPR	Watchdog Timer Prescaler Register	R250	FA	0	XXXXXXXX
WDTCR	Watchdog Timer Control Register	R251	FB	0	00010010
WCR	Wait Control Register	R252	FC	0	01111111

A.12. SPI Initialization

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Binary)
SPIDR	SPI Data Register	R253	FD	0	XXXXXXXX
SPICR	SPI Control Register	R244	F4	0	00100000

A.13. EEPROM Initialization (ST9040 only)

Mnem.	Name	Reg.	Hex	Pg.	Reset Value (Binary)
EECR	EEPROM Control Register	R241	F1	0	00000000

APPENDIX B. EXAMPLES OF ST9 PERIPHERAL CONFIGURATIONS

```
      . sbttl     " ST9030 registers addresses and contents "
      . include "c:\st9\bin\symbols.inc"
      ; The reader should refer to the file containing the
      ; declaration of all the bits and registers of the ST9030
      ; for the symbols used in the following listing.
      ;
      ; .nlist

;*********************************************************************
;* This program demonstrates the configuration of ST9 peripherals*
;*********************************************************************
;**********************
;*RAM Declaration*
;**********************
prescal_t0 :=     r2          ; Value of Timer 0 Prescaler
val_capt_t0       :=    rr4         ; Value of Timer 0 Capture register
nb_event_t0       :=    rr4         ; Number of Timer 0 event
lg_dma     :=     rr6         ; Length of DMA
CPT_AD_DMA :=     RR8         ; DMA Address Register
CPT_LG_DMA :=     RR8         ; DMA Counter Register

ad_conv           :=    r3          ; conversion start address

IT_T0_LEVEL       =     4           ; Timer 0 priority level
IT_CAD_LEVEL      =     6           ; A/D converter priority level

;****************************
;*INTERRUPT VECTOR ADDRESSES*
;****************************
CORE_IT_VECT      :=    00h         ; Core interrupt vectors
T0_IT_VECT :=     10h         ; Timer 0 interrupt vectors
EXT_IT_VECT       :=    20h         ; External interrupt vectors
ADC_IT_VECT       :=    30h         ; A/D Converter  interrupt vectors
SCI_IT     :=     40h         ; SCI interrupt vector

;*******************
;*STACK Declaration*
;*******************
SSTACK     :=     223         ; System stack address group D C
USTACK     :=     191         ; User stack address group B
```

APPENDIX B. EXAMPLES OF ST9 PERIPHERAL CONFIGURATIONS (Continued)

```
;********************
;*Group number names*
;********************
BK0         :=      0
BK1         :=      1
BK2         :=      2
BK3         :=      3
BK4         :=      4
BK5         :=      5
BK6         :=      6
BK7         :=      7
BK8         :=      8
BK9         :=      9
BKA         :=      10
BKB         :=      11
BKC         :=      12
BKD         :=      13
BKE         :=      14
BKF         :=      15
BK_0        :=      BK0 * 2                 ; free user group
BK_BDT:=    BK2 * 2             ; TWD group
BK_CAD:=    BK5 * 2             ; A/D group
BK_T0 :=    BK4 * 2             ; MFTimer 0 group
BK_SCI:=    BK6 * 2             ; SCI group.
BK_F        :=      BKF * 2                 ; paged registers
```

APPENDIX B. EXAMPLES OF ST9 PERIPHERAL CONFIGURATIONS (Continued)

```
;*****************************************
;*Declaration of the interrupt vector table*
;*****************************************
      .text                                ; start of program
           .org    CORE_IT_VECT      ; Core interrupt vector
                                      **********************
              .word      DIV0               ; divide by 0 interrupt vector
              .word      TOP_LEVEL_IT; Top level interrupt vector

           .org    T0_IT_VECT               ; Timer 0 interrupt vector
                                            ; **********************

           .org    T0_IT_VECT + 4           ; unused addresses
              .word      T0_CAP       ; Timer 0 capture interrupt vector
              .word      T0_COMP            ; Timer 0 compare interrupt vector

           .org    EXT_IT_VECT              ; External interrupt vector
                                            ; ************************

WDT_IT:       .word      TEMPO              ; Watchdog Timer interrupt vector

           .org ADC_IT_VECT                 ; ADC interrupt vector
                                            ; ********************

              .word RESET_START             ; Analog Watchdog interrupt vector
              .word ADC_EOC                 ; End of conv. interrupt vector

           .org SCI_IT                      ; SCI interrupt vector
                                            ; ********************
           .org    SCI_IT + 4               ; unused addresses
              .word      REC_DATA           ; receiver interrupt
              .word      TRA_HOLD           ; Transmitter interrupt
```

APPENDIX B. EXAMPLES OF ST9 PERIPHERAL CONFIGURATIONS (Continued)

```
;**********************
;*Start of main module*
;**********************

      .org 100h                        ; start of code

RESET_START:
      ld   MODER,#11100000b            ; CLOCK MODE REGISTER
                                       ; internal stack
                                       ; no prescaling
                                       ; external clock divided by 2

      ld   CICR,#10000111b             ; CENTRAL INTERRUPT
                                       ; CONTROL REGISTER
                                       ; priority level = 7
                                       ; concurrent mode
                                       ; disable interrupt
      clr  FLAGR

      spp  #WDT_PG
      ld   WCR,#wden                   ; watch dog mode disabled,
                                       ; no wait states.
      ld   EIMR,#0                     ; mask all channel interrupts.
                                       ; at reset,Global Counter Enable
                                       ; bit is active.

      ld SSPLR,#SSTACK + 1             ; load system stack pointer
      ld USPLR,#USTACK + 1             ; load user stack pointer

      call INIT_IO              ; init I/O ports

MAIN:
      jxt MAIN                         ; include your Main program here !
```

APPENDIX B. EXAMPLES OF ST9 PERIPHERAL CONFIGURATIONS (Continued)

```
;*************************************************************
;*Configuration of TIMER 0 I/O pins and A/D Converter I/O pins*

proc  INIT_IO [PPR]      {
;.............
;.............. P3.0 (T0INA)  P3.2 (T0INB) INPUT TRISTATE TTL
;.............. P3.1 (T0OUTA) P3.3 (T0OUTB) OUTPUT ALTERNATE FUNCTION
                                    PUSH_PULL  TTL
      spp  #P3C_PG                  ; Port 3 control register page

      ld   P3C0R,#00001111b
      ld   P3C1R,#00001010b
      ld   P3C2R,#00000101b

;.............. end of init. P3

;.............. INITIALIZATION OF A/D CONVERTOR INPUTS
;.............. P4.7 (AIN7) ALTERNATE FUNCTION OPEN DRAIN TTL
;.............. P4.6 (AIN6) ALTERNATE FUNCTION OPEN DRAIN TTL

      spp  #P4C_PG                  ; Port 4 control register page

      ld   P4C0R,#11000000b
      ld   P4C1R,#11000000b
      ld   P4C2R,#11000000b

;.............. end of init. P4

;.............. INITIALIZATION OF SCI I/O
;     P70: Input = Sin.
;     P71: AF = Sout.
;     P72: AF = Txclck.
;     P73: AF = Rxclck.

      spp  #P7C_PG                  ; Port 7 control page.

      ld   P7C0R,#00001111b         ; bit 0 (Sin): IN, TRI, TTL.
      ld   P7C1R,#11111110b         ; bit 1,2,3 (Sout, Txck, Rxck): AF,PP,TTL.
      ld   P7C2R,#00000001b         ; Others : OUT,PP,TTL.
      }
```

APPENDIX B. EXAMPLES OF ST9 PERIPHERAL CONFIGURATIONS (Continued)

```
;*****************************************
;*SECTION CODE FOR THE CORE INTERRUPT ROUTINE*
;*****************************************

;----------------------
;*INTERRUPT ROUTINE FOR ZERO DIVISION*
;----------------------
DIV0:
      nop
      ret

;----------------------
;*INTERRUPT ROUTINE FOR TOP_LEVEL_IT*
;----------------------
TOP_LEVEL_IT:
      nop
      iret

;------------------------
;*INTERRUPT ROUTINE FOR TIMER WATCHDOG INT*
;------------------------
TEMPO:
      nop
      iret
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS

```
;***********************
;*DEFINE TIMER 0 MACROS*
;***********************
.macroT0_START_IT              ; start timer 0, enable interrupts
          spp     #T0D_PG              ; select Timer 0 data register page
          and     T_TCR,#ccl           ; counter clear bit
          or      T_TCR,#cen           ; counter enable bit
          or      T_IDMR,#gtien        ; global interrupt mask

.endm
.macroT0_START_DMA_CAP         ; start timer 0, enable interrupts
                                       ; and DMA
          spp     #T0D_PG              ; select Timer 0 data register page
          or      T_IDMR,#( gtien | cp0d ); global interrupt mask
          or      T_TCR,#cen           ; counter enable bit

.endm

.macroSTOP_T0                          ; stop Timer 0
          spp     #T0D_PG              ; select Timer 0 data register page
          and     T_IDMR,#gtien        ; global interrupt mask
          and     T_TCR,#cen           ; counter enable bit

.endm
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS (Continued)

```
;*********************************************************
proc  GEST_T0_ITCAPT{
      ;Configuration of Timer 0 for IT CAPTURE
      ;TCR:              - stop count
      ;                  - clear on capture
      ;                  - up count
      ;TMR:              - disable output
      ;                  - internal clock
      ;                  - disable bivalue mode
      ;                  - disable retrigger mode
      ;                  - disable REG1 mode
      ;                  - continuous mode
      ;                  - enable REG0 mode
      ;ICR:              - EXTA Trigger
      ;                  - falling edge on EXTA
      ;                  - EXTB No Operation
      ;OACR-OBCR:        - no operation
      ;IDMR:             - Interrupt on capture REG0
      ;DCPR:             - reset value
      ;DAPR:             - 00h
      ;IVR:              - Interrupt vector 10h = T0_IT_VECT
      ;IDCR:             - level 4
      spp  #T0D_PG                    ; Timer 0 data register page
      ld   T_TCR,#01001000b           ; TCR
      ld   T_TMR,#00001010b           ; TMR
      ld   T_ICR,#01010100b           ; ICR
      ld   T_PRSR,prescal_t0          ; PRESCALER
      ld   T_OACR,#11111100b          ; OACR
      ld   T_OBCR,#11111100b          ; OBCR
      ld   T_FLAGR,#00h               ; FLAGR
      ld   T_IDMR,#00100000b          ; IDMR
      spp  #T0C_PG                    ; Timer 0 control register page
      ld   T0_DCPR,#00h               ; DCPR
      ld   T0_DAPR,#0                 ; DAPR
      ld   T0_IVR,#T0_IT_VECT         ; IVR interrupt vector 14h
      ld   T0_IDCR,#IT_T0_LEVEL       ; priority level 4

      T0_START_IT                     ; start Timer 0, enable interrupt
      }
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS (Continued)

```
;*********************************************************************
proc  GEST_T0_EVENT{
      ; Configuration of Timer 0 into EVENT COUNTER MODE
      ; IT COMPARE is serviced when nb_event_t0 is reached

      ;TCR:              - Stop count
      ;                  - Up count
      ;                  - Clear on compare
      ;TMR:              - Disable output 0-1
      ;                  - no Bivalue mode
      ;                  - no Bicapture
      ;                  - Internal clock
      ;                  - Disable retrigger mode
      ;                  - Continuous mode
      ;ICR:              - EXTB Ext.Clock
      ;                  - Falling edge on EXTB
      ;                  - EXTA I/O
      ;OACR-OBCR:        - No operation
      ;FLAG:             - reset value
      ;IDMR:             - IT compare 0
      ;DCPR:             - 00h
      ;DAPR:             - 00h
      ;IVR:              - interrupt vector 10h = T0_IT_VECT
      ;IDCR:             - priority level 4
      ;COMP0

      spp  #T0D_PG                  ; Timer 0 data register page

      ldw  T_CMP0R,nb_event_t0      ; COMP0

      ld   T_TCR,#00111000b         ; TCR
      ld   T_TMR,#00000010b         ; TMR
      ld   T_ICR,#01000010b         ; ICR
      ld   T_PRSR,prescal_t0        ; PRESCALER
      ld   T_OACR,#11111100b        ; OACR
      ld   T_OBCR,#11111100b        ; OBCR
      ld   T_IDMR,#00000100b        ; IDMR
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS (Continued)

```
spp   #T0C_PG                        ; Timer 0 control register page
      ld   T0_DCPR,#0                ; DCPR
      ld   T0_DAPR,#0                ; DAPR
      ld   T0_IVR,#T0_IT_VECT        ; IVR
      ld   T0_IDCR,#IT_T0_LEVEL      ; IDCR
      T0_START_IT

      }

;*********************************************************************
      proc GEST_T0_DMA{

;Configuration of Timer0 in IT CAPTURE associated to the DMA mode
;the length of DMA is given by lg_dma

      ;TCR:              - Stop count
      ;                  - no clear
      ;                  - Up count
      ;TMR:              - disable interrupt
      ;                  - no bivalue mode
      ;                  - no capture
      ;                  - external/internal clock
      ;                  - disable retrigger mode
      ;                  - continuous count
      ;ICR:              - EXTA TRIGGER
      ;                  - Falling edge on EXTA
      ;                  - EXTA no operation
      ;OACR-OBCR:        - no operation
      ;IDMR:             - no interrupt, DMA / CAPTURE REG0
      ;DCPR:             - DMA ext. data/program memory- DMA counter
      ;DAPR:             - DMA external program memory - DMA address
      ;IVR:              - interrupt vector 10h = T0_IT_VECT
      ;IDCR:             - interrupt dma priority level 4

      spp  #T0D_PG                   ; select Timer 0 data register
      ld   T_TCR,#01001000b          ; TCR
      ld   T_TMR,#00001010b          ; TMR
      ld   T_ICR,#01010100b          ; ICR
      ld   T_PRSR,prescal_t0         ; PRESCALER
      ld   T_OACR,#11111100b         ; OACR
      ld   T_OBCR,#11111100b         ; OBCR
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS (Continued)

```
      ld   T_FLAGR,#00h              ; FLAGR
      ld   T_IDMR,#00100000b         ; IDMR
      spp  #T0C_PG                   ; select Timer 0 control register
      ld   T0_DCPR,#CPT_LG_DMA       ; DCPR lg. DMA  = 4ch = rr12
                                     ; = RR76
      ld   T0_DAPR,#CPT_AD_DMA       ; DAPR ad. DMA  = 48h = rr8
                                     ; = RR72
      ld   T0_IVR,#T0_IT_VECT        ; IVR
      ld   T0_IDCR,#IT_T0_LEVEL      ; priority level 4

      ldw  CPT_LG_DMA,lg_dma         ; init DMA counter
      ldw  CPT_AD_DMA,#0ff00h        ; DMA address in ROM is 0FF00h

      T0_START_DMA_CAP               ; enable Interrupt. and DMA

      }
;*********************************************************************
;     Example for Timer 0 and Timer 1 in parallel mode
;     A Toggle is generated on T0OUTB and T1OUTB on each overflow
;*********************************************************************
                ;******************
                ;initialize TIMER 0
                ;******************

TIMER0::
      spp  #T0D_PG                   ; select timer 0 register page
      srp  #BK_F                     ; select working register

      ld   t_tcr,#00011000b          ; Counter clear
                                     ; Software Up
      ld   t_tmr,#10001000b          ; Enable output 1
                                     ; Disable output 0
                                     ; Not bivalue mode
                                     ; REG 1 monitor counter value
                                     ; REG 0 Capture
                                     ; Internal clock
                                     ; Retrigger mode
                                     ; Continuous mode
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS (Continued)

```
ld    t_icr,#00                    ; No action on input pins
      ld   t_prsr,#00              ; No prescaling
      ld   t_oacr,#11111100b       ; No action on OUTPUT0
      ld   t_obcr,#11110100b       ; Toggle on OVF
      ld   t_flagr,#00
      ld   t_idmr,#00

.macroT0_START                ; Start TIMER 0
      spp  #T0D_PG                 ; select Timer 0 data register page
      or   t_tcr,#cen              ; counter enable bit
.endm

                ;******************
                ;initialize TIMER 1
                ;******************

TIMER1::
      spp  #T1D_PG                 ; select timer 1 register page
      srp  #BK_F                   ; select working register

      ld   t_tcr,#00011000b        ; Counter clear
                                   ; Software Up

      ld   t_tmr,#10001100b        ; Enable output 1
                                   ; Disable output 0
                                   ; Not bivalue mode
                                   ; REG 1 monitor counter value
                                   ; REG 0 Capture
                                   ; Parallel mode
                                   ; Retrigger mode
                                   ; Continuous mode

      ld   t_icr,#00               ; No action on input pins
      ld   t_prsr,#00              ; No prescaling
      ld   t_oacr,#11111100b       ; No action on T1OUTA
      ld   t_obcr,#11110100b       ; Toggle on OVF T1OUTB
      ld   t_flagr,#00
      ld   t_idmr,#00
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS (Continued)

```
.macroT1_START                  ; Start TIMER 1
          spp     #T1D_PG             ; select Timer 1 data register page
          and     t_tcr,#ccl          ; counter clear bit
          or      t_tcr,#cen          ; counter enable bit
.endm

          or      CICR,#10000000b     ; Global counter enable

          loop    {
                  }

;*********************************************************************
;     INTERRUPT SUBROUTINES FOR TIMER 0
;*********************************************************************
;These subroutines are serviced on TIMER 0 Interrupts. They come from:

; T0_IT_VECT + 4 for both      - IT/CAPTURE
; and - DMA IT/CAPTURE end of block

; T0_IT_VECT + 6 for     - IT/COMPARE

;*********************************************************************
; Timer 0 CAPTURE Interrupt subroutine:
; - IT Capture on event on EXTA
; - DMA IT/CAPTURE end of block

T0_CAP:
      spp  #T0D_PG                  ; Timer 0 data register page

      tm   T_FLAGR,#ccp0            ; mask successful capture
      jxz  RESET_START              ; this is not an IT CAPTURE
                                    ; == Pb

      tm   T_FLAGR,#ocp0            ; overrun on Capture 0 ?
      jxnz RESET_START              ; yes == RESET

      and  T_FLAGR,#~cp0            ; reset successful capture flags
      and  T_FLAGR,#~ocp0           ; reset overrun on capture 0 flag
      iret                          ; return from interrupt
```

APPENDIX C. EXAMPLES OF TIMER 0 CONFIGURATIONS (Continued)

```
;************************************************************************
;Timer 0 COMPARE interrupt subroutine:
;          - IT / COMPARE

T0_COMP:

      spp  #T0D_PG                  ; Timer 0 data register page

      tm   T_FLAGR,#cm0             ; mask successful compare
      jxz  RESET_START              ; RESET if it is not
                                    ; an IT COMPARE

      tm   T_FLAGR,#ocm0            ; overrun on Compare 0 ?
      jxnz RESET_START              ; yes == RESET

      and  T_FLAGR,#~cm0            ; reset successful compare bit
      and  T_FLAGR,#~ocm0           ; reset overrun compare 0 bit

      iret                          ; return from interrupt

;******** END OF TIMER 0 CONFIGURATION EXAMPLES ************
```

APPENDIX D. EXAMPLES OF A/D CONVERTOR CONFIGURATIONS

```
;*********************************************************************
      proc SG_CONV{

; A/D Converter is configured as follows:
;     - one shot conversion
;     - power up mode
;     - IT upon End of Conversion
;     - Start mode
;     - Autoscan from channel number AD_CONV
;     - No INT upon Analog Compare

      spp  #AD0_PG                  ; A/D converter register page

      ld   AD_CLR,#00000100b        ; Control logic register
                                    ; power up
                                    ; Stop
                                    ; Single mode
                                    ; Channel 0

      ld   AD_CRR,#00h              ; Compare result register
      ld   AD_ICR,#00100000b        ; Interrupt control register
                                    ; mask analog watchdog
                                    ; enable end of conversion
      or    AD_ICR,#IT_CAD_LEVEL    ; Priority level = 6
      ld    AD_IVR,#ADC_IT_VECT     ; Interrupt vector register

      ld   r0,ad_conv               ; AD_CONV = channel number
      swap r0
      rcf
      rlc  r0                       ; mask for channel number
      or   AD_CLR,r0                ; start conversion address

      ld   R10, #40
      loop [R10]   {                ; wait 60µs before start the first
                                    ; conversion
           nop
      }
      or   AD_CLR,#st               ; start conversion

      }
```

APPENDIX D. EXAMPLES OF A/D CONVERTOR CONFIGURATIONS (Continued)

```
;************************************************************
;      A/D END OF CONVERSION INTERRUPT SUBROUTINE

ADC_EOC:
      spp  #AD0_PG                    ; A/D converter register page
                                      ; converter flags

      and  AD_ICR,#~(ecv | awd)       ; end of conversion pending flag
                                      ; analog watch_dog pending flag
      and  AD_CLR,#~(st | pow )       ; stop converter
                                      ; power down mode
      iret
```

APPENDIX E. EXAMPLES OF SCI CONFIGURATIONS

```
;************************
; SCI
;constant declarations.
;************************
PRIORITY_SCI        =       4       ; SCI priority level
DIV_9600            =       78      ; BRG divisor for a 9600 baud clock
                                    ; with a 12 MHz system clock.
DIV_4800            =       156     ; To generate a 4800 bds clock.
DIV_2400            =       312     ; To generate a 2400 bds clock.
DIV_1200            =       614     ; To generate a 1200 bds clock.

VC_9600             :=      4       ; Character for 9600 bauds.
Return      =       00dh
LNG_DMA_SCI         :=      0Fh     ; DMA length.
DEPART_DMA_SCI      :=      0A0h    ; Start DMA address .
                                    ; BK_DMA_SCI reserved for this.

NUM_TDAP            :=      6       ; Contains DMA transmit address pointer value.
NUM_TDCP            :=      7       ; Contains DMA transmit address counter value.

data                :=      r2      ; data hold register
rec_ptr             :=      rr6
rec_cpt             :=      rr8

;*********************************************************************
; function:
;       - I/O ports initialization.
;       - Speed and frame initialization.
;       - Compare register initialization.
;       - Interrupt and DMA configuration.
;
; Interrupt request:
;       - Receive error.
;       - Receiver data.
;       - end of DMA transmit.
;
; inputs: none
;
; outputs:none
;
```

APPENDIX E. EXAMPLES OF SCI CONFIGURATIONS (Continued)

```
;*********************************************************************
proc  INIT_SCI    {

;--  Communication format configuration.
;
; Communication format is configured as follows:
;
;     - 8 data bit transmitted or received character.
;     - 1 stop bit included in data format.
;     - Parity even.
;     - 9600 Baud communication rate.

;-- SCI configuration.
;
;     - No address bit included between the parity bit and the stop bit.
;     - Address mode: Address interrupt if character match.
;     - DMA permits transmission from EEPROM memory to serial line.
;     - Receiver data interrupt unmask (to detect a received data item).
;     - Transmitter data interrupt unmask (to detect DMA end of block).
;     - Receiver error interrupt unmask (to detect overrun, parity or framing error).

      spp  #SCI1_PG           ; SCI register page.
      srp  #BK_F   ; To address SCI registers with r.
      ld   s_brglr,#00        ; Reset SCI

      ld   s_chcr,#( wl8 | sbl0 | pen | ep | am )
                              ; 8 data bit.
                              ; 1 stop bit.
                              ; Parity even.
                              ; No address bit.
                              ; AME = 0, AM = 1.
                              ; = IT if character match.

      ld   s_ccr,#txclk       ; Xmit clock source = BRG.
                              ; Receiver clock source = BRG.
                              ; 16x asynchronous mode.

      ld   s_acr,#RETURN      ; End Of Command acquisition.
```

APPENDIX E. EXAMPLES OF SCI CONFIGURATIONS (Continued)

```
      ;-- Interrupt and DMA configuration.

      ld   s_ivr,#SCI_IT; Interrupt vector  register.
      ld   s_tdcpr,#NUM_TDCP          ; Tx DMA counter in register file.

      ld   s_imr,#( rxdi | rxa | rxe )
                                      ; Mask Transmitter data interrupt.
                                      ; Unmask Receiver data interrupt.
                                      ; Unmask Receiver data error interrupt.
                                      ; Unmask Receiver address interrupt.
                                      ; Reset of the pending bits.

      ld   s_idpr,#PRIORITY_SCI       ; Mask transmitter DMA request.
                                      ; SCI exeptions priority level.
      ld   s_brglr,#DIV_9600          ; BRG divisor for 9600 bauds, start SCI
                                      ; !!! with a 24 Mhz external clock,
                                      ; !!! or 4800 Bds (12 MHz external clock.)

 } ;-- end of proc.

;*********************************************************************
;     SYNC_COM:

proc  SYNC_COM {
      spp  #SCI1_PG
      srp  #BK_F
      ld R#NUM_TDAP,#(DEPART_DMA_SCI)      ; DMA pointer initialisation.
      ld R#NUM_TDCP,#(LNG_DMA_SCI)         ; DMA counter initialisation.
      or s_idpr,#txd
                                           ; Unmask transmitter DMA request.
                                           ; unmask transmitter data interrupt.

      ld s_imr,#txdi
                                           ; Unmask Transmitter data interrupt.
                                           ; Mask Receiver data interrupt.
                                           ; Mask Receiver data  error interrupt.
} ;-- End of proc.
```

APPENDIX E. EXAMPLES OF SCI CONFIGURATIONS (Continued)

```
;*********************************************************************
;      REC_DATA:   Receive interrupt.
REC_DATA:
      pushu       PPR                ; save page pointer.
      pushuw      RPP                ; save register pointer pair.

      spp  #SCI1_PG                  ; SCI register page.
      srp  #BK_SCI                   ; 16 registers reserved for SCI.

      ld   data,S_RXBR               ; Read the data received.
      and  data,#07Fh                ; Mask the parity bit.

      ld   rec_ptr(rec_cpt),data     ; Storage of the received data.
      incw rec_cpt
      cpw  rec_cpt,#7                ; End of the table.

      and  S_ISR,#~rxdp              ; Reset receiver data pending flag.

      popuw       RPP                ; restore register pointer pair
      popu        PPR                ; restore page pointer
      iret
```

APPENDIX E. EXAMPLES OF SCI CONFIGURATIONS (Continued)

```
;************************************************************************
; TRA_HOLD:       End of DMA transmitter Interrupt
; Function:
;     - Check Interrupt source.
;     - Disable DMA mask .
;     - Enable Receiver interrupt mask.

TRA_HOLD:

     pushu       PPR              ; save page pointer.
     pushuw      RPP              ; save register  pointer pair.

     spp  #SCI1_PG                ; SCI register page.
     srp  #BK_F                   ; To address SCI registers with r.
     tm   s_imr,#txeob
     if   [SETZ] {                ; If a Transmitter End Of Block interrupt.
          bres    S_txeob         ; Dis. Transmit end of block pending bit.
          bres    S_txhem         ; Reset transmit holding reg. empty .
          ld      s_imr,#~( rxdi | rxe)
                                  ; Unmask Receiver data interrupt.
                                  ; Unmask Receiver data error interrupt.
                                  ; Mask Transmitter data interrupt.
     } else {
          jx     RESET_START      ; If not a normal interrupt source.
     } ;-- end of if.

     popuw       RPP              ; restore register pointer pair
     popu        PPR              ; restore page pointer
     iret
```

APPENDIX F. EXAMPLES OF WATCHDOG TIMER CONFIGURATIONS

```
;*********************************************************************
;INIT_WDT: This procedure initializes and starts Watchdog Timer.
;
; Watchdog mode is disabled.
; Timer will down count in continuous mode.
; It will generate an interrupt on channel A0 at each End Of Count.
; -- See the external interrupt parameters initialization.
;*********************************************************************
proc  INIT_WDT     {

      spp  #WDT_PG            ; To access in paged registers with r.
      ld   wcr,#wden          ; watch dog mode dis., no wait states.
      clr  wdtpr              ; 333 ns(sys.clock=12 MHz) min. count,
                              ; prescaler = 0.
      ldw  WDTR,#3003         ; (3003 X 333) ns = 1 ms.
      or   wdtcr,#stsp        ; Timer starts down counting.
                              ; Continuous mode.
                              ; Watch Dog disabled.
                              ; Input section disabled.
                              ; Output disabled.
                              ; Interrupt A0 on Timer EOC.
                              ; Top Level Interrupt on SW TRAP.
};-- End of proc.

;*************************************
;*Interrupt on channel A0 initialization*
;*************************************
      spp  #WDT_PG
      srp  #BK_F              ; page 0 reg. direct addressing mode.

      clr  eipr               ; Dis. all external int. pending bits.
      nop                     ; See WARNING (Tech. manual-Chap. 8).
      ld   eivr,#EXT_IT_VECT  ; External interrupt vector.
                              ; IAOS - TLIS = 00 = ...
                              ; ... A0 int. will be on WDT End Of Count.
      ld   eiplr,#0FEh
                              ; Priority level: group INTA0,INTA1 = 4,5.
      ld   eimr,#ia0sm        ; Unmask Interrupt A0 channel
                              ; (WDT End Of Count).
```

USING THE I²C-bus PROTOCOL WITH THE ST9

Myriam Chabaud and Alan Dunworth

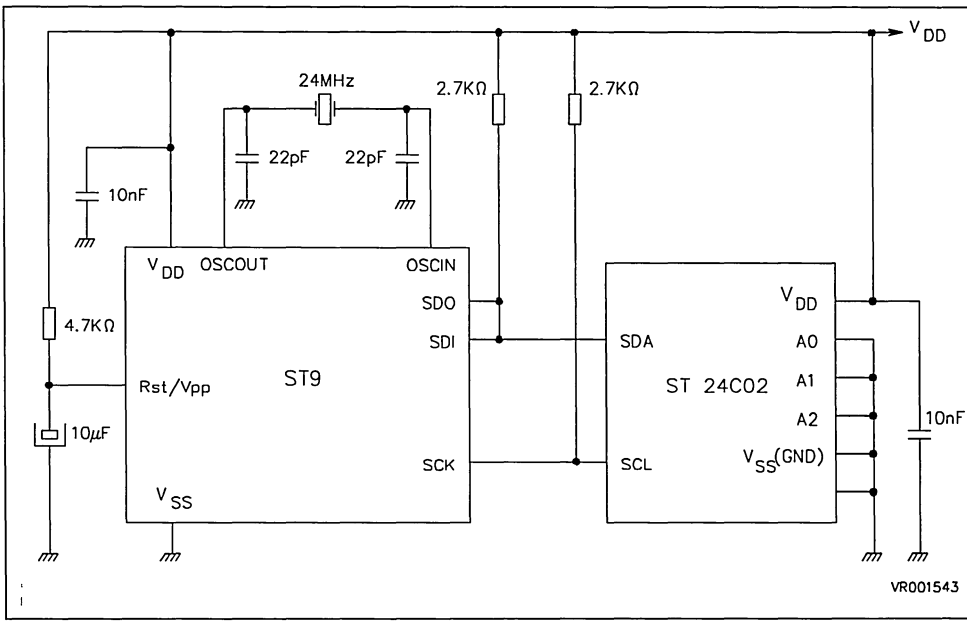
INTRODUCTION

The Serial Peripheral Interface (SPI) in the ST9 has been designed to handle a wide variety of serial bus protocols, including SBUS, IMBUS, and I²C-bus. Certain standard I²C-bus features have not been directly implemented in hardware, but may be realized with simple software routines, based on the SPI, contained in the standard ST9 core. This Application note gives an example of such routines, suitable for interfacing the ST9 with a serial memory device.

CHARACTERISTICS OF THE I²C-bus

The **I²C-bus** comprises two bidirectional lines, one for data signals (SDA) and one for clock signals (SCK). Both the SDA and the SCK lines must be connected to the positive supply via pull-up resistors (Figure 1).

Figure 1. Connection of ST24C02 and ST9 in I²C-bus



Note: Although the ST24C02 2K bit EPROM is shown, this circuit will work with serial EEPROMS up to 16K bit capacity (ST24C16) and all others in the ST24Cxx and ST25Cxx families.

The following basic definitions are applied:

* MASTER:

The device which initiates the transfer, generates the clock signals, and terminates the transfer is referred to as the Master. In our present application the ST9 always acts as the Master.

* SLAVE:

This is the device addressed by the Master (always the serial memory).

* TRANSMITTER:

This is the device which sends data to the bus. In our application the ST9 acts as Transmitter when it is writing data in the serial memory. Conversely, the serial memory serves as Transmitter when the ST9 is reading data from memory.

* RECEIVER:

This is the device which receives data from the bus. In our application this will be the ST9 when reading data, or the serial memory when the ST9 commands a write operation.

The following protocol has been defined:

* DATA TRANSFER

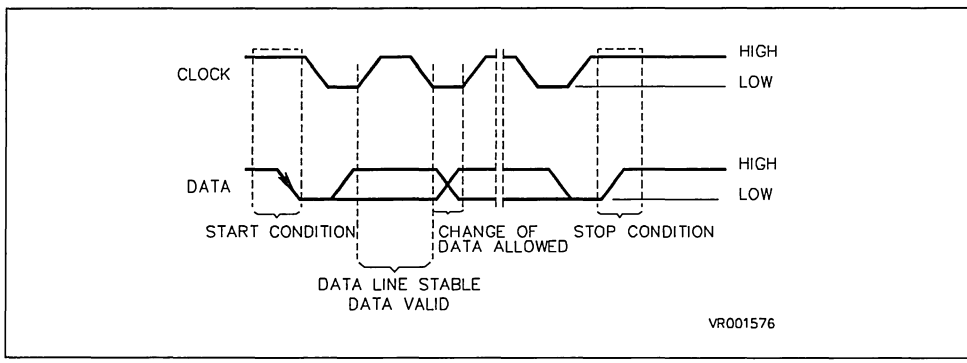
A data transfer may be initiated only when the bus is not busy.

* DATA LINE STABLE:

During data transfer, the data line must remain stable whenever the clock line is HIGH. Changes in the data line while the clock is HIGH will be interpreted as control signals.

Accordingly, the following bus conditions have been defined:

Figure 2. Data Transfer Sequence of the Serial Bus



* START DATA TRANSFER:

A change in the state of the data line from HIGH to LOW, while the clock is HIGH, defines the START condition.

* STOP DATA TRANSFER:

A change in the state of the data line from LOW to HIGH, while the clock is HIGH, defines the STOP condition.

* DATA VALID:

The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on SDA may be changed during the LOW period of the clock signal. There is one clock pulse for each bit of data.

* DATA TRANSFER:

Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes, transferred between the START and STOP conditions, is limited to eight bytes in the ST24C02 Memory device ERASE + WRITE mode, and is not limited in the READ mode.

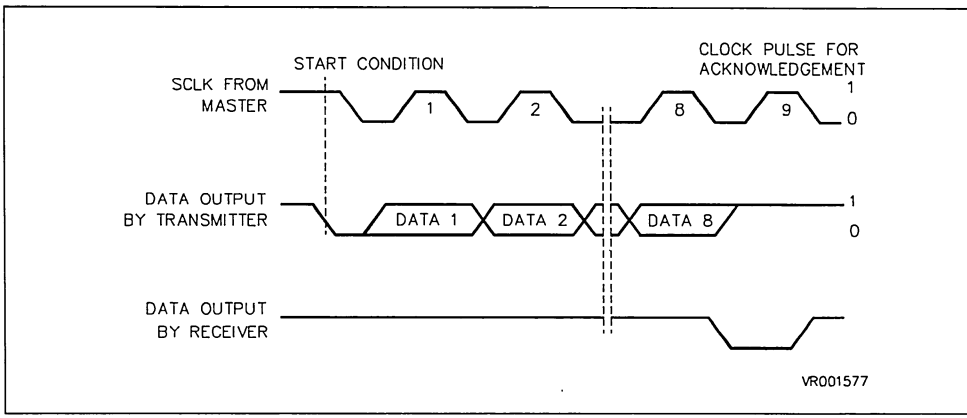
* ACKNOWLEDGE:

Each byte of eight bits is followed by an acknowledge bit. This acknowledge bit is a low level put on the SDA line by the Receiver. At the same time the Transmitter releases the SDA line to the High impedance state, and the MASTER device generates an additional 9th acknowledge-related clock pulse.

The receiving device acknowledges the receipt of the 8-bit byte by pulling the SDA down so that is stable LOW during the 9th clock pulse. Of course, set-up and hold-times must be respected.

The ST9 when acting as a Master Receive device, i.e. during serial memory READ operations, must signal an end of data by not generating an acknowledge on the last byte that has been clocked out of the slave. In this case the serial memory must leave the SDA line high to enable the Master to generate a STOP condition.

Figure 3. Acknowledgement and the 9th Clock Pulse



BASIC SOFTWARE OPERATIONS

The following aspects of the I²C-bus protocol have not been directly implemented but must be simulated in software.

* Generation of START Conditions,
* Generation of STOP Conditions,
* Generation of the Acknowledge pulse (9th clock signal),
* Generation of the Acknowledge , when the ST9 acts as a Receiver, i.e. in READ mode.
* Test of the Acknowledge from the receiver, when the ST9 acts as a Transmitter, i.e. in WRITE mode.

In order to implement these features it is necessary to drive SDA and/or SCK HIGH or LOW in the correct timing sequence.

The SDO and SCK signals are defined as Alternate Functions. These pins are configured with Open-Drain outputs and TTL inputs. The SDI signal is defined as an INPUT.

The SPI unit is enabled or disabled using the flag SPEN, bit 7 in SPICR, the SPI Control Register.

When the SPI is disabled, both SCK and SDO are released to the High impedance state. The presence of Pull-up resistors, as shown in Figure 1, effectively defines both SCK and SDA as HIGH, whenever the SPI is disabled. Note however that SDA may be driven low either by the actions of peripherals connected to the SDA line, or by appropriate action of the ST9 on the SDO line when it is defined as a normal output.

When the SPI is enabled (SPEN = "1"), it may be in either an active or passive state. The active state is entered by loading a byte of data into the SPI Data register. This automatically causes the SPI to generate a sequence of 8 clock pulses, during which data is shifted out on the SDO line, and input Data on the SDI input is clocked into the Serial input register. On completing this sequence the SPI will revert to its passive (Rest) mode.

When the SPI is in its Rest mode, the SCK clock output is in a state selected by CPOL, bit 3 of SPICR. Thus with CPOL set to a value of "1" the SCK output will be LOW. The value of SDO will be LOW (non-programmable) when the SPI is enabled but inactive.

If the SPI is enabled and in the Rest (passive) state SDO and hence SDA will be LOW.

If the SPI is disabled SDO will be released to HIGH impedance, and hence to the HIGH level by the presence of the Pull-up resistor. It may be pulled LOW by loading a Zero into the Port 2 Pin 1 output buffer and then specifying this pin as a normal Port output pin.

Having established these basic preliminaries we can proceed to discuss the provision, by software, of basic **I²C-bus** operations.

SIMULATION OF BASIC I²C-bus OPERATIONS

Using the basic operations described in the above Sections the various **I²C-bus** Protocol features may be implemented as follows.

Generation of START Conditions

The generation of a START condition is implemented in Procedure `INIT_START_I2C` (Appendix A).

a) Disable the SPI unit putting SDA and SCK in the High-impedance state.

b With the SPI disabled and SCK HIGH, pull the SDO line LOW by respecifying SDI as a normal output.

c) Hold the above condition for a period of ~5 µs by calling the `DELAI` Macro (see Appendix A).

d) Enable the SPI, specifying SCK to the rest clock state (LOW).

e) Respecify the SDO output as an Alternate Function.

Generation of STOP Conditions

The generation of a STOP condition is implemented in Procedure `GEN_STOP` (see Appendix A).

a) Pull the SDA line LOW by respecifying SDO as a normal Port output.

b) Release SCK to HIGH by disabling the SPI. Note that SDA will remain LOW.

c) Hold this condition for ~5 µs using `DELAI` Macro (see Appendix A) so as to meet the set-up Time specification

d) Respecify SDO as an Alternate Function and hence allow SDA to be pulled HIGH by the Pull-up resistor.

Generation of 9th Clock Pulse with Acknowledge Test

After the transmission of 8 Data bits a 9th Clock Pulse may be generated and the Acknowledge tested as implemented in Procedure TEST_ACK (see Appendix A).

a) Release SCK and SDA to the HIGH impedance state by disabling the SPI.

b) Wait until the SCK line goes HIGH.

c) Test for LOW on the SDA line placed by the Receiver (Slave).

d) Hold the SCK line HIGH for 5 µs using `DELAI` Macro.

e) Force SCK and SDA to LOW by enabling the SPI.

Generation of 9th Clock and Acknowledge

After the reception of 8 Data bits a 9th Clock Pulse may be generated and an Acknowledge asserted as implemented in Procedure GEN_ACK (see Appendix A).

a) Pull the SDA line LOW by respecifying SDO as a normal Port output.

b) Release SCK to HIGH by disabling the SPI. Note that SDA will remain LOW.

c) Hold the SCK line HIGH for 5 µs using `DELAI` Macro.

d) Force SCK to LOW by enabling the SPI.

e) Finally respecify the SDO Port pin as an Alternate Function.

TYPES OF TRANSFER OPERATION SUPPORTED

The ST9 supports the following three types of transfer with an electrically erasable serial memory (EEPROM) which features an **I²C-bus** protocol, e.g. ST24C02.

* Random Write (1 to 8 bytes),
* Random Read (1 to N bytes),
* Current Address Read (or Verify), (1 to N bytes.)

Random Write Mode

The serial **I²C-bus** protocol for Random Write Operations is shown in Figure 4 (single byte) or Figure 5 (for up to 8 bytes).

Figure 4. I²C-bus Protocol for Random Write Mode (1 byte)

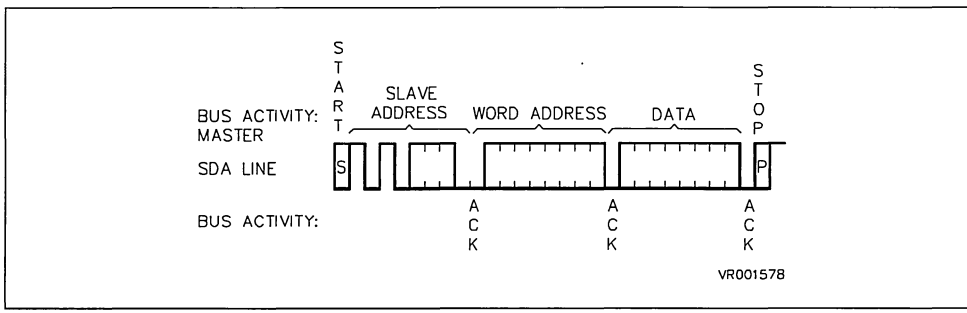
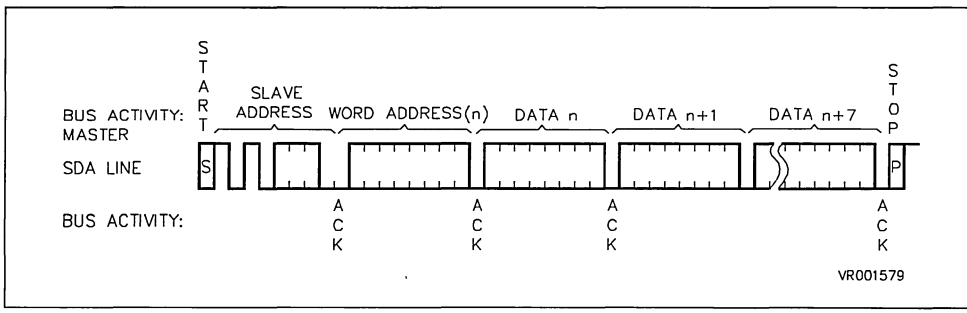


Figure 5. I²C-bus Protocol for Random Write Mode (N bytes)



To Write a single byte the Master ST9 has to transmit a sequence of 3 bytes representing successively:

a) Slave Address: 7 bits + 8th bit = "0" signifying Transmit operation.

b) Word Address: 8 bits.

c) Data value: 8 bits.

The ST9 Master generates the START condition and then transmits the sequence of 3 bytes by successively loading them into the SPI Data Register. Each such Data load generates a sequence of 8 clocks and 8

Data bits, after which the ST9 generates a 9th clock pulse and tests for an Acknowledge from the Slave. After the data pulse has been received and Acknowledged by the Slave the Master terminates the transfer by generating a STOP condition.

To Write a page of N bytes (1< N> 8) the Master ST9 has to transmit the above sequence of 3 bytes followed by the remaining N - 1 data bytes. The Slave Device contains an 8-bit address pointer, the 3 low order bits of which are incremented by 1 after each Read/Write operation with the 5 high order bits remaining constant. Thus a page of up to N = 8 bytes may be written in this way.

The Transfer sequence proceeds as described above except that the Slave continues to accept data words for writing to sequential locations until such time as the Master signals end of Transmission by sending a STOP condition.

Random Read Mode

The serial **I²C-bus** protocol for Random Read Operations is shown in Figure 6.

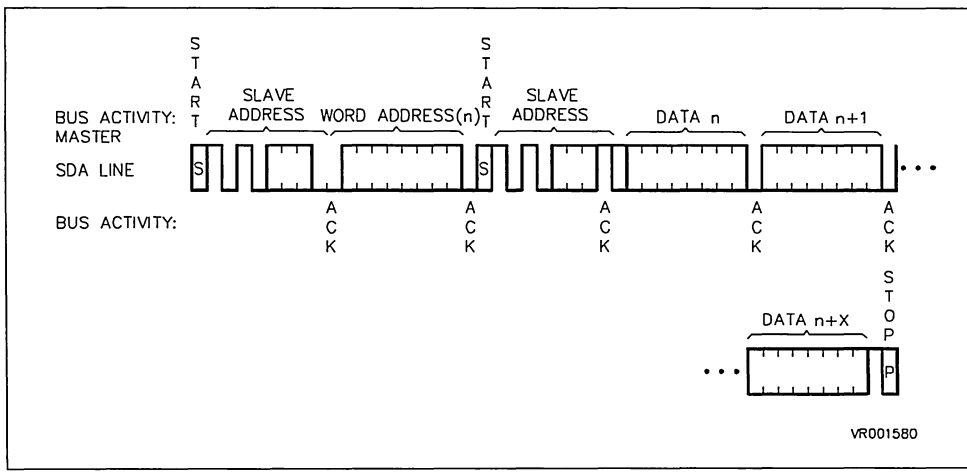
To Read a single byte the Master ST9 has to transmit a sequence of 3 bytes representing successively:

a) Slave Address: 7 bits + 8th bit = "0" signifying Transmit operation.

b) Word Address: 8 bits.

c) Slave Address: 7 bits + 8th bit = "1" signifying Receive operation.

Figure 6. I²C-bus Protocol for Random Read Mode (N bytes)



The ST9 Master generates the START condition and then transmits a dummy Write operation comprising the Slave Address byte, followed by the Word Address. Both these byte operations are followed by a 9th clock pulse and a concurrent test for Slave Acknowledge.

At this point the Master Transmitter must become the Master Receiver. This is achieved by sending another START condition, followed by the retransmission of the Slave Address with the 8th bit set now to "1" to indicate that the subsequent data transfers are from the slave to the ST9 Master.

From this point on the Slave will provide words addressed in sequence as long as the Master continues to Acknowledge receipt of data. Note that the address counter for Read operations increments over all 8 address bits, thus enabling the entire memory to be Read in one operation. The Master can terminate the transfer at any time by generating a STOP condition instead of an Acknowledgement.

Current Address Read Mode

In this alternative Read mode the Master reads from memory at the last location referenced in either Read or Write mode.

The serial **I²C-bus** protocol for Current Address Read Operations is shown in Figure 7.

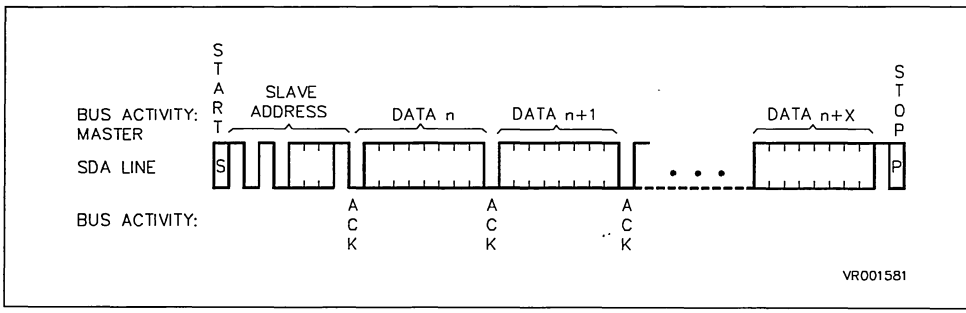
To Read any number of bytes the Master ST9 has to transmit a single byte.

Device Address: 7 bits + 8th bit = "1" signifying Receive operation.

The ST9 Master generates the START condition and then transmits the Slave Address byte. At this point the Master now issues an Acknowledge indicating that it requires additional data.

From this point on the Slave will provide words addressed in sequence as long as the Master continues to Acknowledge receipt of data. The Master can terminate the transfer at any time by issuing a STOP condition instead of an Acknowledgement.

Figure 7. I²C-bus Protocol for Current Address Read mode (N bytes)



EEP_MAN: AN I²C-bus PROTOCOL EEPROM MANAGER

Appendix A contains a detailed Assembler listing of a representative example of an EEPROM manager for a device respecting the I²C-bus serial protocol. This example is not intended to be definitive but should be taken as illustrative example of the use of the ST9 in such applications. Modifications and extensions, depending on the particular application, will readily occur to the Application Engineer, e.g. the use of the ST9 stacks as an alternative mechanism for transferring data and parameters between the Manager and the calling program. Note that Appendix A makes use of a number of Macros which are separately listed and defined in Appendix C.

The EEP_MAN/ Calling Program Interface

A calling program interfaces to EEP_MAN using four registers for calling parameters and a register-file for data.

Parameter/ Transfer-Status Registers

A call to EEP_MAN is initialized by loading parameter values into three registers, viz. EEP_FUNCT, EEP_ADD, and NB_BYTES. The status of a current transfer can be monitored by reading a fourth register, STAT_EEP, in which EEP_MAN records a value giving the status of the EEPROM device.

EEP_FUNCT Register, R3.

This register is loaded with one of the following values to specify the mode of data transfer required:

1: READ_FUNCT: Random READ mode.

2: WRITE_FUNCT: Random WRITE mode.

3: VERIFY_FUNCT: Current Address (Verify) mode.

EEP_ADD Register, R0.

This register should be loaded with the value of the EEPROM byte starting address for Random READ/WRITE operations. For a current address (Verify) operation the contents of this register is a Don't-Care value.

NB_BYTES Register, R6.

This register should be loaded with the number of bytes, #N, which should be transferred in the operation. This value may have a value from 1 to 8 for Write operations, or 1 to 256 for READ operations.

STAT_EEP Register, R4.

EEP_MAN loads this register with one of the following values to specify the current EEPROM Status.

0: EEP_OK: The EEPROM is OK.

1: LECT_ON: The EEPROM is reading a byte (random address mode).

2: VERIF_ON: The EEPROM is reading the current byte.

3: ECR_ON: The EEPROM is programming a byte (random address mode).

4: NO_ACK: The EEPROM has not Acknowledged a byte transferred from the ST9

80h: EEP_FREE_MASK: The EEPROM is available for a new operation.

Transfer of Data Values

DATA_TABLE Register File

A register-file, starting at R32 and of size #N should be reserved for READ/VERIFY operations, or loaded with data to be transferred to the EEPROM for a WRITE operation. The first byte to be transferred should be loaded into register R31+#N, and the last byte should be loaded into register R32.

EEP_MAN Data Transfer Initialization Routines

After loading the Parameter registers and setting up and, if appropriate, loading the Data table, the calling routine tests STAT_EEP to check that the EEPROM is free, and then calls Procedure EEP_MAN.

This procedure first saves the byte address counter value, NB_BYTE, specifies the Port 2 pins SDO and SCK as Alternate Functions, and SDI as an input, and then calls one of the three main initializing routines READ_EEP, VERIF_EEP, or WRITE_EEP, depending on the value transferred in register EEP_FUNCT.

These three procedures essentially carry out identical functions. After verifying that the EEPROM is not busy, they enable the SPI interrupt, generate a START condition, and transfer the EEPROM device address by loading this value into the SPI Data Register, SPIDR.

Note that the EEPROM Device Address is 7 bits long together with an eight bit which is set to "0" for READ or WRITE operations, and set to "1" for VERIFY operations. In addition, a value of "1" is loaded into the Transaction Status Register, STAT_TRANS_SPI to indicate that the Device Address has been transferred. This register is loaded with an appropriate identifying value each time the SPI Data Register is loaded.

STAT_TRANS_SPI Register, R5.

This register serves as an internal Status register, used by EEP_MAN and its associated routines, to maintain a record of the nature of the current ST9 to EEPROM transfer.

1: T_ADD_SLAVE: The EEPROM device address has been transferred.

2: T_ADD_EEP: The EEPROM byte address has been transferred.

3: TRANS_WR_DATA: A WRITE byte has been transferred.

4: TRANS_RD_DATA: A READ byte has been transferred.

After initiating a byte transfer by loading the SPI Data Register, SPIDR, a return is made to the calling routine. At the completion of the byte transfer (8 SCK clock pulses) the SPI raises an interrupt on channel B0 (associated to external interrupt INT2).

The SPI Interrupt Service Routine

This routine is called at the termination of the transmission of each byte representing a Device Address, Word Address, READ data, or Write data. The action effected by this routine (Procedure IT_END_TRANS, see Appendix A) depends upon the values contained in the following registers:

1: STAT_TRANS_SPI Register, R5.

2: STAT_EEP Register, R4.

3: NB_BYTES Register, R6.

4: EEP_FUNCT Register, R3.

The required action depends on the nature of the previously transferred byte, indicated by the value contained in STAT_TRANS_SPI. In the case of data byte transfers the next action also depends on whether the required number of bytes has been transferred, as indicated by the value of NB_BYTES.

The organization of IT_END_TRANS is illustrated by the flow diagram of Figure 8. This will be described by considering in detail the logical flow of events associated with each of the three modes of data transfer.

Random Write Mode

Figure 5 illustrates the sequence of byte transfers involved in writing N bytes in Random Write Mode, observing the I²C-bus protocol.

(i) Transmission of Slave Device Address.

This operation is initiated by Procedure WRITE_EEP which generates a START condition, loads the Device address (with the 8th bit set to 0) in SPIDR, thus initiating the transfer, and then returns to the calling program.

In addition, this routine loads the following values into the Status Registers:

```
STAT_TRANS_SPI          <-1 (#T_ADD_SLAVE)
STAT_EEP                <-3 (#ECR_ON)
```

(ii) Transmission of Word Address.

After transmission of the 8 bits of the Device Address, an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path AI (refer to Figure 8), as a result of which the required random Word address is loaded into SPIDR, so effecting the required byte transfer.

In addition, this routine loads (or retains) the following values in the Status Registers:

```
STAT_TRANS_SPI          <-2 (#T_ADD_EEP)
STAT_EEP                <-3 (#ECR_ON)
```

(iii) Transmission of 1st Data Byte.

After transmission of the 8 bits of the Word Address, an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path ACG (refer to Figures 8, 8b), as a result of which the 1st Data Byte is loaded into SPIDR, so effecting the required byte transfer.

In addition, this routine loads (or retains) the following values in the Status Registers:

```
STAT_TRANS_SPI          <-3 (#TRANS_WR_DATA).
STAT_EEP                <-3 (#ECR_ON)
```

Figure 8. Flow Diagram of the IT_END_TRANS Interrupt Routine

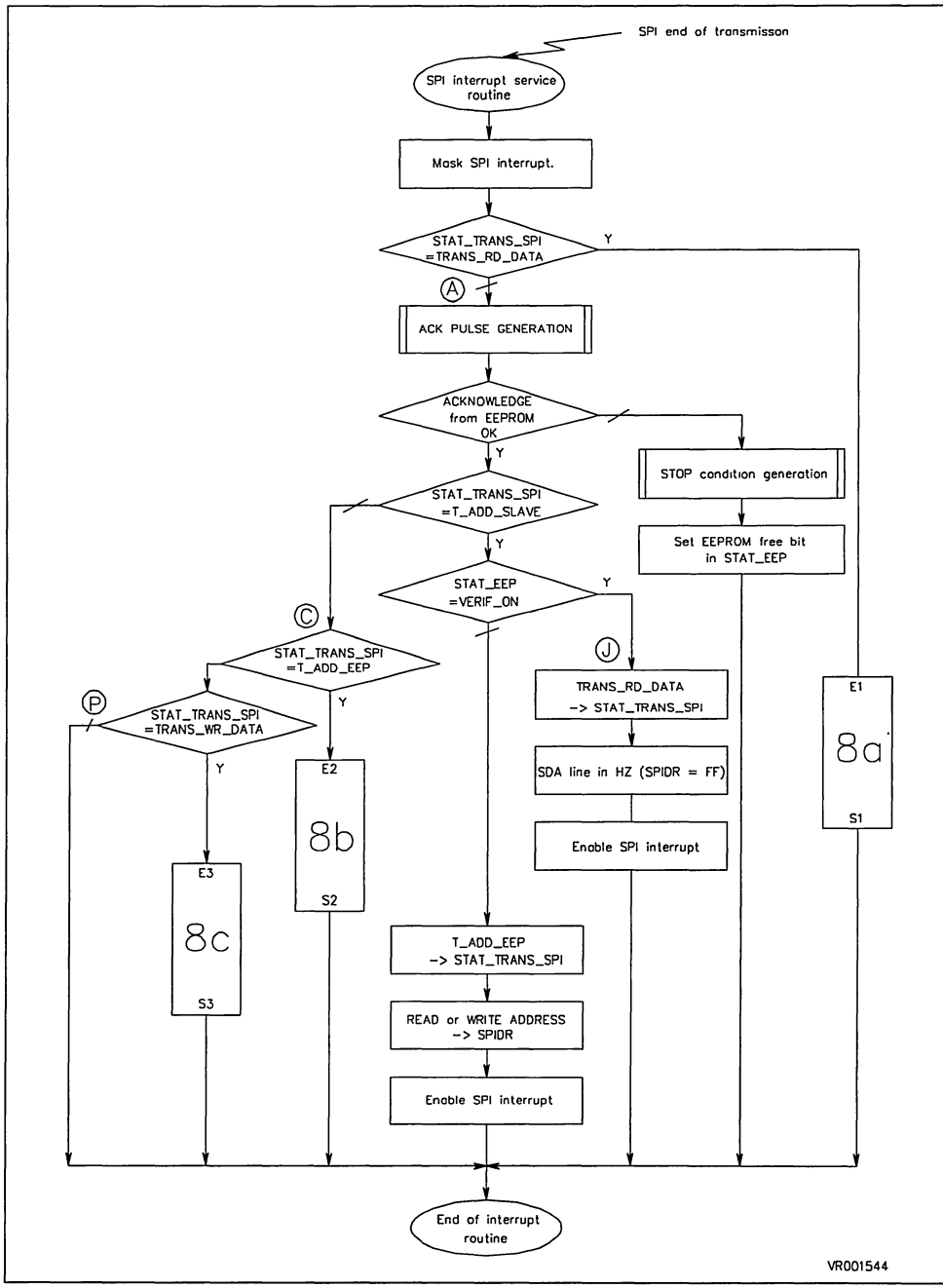


Figure 8a. Flow Diagram of the IT_END_TRANS Interrupt Routine (continued)

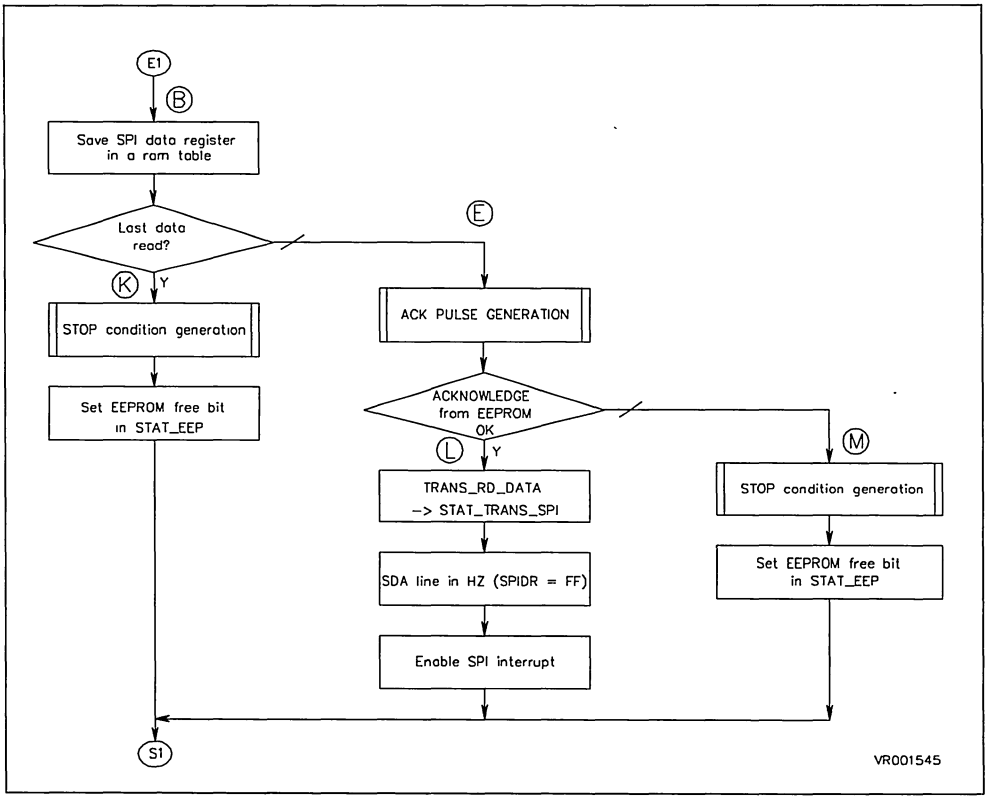


Figure 8b. Flow Diagram of the IT_END_TRANS Interrupt Routine (continued)

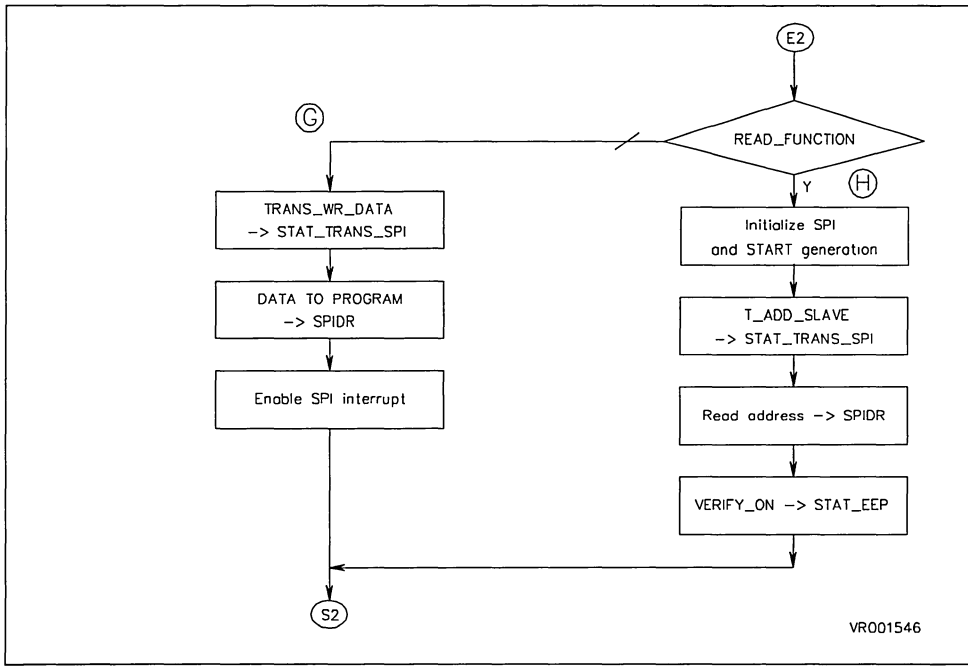
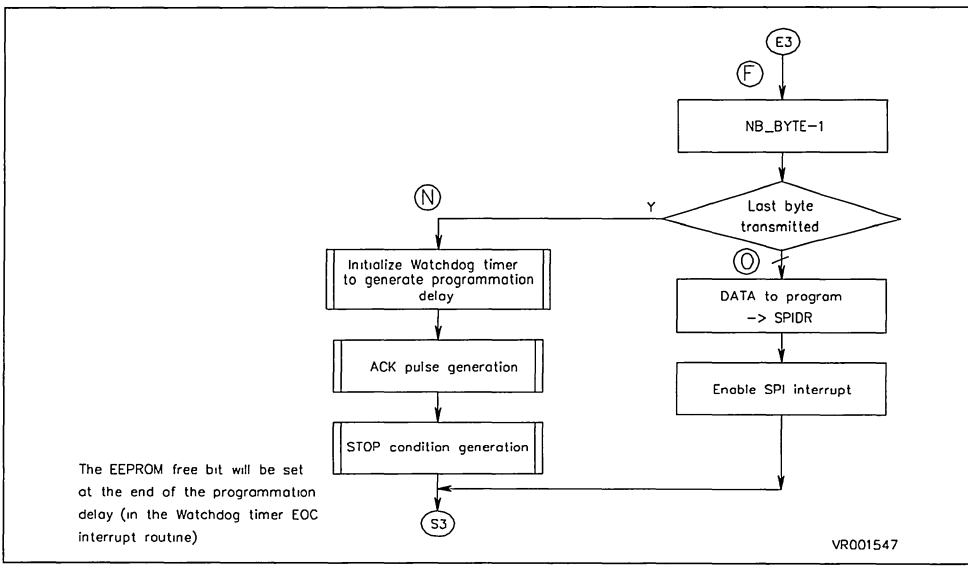


Figure 8c. Flow Diagram of the IT_END_TRANS Interrupt Routine (continued)



(iv) Transmission of Subsequent Data Bytes.

After transmission of Byte #M (1 < M < N), an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path ACFO (refer to Figures 8, 8c) as a result of which Data Byte #M + 1 is loaded into SPIDR, so effecting the required byte transfer.

The following values are retained in the Status Registers:

```
STAT_TRANS_SPI          <-3 (#TRANS_WR_DATA).
STAT_EEP                <-3 (#ECR_ON)
```

(v) Transmission of the final Data Byte.

After transmission of Byte #N, an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path ACFN (refer to Figures 8, 8c). On this occasion the Watch-Dog Timer routine, PROG_DELAY (see Appendix A) is entered to generate a delay equal to N x 5 milliseconds to enable the EEPROM to be programmed with the new data values.

For this purpose the Watch_Dog Timer is initialized in Single Operation, Count-down Mode, and a constant value is loaded into the counter appropriate to the required delay. An interrupt is enabled on Channel A0 for the Timer EOC event, and a return is made to the calling program.

When the Timer times out, entry is made to interrupt routine TEMP0 (see Appendix A). This routine clears the A0 interrupt pending bit, sets the EEP_FREE_MASK bit to 1, and returns to the calling program. At this point the EEPROM is available again for further data transfers.

Random READ Mode

Figure 6 illustrates the sequence of byte transfers involved in reading N bytes in Random Read Mode, observing the I²C-bus protocol.

(i) Transmission of Slave Device Address.

This operation is initiated by Procedure READ_EEP which generates a START condition, loads the Device address in SPIDR (with the 8th bit set to "0"), thus initiating the transfer, and then returns to the calling program.

In addition, this routine loads the following values into the Status Registers:

```
STAT_TRANS_SPI          <-1 (#T_ADD_SLAVE)
STAT_EEP                <-1 (#LECT_ON)
```

(ii) Transmission of Word Address.

After transmission of the 8 bits of the Device Address, an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path AI (refer to Figure 8), as a result of which the required random Word address is loaded into SPIDR, so effecting the required byte transfer.

In addition, this routine loads (or retains) the following values in the Status Registers:

```
STAT_TRANS_SPI          <-2 (#T_ADD_EEP)
STAT_EEP                <-1 (#LECT_ON)
```

(iii) Retransmission of Slave Device Address.

After transmission of the 8 bits of the Word Address, an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path ACH (refer to Figure 8), as a result of which the Device address (with the 8th bit set to "1"), loaded into SPIDR, so effecting the required byte transfer.

In addition, this routine loads the following values in the Status Registers:

```
STAT_TRANS_SPI          <-1 (#T_ADD_SLAVE)
STAT_EEP                <-2 (#VERIF_ON)
```

(iv) Read of 1st Data Byte.

After the retransmission of the 8 bits of the Device Address, an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path AJ (refer to Figure 8), as a result of which a value of 0FFh is loaded into SPIDR, so effecting the required byte transfer from the Slave Memory.

In addition, this routine loads (or retains) the following values in the Status Registers:

```
STAT_TRANS_SPI          <-4 (#TRANS_RD_DATA).
STAT_EEP                <-2 (#VERIF_ON)
```

(v) Read of Subsequent Data Bytes.

After transmission of Byte #M (1 < M < N), an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path BEL (refer to Figure 8), as a result of which Data Byte #M + 1 is loaded into SPIDR, so effecting the required byte transfer.

The following values are retained in the Status Registers:

```
STAT_TRANS_SPI          <-4 (#TRANS_RD_DATA).
STAT_EEP                <-2 (#VERIF_ON)
```

(vi) Read of the final Data Byte.

After transmission of Byte #N, an Interrupt is raised and entry made to Interrupt Procedure IT_END_TRANS. The logical flow then follows the path BK (refer to Figure 8), as a result of which the STOP condition is generated and the EEPROM free bit set in STAT_EEP.

Current Address READ (Verify) Mode

Figure 7 illustrates the sequence of byte transfers involved in reading N bytes in Random Write Mode, observing the **I²C-bus** protocol.

(i) Transmission of Slave Device Address.

This operation is initiated by Procedure `VERIF_EEP` which generates a START condition, loads the Device address (with the 8th bit set to "1") in SPIDR, thus initiating the transfer, and then returns to the calling program.

In addition, this routine loads the following values into the Status Registers:

```
STAT_TRANS_SPI          <-1 (#T_ADD_SLAVE)
STAT_EEP                <-2 (#VERIF_ON)
```

(ii) Read of 1st Data Byte.

After the retransmission of the 8 bits of the Device Address, an Interrupt is raised and entry made to Interrupt Procedure `IT_END_TRANS`. The logical flow then follows the path AJ (refer to Figure 8), as a result of which a value of 0FFh is loaded into SPIDR, so effecting the required byte transfer from the Slave Memory.

In addition, this routine loads (or retains) the following values in the Status Registers:

```
STAT_TRANS_SPI          <-4 (#TRANS_RD_DATA).
STAT_EEP                <-2 (#VERIF_ON)
```

(iii) Read of Subsequent Data Bytes.

After transmission of Byte #M (1 M N), an Interrupt is raised and entry made to Interrupt Procedure `IT_END_TRANS`. The logical flow then follows the path BEL (refer to Figure 8), as a result of which Data Byte #M + 1 is loaded into SPIDR, so effecting the required byte transfer.

The following values are retained in the Status Registers:

```
STAT_TRANS_SPI          <-4 (#TRANS_RD_DATA).
STAT_EEP                <-2 (#VERIF_ON)
```

(iv) Read of the final Data Byte.

After transmission of Byte #N, an Interrupt is raised and entry made to Interrupt Procedure `IT_END_TRANS`. The logical flow then follows the path BK (refer to Figure 8), as a result of which the STOP condition is generated and the EEPROM free bit set in `STAT_EEP`.

ILLUSTRATIVE CALLING ROUTINES

Appendix B contains listing of suitable calling routines to WRITE 4 bytes to the Serial EEPROM or to READ 6 bytes. Included also in Appendix B are the appropriate ST9 Core System and Peripheral initialization routines (see also Reference 1).

These programs make use of the File of ST9 Standard Register and Register Bit Definitions listed in Application Note AN411, **SYMBOLS.INC.**

It will be noted that the calling routines, after initiating the data transfers, wait in test and branch loops until the EEPROM is free. In a practical real-time application this waiting time (>N.5 mS for an N byte WRITE transfer) could be used for useful processing.

REFERENCES

(1) Application Note 413, "Initialization of the ST9", Pierre Guillemin and Alan Dunworth, SGS-THOMSON Microelectronics.

(2) The "ST9 Technical Manual", SGS-THOMSON Microelectronics.

Appendix A. EEPROM I²C-bus Manager Routine

```
.title     " ST9 SPI use with I²C protocol.          January 24 1990 "
.sbttl     " EEPROM manager                            version 2.0 "
.list bex
.global          IT_END_TRANS, TEMPO, EEP_MAN
.extern          RESET_START

;***************************
;* Module Macro Definitions *
;***************************
.library   "c:\st9\inc\bitmacro.inc"  ; change as required
.mcall           ifbit, attbit
;-----------------------------------------
.macro     DELAI  ?loop_var

     ld    COUNTER,#03h          ; 10 Tcy.
loop_var:
     dec   COUNTER               ; 6 Tcy.
     jrnz  loop_var              ; 12 Tcy: A loop = 1.5 fs
                                 ; with a 12 MHz system clock.
.endm
;-----------------------------------------


;-----------------------------------------
.macro     DIS_SPI_IT            ; Disable SPI interrupt.
     and   EIPR,#~ipb0m          ; Reset the B0 ( SPI interrupt) pending bit.
     and   EIMR,#~ib0m           ; Disable SPI channel (B0).
.endm
;-----------------------------------------


;-----------------------------------------
.macro     EN_SPI_IT             ; Enable SPI interrupt.
     and   EIPR,#~ipb0m          ; Clear request on SPI channel (B0).
     nop
     or    EIMR,#ib0m            ; Enable SPI channel (B0).
.endm
;-----------------------------------------


;-----------------------------------------
.macro     INIT_TRANS_READ       ; Initialize SPI register and interrupt
                                 ; for read operation.
     ld    STAT_TRANS_SPI,#TRANS_RD_DATA    ; Initialisation for read operation.
     spp   #0
     ld    SPI_TAMP,#0FFH        ; To read the data from the EEPROM.
.endm
;-----------------------------------------
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
;***********************
; Register declarations.
;***********************

EEP_ADD           =       R0     ; Operation address in the EEPROM.
WRITE_DATA        =       R1     ; Data to be programmed in the EEPROM.
write_data        =       r1
READ_DATA         =       R2     ; Data which has been read from the EEPROM.
read_data         =       r2
EEP_FUNCT         =       R3
STAT_EEP          =       R4
STAT_TRANS_SPI    =       R5
NB_BYTE           =       R6     ; Number of bytes to be written
                                 ; (maximum 8) or to read.
nb_byte           =       r6
SPI_TAMP          =       R7
MEMO_NB_BYTE      =       R14
COUNTER           =       R15
DATA_TABLE        =       31     ; The real beginning of the table
;                         (1Fh)  ; to store data is R20h.

;***********************
; Constant declarations.
;***********************

ADD_EEP_W         ==      0A0h   ; Address the external EEPROM slave
                                 ; for WRITE operation.
ADD_EEP_R         ==      0A1h   ; Address the external EEPROM slave
                                 ; for READ operation.
SDI_MASK          =       02h    ; SDI = bit 1 of port 2.
SCK_MASK          =       04h    ; SCK = bit 2 of port 2.
SDO_MASK          =       08h    ; SDO = bit 3 of port 2.

;—— Status of EEP_FUNCT register.
;     This register is used to indicate the EEPROM manager the
;     function to be executed.

READ_FUNCT        ==      1      ; Read mode: read after transferring the
                                 ; address pointer.
                                 ; ie: Read from the current address.
WRITE_FUNCT       ==      2      ; Write mode.
VERIF_FUNCT       ==      3      ; Alternate read mode:
                                 ;Read operation without programming
                                 ;the address pointer.
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
;—— Status of STAT_TRANS_SPI register.
;     This register permits the EEPROM manager (in the SPI interrupt routine) ...
;     ... to know the type of the byte which has just been transmitted.

T_ADD_SLAVE       ==      1      ; The eeprom address has been transferred.
T_ADD_EEP         ==      2      ; The operation address has been transferred.
TRANS_WR_DATA     ==      3      ; The data to be written has been transferred.
TRANS_RD_DATA     ==      4      ; The data to be read has been received.

;—— Status of STAT_EEP register.
;          This register permits the caller to know the status of the EEPROM.

EEP_OK            ==      0      ; EEPROM is OK.
LECT_ON           ==      1      ; EEPROM is reading a byte.
VERIF_ON          ==      2      ; EEPROM is reading the current byte.
ECR_ON            ==      3      ; EEPROM is programming a byte.
NO_ACK            ==      4      ; EEPROM has not acknowledged.

EEP_FREE_MASK     ==      80h    ; EEPROM is ready for a new operation...
                                 ; ... if this bit is equal to 1.
.text
;*****************************************************************************
; EEP_MANAGER:   EEPROM MANAGER.
;
;*****************************************************************************
proc  EEP_MAN [PPR] {            ; Save page pointer.
      spp  #0
      DIS_SPI_IT
      ld   MEMO_NB_BYTE,NB_BYTE ; Save NB_BYTE Before decrement for
                                 ; programmation tempo.
      switch [ EEP_FUNCT ] {
           case  #READ_FUNCT:
                 call    READ_EEP
           case  #VERIF_FUNCT:
                 call    VERIF_EEP
           case  #WRITE_FUNCT:
                 call    WRITE_EEP
      } ;-- End of switch.
} ;- End of proc.
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
;*****************************************************************************
;     READ_EEP:   Normal read mode.
;                 Read of some bytes after setting the slave address.
;
;*****************************************************************************
proc  READ_EEP [PPR] {
      ifbit STAT_EEP,#EEP_FREE_MASK    ; Test if EEPROM free.
;     {
            call   INIT_START_I2C      ; SPI and related interrupt initialization..
                                       ; ... to support I2C protocol...
                                       ; ... Generate a start condition.
            ld     STAT_TRANS_SPI,#T_ADD_SLAVE; Slave address will be transferred.
            ld     STAT_EEP,#LECT_ON   ; A read condition is started.
                                       ; EEPROM is not FREE = EEP_FREE_BIT = 0.
            ld     SPIDR,#ADD_EEP_W    ; EEPROM address in write mode to transfer
                                       ; pointer.
      } ;-- End of if.
} ;-- end of proc.

;*****************************************************************************
;     VERIF_EEP:  Alternate read mode.
;                 Read of some bytes without setting the address pointer.
;
;*****************************************************************************
proc  VERIF_EEP [PPR] {
      ifbit STAT_EEP,#EEP_FREE_MASK    ; Test if EEPROM free.
;     {
            call   INIT_START_I2C      ; SPI and related interrupt initialization..
                                       ; ... to support I2C protocol...
                                       ; ... Generate a start condition.
            ld     STAT_TRANS_SPI,#T_ADD_SLAVE; Slave address will be transferred.
            ld     STAT_EEP,#VERIF_ON  ; A verif condition is started.
                                       ; EEPROM is not FREE = EEP_FREE_BIT = 0.
            ld     SPIDR,#ADD_EEP_R    ; EEPROM address in read mode.
      } ;-- End of if.
} ;-- end of proc.
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
;*****************************************************************************
;      WRITE_EEP: Write of some bytes.
;
;*****************************************************************************
proc  WRITE_EEP [PPR] {
      ifbit STAT_EEP,#EEP_FREE_MASK   ; Test if EEPROM free.
;     {

            call   INIT_START_I²C       ; SPI and related interrupt initialization..
                                        ; ... to support I²C protocol...
                                        ; ... Generate a start condition.
            ld     STAT_TRANS_SPI,#T_ADD_SLAVE; Slave address will be transferred.
            ld     STAT_EEP,#ECR_ON     ; A write condition is started.
                                        ; EEPROM is not FREE = EEP_FREE_BIT = 0.
            ld     SPIDR,#ADD_EEP_W     ; EEPROM address in write mode.
      } ;-- End of if.
} ;-- end of proc.

;*****************************************************************************
;     INIT_START_I²C:
;           Initialize SPI to support I²C protocol.
;           Generation of a start condition.
;
;*****************************************************************************
proc  INIT_START_I²C [PPR] {
;-- SPI initialization.
      spp   #0                    ; SPI and ext. interrupts registers in page 0.
      ld    SPICR,#042h           ; SPI is Disabled = SDA and SCK in HZ (1).
                                  ; I²C bus mode is selected.
                                  ; SCK frequency # 100 kHz.
;-- START condition generation.
      and   P2DR,#~SDO_MASK       ; Prepare "0" on output buffer of SDO.
      spp   #P2C_PG
      and   P2C0R,#~SDO_MASK      ; SDO line in output - SDA line = "0".
      DELAI                       ; Wait for start condition hold time.
      spp   #0
      or    SPICR,#spen           ; Enable SPI.
      EN_SPI_IT                   ; Enable SPI interrupt.
      spp   #P2C_PG
      or    P2C0R,#SDO_MASK       ; SDO line in AF.
} ;-- End
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
;*****************************************************************************
;     GEN_STOP:   Generation of a stop condition.
;
;*****************************************************************************
proc  GEN_STOP [PPR] {
      spp  #0
      DIS_SPI_IT                ; Disable SPI interrupt.
      and  P2DR,#~SDO_MASK      ; Prepare "0" on output buffer of SDO.
      spp  #P2C_PG
      and  P2C0R,#~SDO_MASK     ; SDO line in output - SDA line = "0".
      spp  #0
      and  SPICR,#~spen         ; Disable SPI - Release SCK line - SCK = "1".
      DELAI                     ; Wait for stop condition setup.
      spp  #P2C_PG
      or   P2C0R,#SDO_MASK      ; SDO in AF - Release SDA line - SDA = "1".
} ;-- End

;*****************************************************************************
;     GEN_ACK:    ACK pulse generation,
;                 and force the SDA line to 0 for Acknowledgement.
;
;*****************************************************************************
proc  GEN_ACK [ PPR ] {
      and  P2DR,#~SDO_MASK      ; Prepare "0" on output buffer of SDO.
      spp  #P2C_PG
      and  P2C0R,#~SDO_MASK     ; SDO line in output - SDA line = "0".
      spp  #0
      and  SPICR,#~spen         ; Disable SPI - Release SCK line - SCK = "1".
      DELAI                     ; Wait for ACK hold time.
      or   SPICR,#spen          ; Enable SPI - Force SDA and SCK low.
      spp  #P2C_PG
      or   P2C0R,#SDO_MASK      ; SDO line in AF.
} ;-- End of proc.
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
;*******************************************************************************
;     TEST_ACK:   ACK pulse generation,
;                 and check the slave acknowledgment.
;*******************************************************************************
proc  TEST_ACK [ PPR ] {
      and         SPICR,#~spen          ; Release SPI lines in disabling it.
      attbit      P2DR,#SCK_MASK        ; Wait for SCK going high.
      ifbitP2DR,  #SDI_MASK             ; Check if receiver has acknowledged.
                                        ;(SDA = 0).
;     {                                 ; If no ACK.
           ld     STAT_EEP,#NO_ACK
      } else {                          ; If ACK ok.
           DELAI                        ; Wait for high period of the clock.
      } ;-- End of if.
      or    SPICR,#spen                 ; Enable SPI - Force SDA low.
} ;-- End of proc.

;*******************************************************************************
;     IT_END_TRANS:       SPI end of transmission interrupt service routine.
;                         This interrupt is connected to channel B0 in the ST9.
;*******************************************************************************
IT_END_TRANS::
      pushu       PPR
      pushuw      RPP
      srp   #0
      spp   #0
      if [ STAT_TRANS_SPI == #TRANS_RD_DATA ] {

;-- A data to be read has been received from EEPROM.
           ld     read_data,SPIDR      ; For the next instruction addressing mode.
           ld     DATA_TABLE(nb_byte),read_data     ; Save the received data.
           dec    nb_byte              ; Number of bytes to be read.
           if     [ SETZ ] {           ; If the last byte has been read.
                  call    GEN_STOP     ; Gnrate STOP condition in I2C protocol.
                  ld      STAT_EEP,#EEP_FREE_MASK   ; Indicates to the caller than
                                                    ;EEPROM is OK and FREE.
           } else {
                  call    GEN_ACK      ; ACK pulse generation and force SDA line
                                       ; to 0.
                  INIT_TRANS_READ
           } ;-- End of else.
      } else {
           call   TEST_ACK             ; ACK pulse generation and test EEPROM
                                       ;response..
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
          if     [ STAT_EEP == #NO_ACK ] {          ; If no acknowledge from EEPROM.
                 call   GEN_STOP                   ; Stop generation.
                 or     STAT_EEP,#EEP_FREE_MASK    ; Indicates to the caller than
                                                   ; EEPROM is free.
                 switch [ STAT_TRANS_SPI ] {
                 case   #T_ADD_SLAVE:
;-- The slave address has been transferred.
                        if     [ STAT_EEP == #VERIF_ON ] {
;-- The slave address has been transmitted for a verif operation.
                        } else {
;-- The slave address has been transmitted for a write or a random read operation.
                               ld     STAT_TRANS_SPI,#T_ADD_EEP
                                                   ; Transfer of the address of
                                                   ; the EEPROM operation.
                               spp    #0
                               ld     SPI_TAMP,EEP_ADD    ; To transfer the read
                                                          ;or write address.
                 case   #T_ADD_EEP:
;-- The write or random read address has been transmitted.
#READ_FUNCT ] {
;-- The random read addresss has been transmitted.
                               call   INIT_START_I2C      ; A start condition is
                                                          ; necessary here.
                               ld     STAT_TRANS_SPI,#T_ADD_SLAVE
                                                          ; The slave address must
                                                          ; be transmitted again.
                               ld     SPI_TAMP,#ADD_EEP_R; EEPROM address in read
                                                          ; mode.
                               ld     STAT_EEP,#VERIF_ON ; The next sequence is
                                                          ; the same than verif
                                                          ; sequence.
;-- The write address has been transmitted.
                        } else {
                               spp    #0
                               ld     STAT_TRANS_SPI,#TRANS_WR_DATA
                                                   ; Initialisation for transfer
                                                   ; of data to be written.
                               ld     write_data,DATA_TABLE(nb_byte)
                                                   ; The first data to programm.
                               ld     SPI_TAMP,write_data
                        } ;-- End of else.

                 case   #TRANS_WR_DATA:

;-- The data to be written has been transmitted.
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
                        spp   #0
                        dec   nb_byte       ; Number of bytes to write.
                        if    [ CLZ ] {     ; If the last byte has not yet
                                            ; been written.
                              ld    write_data,DATA_TABLE(nb_byte)
                              ld    SPI_TAMP,write_data
                        } else {            ; If all data have been programmed.
                                            ; Write sequence is finished.
                              call  PROG_DELAY   ; Initialise watch dog timer
                                                 ;to generate a 5 ms delay.
                              call  GEN_STOP     ; STOP condition generation.
                        } ;-- End of else.
                  } ;-- End of switch.
            } ;-- End of else.
      } ;-- End of else.
      popuwRPP
      ld    SPIDR,SPI_TAMP                ; Data to transmit via SPI.
      popu PPR
      iret
;*****************************************************************************
;     PROG_DELAY  Initialize the watchdog-timer to generate the delay
;                 necessary for programmation.
;
;*****************************************************************************
proc  PROG_DELAY [ PPR ] {
      pushuw      RPP
      spp   #WDT_PG
      srp   # (15 * 2)                 ; To access in paged registers with r.
      ld    wcr,#wden                   ; watch dog mode disabled, no wait states.
      clr   wdtpr                       ; To have 333 ns (with system clock = 12 MHz)
                                        ; in minimum count,
                                        ; prescaler = 0.
      ldw   wdtr,#15015                 ; 15015 * 333 ns = 5 ms.

                  while   [ CLZ ] {
            addw  wdtr,#15015           ; 5 ms delay is multiplied by
                                        ; the number of bytes to programm.
            dec   MEMO_NB_BYTE
      }
      or    wdtcr,#( stsp | sc )        ; Timer starts down counting.

                                        ; Single mode.

                                        ; Watch Dog disabled.
```

Appendix A. EEPROM I²C-bus Manager Routine (Continued)

```
                                        ; Input section disabled.
                                        ; Output disabled.
                                        ; Interrupt A0 on Timer EOC.
                                        ; Top Level Interrupt on SW TRAP.
      popuwRPP
}
;*****************************************************************************
;     TEMPO:      Interrupt service routine of the watchdog timer end of count.
;                 This interrupt is connected to the A0 channel in the ST9.
;
;*****************************************************************************
TEMPO:
      pushuPPR
      spp   #0
      and   EIPR,#~ipa0m                ; Reset of WD/Timer EOC interrupt pending
                                        ; bit.
      or    STAT_EEP,#EEP_FREE_MASK     ; Write sequence is finished.
      popu PPR
      iret
```

Appendix B. Examples of Calling Programs

```
.title     " Main example for EEPPROM manager call          January 24 1990 "


.extern          IT_END_TRANS, EEP_MAN, TEMPO
.global          RESET_START
;*************************
; Module Macro Definitions.
;*************************
.library   "c:\st9\inc\bitmacro.inc"  ; change if required
.mcall           attbit

;***********************
; Register declarations.
;***********************
EEP_ADD          =       R0     ; Operation address in the EEPROM.
WRITE_DATA       =       R1     ; Data to be programmed in the EEPROM.
write_data       =       r1
READ_DATA        =       R2     ; Data which has been read in the EEPROM.
read_data        =       r2
EEP
FUNCT =    R3
STAT_EEP         =       R4
STAT_TRANS_SPI   =       R5
NB_BYTE          =       R6
nb_byte          =       r6
CPT_DELAY        =       RR8
DATA_TABLE       =       31     ; The real beginning of the table to store
                                ; data is R20h.
;                01Fh

;****************************
; INTERRUPT VECTOR ADDRESSES.
;****************************
ORE_IT_VECT      :=      00h    ; Core interrupt vectors
EXT_IT_VECT      :=      20h    ; External interrupt vectors

;******************
; START of PROGRAM.
;******************
START_PROG       :=      100h   ; start address program


;*******************
; STACK Declaration.
;*******************
SSTACK           :=      ( 14 * 16 ) - 1     ; System stack address group D C
USTACK           :=      ( 12 * 16 ) - 1     ; User stack address group B
```

Appendix B. Examples of Calling Programs (Continued)

```
;*****************************************
; Declaration of the interrupt vectors table.
;*****************************************
      .text                          ; start of program
           .org   CORE_IT_VECT       ; Core interrupt vector
                                     ; *********************
                  .word   RESET_START  ; power on interrupt vector
           .org   EXT_IT_VECT        ; External interrupt vector
                                     ; ************************
                  .word   TEMPO      ; Channel A0 for Watchdog Timer.
                  .word   0000       ; Channel A1 not used/
                  .word   IT_END_TRANS ; Channel B0 for SPI.

;**********************
; Start of main module.
;**********************
           .org   START_PROG         ; start of code
RESET_START:
      spp  #0
      ld   MODER,#( sspm | uspm| div2m ); CLOCK MODE REGISTER
                                     ; internal stack
                                     ; no precaling
                                     ; external clock divided by 2
;--- SPI and related I/O initialization.
      spp  #P2C_PG                   ; P21 = SDI: IN/TRI/TTL.
      ld   P2C0R,#00001110b          ; P22 = SCK: AF/OD/TTL.
      ld   P2C1R,#11111101b          ; P23 = SDO: AF/OD/TTL.
      ld   P2C2R,#00001110b          ; Others = OUT/PP/TTL.
      spp  #0
      ld   CICR,#( gcenm | iamm | cplm ); CENTRAL INTERRUPT CONTROL REGISTER
                                     ; priority level = 7
                                     ; Nested Arbitration mode
                                     ; disable interrupt
                                     ; enable counters
      spp  #0

      srp  #(15 * 2)                 ; To access page 0 registers
      clr  eipr                      ; Disable all the external interrupt
                                     ; pending bits.
      nop                            ; See WARNING (Technical Manual - Chapter 8)
      ld   eivr,#EXT_IT_VECT         ; External interrupt vector.
                                     ; IAOS - TLIS = 00 = ...
      ld   eiplr,#0FBh               ; Priority level for group INTA0
                                     ; INTA1 = 6, 7.
```

Appendix B. Examples of Calling Programs (Continued)

```
ld    eimr,#01                          ; Unmask Interrupt A0 channel
                                        ; (WDT End Of Count).
                                        ; ( SPI EOT ).
                                        ; bit is active.
      clr  FLAGR                        ; init flag
      ld   SSPLR,#SSTACK + 1            ; load system stack pointer
      ld   USPLR,#USTACK + 1            ; load user stack pointer
      ld   STAT_EEP,#EEP_FREE_MASK      ; EEPROM is free, no function in service.
      ei
;*******************************************************************************
;Exemple of call to the EEPROM manager to programm 4 bytes from the address 010h.
;*******************************************************************************
begin_write::
      ld   EEP_FUNCT,#WRITE_FUNCT       ; Function to be executed by the
                                        ; EEPROM manager.
      ld   EEP_ADD,#010h                ; 1st address to be programmed.
      ld   NB_BYTE,#4                   ; Number of bytes to program.
      ld   R#(DATA_TABLE+4),#78h        ; 1st data to programm.

      ld   R#(DATA_TABLE+3),#49h        ; 2nd data to programm.
      ld   R#(DATA_TABLE+2),#10h        ; 3rd data to programm.
      ld   R#(DATA_TABLE+1),#94h        ; 4th data to programm.
      call EEP_MAN
      attbit STAT_EEP,#EEP_FREE_MASK    ; Wait for end of WRITE procedure
                                        ; (programming delay also).
                                        ; by the ST9.
      nop                               ; To replace by a JR instruction
                                        ; under SDBST9 for DEBUG.
      nop
;*******************************************************************************
;Example of call to the EEPROM manager to read 6 bytes from the address 0fh.
;This can be a verification of the last programmation.
;*******************************************************************************
begin_read::
      ld   EEP_FUNCT,#READ_FUNCT        ; Function to be executed by the
                                        ; EEPROM manager.
      ld   EEP_ADD,#0fh                 ; Read address in EEPROM.
      ld   NB_BYTE,#6                   ; Number of data to be read.
      call EEP_MAN
      attbit STAT_EEP,#EEP_FREE_MASK    ; Wait for end of read procedure.
                                        ; Here some instructions
                                        ; could be executed by the ST9.
end_read::
      jr   end_read
```

Appendix C. Module Macro Definitions

```
 title     " BITMACRO.INC                                  05 December 1989 "
;*******************************************************************************
;*******************************************************************************
; BITMACRO:       Macro file allowing bit test like PSEUDO_MACROS programmation,
; User must declare the macro used in his ST9 source file like the following
example

;.library  "c:\st9\inc\bitmacro.inc"
;.mcall           ifbit, ifnobit, and so on
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************
;--- macro-instruction IFBIT: test if a bit is 1.
;    Parameters: - destination: All addressing mode allowed by
;                                the"tm*instruction.
;                           - mask selecting the bit to be tested.
;                                ex: 00000010b for bit 1 test.
;
; !!! DO not forget the "}" after instructions executed when the condition is
;      TRUE.
; application example
;     ifbitdest,mask
;          ...
;          ...
;     }
.macro     ifbit  dest,mask
      tm    dest,mask
      if    [ CLZ ] {            ; The bit is set to 1.
.endm
;*******************************************************************************
;*******************************************************************************
;     macro-instruction WHILEBIT: DO WHILE bit is 1.
;    Parameters: - destination: All addressing mode used for "tm" instruction.
;                           - mask selecting the bit to be tested.
;                                ex: 00000010b to test bit 1.
;
; application example
      do {
;          ...
;          ...
      whilebit    dest,mask
.macro     whilebit       dest,mask
      tm    dest,mask
      }     while  [ CLZ ]       ; The bit is set to 1.
.endm
```

Appendix C. Module Macro Definitions (Continued)

```
;*****************************************************************************
;*****************************************************************************
;--- macro-instruction IFNOBIT: test if a bit is 0.
;    Parameters: - destination: All the addressing mode used for the "tm"
;                                instruction.
;                           - mask selecting the bit to be tested.
;                                ex: 00000010b to test bit 1.
;
; !!! Do not forget the "}" after instructions executed when the condition is
;      TRUE.
; application example
;     ifnobit     dest,mak
;           ...
;           ...
;     }
.macro      ifnobit        dest,mask
      tm    dest,mask
      if    [ SETZ ] {           ; the bit is set to 1.
.endm
;*****************************************************************************
;*****************************************************************************
;--- macro-instruction WHILENOBIT: DO WHILE bit = 0.
;    Parameters: - destination: All the addressing mode used for the "tm"
;                                instruction.
;                           - mask selecting the bit to be tested.
;                                ex: 00000010b to test bit 1.
;
; application example
      do    {
;           ...
;           ...
      whilenobit  dest,mak
.macro      whilenobit     dest,mask
      tm    dest,mask
      }     while  [ SETZ ]      ; The bit is set to 1.
.endm
;*****************************************************************************
;*****************************************************************************
;--- WAITBIT: waiting for a bit to be 1.
;    Parameters: - destination: All the addressing mode used for "tm"
;                  instruction.
;                        - mask selecting the bit to be tested.
;                             ex: 00000010b to test bit 1.
;
.macro      waitbit        dest,mask
      ********
      do {
           tm     dest,mask
      } while [ SETZ ]          ; WAITING for bit = 1.
.endm
```

Appendix C. Module Macro Definitions (Continued)

```
;*******************************************************************************
;*******************************************************************************
WAITNOBIT: waiting for a bit to be a 0.
;     Parameters: - destination: All the addressing mode used with the "tm"
;                   instruction.
;                          - mask selecting the bit to be testesd.
;                               ex: 00000010b to test bit 1.
;
.macro     waitnobit     dest,mask
      do {
           tm     dest,mask
      } while [ CLZ ]                ; WAITING for the bit = 0.
.endm
;*******************************************************************************
```

SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

EXTERNAL DMA MODE I/O DATA TRANSFER SYNCHRONIZED BY TIMER

Pierre Guillemin

INTRODUCTION

ST9 provides a powerful features allowing DMA transfers between I/O port and Register file or memory spaces (Program/Data memory). Furthermore DMA operations on I/O port can be done under handshake control and with swap mode capability.

The DMA transfer between external I/O port and memory fields (Register file, Program memory, Data memory) is possible with the help of the two Timer DMA channels (COMPARE 0 and CAPTURE 0) and only one byte transfer is performed at any request (instead of two bytes when DMA takes place between Register file/Memory spaces and Compare/Capture register).

Three Timer DMA external modes on I/O port are possible:

- COMPARE 0 channel external mode allowing only output data transfer.
- CAPTURE 0 channel external mode allowing bidirectional data transfer. (The direction of the data transfer is set by a bit in the I/O port control register.)
- CAPTURE 0 channel external mode synchronized by a COMPARE 0 event.

For these three modes, the synchronization is accomplished by an internal synchronization signal or by a Timer On-Chip Event signal (ie. COMPARE 0 or OVERFLOW/UNDERFLOW event).

To enable such transfers, user has to program:

- the Timer in CAPTURE or COMPARE or both DMA mode
- two Timer control bits (DCTS and DCTD) in Interrupt/DMA Control Register IDCR
- the Timer On-Chip Event and the handshake /DMA control register of the relevant I/O port.

Please refer to the note for a better understanding of the internal connection between ST9 TIMER and I/O port.

Note: On **ST9030**, the On-Chip Event of **TIMER 1** controls the handshake function with I/O Port 5, the On-Chip Event of Timer 0 is connected to the internal trigger of the A/D converter.

On **ST9020**, the On-Chip Event of **TIMER 0** controls the handshake function with I/O Port 5.

On **ST9050**, the On-Chip Event of **TIMER 0** controls the handshake function with I/O Port 4, the On-Chip Event output of Timer 1 controls the handshake function with I/O Port 5, the On-Chip Event of Timer 3 is connected to the internal trigger of the A/D Converter.

Furthermore, the Timer's output signals T0OUTA and T1OUTA may be software connected respectively to the same Timer's input signals T0INA and T1INA.

TIMER DMA EXTERNAL MODE CONTROL BITS

To program the Timer and I/O port in DMA external mode, user has to set:
- two control bits in Timer Interrupt/DMA Control Register (IDCR)
- two bits in the I/O Connection Register (IOCR)
- the Timer On-Chip Event and four control bit in the handshake/DMA Control Register (HDCTL) of the relevant I/O port.

TIMER CONTROL BITS

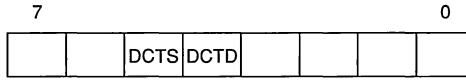
DMA transfer configuration bits

Two bits located in the Timer Interrupt/DMA Control Register (IDCR) select the source or the destination of the DMA transfer.

IDCR R243 (F3h) or R247 (F7h)

Interrupt/DMA Control Register

Page 9 or 13 Read/Write
Reset value: 10100111b (A7h)



b5 = **DCTS:** *DMA Capture Transfer Source.* This bit selects the source of the DMA operation related to the channel associated to CAPTURE 0.

DCTS = "0": The DMA transfer source is the Capture 0 register

DCTS = "1": The DMA transfer source is the I/O port

b4 = **DCTD**: *DMA Compare Transfer Destination.* This bit selects the destination of the DMA operation related to the channel associated to the COMPARE 0.

DCTD = "0": The DMA transfer destination is the COMPARE 0 register

DCTD = "1": The DMA transfer destination is the I/O port

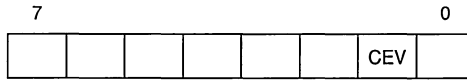
OEV, CEV: Timer On-Chip Event

These two bits, located in the output control registers (OACR and OBCR), select the event source strobing the data transfer on I/O port.

OACR R252 (FCh)

Output A Control Register

Page 8 (10, 12, 14) Read/Write
Reset value: xxxxxx0xb



b1 = **CEV** : *On-Chip Event on COMPARE 0*

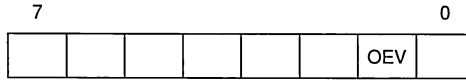
CEV = "1": On-Chip Event on successful COMPARE 0 event

TIMER CONTROL BITS (Continued)

OBCR R253 (FDh)

Output B Control Register

Page 8 (10,12,14) Read/Write
Reset value: xxxxxx0xb



b1 = **OEV :** *On-Chip Event on OVERFLOW/UNDERFLOW*

OEV = "1": On-Chip Event on OVERFLOW/UNDERFLOW *event*

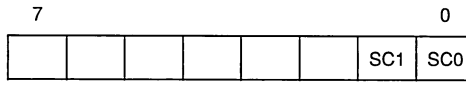
SC1, SC2: Timer I/O Connection bits

These two bits, located in the I/O Connection Register (IOCR), select (or not) an on-chip connection between input A and output B of the same Timer.

IOCR R248 (F8h)

I/O Connection Register

Page 9 or 13 Read/Write
Reset value: xxxxxx00b



b1 = **SC1**: *Select connection odd.*
Selects if connection between TxOUTA and TxINA for odd timer (x = 1 or 3) is done on-chip or externally.

SC1 = "0" TxOUTA and TxINA unconnected

SC1 = "1" TxOUTA and TxINA connected

b0 = **SC0**: *Select connection even.*
Selects if connection between TxOUTA and TxINA for even timer (x = 0 or 2) is done on-chip or externally.

SC0 = "0" TxOUTA and TxINA unconnected

SC0 = "1" TxOUTA and TxINA connected

I/O PORT CONTROL BITS

Apart from the three bits used for handshake programing, four bits located in the Handshake/DMA Control Register (HDCTL) of the relevant I/O port are used to control the DMA mode.

HDCTL

Handshake/DMA Control Register

R255 (FFh) Page 2 Handshake on Port 3

R251 (FBh) Page 2 Handshake on Port 2

R247 (F7h) Page 3 Handshake on Port 5

R243 (F3h) Page 3 Handshake on Port 4

Read/Write
Reset value: FFh

7							0
			DEN	DD	DST	DCH	1

b4 = **DEN:** *DMA enable bit*

DEN = 0: enable the DMA mode on I/O port

DEN = 1: disable the DMA mode on I/O port

b3 = **DD:** *DMA DATA Direction bit*

DD sets the DMA direction for the DMA transfer on CAPTURE 0 channel

DD = 0: output data on I/O port

DD = 1: input data from I/O port

b2 = **DST**: *DMA strobe bit*

DST bit selects I/O port strobe from internal synchronisation signal or from Timer On-Chip Event.

DST = 0: Internal synchronization strobe

DST = 1: Timer On-Chip Event strobe

b1 = **DCH**: *DMA channel mode*

DCH bit selects the DMA channel from CAPTURE 0 or COMPARE 0

DCH = 0: CAPTURE 0 DMA channel

DCH = 1: COMPARE 0 DMA channel

TIMER DMA TRANSFER ON COMPARE 0 CHANNEL

Principle

This mode, enabled when DCTD (DMA Compare Transaction Destination) bit is equal to "1", allows output transfer from Register File/memory to an I/O port at fixed period. In this mode, DMA direction transfer always outputs Data on I/O port. A one byte output DMA transfer is done on a COMPARE 0 request caused by a COMPARE 0 event or a software COMPARE 0 request (by writing "1" in the CM0 bit in Timer Flag Register).

The data strobe is made by an internal synchronization signal on COMPARE 0 event or by a Timer OVERFLOW/UNDERFLOW On-Chip Event.

Figure 1 shows the principle of COMPARE 0 channel external mode.

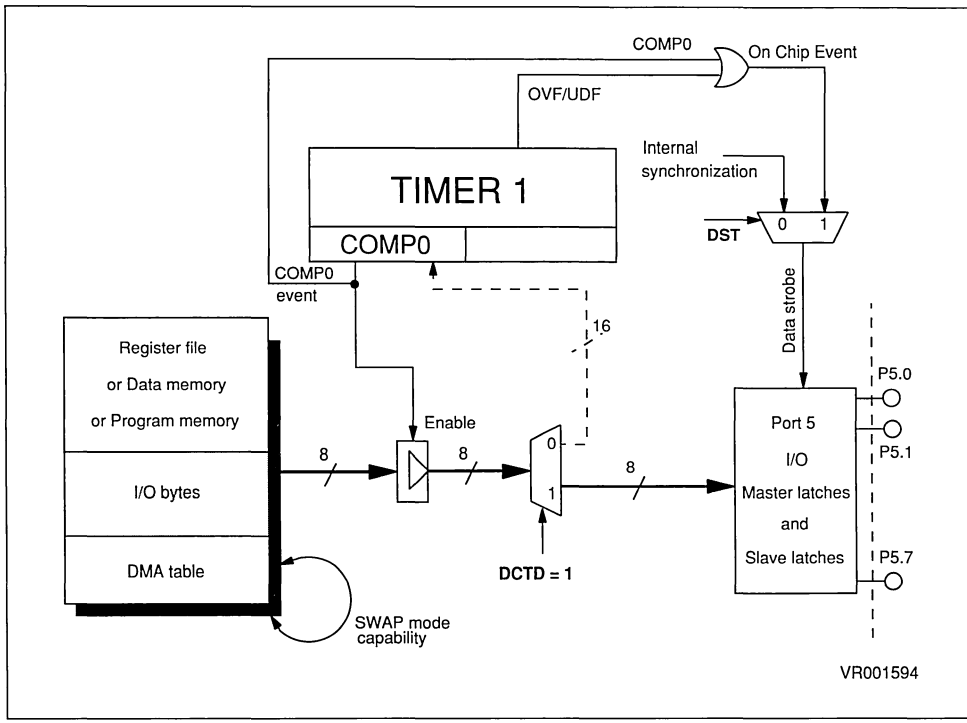
Programmation

To program Timer in DMA transfer mode on COMPARE 0 channel, user has to:

- program the Timer in COMPARE 0 channel DMA mode
- select the I/O port as destination for DMA transfer by setting the DCTD bit in IDCR
- select the data strobe mode by programming the on chip event (internal data synchronization, Timer on chip event data strobe)
- Program the relevant I/O port by clearing the DEN bit to enable DMA mode, choose the data strobe mode (DST bit) and select the COMPARE 0 DMA channel (DCH bit).
- start DMA transfer by enabling the Timer count.

Such a programmation is shown in appendix A.

Figure 1. Timer DMA Transfer on COMPARE 0 Channel Principle



TIMER DMA TRANSFER ON CAPTURE 0 CHANNEL

Principle

This mode, enabled when DCTS (DMA Capture Transaction Source) bit is equal to "1", allows bidirectional transfer from register file/memory to/from I/O port. In this mode, the DMA transfer direction is set by the DMA direction bit in HDCTL register. The DMA transfer is done on a CAPTURE 0 request caused by an external CAPTURE event or by a software CAPTURE 0 request (by writing "1" in the CP0 bit in Timer Flag Register). The data strobe is either an internal synchronization signal on CAPTURE 0 event or a Timer COMPARE 0 or OVERFLOW/UNDERFLOW On-Chip Event.

Figure 2 shows the principle of CAPTURE 0 channel external mode

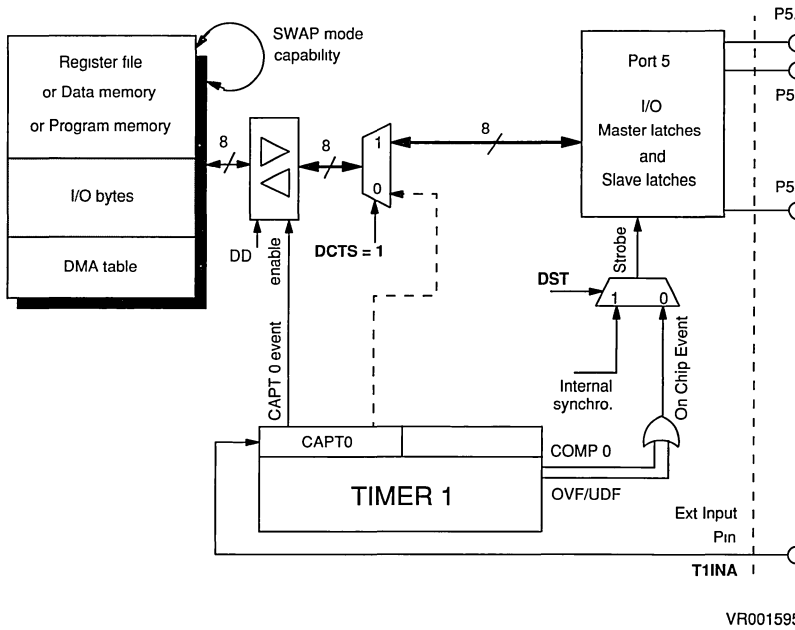
Programmation

To program Timer in DMA transfer mode on CAPTURE 0 channel, user has to:

- program the Timer in CAPTURE 0 channel DMA mode
- select the I/O port as destination for DMA transfer by setting the DCTS bit (in IDCR)
- select the data strobe mode by programming the On-Chip Event (internal data synchronization or Timer On-Chip Event strobe)
- program the relevant I/O port by clearing the DEN bit to enable DMA mode, set the DMA transfer direction (DD bit), choose the Data strobe mode (DST bit) and select the CAPTURE 0 DMA channel (DCH bit).
- start DMA transfer by enabling the Timer count.

Such a programmation is shown in appendix B.

Figure 2. Timer DMA Transfer on CAPTURE 0 Channel Principle



DMA SEQUENCER: CAPTURE 0 CHANNEL SYNCHRONIZED BY COMPARE 0 CHANNEL

Principle

This mode, using two DMA channels, allows bidirectional DMA Data transfer between I/O port and register file/memory spaces at variable throughput. This Timer DMA transfer mode uses the CAPTURE 0 channel to provide bidirectional transfer on each CAPTURE 0 event. This Capture 0 event is triggered by the COMPARE 0 channel by an output toggle on COMPARE 0 internally fed back to CAPTURE 0. Like in the previous mode, the data strobe is made by an internal synchronization signal on CAPTURE event or by a Timer COMPARE 0 or OVERFLOW/UNDERFLOW On-Chip Event.

Figure 3 shows the principle of CAPTURE 0 synchronized by COMPARE 0 event.

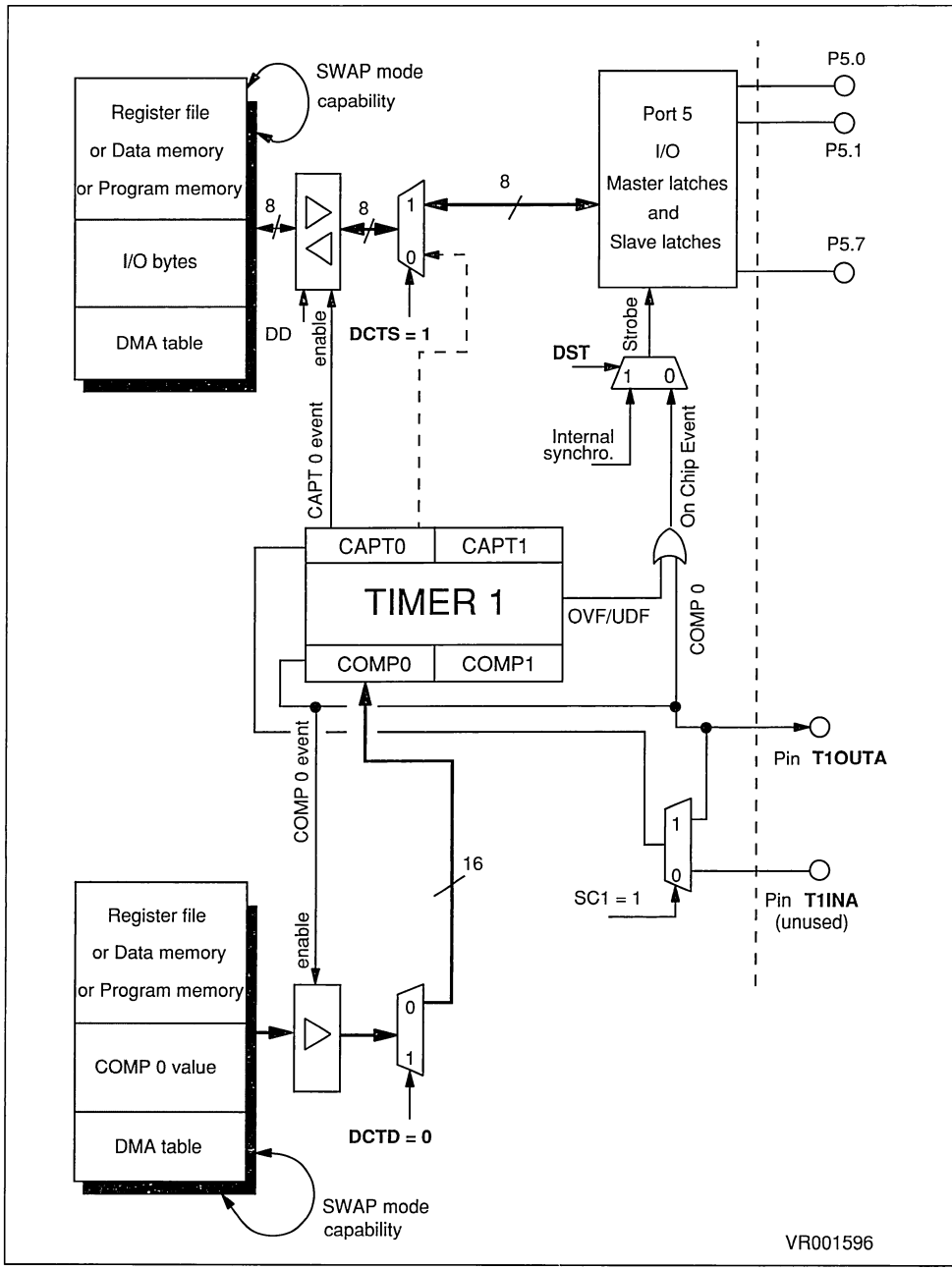
Programmation

To program Timer in DMA transfer on CAPTURE 0 synchronized by COMPARE channel, the user has to:

- program the timer in CAPTURE 0 channel DMA external mode and in COMPARE 0 DMA mode with an output toggle action on T1OUTA
- select I/O port as destination for DMA transfer by setting the DCTS bit in IDCR
- select the data strobe mode by programming the On-Chip Event (internal synchronization or Timer On-Chip Event strobe)
- program the relevant I/O port by clearing the DEN bit to enable DMA mode, set the DMA transfer destination (DD bit), choose the Data strobe mode (DST bit) and select the Capture 0 DMA channel (DCH bit)
- internally connect the timer 1 output A (T1OUTA) on the timer input A (T1INA) by setting SC1 bit in I/O Control Register (IOCR)
- start the DMA transfer by enabling the Timer count.

Such a programmation is shown in appendix C.

Figure 3. Timer DMA Transfer on CAPTURE 0 Channel Principle

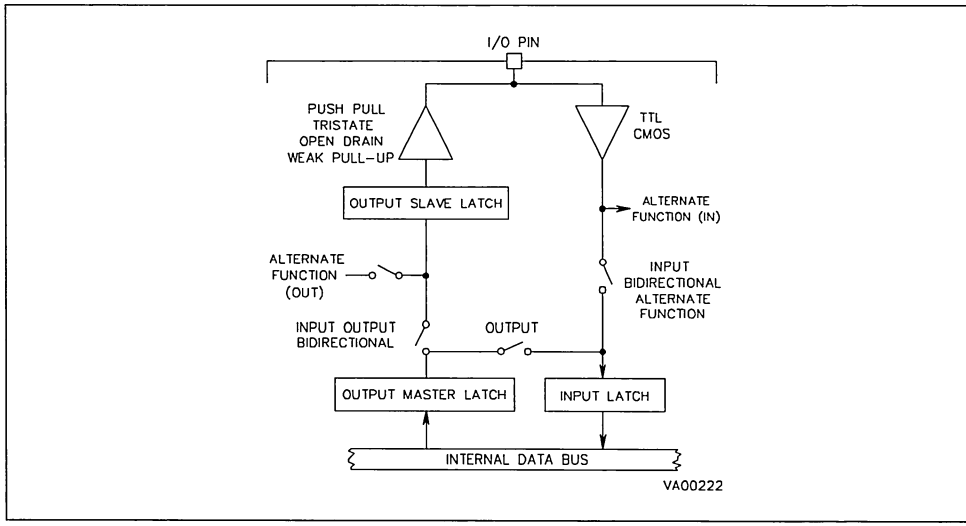


DATA OUTPUT SYNCHRONIZATION ON I/O PORT

Data Output During an Instruction Execution

The basic structure of an I/O pin shows that an I/O port is driven by an output slave latch and by an output master latch as shown in the following Figure.

Figure 4. Basic Structure of an I/O Port Pin

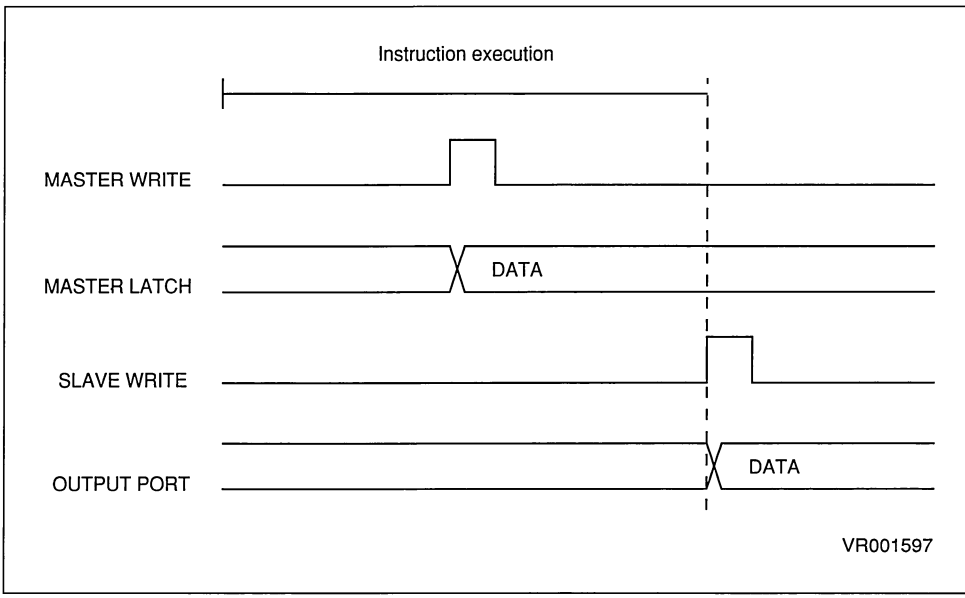


DATA OUTPUT SYNCHRONIZATION ON I/O PORT (Continued)

The data present on the internal data bus is copied in the output master latch during the execution of each instruction. The data stored into the output master latch is copied into the output slave latch (driving the I/O pin) at the end of each instruction. In input Mode data present on the I/O pin is sampled into the input data latch at the beginning of the execution of each instruction.

Figure 5 shows the timing of such a transfer.

Figure 5. Data Output During an Instruction Execution



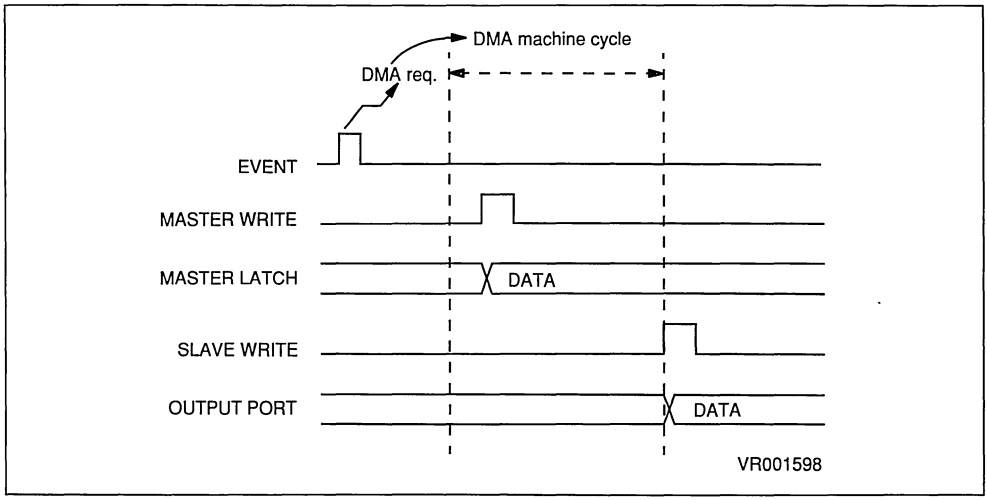
DATA OUTPUT SYNCHRONIZATION ON I/O PORT (Continued)

Data Output when Using Regular DMA Mode

In regular DMA mode, an internal synchronization signal, (depending on the DMA channel used: CAPTURE 0 channel or COMPARE 0 channel), is used to strobe the data on I/O port. In this mode, the data present on the internal data bus is copied into the output master latch during the DMA machine cycle. The data stored into the output master latch is copied into the output slave latch driving I/O pin at the end of the DMA machine cycle.

Figure 6 shows the timing of such a transfer.

Figure 6. Data Output with Regular DMA Mode



DATA OUTPUT SYNCHRONIZATION ON I/O PORT (Continued)

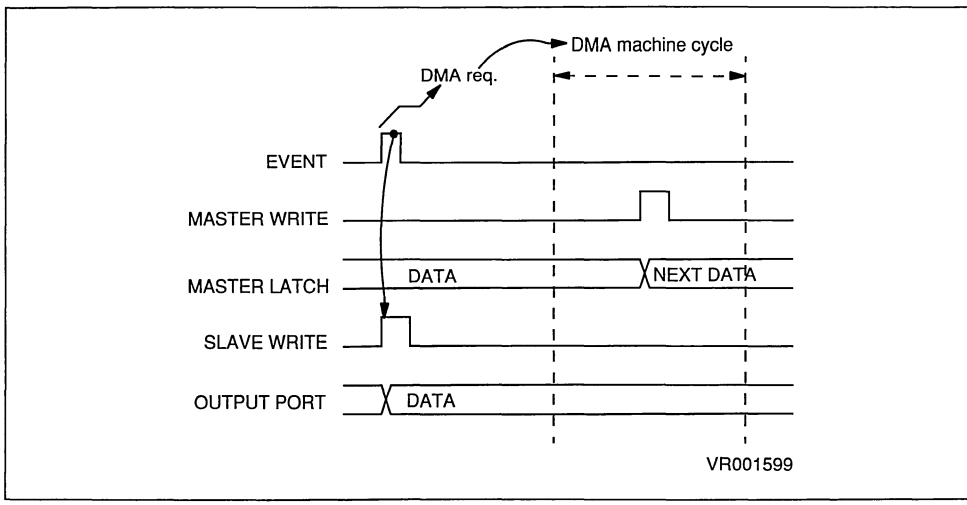
Data Output with DMA "On-Chip Event" Mode

In this mode, the data synchronization on I/O port is done by the Timer On-Chip Event signal (COMPARE 0 On-Chip Event, OVERFLOW/UNDERFLOW On-Chip Event).

The data present in the output master latch is copied into the output slave latch driving the I/O pin when the On-Chip Event occurs. The data present on the internal data bus is copied into the output master latch during the following DMA machine cycle.

Figure 7 shows the timing of such a transfer.

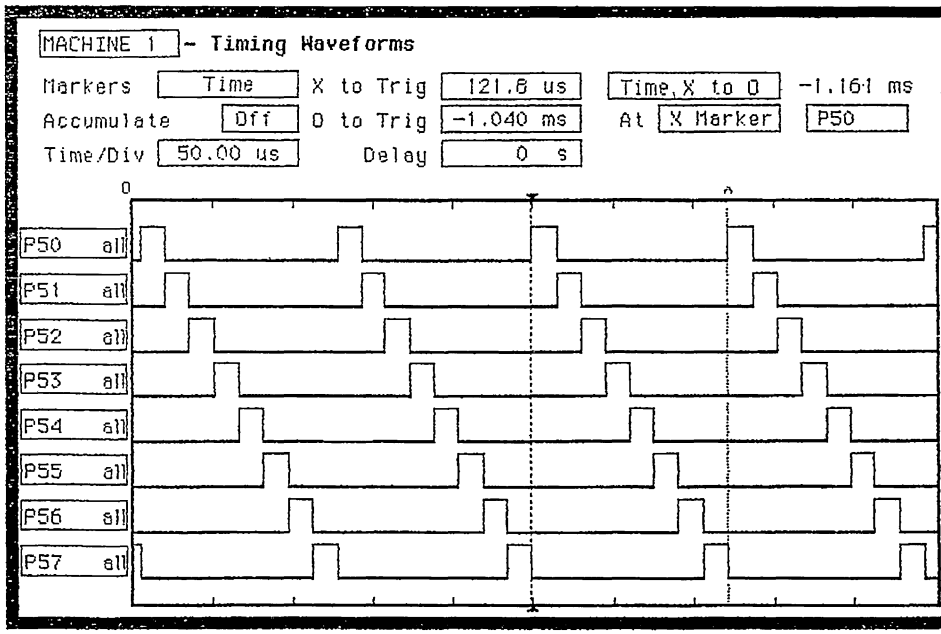
Figure 7. Data Output with DMA "On-Chip Event" mode



APPENDIX A: COMPARE 0 CHANNEL IN EXTERNAL MODE

The following examples details how to program the Timer in order to generate the waveform shown on Figure 8 .

Figure 8. Timing Waveform Example: COMPARE 0 Channel



To generate this specific waveform, the Timer is programmed to output data by a COMPARE 0 channel DMA transfer between a pattern table located in program memory (DMA_TABLE_OUT) and I/O port 5 (see TIMER_1 subroutine). A COMPARE 0 interrupt (DMA end of block interrupt) clears the successful COMPARE 0 flag and restarts the DMA transfer (see COMPARE 0 interrupt routine). I/O port 5 is configured in output in DMA mode (see INIT_IO subroutine).

APPENDIX A: COMPARE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
.title  "DMA between program memory and I/O port 5 with TIMER 1 COMPARE 0 channel"

;This program is a small example of using Timer 1 DMA external mode on I/O port
;The timer is programmed in COMPARE0 DMA channel EXT mode

;****************************
;*INTERRUPT VECTOR ADDRESSES*
;****************************

CORE_IT_VECT    :=          00h             ; Core interrupts vectors
T1_IT_VECT      :=          10h             ; Timer 1 interruts vectors

CAPT_IT_VECT    =           4               ; Capture event interrupt address
COMP_IT_VECT    =           6               ; COMPARE event interrupt address
T1_LEVEL        :=          4               ; Timer 1 priority level
LG_DMA          =           8               ; length of DMA

; Define the DMA pointer register
; DMA transfer from program memory

AD_DMA_BR       =           48h             ; DMA address base register
LG_DMA_BR       =           4ch             ; DMA counter base register
CMP_AD_DMA      :=          RR#AD_DMA_BR+2  ; Compare DMA address register ptr
CMP_LG_DMA      :=          RR#LG_DMA_BR+2  ; Compare DMA counter register ptr

; Define global references

.global         RESET_START, TIMER_1, INIT_IO, COMPARE0, DMA_TABLE_OUT

;********************
;*Group number names*
;********************

BKC             :=    12
BKD             :=    13
BKE             :=    14
BKF             :=    15
BK_F            :=    BKF * 2               ; group F: page registers

;************************
;* Start Timer 1 macro *
;************************

.macro  START_T1
        spp     #T1D_PG                     ; select Timer 0 page
        or      T_TCR,#( cen | ccl )        ; counter enable bit, clear counter
.endm;*******************
;*START of PROGRAM*
;*******************

START_PROG      :=    110h                  ; start address program

;********************
;*STACK Declaration*
;********************

SSTACK          :=    ( BKE * 16 ) - 1      ; System stack address group D C
USTACK          :=    ( BKC * 16 ) - 1      ; User stack address group B
```

APPENDIX A: COMPARE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
;*****************************************
;*Declaration of the interrupt vectors table*
;*****************************************

        .text                                   ; start of program

                .org  CORE_IT_VECT              ; Core interrupt vector
                                                ; *********************
                .word RESET_START               ; power on interrupt vector
                .word DIV0                      ; divided by 0 subroutine trap vector
                .word TOP_LEVEL_IT              ; Top level interrupt vector

                .org  T1_IT_VECT                ; Timer 1 interrupt vectors
                                                ; ************************

                .org  T1_IT_VECT + 6            ; unused addres
                .word COMPARE0                  ; Timer 1 compare 0 interrupt

;*******************************
;* Output data table to Port 5 *
;*******************************

                .org  100h

DMA_TABLE_OUT:  .byte 01h, 02h, 04h, 08h
                .byte 10h, 20h, 40h, 80h        ; output DMA table

;**********************
;*Start of main module*
;**********************

                .org  START_PROG                ; start of code

RESET_START:

        ld      MODER,#11100000b                ; CLOCK MODE REGISTER
                                                ; internal stack
                                                ; no prescaling
                                                ; external clock divided by 2

        ld      CICR,#10001111b                 ; CENTRAL INTERRUPT CONTROL REGISTER
                                                ; priority level = 7
                                                ; Nested Arbitration mode
                                                ; disable interrupt
                                                ; enable counters
                                                ; At reset, Global Counter Enable bit is active.

        ld      SSPLR,#SSTACK + 1               ; load system stack pointer
        ld      USPLR,#USTACK + 1               ; load user stack pointer

        call    TIMER_1                         ; Timer 1 initialization in DMA mode

        call    INIT_IO                         ; Port 5 init. DMA mode

        START_T1                                ; Start Timer 1

        ei                                      ; enable all interrupts

;MAIN PROGRAMM

        loop    {
;               wfi
        }
```

APPENDIX A: COMPARE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
;******************
;initialize TIMER 1
;******************

proc    TIMER_1 {

        srp     #BK_F                       ; select working register
        spp     #T1D_PG                     ; select timer 1 register page

        ld      t_tcr,#( ccmp0 | udc )      ; count up
                                            ; clear on compare  0

        ld      t_tmr,#0                    ; Disable output B
                                            ; Disable output A
                                            ; Internal clock
                                            ; Countinuous mode

        clr     t_icr                       ; No action on input pins
        clr     t_prsr                      ; No prescaling

        ld      t_oacr,#( ou_nop | c1_nop | c0_nop ); No action on OUTPUT0
        ld      t_obcr,#( c0_nop | c1_nop | ou_nop ); No action on OUTPUT1
        clr     t_flagr

        spp     #T1C_PG                     ; Timer 1 Control page register
        ld      t1_dcpr,#LG_DMA_BR          ; DMA counter register base address
        ld      t1_dapr,#AD_DMA_BR          ; DMA address register base address
        ld      t1_ivr,#T1_IT_VECT
        ld      t1_idcr,#( T1_LEVEL | dctd ); DMA compare transaction
        ldw     CMP_LG_DMA,#LG_DMA          ; Compare DMA counter init.
        ldw     CMP_AD_DMA,#DMA_TABLE_OUT   ; Compare DMA addres init.

        spp     #T1D_PG                     ; Timer 1 Data page
        ld      t_idmr,#( gtien | cm0i | cm0d ); Compare 0 INT and DMA

        ldw     t_reg0r,#0                  ; reg 0
        ldw     t_cmp0r,#3ch                ; 15 µs

        }
```

APPENDIX A: COMPARE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
;*****************************************************************************
;                    COMPARE 0 INTERRUPT ROUTINE
;                    DMA Interrupt End of block
;*****************************************************************************

COMPARE0:

begin   [ PPR, RP0R, RP1R ] {              ; save page pointer
                                           ; save register pointer pair

        spp     #T1D_PG                    ; Timer 1 data register page
        srp     #BK_F                      ; select group F

        and     t_flagr,#~( cm0 | ocm0 )   ; reset successful compare 0
                                           ; reset overrun on compare 0

        ldw     CMP_LG_DMA,#LG_DMA
        ldw     CMP_AD_DMA,#DMA_TABLE_OUT
        or      t_idmr,#cm0d               ; restart DMA compare 0 channel

}

        iret                               ; return from interrupt

;*****************************************************************************

;                    I/O port initialization

proc    INIT_IO [ PPR ]     {
;programmation Port 5 in OUTPUT in DMA mode

        spp     #P5C_PG                    ; Port 5 control register page
                                           ; Port 5 in DMA mode
                                           ; Port 5 Handshake disabled
                                           ; DMA on Compare 0 channel

;                    76543210

        ld      P5C0R,#00000000b
        ld      P5C1R,#11111111b
        ld      P5C2R,#00000000b
        ld      P5DR,#0

        ld      HDC5R,#( hsdis | den | ddw | dcm0 )

;.....................end init P5

}
```

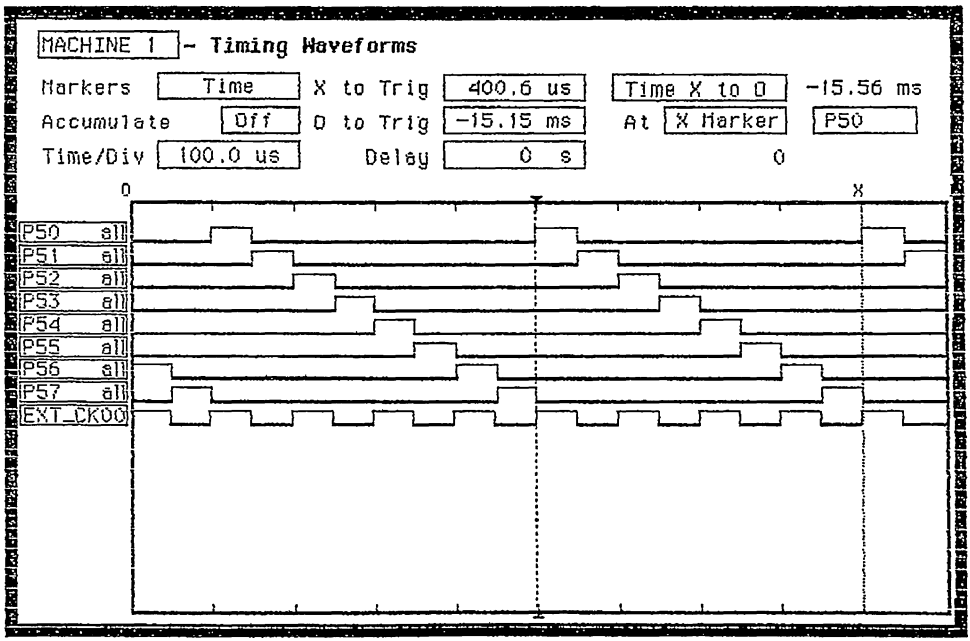
APPENDIX A: COMPARE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
;*******************************************************************************
;               SECTION CODE FOR THE CORE INTERRUPT ROUTINE
;*******************************************************************************


;-------------------------------------------------------------------------------
;               INTERRUPT ROUTINE FOR ZERO DIVISION
;-------------------------------------------------------------------------------

DIV0:   jx      DIV0                ; debug loop
        ret


;-------------------------------------------------------------------------------
;               INTERRUPT ROUTINE FOR TOP_LEVEL_IT
;-------------------------------------------------------------------------------

TOP_LEVEL_IT:

        jx      TOP_LEVEL_IT        ; debug loop
        iret
```

APPENDIX B: CAPTURE 0 CHANNEL IN EXTERNAL MODE

The goal of the following example is to generate the following waveform synchronized by an external clock signal.

Figure 9. Timing Waveform Example: CAPTURE 0 Channel in External Mode



In this mode the Timer is programmed in CAPTURE 0 DMA external mode in order to output data located in program memory to I/O port 5. The data is output on Port 5 on each rising and falling edge of the external input clock (see the Timer initialization routine TIMER_1). A CAPTURE 0 interrupt (DMA end of block interrupt) resets the successful CAPTURE 0 flag and restarts the DMA transfer. I/O Port 5 is programmed in output in DMA mode and T1INA in input mode (see INIT_IO subroutine).

APPENDIX B: CAPTURE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
.title  "DMA between program memory and I/O port 5 with timer 1 CAPTURE 0 channel"

;This program is a small example of using Timer 1 DMA external mode on I/O port

;The timer is programmed in CAPTURE0 DMA channel EXT mode

;****************************
;*INTERRUPT VECTOR ADDRESSES*
;****************************

CORE_IT_VECT          :=    00h             ; Core interrupts vectors
T1_IT_VECT            :=    10h             ; Timer 1 interruts vectors

CAPT_IT_VECT          =     4               ; CAPTURE event interrupt addres

COMP_IT_VECT          =     6               ; COMPARE event interrupt addres

T1_LEVEL              :=    4               ; Timer 1 priority level

LG_DMA                =     8               ; length of DMA

; Define the DMA pointer register

; DMA transfer from program memory

AD_DMA_BR             =     48h             ; DMA address base register

LG_DMA_BR             =     4ch             ; DMA counter base register

CPT_AD_DMA            :=    RR#AD_DMA_BR    ; Capture DMA address register pointer

CPT_LG_DMA            :=    RR#LG_DMA_BR    ; Capture DMA counter register pointer

; Define global references
.global         CAPTURE0, TIMER_1, INIT_IO, RESET_START, DMA_TABLE_OUT

;********************
;*Group number names*
;********************

BKC     :=      12

BKD     :=      13

BKE     :=      14

BKF     :=      15

BK_F    :=      BKF * 2                     ; group F: page registers

;***********************
;* Start Timer 1 macro *
;***********************

.macro  START_T1

        spp     #T1D_PG                     ; select Timer 0 page

        or      T_TCR,#( cen | ccl )        ; counter enable bit, clear counter

.endm
```

APPENDIX B: CAPTURE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
;******************
;*START of PROGRAM*
;******************

START_PROG            :=    110h                    ; start address program

;******************
;*STACK Declaration*
;******************

SSTACK                :=    ( BKE * 16 ) - 1        ; System stack address group D C

USTACK                :=    ( BKC * 16 ) - 1        ; User stack address group B

;*****************************************
;*Declaration of the interrupt vectors table*
;*****************************************

                .text                               ; Start of program

                .org  CORE_IT_VECT                  ; Core interrupt vector
                                                    ; *********************

                .word RESET_START                   ; Power on interrupt vector

                .word DIV0                          ; Divided by 0 trap vector
                .word TOP_LEVEL_IT                  ; Top level interrupt vector
                .org  T1_IT_VECT                    ; Timer 1 interrupt vectors
                                                    ; ************************

                .org  T1_IT_VECT + CAPT_IT_VECT     ; Unused address

                .word CAPTURE0                      ; Timer 1 capture 0 interrupt

;*****************************
;* Output data table to Port 5 *
;*****************************

                .org  100h

DMA_TABLE_OUT:  .byte 01h, 02h, 04h, 08h

                .byte 10h, 20h, 40h, 80h            ; Output DMA table

;**********************
;*Start of main module*
;**********************

                .org  START_PROG                    ; Start of code

RESET_START:

        ld      MODER,#11100000b                    ; CLOCK MODE REGISTER
                                                    ; internal stack
                                                    ; no prescaling
                                                    ; external clock divided by 2

        ld      CICR,#10001111b                     ; CENTRAL INTERRUPT CONTROL REGISTER
                                                    ; priority level = 7
                                                    ; Nested Arbitration mode
                                                    ; disable interrupt
                                                    ; enable counters
                                                    ; At reset, Global Counter Enable
                                                    ; bit is active.

        ld      SSPLR,#SSTACK + 1                   ; load system stack pointer
        ld      USPLR,#USTACK + 1                   ; load user stack pointer
```

APPENDIX B: CAPTURE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
        call    TIMER_1                                 ; Timer 1 initialization in DMA mode
        call    INIT_IO                                 ; Port 5 init. DMA mode
        START_T1                                        ; Start Timer 1
        ei                                              ; enable all interrupts
;MAIN PROGRAMM
        loop    {
;               wfi
        }
;******************
;initialize TIMER 1
;******************
proc    TIMER_1 {
        srp     #BK_F                                   ; select working register
        spp     #T1D_PG                                 ; select timer 1 register page
        ld      t_tcr,#( ccp0 | udc )                   ; count up
                                                        ; clear on capture 0
        ld      t_tmr,#rm0                              ; Disable output B
                                                        ; Disable output A
                                                        ; Internal clock
                                                        ; Countinuous mode
                                                        ; Capture on REG0
        ld      t_icr,#( ab_ti | exa_rf )               ; T1INA trigger, T1INB I/O
                                                        ; T1INA rising/falling edge sensitive
        clr     t_prsr                                  ; No prescaling
        ld      t_oacr,#( ou_nop | c1_nop | c0_nop );   No action on OUTPUT0
        ld      t_obcr,#( ou_nop | c1_nop | c0_nop );   No action on COMPARE 0
                                                        ; on OUTPUT1
        clr     t_flagr
        spp     #T1C_PG                                 ; Timer 1 Control page register
        ld      t1_dcpr,#LG_DMA_BR                      ; DMA counter register base address
        ld      t1_dapr,#AD_DMA_BR                      ; DMA address register base address
        ld      t1_ivr,#T1_IT_VECT
        ld      t1_idcr,#( T1_LEVEL | dcts )            ; DMA capture transaction source
        ldw     CPT_LG_DMA,#LG_DMA                      ; Capture DMA counter init.
        ldw     CPT_AD_DMA,#DMA_TABLE_OUT               ; Capture DMA addres init.
        spp     #T1D_PG                                 ; Timer 1 Data page
        ld      t_idmr,#( gtien | cp0i | cp0d )         ; Capture 0 INT and DMA
}
```

APPENDIX B: CAPTURE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
;*****************************************************************************
;                    CAPTURE 0 INTERRUPT ROUTINE
;                    DMA Interrupt End of block
;*****************************************************************************

CAPTURE0:

begin   [ PPR, RP0R, RP1R ] {              ; save page pointer
                                           ; save register pointer pair
        spp     #T1D_PG                    ; Timer 1 data register page
        srp     #BK_F                      ; select group F

        and     t_flagr,#~( cp0 | ocp0 )   ; reset successful capture 0
                                           ; reset overrun on capture 0

        ldw     CPT_LG_DMA,#LG_DMA

        ldw     CPT_AD_DMA,#DMA_TABLE_OUT

        or      t_idmr,#cp0d               ; restart DMA capture 0 channel

}

        iret                               ; return from interrupt

;*****************************************************************************
;                    I/O port initialization

proc    INIT_IO [ PPR ]     {

;programmation Port 5 in OUTPUT in DMA mode

        spp     #P5C_PG                    ; Port 5 control register page
                                           ; Port 5 in DMA mode
                                           ; Port 5 Handshake disbled
                                           ; DMA on Capture 0 channel
                                           ; DMA direction = output

;                    76543210

        ld      P5C0R,#00000000b
        ld      P5C1R,#11111111b
        ld      P5C2R,#00000000b
        ld      P5DR,#0

        ld      HDC5R,#( hsdis | den | ddw | dcp0 )

;..................... end init P5

; programmation P3.4 ( T1INA ) in INPUT, TRISTATE, TTL

        spp     #P3C_PG
        ld      P3C0R,#00010000b
        ld      P3C1R,#00000000b
        ld      P3C2R,#00010000b

}
```

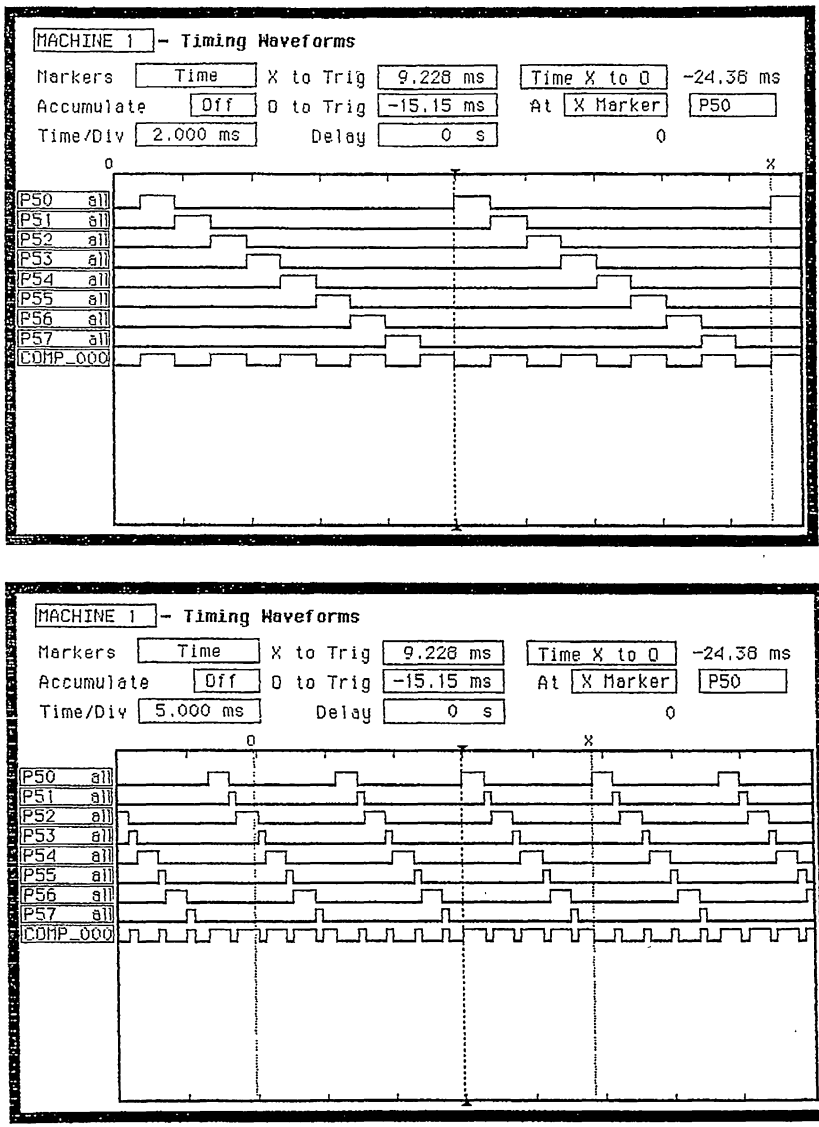
APPENDIX B: CAPTURE 0 CHANNEL IN EXTERNAL MODE (Continued)

```
;*******************************************************************************
;               SECTION CODE FOR THE CORE INTERRUPT ROUTINE
;*******************************************************************************

;-------------------------------------------------------------------------------
;               INTERRUPT ROUTINE FOR ZERO DIVISION
;-------------------------------------------------------------------------------

DIV0:

        jx      DIV0                ; debug loop
        ret

;-------------------------------------------------------------------------------
;               INTERRUPT ROUTINE FOR TOP_LEVEL_IT
;-------------------------------------------------------------------------------

TOP_LEVEL_IT:

        jx      TOP_LEVEL_IT        ; debug loop

        iret
```

APPENDIX C: CAPTURE 0 CHANNEL SYNCHRONIZED BY COMPARE 0 CHANNEL

This example output data on the CAPTURE 0 channel in external mode is synchronized by a COMPARE 0 data transfer to provide the following waveform.

Figure 10. Timing Waveform Example: DMA Channel



APPENDIX C: CAPTURE 0 CHANNEL SYNCHRONIZED BY COMPARE 0 CHANNEL (Continued)

The DMA transfer uses two data tables located in program memory : DMA_TABLE_OUT, a list of data to be output on I/O Port 5 and DMA_TIME_TABLE, a list of the time value to be loaded in the COMPARE 0 register. In this mode the Timer is programmed on CAPTURE mode with DMA transfer on CAPTURE 0 and COMPARE 0 event. T1OUTA, toggled on each COMPARE 0 event, and T1INA are connected together in order to synchronize the data transfer on the I/O port (see TIMER_1 subroutine). A COMPARE 0 interrupt (end of DMA interrupt) resets the successful CAPTURE 0 and COMPARE 0 flag and restarts the two DMA transfers (see COMPARE 0 interrupt routine). I/O Port 5 is programmed in output in DMA mode, T1INA in input mode and T1OUTA in alternate function.

APPENDIX C: CAPTURE 0 CHANNEL SYNCHRONIZED BY COMPARE 0 CHANNEL (Continued)

```
.title  "DMA between program memory I/O port 5 with timer 1 DMA channel synchronization"

;This program is a small example of using Timer 1 DMA external mode on I/O port
;The timer is programmed in COMPARE 0 and CAPTURE 0 DMA mode with DMA channel
;Synchronization

;****************************
;*INTERRUPT VECTOR ADDRESSES*
;****************************

CORE_IT_VECT          :=    00h             ; Core interrupts vectors
T1_IT_VECT            :=    10h             ; Timer 1 interruts vectors

COMP_IT_VECT          =     6               ; COMPARE event interrupt addres
T1_LEVEL              =     4               ; Timer 1 priority level
LG_DMA                =     10              ; Length of DMA

; Define the DMA pointer register
; DMA transfer from program memory

AD_DMA_BR             =     48h             ; DMA address base register
LG_DMA_BR             =     4ch             ; DMA counter base register
CPT_AD_DMA            :=    RR#AD_DMA_BR    ; Capture DMA address register pointer
CPT_LG_DMA            :=    RR#LG_DMA_BR    ; Capture DMA counter register pointer
CMP_AD_DMA            :=    RR#AD_DMA_BR+2  ; Compare DMA address register pointer
CMP_LG_DMA            :=    RR#LG_DMA_BR+2  ; Compare DMA counter register pointer

; Define global references

.global         TIMER_1, INIT_IO, RESET_START, DMA_TABLE_OUT
.global         DMA_TIME_TABLE, COMPARE0

;********************
;*Group number names*
;********************

BKC                   :=    12
BKD                   :=    13
BKE                   :=    14
BKF                   :=    15
BK_F                  :=    BKF * 2         ; group F: page registers

;************************
;* Start Timer 1 macro *
;************************

.macro  START_T1

        spp     #T1D_PG                     ; select Timer 0 page
        or      T_TCR,#( cen | ccl )        ; counter enable bit, clear counter
.endm

;******************
;*START of PROGRAM*
;******************

START_PROG            :=    200h            ; start address program

;********************
;*STACK Declaration*
;********************

SSTACK                :=    ( BKE * 16 ) - 1; System stack address group D C
USTACK                :=    ( BKC * 16 ) - 1; User stack address group B
```

APPENDIX C: CAPTURE 0 CHANNEL SYNCHRONIZED BY COMPARE 0 CHANNEL (Continued)

```
;*****************************************
;*Declaration of the interrupt vectors table*
;*****************************************

        .text                                   ; start of program

                .org  CORE_IT_VECT              ; Core interrupt vector
                                                ; *********************

                .word RESET_START               ; power on interrupt vector
                .word DIV0                      ; divided by 0 interrupt vector
                .word TOP_LEVEL_IT              ; Top level interrupt vector

                .org  T1_IT_VECT                ; Timer 1 interrupt vectors
                                                ; ************************

                .org  T1_IT_VECT + COMP_IT_VECT ; unused addresses
                .word COMPARE0                  ; Timer 1 compare 0 interrupt

;*******************************
;* Output data table on Port 5 *
;*******************************

                .org  100h

DMA_TABLE_OUT:  .byte 00h, 01h, 02h, 04h, 08h
                .byte 10h, 20h, 40h, 80h, 00h           ; Output DMA table

;***********************************
;* Compare 0 channel DMA time table *
;***********************************

DMA_TIME_TABLE: .word 1000h, 2000h, 3000h, 4000h, 5000h
                .word 6000h, 7000h, 8000h, 9000h, 1000h     ; Compare 0 Time Table

;**********************
;*Start of main module*
;**********************

                .org  START_PROG                ; Start of code
RESET_START:
        ld      MODER,#11100000b                ; CLOCK MODE REGISTER
                                                ; internal stack
                                                ; no prescaling
                                                ; external clock divided by 2

        ld      CICR,#10001111b                 ; CENTRAL INTERRUPT CONTROL REGISTER
                                                ; priority level = 7
                                                ; Nested Arbitration mode
                                                ; disable interrupt
                                                ; enable counters
                                                ; At reset, Global Counter Enable
                                                ; bit is active.

        ld      SSPLR,#SSTACK + 1               ; load system stack pointer
        ld      USPLR,#USTACK + 1               ; load user stack pointer

        call    INIT_IO                         ; Port 5 init. DMA mode

        call    TIMER_1                         ; Timer 1 initialization in DMA mode

        START_T1                                ; Start Timer 1

        ei                                      ; enable all interrupts

;MAIN PROGRAMM
        loop    {
;               wfi
        }
; END OF MAIN PROGRAM
```

APPENDIX C: CAPTURE 0 CHANNEL SYNCHRONIZED BY COMPARE 0 CHANNEL (Continued)

```
;******************
;initialize TIMER 1
;******************

proc    TIMER_1 {

        srp     #BK_F                                   ; select working register
        spp     #T1D_PG                                 ; select timer 1 register page

        ld      t_tcr,#udc                              ; count up

        ld      t_tmr,#( oe0 | rm0 )                    ; Disable output B
                                                        ; Enable output A
                                                        ; Internal clock
                                                        ; Countinuous mode
                                                        ; Capture on REG0

        ld      t_icr,#( ab_ti | exa_rf )               ; T1INA trigger, T1INB I/O
                                                        ; T1INA rising and falling edge sensitive

        clr     t_prsr                                  ; No prescaling
        ld      t_oacr,#( ou_nop | c1_nop | c0_tog ); Toggle on Compare 0 event on T1OUTA
        ld      t_obcr,#( ou_nop | c1_nop | c0_nop ); No action on COMPARE 0 on T1OUTB

        clr     t_flagr

        spp     #T1C_PG                                 ; Timer 1 Control page register
        ld      t1_dcpr,#LG_DMA_BR                      ; DMA counter register base addres
        ld      t1_dapr,#AD_DMA_BR                      ; DMA address register base addres

        ld      t1_ivr,#T1_IT_VECT

        ld      t1_idcr,#( T1_LEVEL | dcts )            ; DMA capture transfer source
        ldw     CPT_LG_DMA,#LG_DMA                      ; Capture DMA counter init.
        ldw     CPT_AD_DMA,#DMA_TABLE_OUT               ; Capture DMA address init.
        ldw     CMP_LG_DMA,#( LG_DMA * 2 )              ; Compare DMA counter init.
        ldw     CMP_AD_DMA,#DMA_TIME_TABLE              ; Compare DMA address init.

        spp     #T1D_PG                                 ; Timer 1 Data page
        ldw     t_cmp0r,#0                              ; Clear Compare 0 register
        ld      t_idmr,#( gtien | cm0i | cm0d | cp0d ); DMA on Capture 0
                                                        ; Compare 0 INT and DMA

}

;*****************************************************************************
;                    COMPARE 0 INTERRUPT ROUTINE
;                    DMA Interrupt End of block
;*****************************************************************************

COMPARE0:

begin   [ PPR, RP0R, RP1R ] {                           ; save page pointer
                                                        ; save register pointer pair

        spp     #T1D_PG                                 ; Timer 1 data register page
        srp     #BK_F                                   ; select group F

        and     t_flagr,#~( cm0 | ocm0 | cp0 | ocp0 ); Reset successful Compare 0
                                                        ; Reset overrun on Compare 0
                                                        ; Reset successful Capture 0
                                                        ; Reset overrun on Capture 0
        or      t_tcr,#ccl                              ; Counter clear
```

APPENDIX C: CAPTURE 0 CHANNEL SYNCHRONIZED BY COMPARE 0 CHANNEL (Continued)

```
        ldw     CPT_LG_DMA,#LG_DMA-1
        ldw     CPT_AD_DMA,#DMA_TABLE_OUT+1
        ldw     CMP_LG_DMA,#( (LG_DMA-1) * 2 )
        ldw     CMP_AD_DMA,#DMA_TIME_TABLE+2
        or      t_idmr,#( cp0d | cm0d )                 ; restart DMA capture 0 channel
                                                        ; restart DMA compare 0 channel

}

        iret                                            ; return from interrupt

;*****************************************************************************
;                       I/O port initialization

proc    INIT_IO [ PPR ]     {
;programmation Port 5 in OUTPUT in DMA mode

        spp     #P5C_PG                                 ; Port 5 control register page
                                                        ; Port 5 in DMA mode
                                                        ; Port 5 Handshake disbled
                                                        ; DMA on Capture 0 channel
                                                        ; DMA direction = output

;                       76543210

        ld      P5C0R,#00000000b

        ld      P5C1R,#11111111b

        ld      P5C2R,#00000000b

        ld      P5DR,#0

        ld      HDC5R,#( hsdis | den | ddw | dcp0 )

;.........................end init P5

; programmation P3.4 ( T1INA ) in INPUT, TRISTATE, TTL
; programmation P3.5 ( T1OUTA ) in ALTERNATE FUNCTION, PUSH-PULL, TTL

        spp     #P3C_PG
        ld      P3C0R,#00110000b
        ld      P3C1R,#00100000b
        ld      P3C2R,#00010000b

}

;*****************************************************************************
;               SECTION CODE FOR THE CORE INTERRUPT ROUTINE
;*****************************************************************************

;-----------------------------------------------------------------------------
;               INTERRUPT ROUTINE FOR ZERO DIVISION
;-----------------------------------------------------------------------------

DIV0:

        jx      DIV0                                    ; debug loop

        ret

;-----------------------------------------------------------------------------
;               INTERRUPT ROUTINE FOR TOP_LEVEL_IT
;-----------------------------------------------------------------------------

TOP_LEVEL_IT:

        jx      TOP_LEVEL_IT                            ; debug loop
        iret
```

SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

STACK OVERFLOW DETECTION USING THE ST9 TIMER/WATCHDOG

Pierre Guillemin

INTRODUCTION

In real time applications, the implemention of software protection is not always easy, but allows reaching a high security level for the software against malfunction. This is particulary true for in-board applications in disturbed environments, such as automotive, power meter or industrial applications.

To help avoid non-controlled functionality and damage to real time system due to possible perturbations on the ST9 microcontoller core and I/O ports, a special peripheral able to act as a watchdog is available on all ST9 family members: the Timer Watchdog.

A periodic restarting of the Timer Watchdog by program, associated with the automatic detection of possible stack overflow, add to the protection of real time application software.

This application note shows how to detect stack overflow by using the Timer Watchdog in watchdog mode.

STACK OVERFLOW DETECTION PRINCIPLE

Summary of Timer Watchdog Features

The ST9 core include a 16-bit down counter with an 8-bit prescaler offering the possibility of a watchdog mode. This timer, driven by a clock equal to INTCLK divided by 4, is able to provide time periods within the range of 333ns to 5.59s (using a 12 MHz internal clock).

In watchdog mode, the Timer Watchdog generates a fixed time base according to the Timer Watchdog registers and prescaler, and to INTCLK. This time base can be modified on the fly by changing the prescaler value. The new value will be taken into account only after an End Of Count event. In watchdog mode, the End Of Count occurence generates a system reset.

In order to prevent the reset, the byte sequence AAh, 55h should be written into the Timer Watchdog register Low. Once the writing of 55h has been performed, the timer reloads the prescaler register and the counting restarts from this value (the prescaler register value may be modified between two End Of Count events).

Note 1. For a better understanding of this application note; please refer to the ST9 Technical Manual chapter on the 16-bit programmable Timer/Watchdog.

Note 2. INTCLK: Internal Clock. This clock issued from the oscillator circuitry, divided or not by 2, is the ST9 Internal Clock driving the peripherals. The maximum frequency allowed for INTCLK is 12MHz.

Stack Overflow Detection

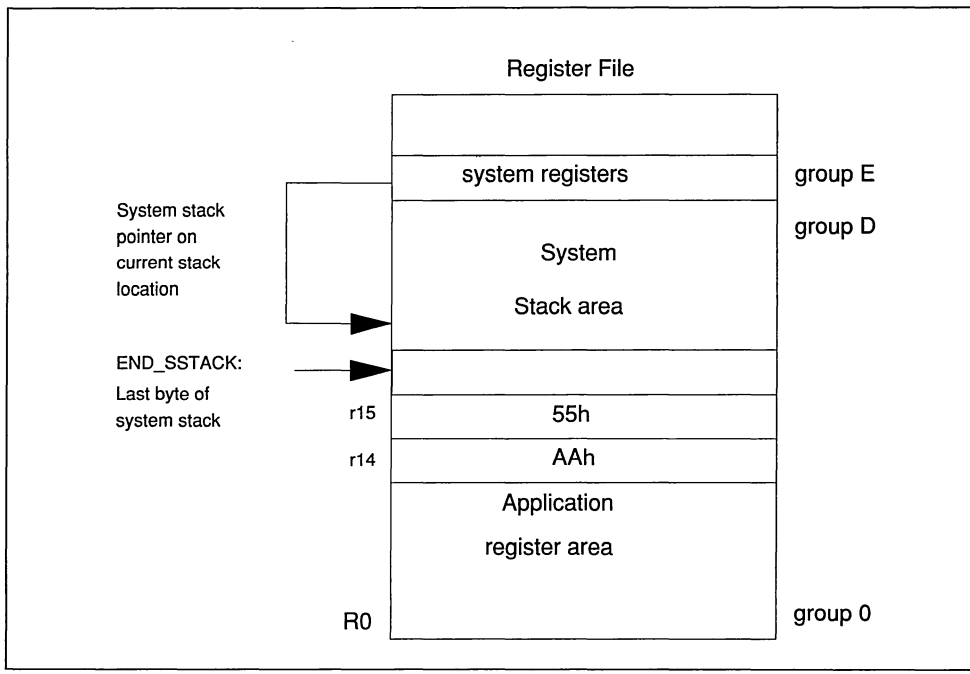
In many software applications, for example when running on ST9 ROMLESS versions or without external memory space, the size of the stack is limited.

On ST9 devices, the system stack may be located in the Register File or in data memory space. The ST9 stack pointer moves from the top to the bottom of the stack area.

A solution to detect stack overflow is to reserve the first two bytes after the bottom of the stack and to store in these locations the Timer Watchdog restart value, ie AAh, 55h.

In the case of stack overflow, the data will be overwritten and thus destroyed and a system reset will be generated on the next Timer Watchdog End Of Count.

Figure 1. Example of Stack overflow detection in Register File



SOFTWARE DESCRIPTION

Stack Initialization

The following example initializes the system stack in groups D and C of the Register File.

In the stack management of the ST9, the stack pointer is automatically pre-decremented before the data is stored on the stack. So the expression:

```
SSTACK = (BKE * 16) - 1
```

defines the first location of the system stack in group D and C within the Register File, while the instruction:

```
ld SSPLR,#SSTACK + 1
```

initializes the system stack pointer in the system register. The instruction:

```
ldw RR#END_SSTACK,#0AA55h
```

initializes the first two bytes following the bottom of the system stack with the value used to restart the Timer Watchdog.

Figure 2. System stack initialisation

```
;*****************************************************************
;          STACK Declaration and end of stack initialisation
;                  in RAM space or Register File
;*****************************************************************

; Initialisation in Register FIle

SSTACK       := (BKE * 16) - 1   ; Sys.stack add.group
LG-SSTACK    := 32               ; Sys.stack length
END_SSTACK   := (BKE * 16) - LG_SSTACK ; Last sys.stack byte

ld     SSPLR,#SSTACK + 1         ; Load sys.stack pointer
ldw    RR#END_STACK - 2,#0AA55h  ; Init end of stack.

; Initialisation in RAM space

SSTACK       := 2000h            ; top of sys.stack
END-STACK    := 1000h            ; Init end of stack
essp          = rr0

sdm
ldw    SSPR,#SSTACK                ; Select data space
ld     essp,#END_SSTACK          ; Init End of sys.stack
ldw    -2(essp),#0AA55h
```

Timer Watchdog Programmation

As an example, the Timer Watchdog is initialized in order to provide a time base of 10ms (with a ST9 driven by a clock frequency of 24MHz internally divided by two). To enable the Watchdog mode, the requirement is to initialize Timer prescaler and counter, to initialize the Timer Watchdog Control Register with its reset value, and then to enable the watchdog mode by clearing the WDGEN bit in the Wait Control Register in page 0. Resetting this bit causes the counter to start in Watchdog mode regardless of the start/stop, Single/Continuous and input mode bits.

Figure 3. Timer/Watchdog Initialisation

```
;*****************************************************************
;                    WATCHDOG INITIALISATION
;*****************************************************************

proc   INIT_WGT[PPR] {

       spp   #0
       ld    WDTPR,#0            ; TWD prescaler register
       ld    WDTLR,#-30h    ;    ; TWD Timer counter low
       ld    WDTHR,#075h    ;    ; TWD Timer counter high
       }

       call  INIT_WGT            ; call TWD initialisation
       spp   #0            ;     ; select page 0 register
       ld    WCR,#00111111b;     ; Enable the Watchdog
       ei                  ;     ; Enable Interrupt
```

Note 3. A bit (DIV2 located in the MODE Register MODER, R235 in the system group) controls the divide by two circuit which operates on the OSCIN clock driving the ST9. The maximum Internal Clock (INTCLK) allowable for the ST9 is 12MHz. This internal clock drives all the ST9 peripherals, while this same clock, optionally slowed down by the ST9 Core clock programmable prescaler and by wait cycle insertion, drives the ST9 Core.
After a reset cycle, the clock frequency applied to the ST9 is divided by two and no Core clock prescaling is done.

Timer Watchdog Restart

This example shows how to restart the Timer Watchdog when the stack is located in Register File or in RAM space. In the register file, the two instructions:

```
ld WDTLR,#END_SSTACK-2
ld WDTLR,#END_SSTACK-1
```

load the restart value of Timer Watchdog.

When the system stack is located in RAM space, a register essp (end of system stack pointer) must be used to load the sequence AAh, 55h in the Timer Watchdog counter register low.

Figure 4. Restarting the Timer/Watchdog

```
;            In Register File

spp    #0                  ; TWD register page
ld     WDTLR,R#END_SSTACK-2     ; Load AAh
ld     WDTLR,R#END_SSTACK-1     ; Load 55h


;            In RAM space

spp    #0                  ; TWD register page
sdm                        ; Select RAM space
ld     essp,#END_SSTACK         ; End stack pointer
ld     WDTLR,-2(essp)           ; Load AAh
ld     WDTLR,-1(essp)           ; Load 55h
```

SUMMARY

Protection of software against externally generated perturbations can be made by additional test routines. This protection can easily be increased by using the ST9 Timer Watchdog bringing software reliability and security. With the Timer Watchdog the ST9 programmer may control the software execution. Additionally, when restarting the Timer Watchdog from values (AAh, 55h) located at the bottom of the system stack two new securities are added:

- test of the integrity of the Register File or the RAM space
- provision of a system reset in the case of stack overflow.

SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

FREQUENCY DOUBLER DEMONSTRATION SYSTEM

Myriam Chabaud and Alan Dunworth

INTRODUCTION

This Application Note is intended to provide the interested designer of ST9 system applications with a further insight into methods of exploiting the powerful capabilities offered by the ST9 chip in conjunction with the ST9 Technical Manual (Ref. 3). For this purpose we present full software and broad hardware details of a complete system application (Appendices A and B).

BASIC SPECIFICATION OF A FREQUENCY DOUBLER SYSTEM.

An analogue signal (speech signal) is sampled at a fixed rate and digitized sample values are stored in internal RAM. After a delay period, equal to the period of the lowest frequency component in the input signal, the samples are read from RAM at twice the input sampling frequency and converted to analogue form by using Pulse Width modulation techniques coupled with external filtering.

This process is illustrated by the waveforms shown in Figure 1 which show the nature and timing of the output signal resulting from sampling and frequency doubling the given input signal.

USE OF ST9 SYSTEM RESOURCES.

The demonstration application makes use of the following basic ST9 system resources:

- A/D Convertor
- Multifunction Timer/Counter
- Internal RAM (Register File)
- Input/Output Ports

The A/D convertor

One only of the eight available analogue input channels of the ST9 A/D Convertor is used in this application. The input speech signal, band-restricted by filtering to remove frequency components below a lower, or above an upper limit, is fed to the input channel. The A/D Convertor is operated in continuous scanning mode, each conversion being started by an internal On Chip Event signal generated by the Multifunction Timer underflow event.

USE OF ST9 SYSTEM RESOURCES (Continued)

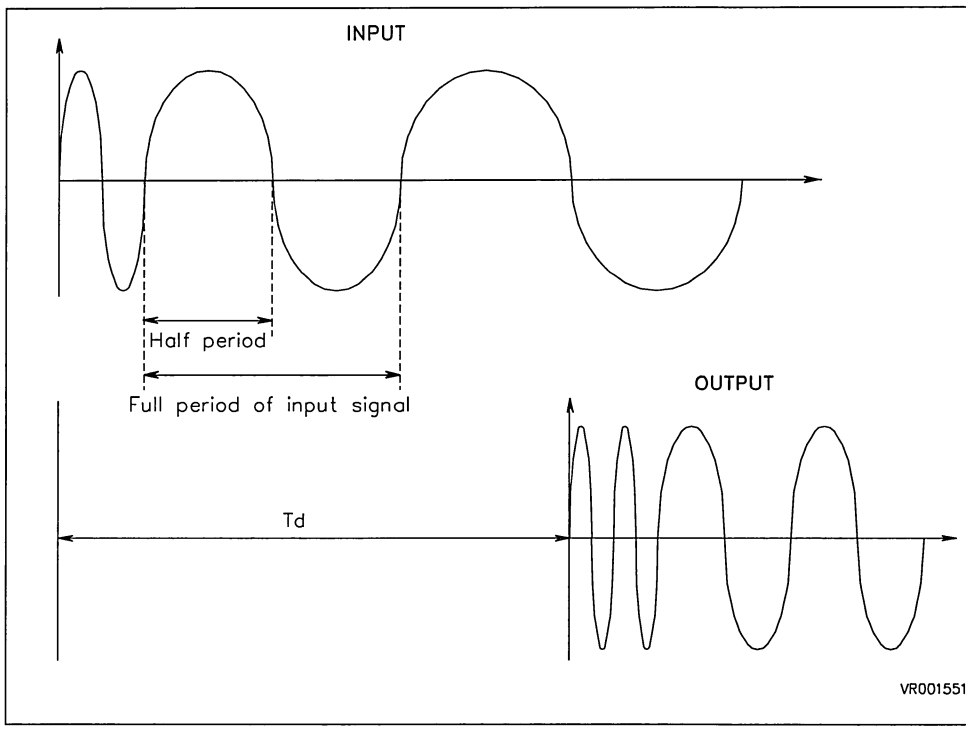
The Multifunction Timer/Counter

The Multifunction Timer is operated in continuous mode with count down from a fixed value of 508, each underflow resulting in the generation of an On-Chip Event signal and reload of the fixed initial counter value (508). The initial count value, prescaler, and clock rate are chosen to give a sampling rate of ~7.8kHz.

Comparison register 1, CMP1, is used to obtain the double sample output rate by successively loading this register with the maximum count value (508), and one half the maximum count value (254). In this way two CMP1 event pulses, and the related sample output, will be obtained for each Timer count-down period, i.e. for each input sampling event.

Comparison register 0, CMP0, is used to obtain the Pulse-Width Modulated output. This is achieved by successively loading this register with a value equal to 508 - S, where S is the sample value, or 254 - S. Timer output OUT1 is then set to a "one" value by an OVF or CMP1 event, i.e. effectively at count values 508 or 254, and reset to a "zero" value by a CMP0 event. This results in an output pulse train in which the output is set to "one" for the period of time it takes for the counter to count down from 508 to (508 - S), or from 254 to (254 - S). In either case the "ON" period of the output pulse is proportional to the sample value, S, (range 0 to 254).

Figure 1. Illustration of Frequency Doubling

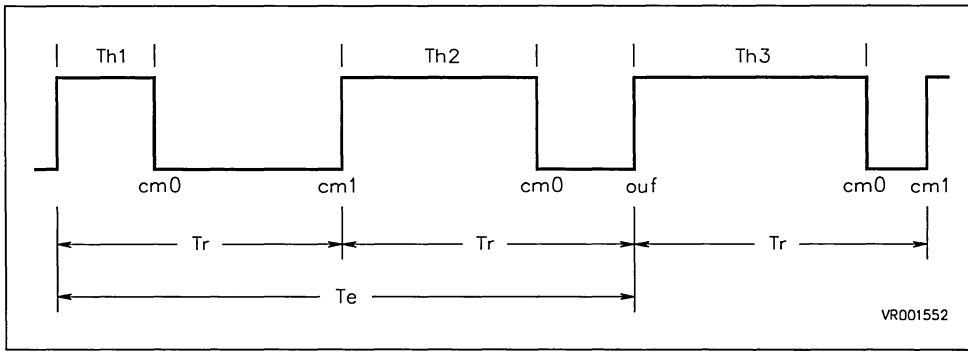


USE OF ST9 SYSTEM RESOURCES (Continued)

The output signal is delayed by an amount T_d relative to the input signal. The minimum input frequency must be no lower than $1/T_d$ since there must be at least three zero-crossings during the period T_d for correct restitution of the double frequency output signal.

Figure 2 shows the Timing control events with T_r, the output sampling rate equal to $T_e/2$, i.e. the input sampling period, T_e, divided by two. In addition, it will be noticed how the sequence of values, T_{hn}, the mark periods of the output, are controlled by the CMP0 values.

Figure 2.Timer Output Controlled by 3 Counter Events



Use of Internal RAM (the Register File).

The ST9030 has 224 bytes of available internal RAM storage, addressable as registers. Two hundred locations have been allocated to the storage of 200 8-bit digitized samples, which leaves 24 register locations for the stack or for storage of temporary values.

The delay of T_d, required before output samples can be read out, is equal to the time required to read and store 100 samples. During this time there must be at least three zero-crossings for the lowest frequency component in the input signal (see Figure 1). For a sampling rate of 7.8kHz this defines the minimum input frequency to be F_{min} ~78Hz.

Use of Port terminals for special input/output.

Two of the Port pins must be used for the analogue signal input and for the PWM output signal. These pins can be initialized for A/D input or Alternate Function Output as described in the initialization section.

ASSOCIATED ANALOGUE CIRCUITRY.

The sampling frequency, F_e, for the input analogue signal is determined by the Timer parameters to be 7.8kHz., which, by Shannon's Theorem implies that the maximun input frequency be limited to no more than half this value, i.e. 3.9kHz. The input circuit must hence include a low-pass filter to avoid any resultant aliasing.

The input frequency must also be limited at the lower frequency end. This arises because the maximum number of samples which can be stored is limited to 200, and furthermore a minimum delay must exist between the input signal and the reconstituted, frequency-doubled output. This minimum delay, equivalent to the period of the lowest frequency component present in the input signal, in conjunction with the sample size limit, implies that the minimum frequency must be no lower than $F_e/100$, i.e. 78Hz.

Effectively, therefore, the input signal must be fed to the A/D sampling input via a Band-Pass filter. In our Application, the filter used had -3db cut-off frequencies of 85Hz and 1.5kHz.

The output signal from the ST9 consists of a PWM pulse train at a frequency of $2xF_e$. The use of a Low Pass filter is hence indicated to recover the required output signal. In our example a high-order Pass-band filter with -3db frequencies of 200Hz and 3kHz was used (Appendix A).

Full details of the design considerations and performance characteristics of the input and output filters do not fall within the scope of this short note.

PROGRAM DETAILS.

The software associated with this design example comprises the following four components:

(i) Initialization of ST9 core and on-chip peripherals.

(ii) The Main Program.

(iii) The Interrupt routine controlling the input sampling and storage of data (`SOUND_IN`).

(iv) The Interrupt routine controlling the output of frequency-doubled samples (`SOUND_OUT`).

Core Initialization.

The ST9 is initialized in the following manner:

(i) The User and System Stacks are set up in Internal RAM,

(ii) The Internal Clock frequency is set to 12MHz,

(iii) The Priority level of the RESET interrupt is set to 7

Initialization of the Input/Output Ports.

Two Input/Outputs only are required, corresponding to the Analog input and output. The corresponding Port pins are initialized either as an A/D input or as an Alternate Function output, i.e. they are linked to ST9 internal signals, as follows:

(i) T0OUTB: Output B from Timer 0,

This output was chosen as it can be activated by the CMP0 event of Timer 0

(ii) AIN7: Input No. 7 of the A/D Convertor

Note: only the one (out of 8 possible) analogue entries is used.

PROGRAM DETAILS (Continued)

Initialization of the A/D Convertor.

* the software activation bit is set to disable,
* an A/D Conversion is triggered by an OCE (On-Chip Event),
* the Interrupt Vector Address is set up at Table Address 20h in EPROM,
* the Analog Watch-Dog Interrupt is disabled,
* the Interrupt generated by "End of Conversion" is validated.

Initialization of the Timer/Counter.

Timer 0 is initialized as follows:

* Count-down mode is selected,
* Continuous Sampling Mode is enabled,
* the internal Clock is selected (4MHz).
* Output, OUTB is:
  - . initialized to 1
  - . Set to 0 by the CMP0 event,
  - . Set to 1 by the CMP1 and OVF events,
* the OCE signal is generated on Counter Underflow,
* the Vector Interrupt Address is specified as 30h in EPROM
* the Interrupt Priority level is set to 6.

MAIN PROGRAM

The Main program, (see Appendix B for a full listing), is automatically entered on System Reset since the address, 38h, has been loaded in the Interrupt Vector Table at locations 0 and 1. Program Main first initializes the Clock, stacks, Multifunction Timer 0, A/D Convertor, and Ports, using the sub-routine `periph_init`. The RAM table pointer is initialized together with pointers and counters which are used to record the number of input waveform zero-crossings, and the number of times the output waveform has been repeated (N.B. there are two repeated output periods for each complete input period).

At this point the `Timer 0 start` Macro is executed which causes the counter to start counting down towards zero from an initial value of 508. Each time the counter clears to zero an On-Chip Event signal will be generated internally in the ST9 chip which will initiate the next A/D input sample conversion.

The main program loops around label "`here`" and the two following statements until such time as 100 input samples have been acquired, as indicated by equality between `cpt_in`, the input sample counter and `#ptr_moy` (64 Hex). The main program then proceeds to set bit 1 in working register 10 (myflags.1 is equivalent to the `start_out` flag in Figure 3).

From this point on the Main program loops around the statements following "`there`" until such time as a System Reset is applied. Each Timer 0 OCE pulse initiates an input sample conversion, and each successful CMP0 comparison event initiates an output sample, these two operations being effected by the Interrupt subroutines `SOUND_IN` and `SOUND_OUT`, respectively.

The overall working of the Main program may be readily visualized by reference to Figure 3.

THE SOUND_IN INTERRUPT ROUTINE.

The organization of the SOUND_IN routine is illustrated by the flow-diagram of Figure 4, and the program details are shown in Appendix B.

This routine is entered whenever the A/D Convertor raises an End of Conversion Interrupt, and will thus occur shortly after the OCE pulse produced by Timer 0 counting down to zero.

After saving the current CPU context (working register pointer and page registers) and selecting the A/D system register page, this routine loads the current input sample into RAM, resetting the RAM table pointer if the end of Table has been reached.

The routine then resets the Timer interrupt pending flags and exits after restoring the CPU context.

Figure 3. Flow Chart of the Main Program

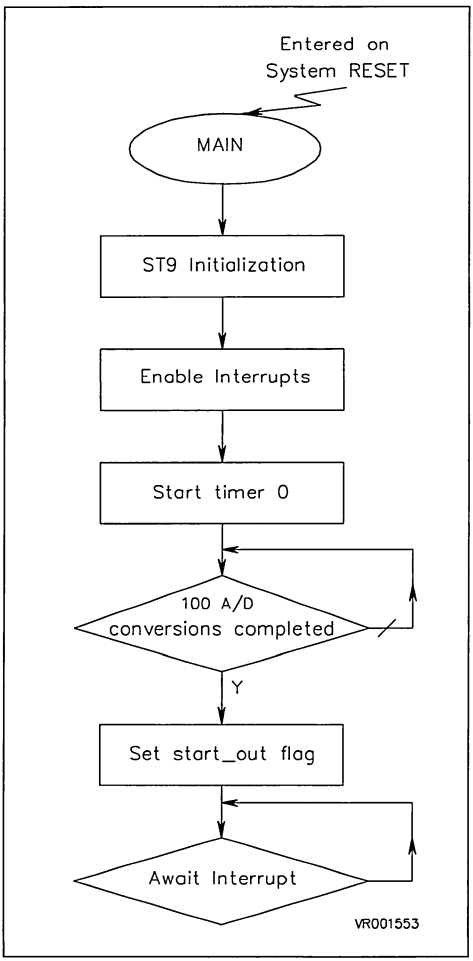
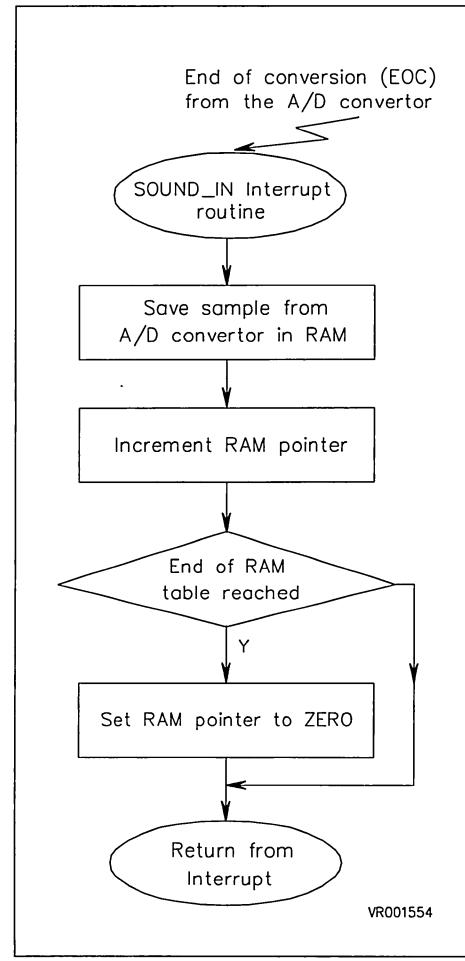


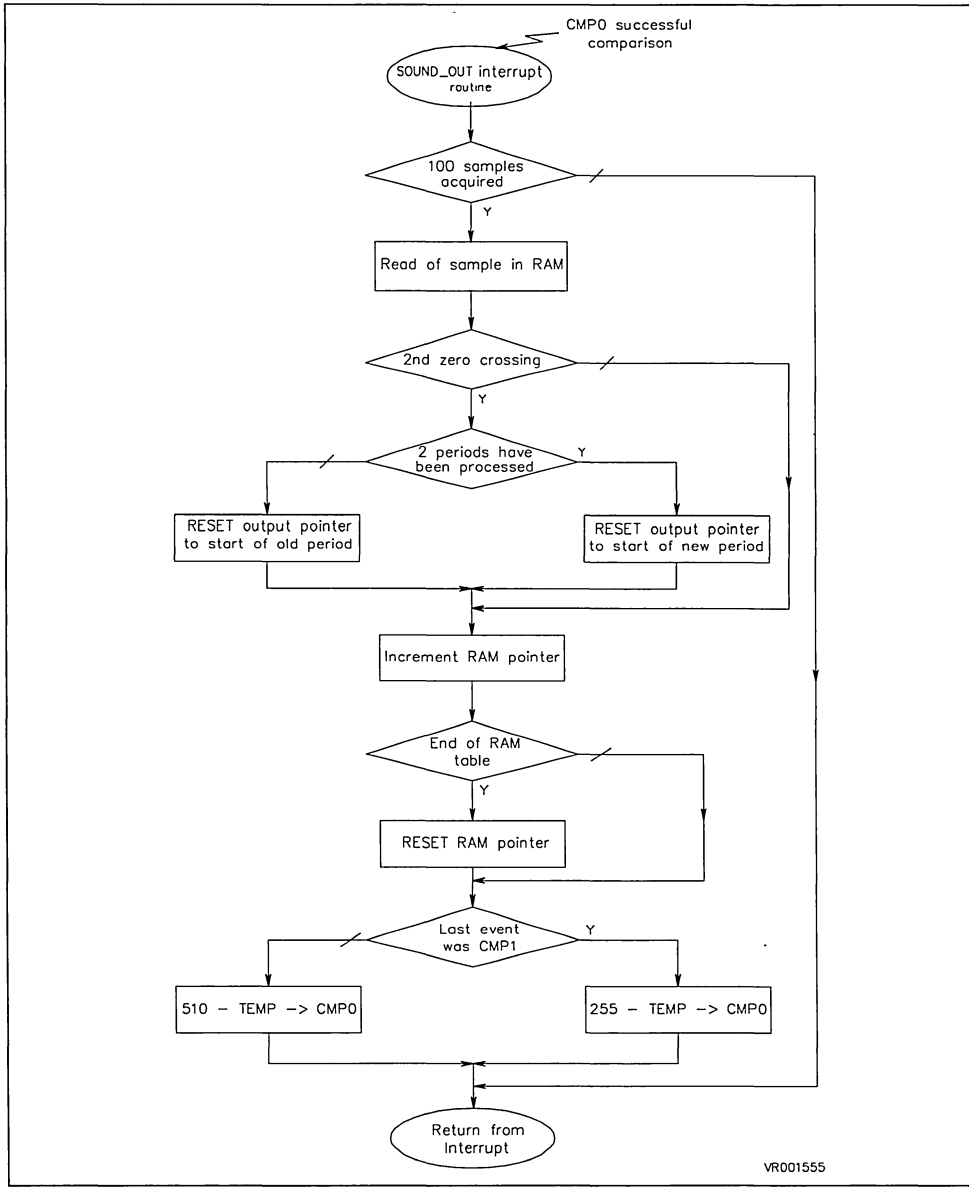
Figure 4. Flow Chart of the SOUND_IN Routine



THE SOUND_OUT INTERRUPT ROUTINE.

The organization of the SOUND_OUT routine is illustrated by the flow-diagram of Figure 5, and the program details are shown in Appendix B.

Figure 5. Flow Chart of the SOUND_OUT Routine



THE SOUND_OUT INTERRUPT ROUTINE (Continued)

This Interrupt routine is entered after each successful CMP0 comparison, i.e. at the conclusion of the "ON" period of the output pulse train (refer to Figure 2).

After saving the current context (working register pointer and page registers) and specifying a set of current working registers, this routine tests to see whether at least 100 samples have been accumulated since the last System Reset. If this is not the case an immediate exit is made by means of a branch to `end_out`. Otherwise the routine proceeds by copying the current output sample value into working registers r8 and r9, These registers comprise respectively the higher and lower components, `tamph` (normally zero) and `tampl`, of the 16-bit extended version (`tampw`) of the basic 8-bit sampled input value.

The next step is to establish whether a zero-crossing has occurred. This is established if the "zero" value (actually "zero" equals the mid-range value of 128) lies between the values of the current and the previous sample values. If a zero-crossing has occurred a further test is made to establish whether this is the first or second such zero-crossing, i.e. whether we have encountered the end of an input sample half- or full-period (refer to Figure 1). If the end of a complete period has been reached a further test is made to see if this complete period has been outputted once, in which case a further copy of the same period is required, or twice, in which case we can proceed to the next input period. The appropriate counter values are updated and a branch is made to `next_sample`.

At this point the sample value is saved for the next zero-crossing test, and the value of the register, CMP1 is loaded with 508 or 254 as appropriate.

Register CMP0 is then loaded with a value of 508 - S, or 254 - S, where S is the current sample. The appropriate choice for the CMP1 and CMP0 values is made on the basis of whether the previous CMP1 value was 508 or 254. In the former case (508) CMP1 is loaded with 254, and CMP0 with 254 - S. In the latter case (254) CMP1 is loaded with 508 and CMP0 is loaded with 508 - S.

Finally the Timer 0 pending event flags are reset, the CPU contest is restored, and exit made by the `IRET` instruction.

REFERENCES

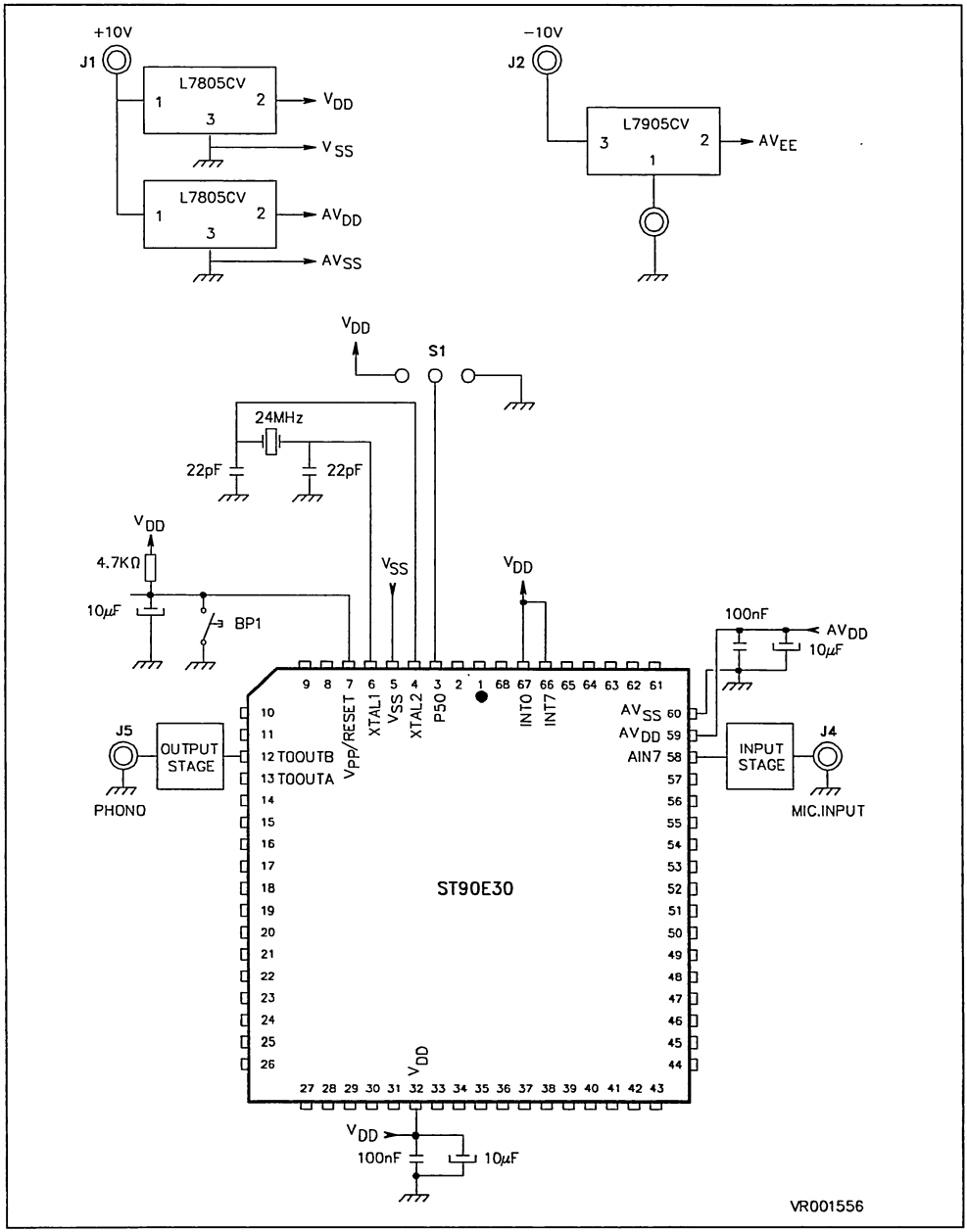
(1) Application Note AN413, "Initialization of the ST9",
Pierre Guillemin and Alan Dunworth, SGS-THOMSON, Rousset.

(2) "ST9 Technical Manual",
SGS-THOMSON Microelectronics.

(3) Application Note AN411, "SYMBOLS.INC, ST9 Register Address and Content Names",
Pierre Guillemin, SGS-THOMSON Microelectronics.

APPENDIX A. CIRCUIT SCHEMATICS

Figure 6 . Frequency Doubler Demonstration System Overview



APPENDIX A. CIRCUIT SCHEMATICS (Continued)

Figure 7. Input Stage Overview

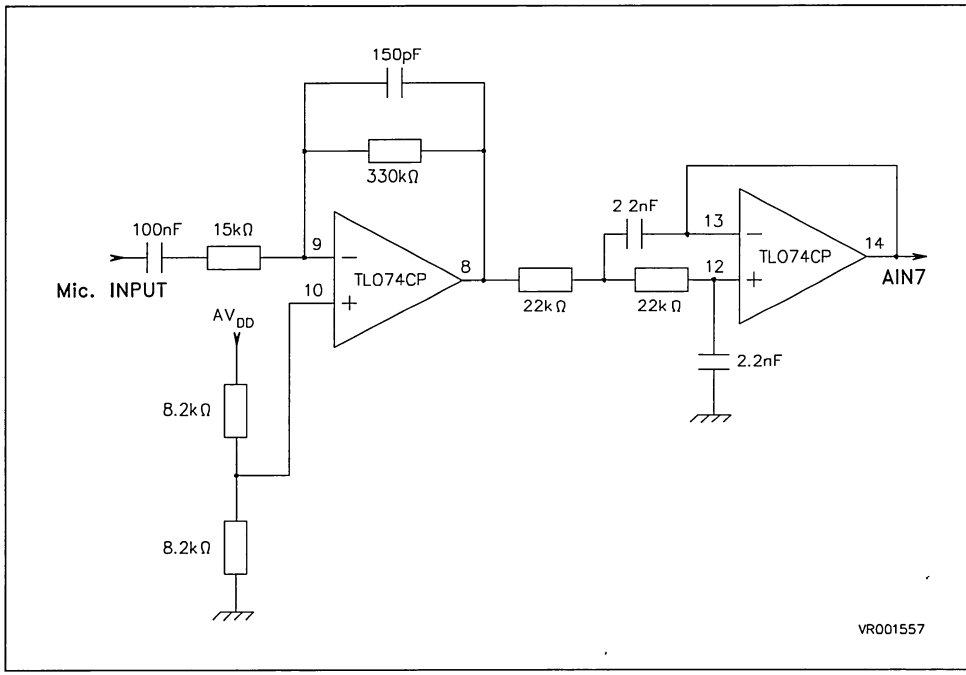
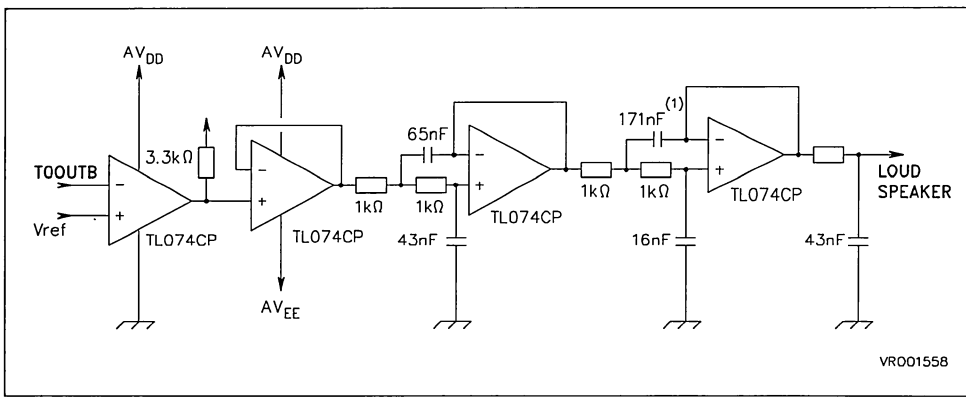


Figure 8. Output Stage



Note 1: This value must be generated

APPENDIX B. PROGRAM LISTING (for ST9030)

```
;*****************************************************************************
;
;          ST90E30  DEMOBOARD
;                DEMO.ST9
;*****************************************************************************
; REGISTER DEFINITIONS
;*********************

memo_sample      =     r15   ; R0DFh  To detect a zero crossing.
pos_zc           =     r14   ; R0DEh  The first sample location of a period.
cpt_zc           =     r13   ; R0DDh  Zero Crossing counter .
cpt_out          =     r12   ; R0DCh  To scan table in output.
cpt_in           =     r11   ; R0DBh  To scan table in input.
myflags          =     r10   ; R0DAh  My 8 flags.
tampl            =     r9    ; R0D9h
tamph            =     r8    ; R0D8h
tampw            =     rr8   ; R0D8h-R0D9h.

tamp_sub         =     rr6   ; R0D6h-R0D7h
cpt_repeat       =     r5    ; Period repeat counter.

sys_stack        =     0D4h  ; !!! system stack is limited by bank 0 registers.

;                R0C7h       ; The end of the system stack.

;*********************
; CONSTANTS DEFINITION
;*********************

work_reg_page0   =     (0Dh*2)
work_reg_page1   =     (0Dh*2)+1

max_count        =     508   ; The maximum count of timer 0.
mid_count        =     (max_count/2)

zc_value         =     07Fh  ; To detect a zero crossing.

;                      R0C8h   The last register of table.
table            =     00h   ; The first register of table.
```

APPENDIX B. PROGRAM LISTING (Continued)

```
ptr_moy          =     100   ; The middle of the table.

ptr_max          =     200   ; Table contains 200 registers.
                             ; Input signal frequency min = 100 Hz .

P_debug          =     R229  ; Port 5 is used for debug.

ad_vect          =     020h  ; Start of A/D vector table.

t0_vect          =     030h  ; Start of Timer 0 vector table.
t0_cmp_vect      =     036h  ; Timer 0 compare event interrupts.


; Flags of MYFLAGS register.
;---------------------------
start_out        =     1     ; 1 => sound output can begin with the 100
                             ; first samples in table.
mask_start_out   =     (1<-start_out)


;******************
; MACROS DEFINITION
;******************

.macro t0start
     spp  #T0D_PG
     or   T_IDMR,#gtien      ; T0 Global interrupt mask disabled.
     or   T_TCR,#cen         ; Counter enabled.
.endm

.macro t0stop
     spp  #T0D_PG
     and  T_TCR,#~cen        ; Counter disabled.
     and  T_IDMR,#~gtiem     ; T0 Global interrupt mask enabled.
.endm
```

APPENDIX B. PROGRAM LISTING (Continued)

```
*******************
; INTERRUPT VECTORS
;*******************

power_on::
     .text

     .word       main         ; RESET vector.
     .word       main         ; Divide by 0 vector.

     .org ad_vect
     .word       main         ; Analog Watchdog Request not used.
     .word       sound_in     ; A/D End Of Conversion Request.

     .org t0_cmp_vect
     .word       sound_out    ; Compare 0 of timer 0 request.

;*************
; MAIN PROGRAM
;*************

main::

     ld   MODER,#11100000b    ; Ext clock prescale by 2.
                              ; Internal system and user stacks.
     ldw  SSPR,#sys_stack     ; System stack pointer.
     spm                      ; Data memory selected.

     spp  #WDT_PG
     ld   wcr,#wden           ; Watch dog mode disabled, no wait states.
     ld   EIMR,#0             ; Mask all channels interrupts.
                              ; At reset, Global Counter Enable bit is active.

     srp0 #work_reg_page0
     srp1 #work_reg_page1

     call periph_init         ; Initialization of ports, timer0, ADC.

     clr  cpt_in              ; To input samples in table.
```

APPENDIX B. PROGRAM LISTING (Continued)

```
     clr  cpt_out              ; To output samples from table.

     ld   cpt_repeat,#2        ; Period repeat counter.
     ldw  tampw,#381           ; The first value.
     clr  memo_sample
     clr  myflags

     ei

     t0start                   ; T0 ,in PWM mode, will effect D/A conversion.

here::
     cp   cpt_in,#ptr_moy
     jrne here

     bset myflags.1            ; To permit output of the 100 first samples.

there::
     wfi
     jr   there

;****************
; INITIALIZATIONS
;****************

periph_init::
; ----- T0_OUTB.

     spp  #P3C_PG
     ld   P3C0R,#00001000b     ; port3: b3:af,pp,ttl.
     ld   P3C1R,#0ffh          ; others:out,pp,ttl
     ld   P3C2R,#0

; ----- AIN7 input.

     spp  #P4C_PG
     ld   P4C0R,#10000000b     ; port4: b7:af,od,ttl.
     ld   P4C1R,#11111111b     ; others:out,pp,ttl
     ld   P4C2R,#10000000b
```

APPENDIX B. PROGRAM LISTING (Continued)

```
; tim0 init ****************************************
;         d/a 8 bit in pwm
;
;         outB is preset to 1.
;                set by OUF and CMP1.
;                reset by CMP0.
;
;         prescale=0,continuous mode.
;         On Chip Event (OCE) generated by OUF.
;         sound_out interrupt generated by CMP0.
;**************************************************

init_t0::

     spp  #T0D_PG              ; T0 data page
                               ; (Xtal clock / 6) => 4 MHz (at reset).
     srp  #(15*2)              ; To access bank F with "r" addressing mode.

     ldw  t_reg0r,#max_count   ; 508 counts at 4MHz => 127 micro sec. (7.8 kHz).
     ldw  t_cmp1r,#max_count   ; CMP1 value will change for
                               ; 4 times between 2 acquisitions.
     ldw  t_cmp0r,#(381)       ; 508 - 127.
     ld   t_tcr,#00000000b     ; Timer 0 stops, software down count.
     ld   t_tmr,#10000000b     ; OUTB enabled, OUTA disabled.
                               ; Retrigger mode enabled.
                               ; Continuous mode selected.
     ld   t_icr,#0             ; Nop on inputs.
     ld   t_oacr,#11111101b    ; OUTA is disabled.
     ld   t_obcr,#10000011b    ; OUTB is preset to 1,reset by cmp0,
                               ; OUTB is set by ouf and cmp1.
                               ; OCE (a single pulse) is generated by ouf.
                               ; Flags register is cleared at reset.

     ld   t_idmr,#00000100b    ; Only cmp0 interrupt will be enabled.

     spp  #T0C_PG              ; Timer 0 control page.

     ld   t0_ivr,#t0_vect      ; Timer 0 interrupt vector table.
     ld   t0_idcr,#0C6h        ; Priority level 6.
```

APPENDIX B. PROGRAM LISTING (for ST9030) (Continued)

```
; A/D converter ************************************************************
;         speech or sound input
;
;         Start Conversion is triggered by On Chip Event signal (OCE).
;         ADC frequency = 7.8 kHz.
;
;         Continuous scanning channel 7.
;         Interrupt on End Of Conversion (EOC).
;*****************************************************************************

init_ad::

     spp  #AD0_PG
     ld   AD_CLR,#11001100b  ; OCE starts conversion (single mode).
                             ; Power up, only the channel 7 is converted.
     ld   AD_IVR,#ad_vect    ; Vector pointing the A/D int. routine starting
                             ; address.
     ld   AD_ICR,#00100110b  ; Enables the End Of Conversion interrupt request.
                             ; Masks the Analog Watchdog interrupt request.
                             ; Priority level 6.
     ret
```

APPENDIX B. PROGRAM LISTING (Continued)

```
; ************************************************
; Sound acquisition routine.
; Called by A/D End Of Conversion.
; A/D EOC occurs every 127 µs (Fin = 7.8 kHz)
; digital value -> table(cpt_in),
; ************************************************

sound_in::

     pushw RPP                ; Register pointer pair.
     push  PPR                ; Page pointer register.

     srp0  #work_reg_page0    ; Selects the Working registers bank 0.
     srp1 #work_reg_page1     ; Selects the Working registers bank 1.

     spp  #AD0_PG             ; ADC page.
     ld   tampl,AD_D6R        ; Load sample (from A/D data register)
                              ; in RAM table.

     ld   table(cpt_in),tampl
     inc  cpt_in

     cp   cpt_in,#ptr_max
     jrne skip_cpt_in

     clr  cpt_in              ; When the end of the table is encountered.

skip_cpt_in::
     and  AD_ICR,#~(awd+ecv)
                              ; Reset of the request flags.
     pop  PPR
     popw RPP
     iret
```

APPENDIX B. PROGRAM LISTING (Continued)

```
;***************************************************************************
; sound generation routine.
; called by cmp0 of timer 0.
; this routine will be executed every 64 micro sec.
; cmp0 register of timer 0 is here initialized for D/A conversion in PWM mode.
; ************************************************************************
sound_out::

     pushw RPP              ;  Register pointer pair.
     push  PPR              ;  Page pointer register.

     spp  #T0D_PG

     srp0 #work_reg_page0
     srp1 #work_reg_page1

     tcm  myflags,#mask_start_out
     jrne end_out

     clr  tamph
     ld   tampl,table(cpt_out); Load sample value from table.

; -------- test if zero crossing.
     cp   memo_sample,#zc_value
     jrmi small             ; If memo_sample < zc_value.

     cp   tampl,#zc_value
     jrpl end_zc            ; If tamp and memo_sample > zc_value.
     jr   end_test_zc       ; If tamp > zc_value and memo_sample < zc_value.

small::
     cp   tampl,#zc_value
     jrmi end_zc            ; If tamp and memo_sample < zc_value.

; -------- there is a zero_crossing.
end_test_zc::
     djnz cpt_zc,end_zc     ; Test if it's the second zero crossing.

     ld   cpt_zc,#2         ; To count again 2 zero crossing.
```

APPENDIX B. PROGRAM LISTING (Continued)

```
     djnz cpt_repeat,again    ; Period repeat counter.
     ld   cpt_repeat,#2
     ld   pos_zc,cpt_out      ; Save the zero crossing position.
     jr   end_zc

again::
     ld   cpt_out,pos_zc      ; Load the zc position to repeat a period.

     ; -------- zero crossing has been treated.
end_zc::

     inc   cpt_out

test_cpt_out::
     cp   cpt_out,#ptr_max
     jrne next_sample

     clr  cpt_out

next_sample::
     ld   memo_sample,tampl   ; Save the sample for the next zero crossing test.

     subw T_CMP1R,#mid_count ; The next value to set OUT0B.
     jrne init_cmp0
     ldw  T_CMP1R,#max_count

init_cmp0::
     ldw  tamp_sub,T_CMP1R
     subw tamp_sub,tampw
     ldw  T_CMP0R,tamp_sub    ; The next value to reset OUT0B.

end_out::
     clr  T_FLAGR             ; Resets the timer0 event flags.

     pop  PPR
     popw RPP
     iret
```

EUROPE

DENMARK

2730 HERLEV
Herlev Torv, 4
Tel. (45-44) 94 85 33
Telex 35411
Telefax (45-44) 948694

FINLAND

LOHJA SF-08150
Ratakatu, 26
Tel (358-12) 155 11
Telefax (358-12) 155 66

FRANCE

94253 GENTILLY Cedex
7 - avenue Gallieni - BP 93
Tel.. (33-1) 47.40 75.75
Telex 632570 STMHQ
Telefax (33-1) 47 40 79 10

67000 STRASBOURG
20, Place des Halles
Tel (33-88) 75 50 66
Telefax (33-88) 22 29 32

GERMANY

8011 GRASBRUNN
Bretonischer Ring 4
Postfach 1122
Tel · (49-89) 460060
Telefax (49-89) 4605454
Teletex 897107=STDISTR

6000 FRANKFURT
Gutleutstrasse 322
Tel (49-69) 237492-3
Telefax (49-69) 231957
Teletex 6997689=STVBF

3000 HANNOVER 51
Rotenburger Strasse 28A
Tel (49-511) 615960-3
Teletex 5118418 CSFBEH
Telefax (49-511) 6151243

8500 NÜRNBERG 20
Erlenstegenstrasse, 72
Tel · (49-911) 59893-0
Telefax (49-911) 5980701

7000 STUTTGART 31
Mittlerer Pfad 2-4
Tel (49-711) 13968-0
Telefax. (49-711) 8661427

ITALY

20090 ASSAGO (MI)
Vle Milanofiori - Strada 4 - Palazzo A/4/A
Tel (39-2) 89213 1 (10 linee)
Telex 330131 - 330141 SGSAGR
Telefax (39-2) 8250449

40033 CASALECCHIO DI RENO (BO)
Via R Fucini, 12
Tel (39-51) 593029
Telex 512442
Telefax (39-51) 591305

00161 ROMA
Via A Torlonia, 15
Tel (39-6) 8443341
Telex 620653 SGSATE I
Telefax (39-6) 8444474

NETHERLANDS

5652 AR EINDHOVEN
Meerenakkerweg 1
Tel (31-40) 550015
Telex 51186
Telefax (31-40) 528835

SPAIN

08021 BARCELONA
Calle Platon, 6 4th Floor, 5th Door
Tel (34-3) 4143300-4143361
Telefax (34-3) 2021461

28027 MADRID
Calle Albacete, 5
Tel (34-1) 4051615
Telex 46033 TCCEE
Telefax (34-1) 4031134

SWEDEN

S-16421 KISTA
Borgarfjordsgatan, 13 - Box 1094
Tel (46-8) 7939220
Telex 12078 THSWS
Telefax (46-8) 7504950

SWITZERLAND

1218 GRAND-SACONNEX (GENEVA)
Chemin Francois-Lehmann, 18/A
Tel (41-22) 7986462
Telex 415493 STM CH
Telefax· (41-22) 7984869

UNITED KINGDOM and EIRE

MARLOW, BUCKS
Planar House, Parkway
Globe Park
Tel (44-628) 890800
Telex 847458
Telefax (44-628) 890391

AMERICAS

BRAZIL

05413 SÃO PAULO
R Henrique Schaumann 286-CJ33
Tel (55-11) 883-5455
Telex (391)11-37988 "UMBR BR"
Telefax (55-11) 282-2367

CANADA

NEPEAN ONTARIO K2H 9C4
301 Moodie Drive
Suite 307
Tel (613) 829-9944

U.S.A.

NORTH & SOUTH AMERICAN
MARKETING HEADQUARTERS
1000 East Bell Road
Phoenix, AZ 85022
(1-602) 867-6100

SALES COVERAGE BY STATE

ALABAMA
Huntsville - (205) 533-5995

ARIZONA
Phoenix - (602) 867-6217

CALIFORNIA
Santa Ana - (714) 957-6018
San Jose - (408) 452-8585

COLORADO
Boulder (303) 449-9000

ILLINOIS
Schaumburg - (708) 517-1890

INDIANA
Kokomo - (317) 455-3500

MASSACHUSETTS
Lincoln - (617) 259-0300

MICHIGAN
Livonia - (313) 953-1700

NEW JERSEY
Voorhees - (609) 772-6222

NEW YORK
Poughkeepsie - (914) 454-8813

NORTH CAROLINA
Raleigh - (919) 787-6555

TEXAS
Carrollton - (214) 466-8844

FOR RF AND MICROWAVE
POWER TRANSISTORS CON-
TACT
THE FOLLOWING REGIONAL
OFFICE IN THE U.S.A.

PENNSYLVANIA
Montgomeryville - (215) 361-6400

ASIA / PACIFIC

AUSTRALIA

NSW 2220 HURTSVILLE
Suite 3, Level 7, Otis House
43 Bridge Street
Tel (61-2) 5803811
Telefax (61-2) 5806440

HONG KONG

WANCHAI
22nd Floor - Hopewell centre
183 Queen's Road East
Tel (852) 8615788
Telex. 60955 ESGIES HX
Telefax (852) 8656589

INDIA

NEW DELHI 110001
LiasonOffice
62, Upper Ground Floor
World Trade Centre
Barakhamba Lane
Tel (91-11) 3715191
Telex 031-66816 STMI IN
Telefax (91-11) 3715192

MALAYSIA

PETALING JAYA, 47400
11C, Jalan SS21/60
Damansara Utama
Tel (03) 717 3976
Telefax (03) 719 9512

PULAU PINANG 10400
4th Floor - Suite 4-03
Bangunan FOP-123D Jalan Anson
Tel (04) 379735
Telefax (04) 379816

KOREA

SEOUL 121
8th floor Shinwon Building
823-14, Yuksam-Dong
Kang-Nam-Gu
Tel (82-2) 553-0399
Telex SGSKOR K29998
Telefax (82-2) 552-1051

SINGAPORE

SINGAPORE 2056
28 Ang Mo Kio - Industrial Park 2
Tel (65) 4821411
Telex RS 55201 ESGIES
Telefax (65) 4820240

TAIWAN

TAIPEI
12th Floor
325, Section 1 Tun Hua South Road
Tel (886-2) 755-4111
Telex 10310 ESGIE TW
Telefax (886-2) 755-4008

JAPAN

TOKYO 108
Nisseki - Takanawa Bld 4F
2-18-10 Takanawa
Minato-Ku
Tel (81-3) 3280-4121
Telefax (81-3) 3280-4131


ORDER CODE: DBST9040FAST/1