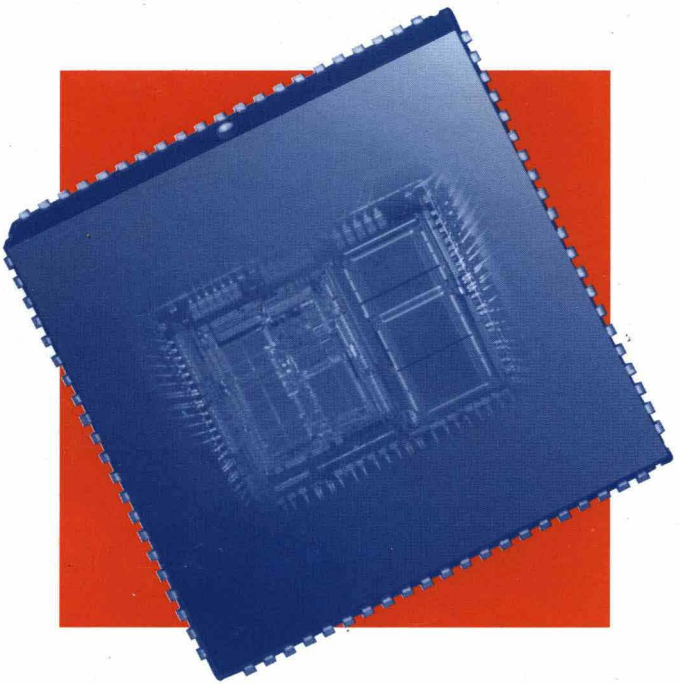
1992 IDT R3051™/R3081™ Application Guide

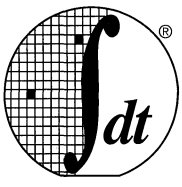
Integrated Device Technology, Inc.

1992
IDT R3051™/R3081™
Application Guide



Integrated Device Technology, Inc.

1992
IDT R3051™/R3081™
Application Guide



Integrated Device Technology, Inc.

LIFE SUPPORT POLICY

Integrated Device Technology's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the manufacturer and an officer of IDT.

1. **Life support devices or systems are devices or systems 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 in the labeling, can be reasonably expected to result in a significant injury to the user.**
2. **A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.**

Integrated Device Technology, Inc.

TABLE OF CONTENTS

Integrated Device Technology, Inc.

INTRODUCTION

This manual is a collection of various applications notes and conference papers written to describe the behavior and use of the IDT79R3051™ family of RISController™ devices.

The application notes include descriptions of design techniques, development environments, and software development tools. The reader is encouraged to review the introduction of the various application notes as a brief summary of the topic of that paper.

This manual is complemented by other documentation, also available from your IDT sales representative. These documents include:

- The RISC data book, which contains data sheets for these devices. Also included are the electrical specifications, pinout, current speed grades, and package dimensions.
- The R3051 Hardware User's Manual, which contains a detailed description of the hardware and software interface of the R3051 and R3052.
- The R3081 Hardware User's Manual, which contains a detailed description of the hardware and software interface of the R3081.
- The DRAM Design Using the IDT RISChipset manual, which describes the use of the R3721 DRAM controller with the R3051 or R3081.
- The IDT Development Products Catalog, which contains an overview description of various development tools manufactured and sold directly by IDT.
- The various user's manuals on the IDT software tools, and the user's manual for the IDT7RS385 Evaluation Board.
- The third-party support list, detailing various third-party tools, such as real-time OS, in-circuit emulation, logic analyzer support, and program development tools available to support applications development around the IDT R3051 family.

Integrated Device Technology, Inc.

IDT79R3051™ SYSTEM DESIGN EXAMPLE

APPLICATION NOTE AN-86

By Andrew Ng

INTRODUCTION

This application note describes a memory evaluation board that is an example of many of the design considerations for systems based on an IDT79R3051™ RISController™ family CPU.

The memory board, illustrated in Figure 1, consists of:

- An R3051 CPU
- Reset circuitry
- An address demultiplexer
- A data transceiver
- Wait-state and memory control logic
- 128K bytes of SRAM
- 128K bytes of EPROM
- A dual channel UART
- A real time counter
- An interrupt controller

In addition, an expansion connector supplies all the CPU signals for the addition of external modules such as DRAM memory systems or other application specific I/O systems. The memory and I/O system on the example board are compatible with the IDT7RS382 R3000 Evaluation Board. Thus 7RS382 software such as the IDT/sim PROM Debug Monitor can run on the example board. The board is typical of an embedded controller core such as for LAN adapters, laser printers, facsimiles, and avionics applications. The differences would appear in which peripherals are used and memory type, size, and speed requirements.

The board was designed as a generic example of the construction of a system using the IDT79R3051 RISController with both low parts count and cost sensitive requirements. However, since many generalities were taken into consideration, many systems can reduce both parts count and cost even further. Although the board is not populated with parts that have the highest performance achievable, its design can be easily modified to do so. In addition, PAL® support for further experiments with optimizations and trade-offs can be done to accommodate different kinds and speeds of memory and I/O. While the board is designed with SRAM for the simplicity of a design example, the extension to a DRAM system with CAS before RAS refresh is only slightly more complex.

THE R3051 RISController CPU

The IDT79R3051 family is a series of high-performance 32-bit microprocessor RISControllers designed to bring the high-performance inherent in the MIPS® RISC architecture into low cost, simplified, and power sensitive applications.

The instruction set is compatible with the 79R3000A and 79R3001 RISC CPUs. Features of the R3051 family include:

- 4kB (R3051) to 8kB (R3052) of Instruction Cache on-chip
- 2kB of Data Cache on-chip
- Clocked from a single, double-frequency clock input
- On-chip 4-deep read and write buffer
- On-chip DMA arbiter
- Flexible burst/simple block bus interface
- Multiplexed address and data bus for low cost packaging, simplicity of use
- Base versions use fixed address translation to simplify software
- Extended architecture versions use 64-entry, fully associative Translation Lookaside Buffer (TLB) to support page.

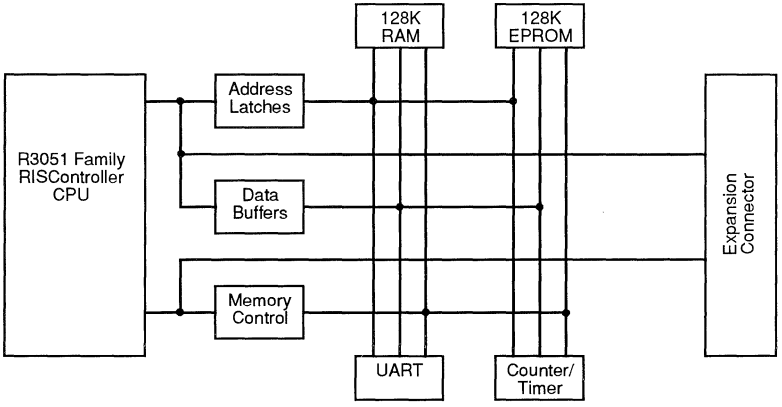


Figure 1. System Block Diagram

The R3051 RISController combines a similarly featured R3000A CPU system consisting of over 50 LSI/MSI parts into a single integrated chip.

DETAILED DESIGN REVIEW

The following sections give a detailed review of how each functional block relates specifically to designing with the R3051 RISController. Particular attention is focused on alternative design strategies that could reduce parts count and improve performance as well as on a description of the original design. The subsystem block designs include:

- Analog reset logic
- A PAL-based memory controller (3x PALs)
- Address demultiplexer (4x IDT74FCT373T)
- Data transceiver (4x IDT74FCT623T)
- 128kB of SRAM (4x IDT71256 32kx8 45ns SRAM)
- 128kB of EPROM (4x 27256 32kx8 125ns EPROM)
- 68681 DUART
- 8254 Timer
- Interrupt controller (1x PAL)
- Off-card connector

Reset, Reset Vector, and Clock Buffer Circuitry

The $\overline{\text{Reset}}$ signal is based on a linear integrated circuit, a TI TL7705A supply voltage supervisor with a Power-On Reset Generator. A 1μF capacitor is used to program the reset generator for a 13ms Reset period.

Note that because the R3051 synchronizes the $\overline{\text{Reset}}$ input signal internally, an RC circuit can be used instead. An example is to pull $\overline{\text{Reset}}$ high with a resistor of about 10kΩ, tie $\overline{\text{Reset}}$ to a 22μF capacitor which is tied to ground, and tie $\overline{\text{Reset}}$ to a push button switch that is tied to ground. Then the RC circuit should be gated through a buffer or synchronizer.

Certain configuration options (the reset vector) are selected in the R3051 by using the interrupt pins at the rising edge of $\overline{\text{Reset}}$. On the example board, the interrupt pins are simply pulled up (or down) since $\overline{\text{SInt}}$(2:0) are not used in this system (software can permanently mask these interrupt inputs in the Status Register). However, if they are used (via the expansion connector) they would need to be multiplexed with the reset function. There are a number of techniques to perform this multiplexing: for example, if the interrupting agent is not capable of tri-stating its interrupt during $\overline{\text{Reset}}$, an external multiplexer such as an IDT74FCT257T can be used, with the enable always tied active and the select tied to $\overline{\text{Reset}}$. If the interrupting agent tri-states its interrupt during Reset, then using simple pull-ups or pull-downs will still operate properly.

The clocks on the board are buffered by an IDT74FCT240C(T) inverting tri-state buffer. This buffer was selected partially to provide a board testability path for injecting a test clock, as well as to buffer the signals to increase their drive. The primary reason for the buffer, however, is to invert $\overline{\text{SysClk}}$ to form SysClk, the signal that is used to clock the state machines on this board. Buffer output pins closest to the ground pin (pins with the lowest pin inductance) were used first to help lessen potential noise and ground bounce problems. The Clk2xIn oscillator is socketed, so that the board may be populated with different speed parts.

In this design, the FCT240C(T) enables are pulled down to be active all of the time. Since $\overline{\text{SysClk}}$ does not tri-state when $\overline{\text{Tri-State}}$ ($\overline{\text{SInt}}$(1)) is active during the reset vector, it is helpful to an ATE programmer to be able to tri-state the inverter.

Memory Controller

The example board's Memory Controller consists of three 22V10 PALs. The first PAL is used for address decoding, the second for wait state and cycle counting, and the third for byte enables. The PALs are functionally described in the following paragraphs. The PAL equations are included in the appendices. The PALs are all placed in sockets, and thus can easily be reprogrammed for various experiments.

Address Decoder

The Address Decoder PAL, MEMDEC.JED, uses Address(31:17) to generate chip selects. The chip selects are decoded according to the 7RS382 address map as described in the 7RS382 Hardware User's Guide. Three spare I/O pins are provided, which could be used to decode additional chip selects. These spare outputs are in place of the 'USER CS1X*' chip selects provided for on the 7RS382 board, but not explicitly supplied by this example board.

The address decoder does not wait for ALE to begin generating the chip-select outputs. It does this so that maximum performance may be achieved, since the Chip Select outputs will be generated earlier in the transfer. However, as a result, the CS outputs may tend to "glitch" as a valid address is driven. Thus, the Read Enable and Write Enable seen in the memory system must be synchronized so that they are valid only within the time that the CPU is attempting a read or write transfer. This combination allows maximum performance: address and chip enables are seen early in the transfer, but the Read and Write signals are generated synchronously to insure proper system operation.

One of the extra I/O pins can be used as a test enable input to tri-state the outputs for board level ATE. Some systems will not need to decode as many address bits or may have a fixed map, and thus may able to use FCT138's or 16V8's to do the address decoding instead of the relatively expensive 22V10 part.

Memory Cycle Controller

The purpose of the Memory Cycle Controller is to provide a wait-state generator which stalls the R3051's Bus Interface Unit, so that various types and speeds of memory can be used. The Memory Cycle Controller is implemented with a 22V10 PAL called MEMCONT.JED. Note that this PAL was selected in order to make the PAL equations more readable. A lower cost solution may implement the state machine in two 16R8 PALs.

The Memory Cycle Controller allows various speeds of memory devices to be used, by using the throttled read supported by the R3051 bus interface. Other kinds of transactions are treated as simplified cases of the throttled read.

The basic state machine looks for the start of a read or write transaction by looking for an asserting edge of $\overline{\text{Rd}}$ or $\overline{\text{Wr}}$. When

a transaction is begun, the state machine starts a 5-bit binary up counter, C(4:0). C(4:0) then increments on each SysClk rising edge. C(4:0) is used as the basic timing master for all of the other control signals generated in the state machine.

In the memory scheme used here, rather than search for the negating edge of $\overline{Rd}$ or $\overline{Wr}$ at the end of the transaction, a $\overline{CycEnd}$ synchronous decoder is used to tell the C counter when the end of the memory cycle occurs. This type of strategy is used because the de-asserting edges of $\overline{Rd}$ and $\overline{Wr}$ occur within the setup and hold times of a buffered/inverted (FCT240C(T)) $\overline{SysClk}$. Typically, the de-asserting edge of $\overline{Rd}$, $\overline{Wr}$, and $\overline{Burst}$ should not be used to control a SysClk based state machine. Similarly, the rapid negation of ALE by the processor makes it difficult to synchronously sample ALE when using a state machine driven by a buffered clock.

$\overline{CycEnd}$ serves to synchronously reset the state machine when a de-asserting $\overline{Rd}$ or $\overline{Wr}$ edge is expected, whether or not the $\overline{Rd}$ or $\overline{Wr}$ de-asserting edge meets the setup and hold times of the state machine. Another output, $\overline{EnStart}$ is used to start the byte enables by waiting a number of cycles before asserting. The amount of time the transfer waits is used to allow drivers used in the previous transfer to tri-state, and may be necessary in systems which employ devices whose output disable time is long relative to the system clock frequency.

Other outputs from the Memory Cycle Controller PAL include the R3051 transfer termination inputs $\overline{RdCEn}$, $\overline{Ack}$, and $\overline{BusError}$. On a read transfer, $\overline{Burst}$ and one of the Chip Enable inputs from the Address Decoder are used to determine the timing and quantity of $\overline{RdCEn}$ signals to be asserted for this transfer (according to the requested transfer size and the memory device speed).

$\overline{Ack}$ is asserted at the end of a write cycle to indicate completion of the transfer, and optionally towards the end of a Quad Word (Burst) read cycle. A description of the various kinds and options of read and write cycles is thoroughly explained in the R3051 Family Hardware User's Guide. The number of cycles before and between the assertion of $\overline{Ack}$ and $\overline{RdCEn}$ is programmable, allowing flexibility for various types of memories.

Finally, the $\overline{BusError}$ output is used to end an undecoded memory cycle. In the R3051, $\overline{Rd}$ is negated one-half cycle after the $\overline{BusError}$ input is asserted.

Other Approaches

Of course, alternative methods and techniques to memory interfacing with an R3051 family CPU exist. Four approaches easily implemented in discrete components include:

- using a SysClk based CycEnd counter (as used in this example)
- using asynchronously resettable registers for the counter
- using interlocking SysClk and SysClk registers
- using an unbuffered SysClk

All of these methods can be used to design for the clocking scheme of the R3051 Family, which uses both the rising and falling edges to control its outputs. The use of both edges of the clock allows the R3051 to mitigate the 1 clock inter-transaction latency that is associated with most other CPUs that need the extra clock to fixup and start new memory cycles. However, because the R3051 Family asserts and de-asserts its edges the same way on both $\overline{Rd}$ and $\overline{Wr}$ cycles, specific methods can be employed so that the memory system is always clocked from one edge of $\overline{SysClk}$. An example of this is the $\overline{CycEnd}$ method used on this board, which ignores the edges that are not synchronized with the state machine. Although traditional high-performance CPUs require complex state machines to operate efficiently, the beauty of the R3051 family is the simplicity of its interface. Memory control state machines for the R3051 family are really only minor variations on traditional wait-state machines, and can also easily take advantage of the 1/2 clock inter-transaction savings provided by the CPU interface.

Each of the four approaches has advantages as well as drawbacks relative to each other. The following paragraphs will give a brief description of each technique. Each of the

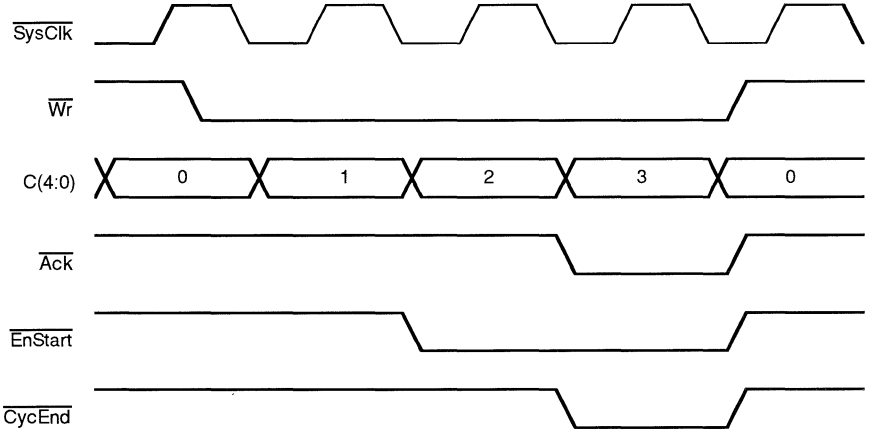


Figure 2. Timing of CycEnd

methods could be used by themselves or combined with one or more of the other methods, to achieve the optimal price/performance/parts count for a given application. Systems employing dedicated interface chips (such as the IDT R372x family, or customer specific ASIC or Gate Array devices), may choose to make different trade-offs than those using discrete component based solutions.

Using SysClk and generating a Cycle End indicator

The SysClk based CycEnd approach as described above is straightforward because of its similarity to traditional wait-state machines. As mentioned above, it does not require the terminating edge of Rd or Wr to complete a transaction.

The system implemented in this design example is limited in speed by:

tclk/2 >= t240 + tpalco + t3051setup + tcap + twire

which works out to 28MHz for a 10ns 16V8, over 40MHz for a 5ns 16R8 PAL, and 33MHz for a 10ns 22V10 PAL.

Using Asynchronous Reset to terminate the Cycle Counter

The second potential method, which uses an asynchronous reset to terminate the cycle, requires ANDing together Rd and Wr into the the reset line of the counter C(4:0) and can be demonstrated by reprogramming the PAL on the example board. The reset-to-valid output, reset width, and the reset recovery time to clock are among the speed limiting paths in this approach when implemented in PALs. Unfortunately, the reset-to-output delay of a PAL is usually less optimized and relatively slow.

tasyncreset <= tclk/2 - trdn - tcap - twire

For example, a 20MHz system would require a reset-to-output delay of 17ns, which can be found in a 10ns 22V10 PAL (with a 15ns reset-to-valid output data time).

Using interlocking PALs clocked on opposite edges

The third potential approach uses a SysClk based register to detect asserting edges and a SysClk based register to detect de-asserting edges. The outputs of each of the PALs interlock by controlling the outputs of the other PALs. This allows the flexibility of seeing all edges and being able to control outputs optimally by using any 1/2 clock edge (such as output enables). Such an approach obviously requires more PALs, and is somewhat speed limited by:

tclk/2 >= t240 + tpalco + tpalsetup + tcap + twire

which works out to 20MHz for a 10ns 16V8 PAL.

In systems using chips designed specifically to interface to the R3051 family (such as the IDT R3721 DRAM controller), this approach is simpler to implement and leads to the highest levels of performance.

Using an unbuffered SysClk

The fourth potential approach uses an unbuffered SysClk based state machine. This leads to the requirement of having 0 hold time on the registers as well as a 2ns minimum propagation delay time to meet the R3051 timing requirements (note that using a buffered SysClk instead of the unbuffered version would require negative hold time on the registers). Despite these restrictions, some PALs can be found that meet all of these requirements. This approach leads to a one cycle latency in reacting to R3051 output assertions. An asserting Rd or Wr would be seen a clock too late to bring RdCEn or Ack LOW during their first possible sampling clock. Using an unbuffered SysClk has a speed advantage over the other techniques:

tclk >= tpalco + t3051setup + tcap + twire

tclk/2 >= t3051prop + tpalsetup + tcap + twire

which can support designs of 35MHz for a 10ns 16V8 PAL and well over 40MHz with a 7.5ns 16R8 PAL.

An additional consideration relative to using an unbuffered SysClk is the amount of loading placed on the clock, and the impact of additional loading on R3051 AC parameters. Of course, when using a single chip memory controller such as the IDT R3721 or a customer designed ASIC, these loading considerations are minimal.

In summary, the R3051 Family uses both edges of the clock to assert control signals in order to reduce inter-transaction delay between external bus cycles. However, by using one or a combination of the above techniques in a design, a traditional wait-state machine can still be used with the addition of only minor variations.

Read and Write Enables

The Read and Write Enables PAL, MEMEN.JED, uses EnStart and CycEnd to control the initiation and length of the output enable and write enable assertions. Rd and Wr are used to select between read and write cycles. Note that it would have been possible to combine individual bank selects with the address decoder PAL, rather than use a distinct PAL to control the timing of the assertion of Write and Read Byte Strobes.

On read cycles, RdEn is asserted as the system's primary output enable signal. RdDataEn is used to enable the FCT623T data transceiver bank. RdDataEn in most systems would simply be 'DataEn' as supplied straight from the processor. This system provides RdDataEn in case other transceiver banks are added to the system.

The byte enables are used to support partial word writes which are used during byte, halfword, and tri-byte operations. Write cycles combine the byte enables, BE(3:0), with Wr, EnStart, and CycEnd to form the write enable outputs WrEn(D:A) which are attached to the byte banks within the memory system. Whether or not the system is Little or Big Endian, WrEn(A) is always attached to the LSB. WrEn(D:A) can also be implemented using an FCT257T multiplexer. WrDataEn is used to control the FCT623T data transceiver bank and must be held extra long to provide memory data hold time.

Finally, the Byte Enable PAL also has a synchronized Pon used to update wReset output called Reset and a "guarded" GUARTCS. The guarded chip select, GUARTCS is an example of interfacing R3051 signals to a Motorola-type I/O Device as opposed to an Intel-type I/O Device.

Motorola-type devices multiplex their read/write input pin and expect a data strobe pin to validate the data out or to latch the data in, while Intel-type devices have separate read and write strobes. Since the MC68681 DUART is a Motorola device, the data strobe must start late and end early, so that read/write is held throughout that period. Additionally, the MC68681 uses its chip select pin as a data strobe. As a data strobe, it is important not to have decoder glitches on the chip select since reads in I/O devices are often used to update FIFO pointers. Thus, the guarded $\overline{\text{GUARTCS}}$ uses $\overline{\text{EnStart}}$ and $\overline{\text{CycEnd}}$ to shorten up $\overline{\text{UARTCS}}$. Finally, $\overline{\text{WrEn}}$ is provided to extend $\overline{\text{Wr}}$ to allow additional data hold time at the end of the write cycle. $\overline{\text{WrEn}}$ could easily be inserted with another OR term into $\overline{\text{WrEn}}$(A).

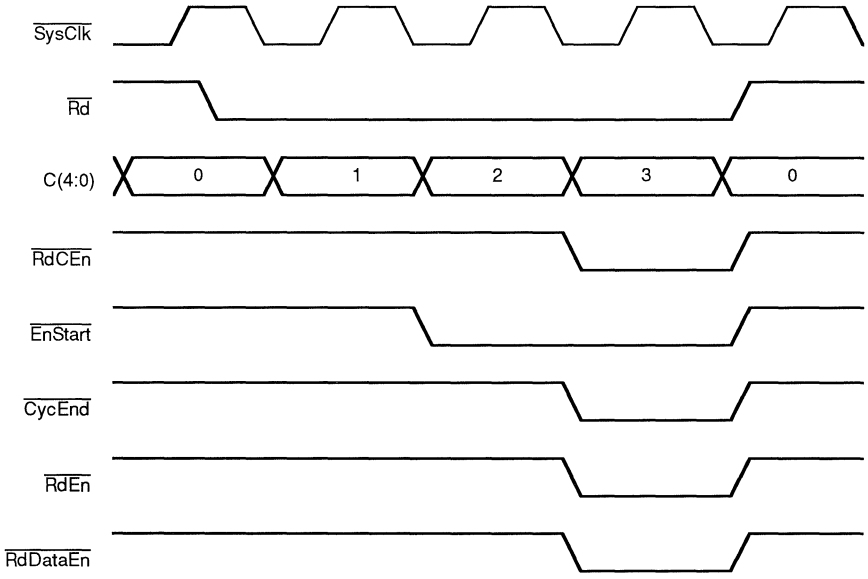


Figure 3. Timing Diagram of RdEn

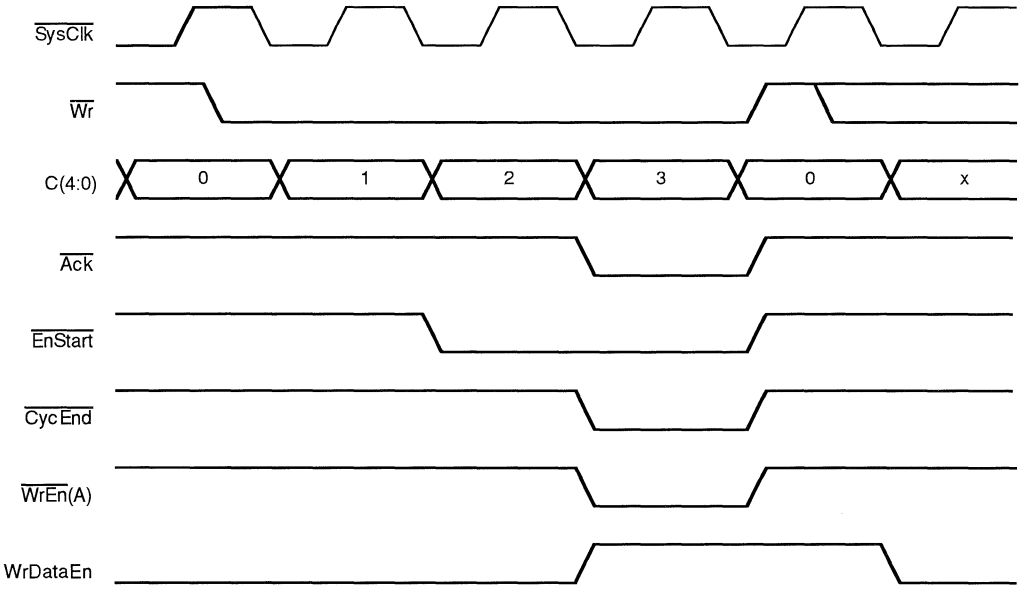


Figure 4. Timing Diagram of WrEn**(A)**

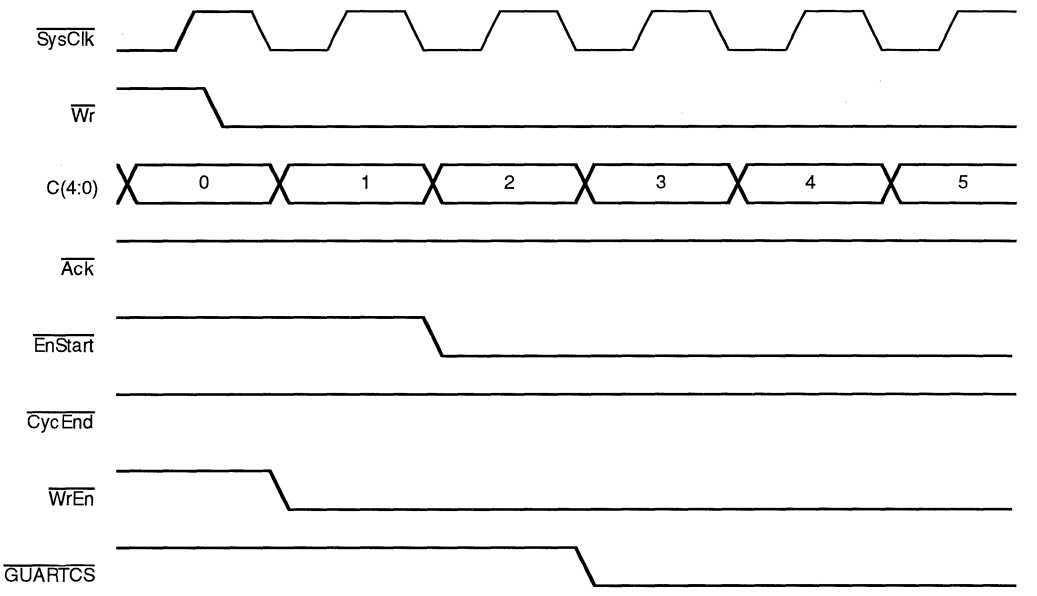


Figure 5. Timing Diagram of Start of GUARTCS

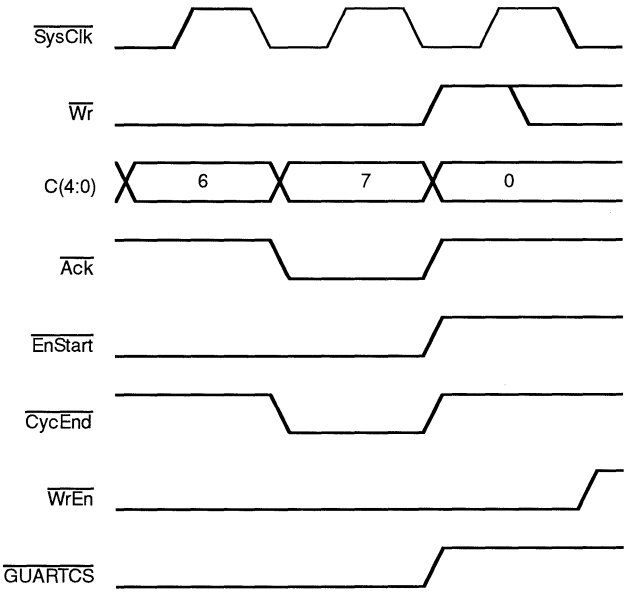


Figure 6. Timing Diagram of End of GUARTCS

Address Latch and Transceiver Demultiplexer

The address latch bank consists of four FCT373T 8-bit transparent latches. ALE is used for the latch enable on the FCT373Ts. The transparent phase allows extra address decoding time during the time that ALE is HIGH; the outputs of the latches are fed directly to the address decode PAL and to the memory devices. In order to insure that address hold time to the latches are met, it is important to take care with the use of the ALE signal. The number and length of the ALE traces is critical and should be kept to a minimum.

Rather than use FCT373s, DRAM systems may want to use FCT821s or FCT823s, which are wider latches. RAS/CAS address multiplexing can be performed by sequencing the output enables of the latches and having the outputs of the latches tied together and driving the DRAM address bus.

The data transceiver bank on the example board uses four FCT623T 8-bit transceivers. FCT623Ts were chosen over the similar 10-bit FCT861's and 9-bit FCT863s simply to reduce pin count. The FCT861/3s provide a more conventional interface, since both output enables are active-LOW, instead of one enable active-HIGH, and the other active-LOW as in the FCT623Ts. However, since this system uses PALs to control the transceivers, the use of FCT623s poses no additional complexity to the design.

FCT623Ts were selected instead of FCT245s because of the ease of interfacing to dual output enable pins instead of a direction and enable pins as in the FCT245. Interfacing with FCT245 controls would ideally require that the direction control only be changed when the output enable is disabled. This requires extending a combined (latched) $\overline{Rd}$ and $\overline{Wr}$ based signal for an extra cycle at the end of a memory transaction, which may be the beginning of the next memory cycle. Unless the direction pin is controlled with a $\overline{SysClk}$ based state machine, a signal like $\overline{EnStart}$ would be necessary to keep the enable pin de-asserted in the subsequent cycle until the direction pin control becomes valid. Some systems with high noise tolerance, e.g., IBM-PC adapter boards, forgo the extra cycle ideal and simply bus contend for a very short time (a few ns) into its memory system by having the read strobe directly control the direction. $\overline{DataEn}$, output from the CPU, can be used in such systems to simplify control signal generation.

When there are no pending DMA, read, or write requests, the R3051 tri-states the A/D(31:0) bus during these non-bus clock cycles to reduce power consumption. One can optionally add external pullup or pulldown resistors so that the A/D(31:0) bus is always defined for board level ATE and so that the input pins of the latches and transceivers are stabilized.

Finally, systems that can output disable (OE to Z-state) all memory readable devices within:

tdisable < tclk/2 - t3051dataenn + taddr - tcap - twire

might not require the transceiver bank and thus could reduce the parts count by 4.

EPROM and Static RAM Memory

The memory on the example board is populated with 125ns Erasable PROMs (EPROMs) and 45ns Static RAMs (SRAMs). Four 27C256 32kx8 EPROMs are used to form 128kB of ROM. The EPROMs are placed in sockets and thus can easily be removed for reprogramming or replacement; alternative designs may wish to add circuitry to allow in-board programming of the EPROMs (e.g. Flash Erase EPROMs).

The EPROMs have a relatively long output disable time (OE to Z-state), typical of ROMs and thus require data buffers to prevent contention on the multiplexed AD(31:0) bus, since the following equation is not met:

tclk/2 >= tdisablecontrol + tdisable - taddr + tcap + twire

In addition, the disable time for these EPROMs is long enough that, except for relatively slow systems (under 20MHz), extra clocks need to be added to the next bus cycle to prevent bus contention with other memory banks. This is determined by:

tclk >= tdisablecontrol + tdisable - tdata + tcap + twire

The SRAM bank is formed using four IDT71256 32kx8 SRAMs for a total of 128kB. The RAM chips have common data I/O pins, separate read and write strobes, and chip selects. RAMs without a separate read strobe (output enable pin) may require more complex address decoding when used in a multiple bank configuration.

DUART, Timer, and Interrupt Controller

An MC68681 DUART and an MAX235 RS232 transceiver are used to form two RS232 serial communication links. The DUART control registers are word addressed, but only D(7:0) are used. The MC68681 is an example of a Motorola-type I/O interface as explained above.

An iP8254 timer/counter chip is used for a real-time clock or timer. The iP8254 is an example of an Intel-type I/O interface. The iP8254s need for separate read and write strobes matches up well with the R3051.

Software control of these chips is best described by their respective data sheets. Typically, most software programs for the 7RS382 have used the DUART in a polling mode and the timer in a square wave mode. Interrupts $\overline{Int}$(5:3) are controlled by UARTIntOC, Timer OutB, and Timer OutA respectively from MSB to LSB. The 16R8 PAL, called MEMINT.JED, is used to control these interrupts latches in the assertion transition of the original interrupt lines.

The controller holds the interrupt line to the processor for Timer A and Timer B until they are acknowledged (as required by the R3051). Acknowledgement is indicated by reading the interrupt controller at Virtual Address BF800010 and BF800014 (Physical Address 1F800010 and 1F800014) respectively. This action incidentally reads extraneous data from the Timer chip itself on D(7:0). The DUART interrupt must be acknowledged by using the DUART control registers.

The output disable to data in Z-state time for these I/O peripherals is relatively long, as is typical for I/O devices. This forms the critical timing path for the placement of $\overline{\text{EnStart}}$ in the Memory Controller and Memory Enable PALs.

Expansion Connector

Two 50-pin connectors are provided which bring out the R3051 RISController pins to allow off-board expansion. The $\overline{\text{BusReq}}$ and $\overline{\text{BusGnt}}$ pins are not presently used on this board. If DMA is to be used, the R3051 control outputs $\overline{\text{Rd}}$, $\overline{\text{Wr}}$, $\overline{\text{Burst}}$, $\overline{\text{DataEn}}$, and ALE are pulled HIGH or LOW so that they remain inactive when tri-stated.

SCHEMATICS AND PAL EQUATIONS

Appendices include the System Design Example Board Schematics and the PAL equations.

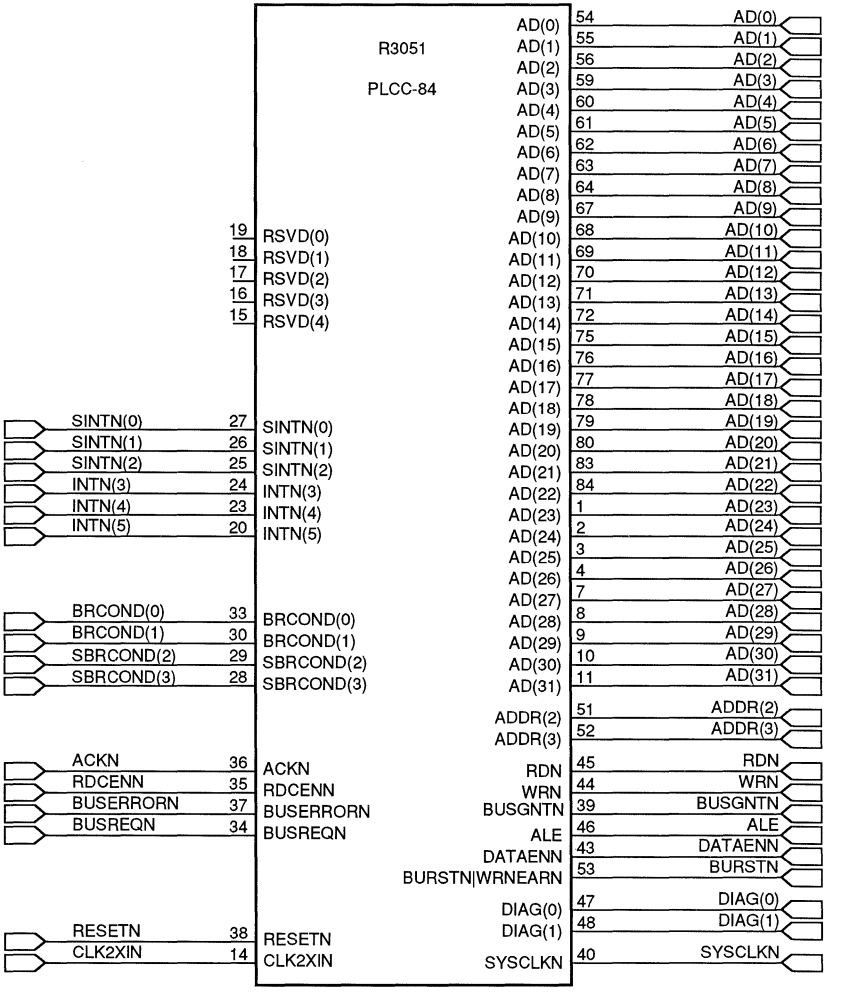


Figure 7. R3051 RISController

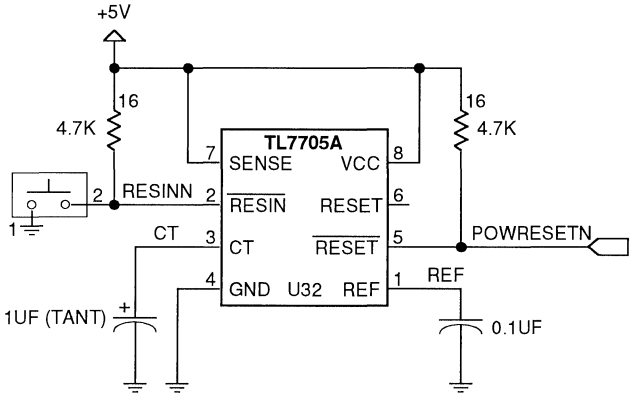


Figure 8. Reset Logic

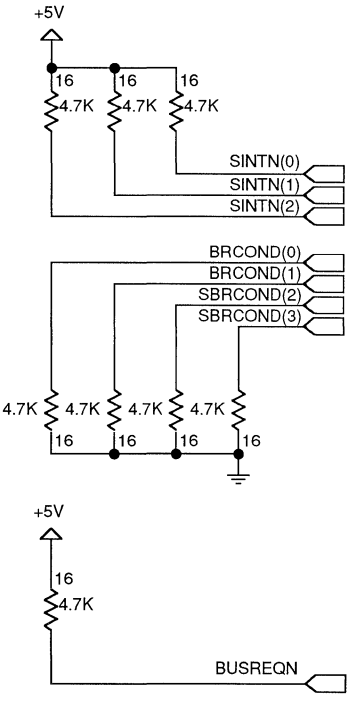


Figure 9. Unused Inputs

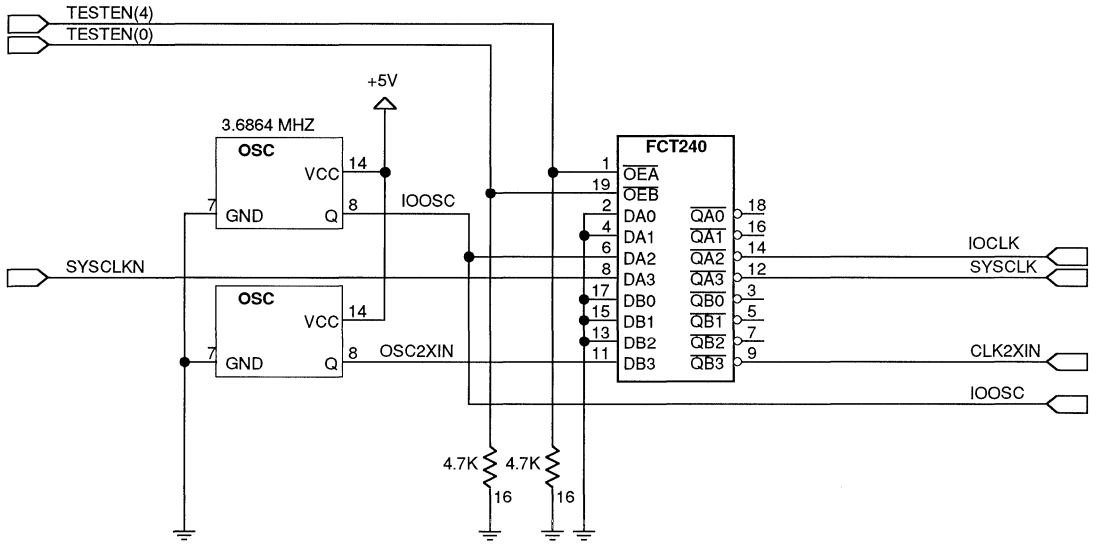


Figure 10. Clock Logic

NOTES:
MEMSPARE0 -- CARDCSN | XCSN0
MEMSPARE1 -- C4 | WRLASTN | WORLDBOOTN
MEMSPARE2 -- TESTEN | SHADOW RAM | DATAENN | XCSN1

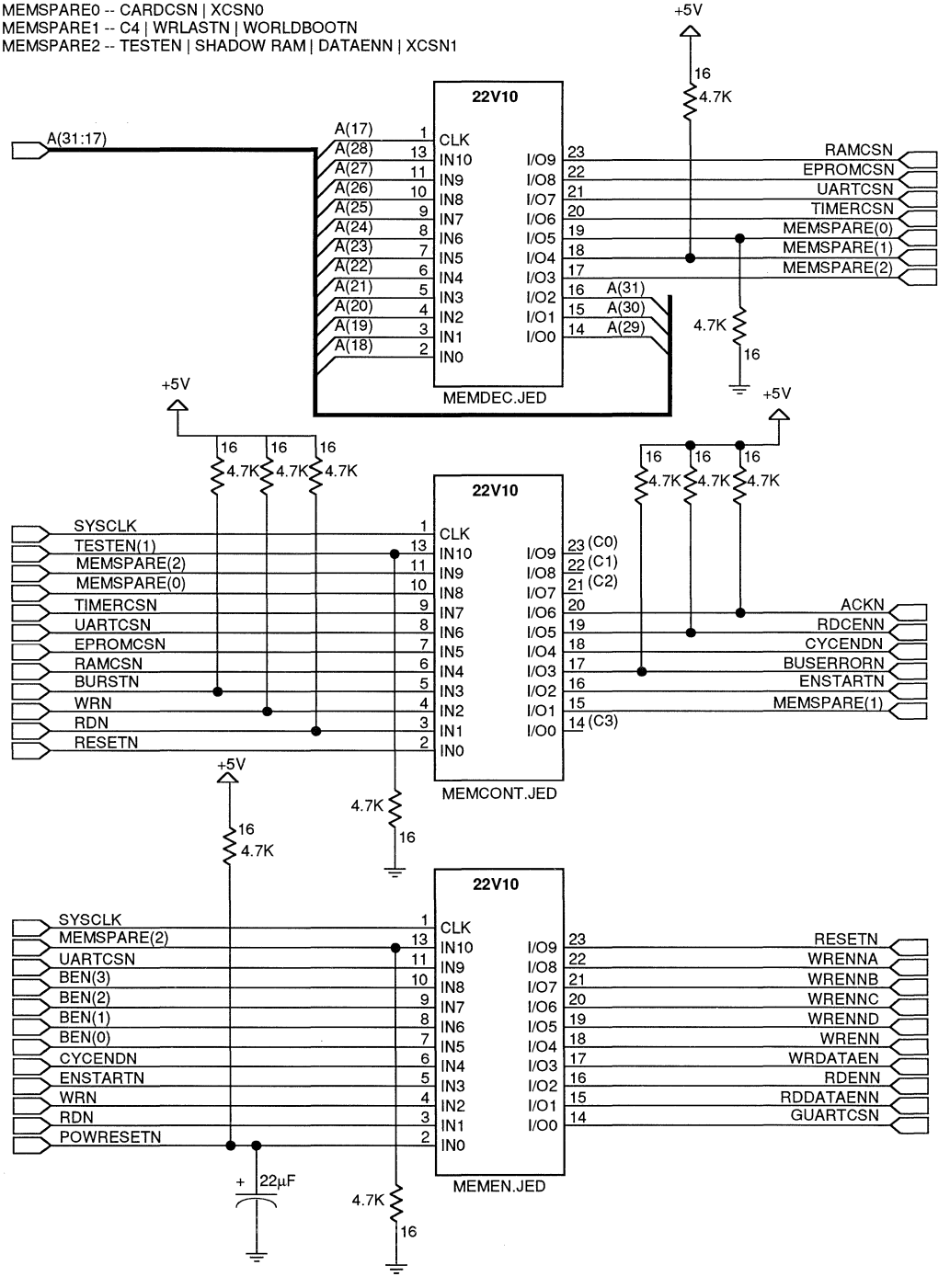


Figure 11. Memory Controller

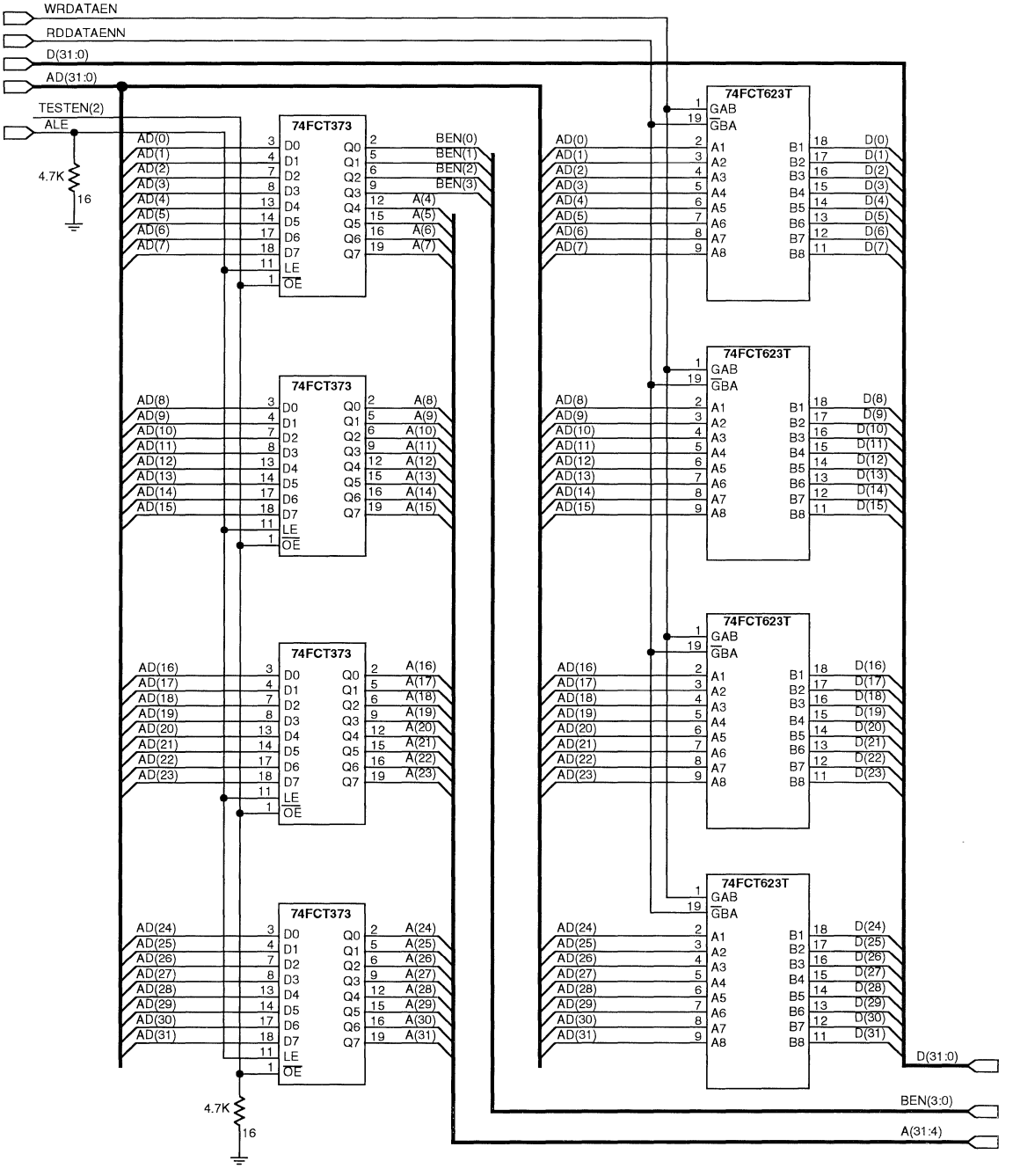
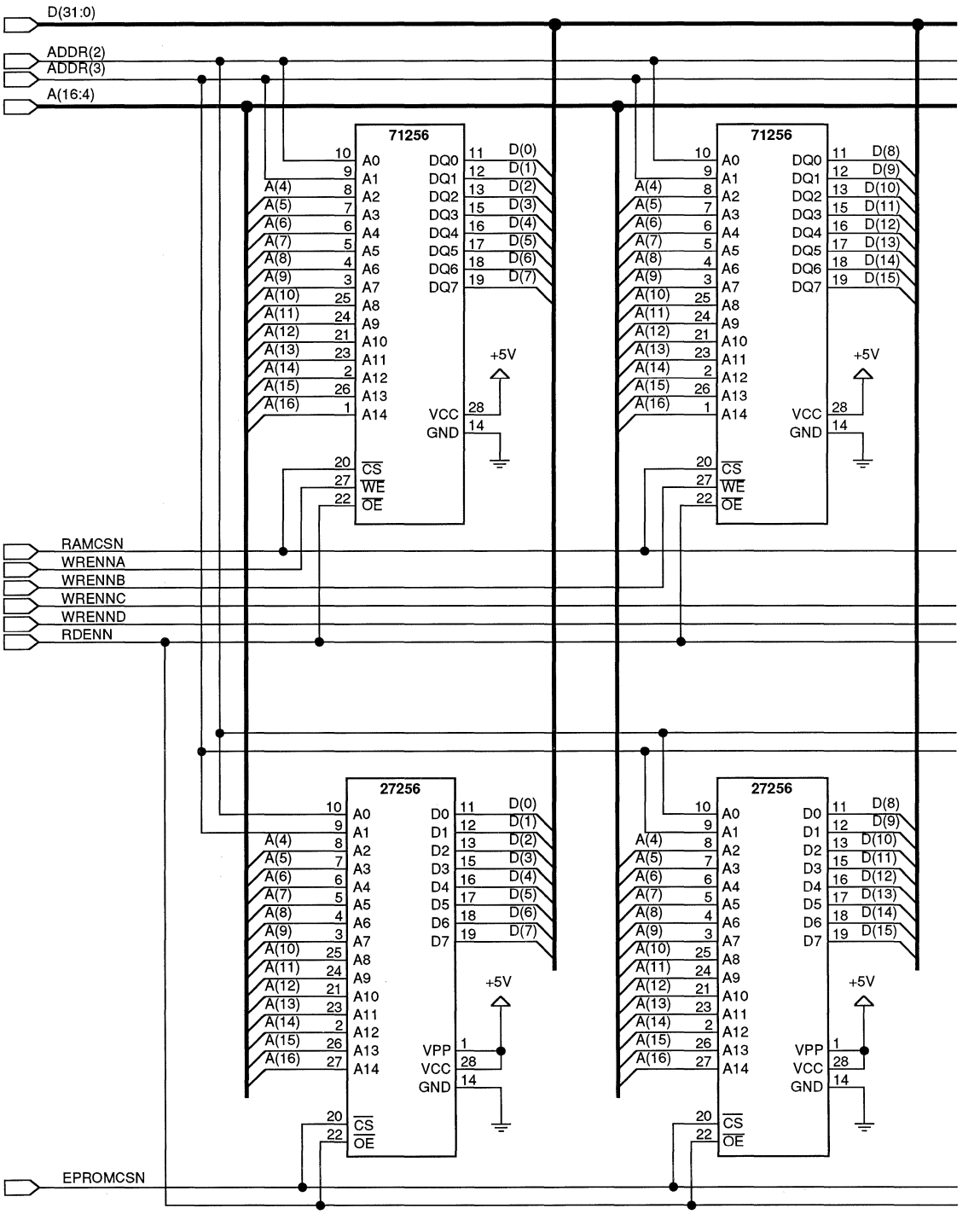


Figure 12. Address Latch Data Transceiver Demultiplexer



NOTE: BANK A - - LITTLE ENDIAN LSB BYTE 0
- - BIG ENDIAN LSB BYTE 3

Figure 13. ROM and Static RAM Memory

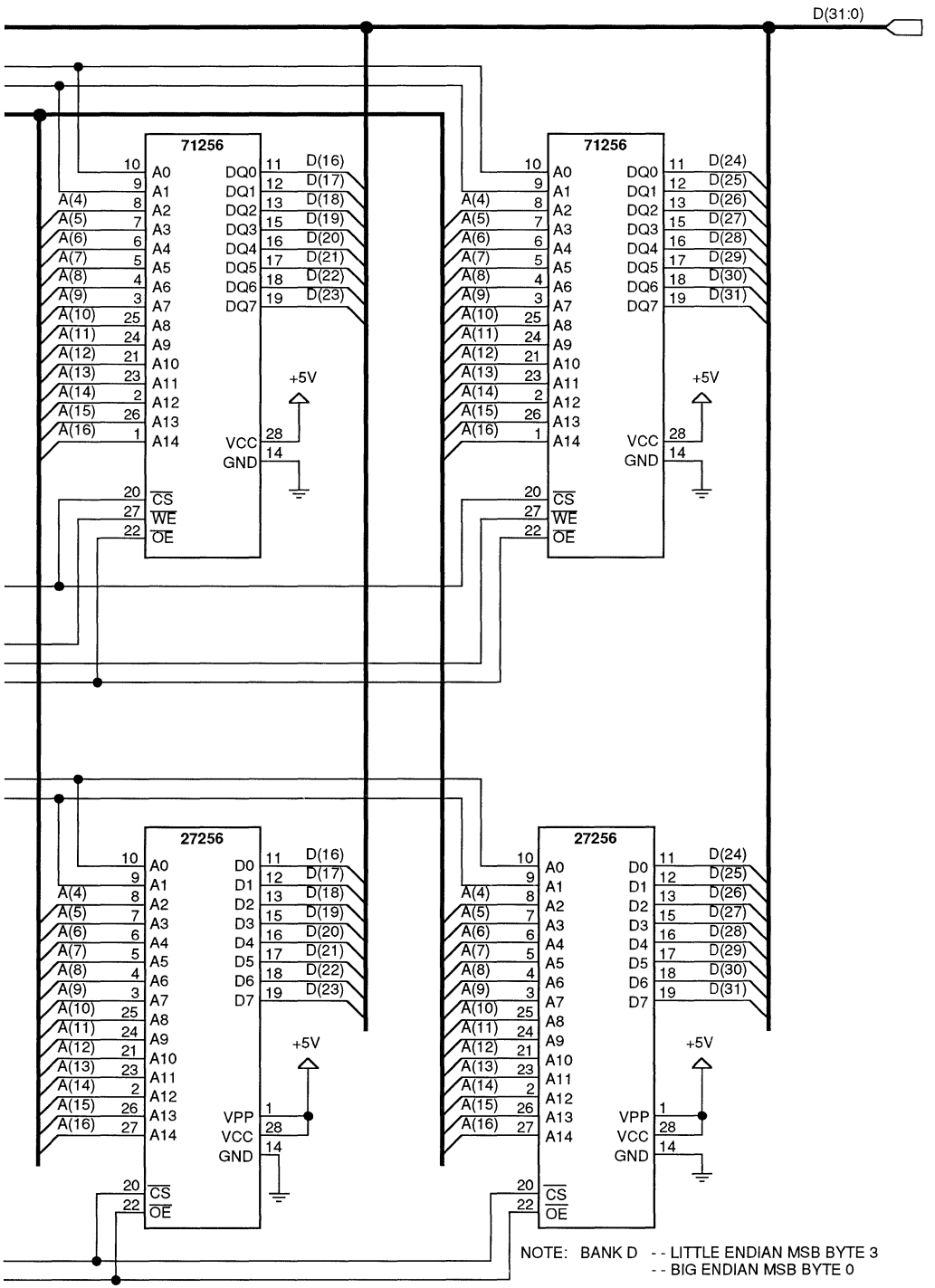


Figure 13. ROM and Static RAM Memory

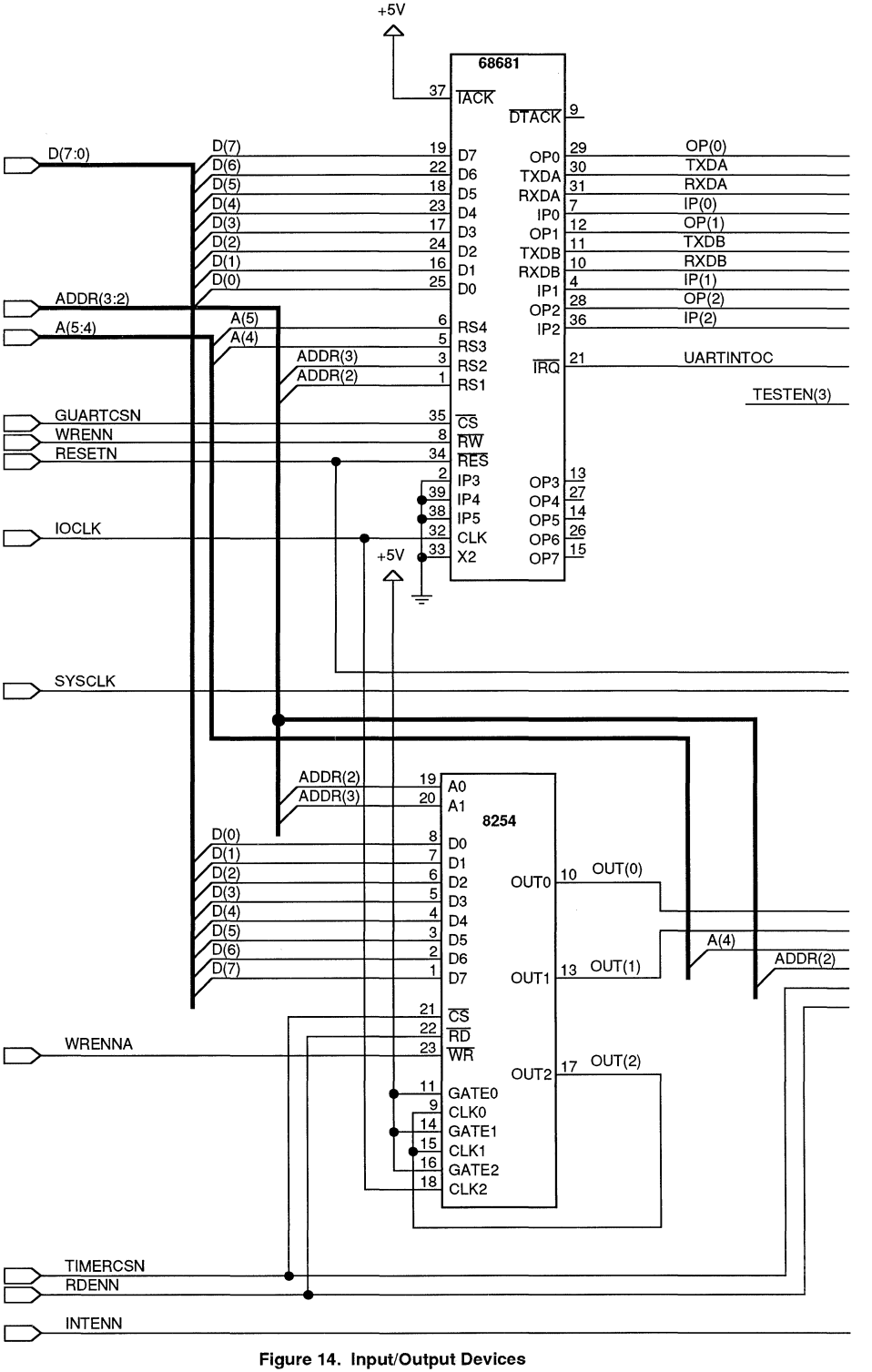


Figure 14. Input/Output Devices

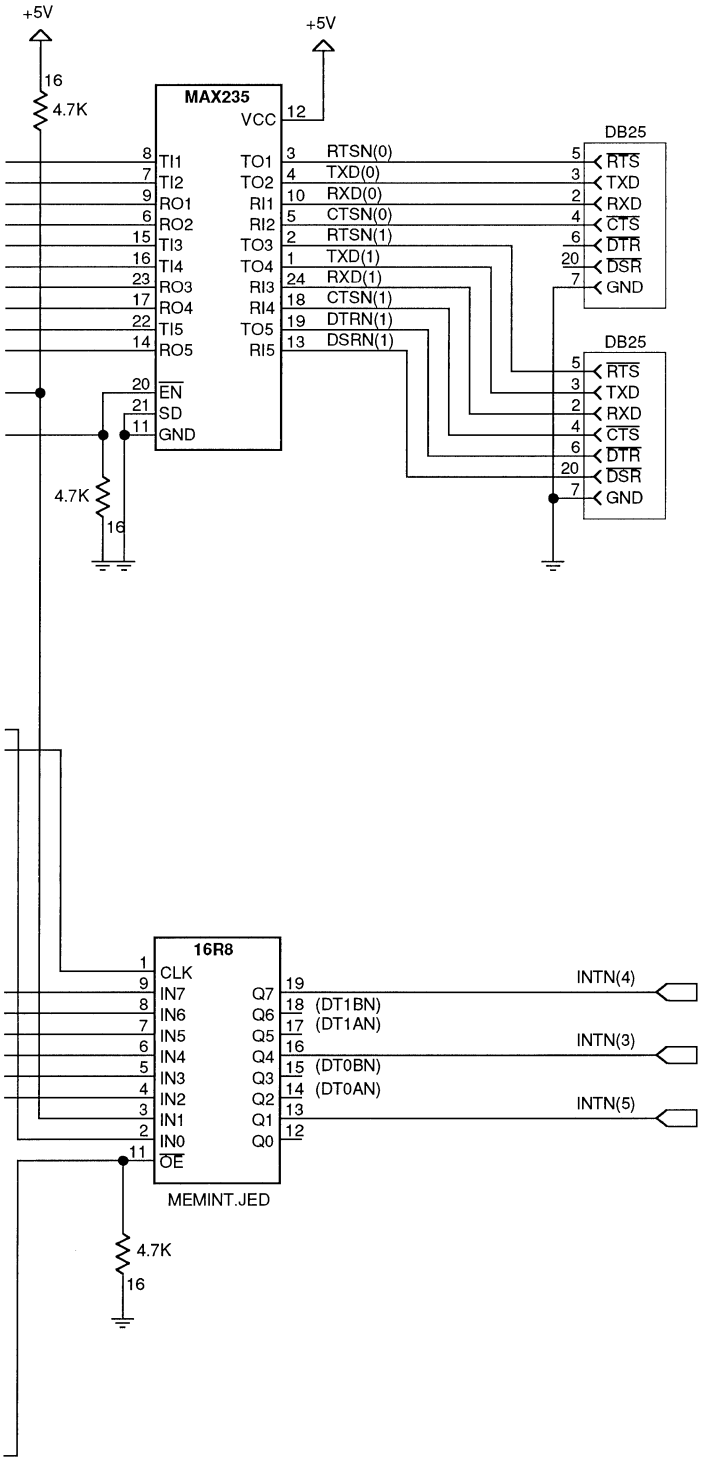


Figure 14. Input/Output Devices

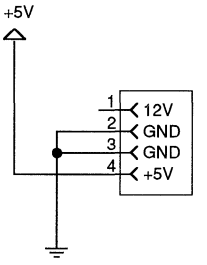


Figure 15. Power Connector

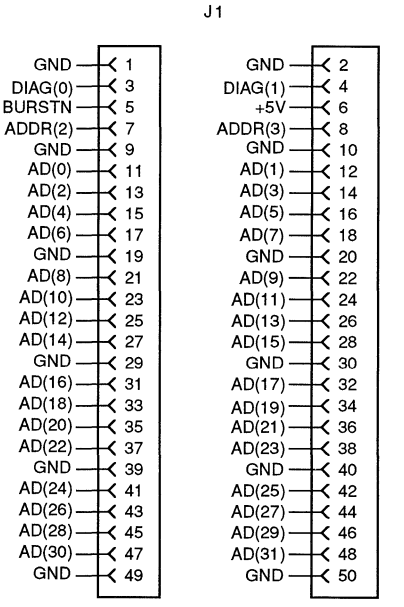


Figure 16. 50-Pin Connector

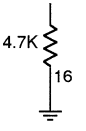


Figure 17. Spares

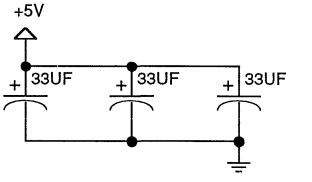


Figure 18. Primary Power Decoupling Capacitors

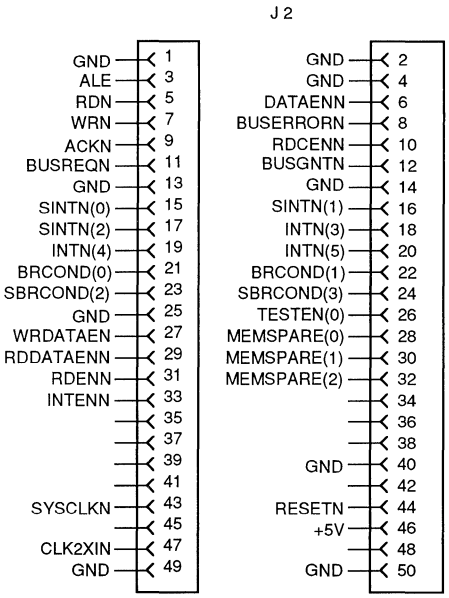


Figure 19. 50-Pin Connector

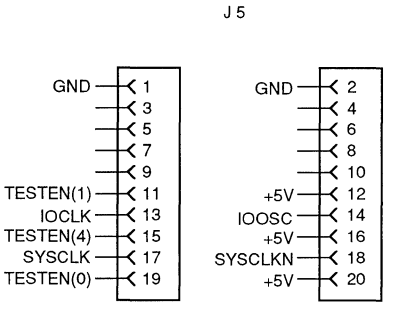


Figure 20. 100 Mil Jumper Headers

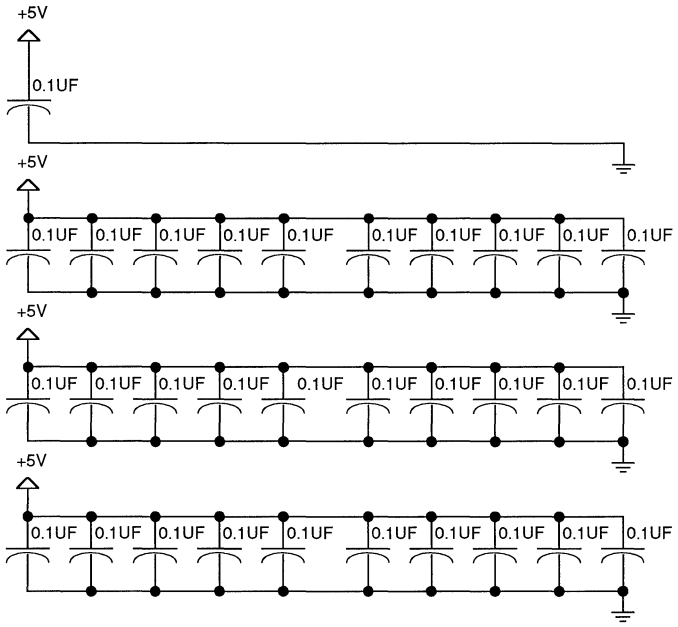


Figure 21. Decoupling Capacitors

```
{ TITLE   : MEMDEC.LPLC
            UPAL1 MEMORY AND I/O ADDRESS DECODER PAL FOR THE R305X
            BEHAVIORAL BUS EMULATOR MEMORY EVALUATION BOARD
  PURPOSE : DECODES DEMULTIPLEXED ADDRESS TO GENERATE CHIP SELECTS.
  LANG    : LPLC - TM OF CAPILANO COMPUTING SYSTEMS
  AUTHOR  : ANDY NG, IDT INC.
  UPDATES : C2503 03-18-91 AP NOTE FIRST RELEASE
}

MODULE UPAL1     ;
TITLE  UPAL1     ;
TYPE   AMD 22V10 ;

INPUTS ;
        {  DEMULTIPLEXED MEMORY ADDRESS LINES }
        A17             NODE[PIN1]  ;          { MSB ADDRESS LINES 31-17       }
        A18             NODE[PIN2]  ;
        A19             NODE[PIN3]  ;
        A20             NODE[PIN4]  ;
        A21             NODE[PIN5]  ;
        A22             NODE[PIN6]  ;
        A23             NODE[PIN7]  ;
        A24             NODE[PIN8]  ;
        A25             NODE[PIN9]  ;
        A26             NODE[PIN10] ;
        A27             NODE[PIN11] ;
        A28             NODE[PIN13] ;


        { OUTPUT FEEDBACK NODES (NEEDED FOR LPLC'ISM) }
        A29             NODE[PIN16] ;
        A30             NODE[PIN15] ;
        A31             NODE[PIN14] ;
        MEMSPARE0       NODE[PIN19] ;
        MEMSPARE1       NODE[PIN18] ;
        MEMSPARE2       NODE[PIN17] ;

OUTPUTS ; { ATTRIBUTES C - COMBINATIONAL, R - REGISTERED, H - HIGH, L - LOW }

        { CHIP SELECTS }
        RAMCSN          NODE[PIN23] ATTR[CL] ;  { STATIC RAM CHIP SELECT        }
        EPROMCSN        NODE[PIN22] ATTR[CL] ;  { EPROM CHIP SELECT             }
        UARTCSN         NODE[PIN21] ATTR[CL] ;  { UNGATED UART CHIP SELECT      }
        TIMERCSN        NODE[PIN20] ATTR[CL] ;  { TIMER CHIP SELECT             }

        { I/O PINS USED AS INPUTS }
        A29             NODE[PIN14] ATTR[CL] ;  { MSB ADDRESS LINES 31-17       }
        A30             NODE[PIN15] ATTR[CL] ;
        A31             NODE[PIN16] ATTR[CL] ;
        MEMSPARE0       NODE[PIN19] ATTR[CL] ;
        MEMSPARE1       NODE[PIN18] ATTR[CL] ;
        MEMSPARE2       NODE[PIN17] ATTR[CL] ;

        { OUTPUT ENABLES }
        RAMCSNEN        NODE[PIN23EN] ;
        EPROMCSNEN      NODE[PIN22EN] ;
        UARTCSNEN       NODE[PIN21EN] ;
```

```
        TIMERCSNEN      NODE[PIN20EN] ;
        A29EN           NODE[PIN14EN] ;
        A30EN           NODE[PIN15EN] ;
        A31EN           NODE[PIN16EN] ;
        MEMSPARE0EN     NODE[PIN19EN] ;
        MEMSPARE1EN     NODE[PIN18EN] ;
        MEMSPARE2EN     NODE[PIN17EN] ;

        { ASYNCHRONOUS RESET AND SYNCHRONOUS PRESET NODES }
        RESETEN         NODE[RESET]   ;
        PRESETEN        NODE[PRESET]  ;


        { 7RS382 COMPATIBLE PHYSICAL ADDRESS DECODE MAP }
        {    RAM       00000000H - 0001FFFFH    32K    }
        {    EPROM     1FC00000H - 1FC1FFFFH    32K    }
        {    UART      1FE00000H - 1FE0003FH           }
        {    TIMER     1F800000H - 1F80002CH           }


TERMS ; { LPLC "TABLE" ALGORITHM TAKES TOO LONG TO COMPILE }

        { NOTES: MEMSPARE0 IS BEING USED FOR A BOARD CHIP SELECT
                 DRIVABLE BY ANOTHER MEMORY SYSTEM.  WITHOUT IT
                 ASSERTED LOW, THIS BOARD WILL NOT ISSUE ANY MEMORY
                 SIGNALS NOR OUTPUT ENABLE SHARED CONTROL PINS.          }
        { NOTES: MEMSPARE1 IS NOT BEING USED.  IT COULD BE USED AS AN
                 OUTPUT IF IT OR THE UPAL2 OUTPUT IT IS CONNECTED TO IS
                 TRISTATED.                                              }
        { NOTES: MEMSPARE2 IS BEING USED AS A TESTEN INPUT PIN TO
                 TRISTATE THE OUTPUTS DURING BOARD TESTING.  ANOTHER
                 USE WOULD BE FOR A BOARD CHIP SELECT - MEMCSN.
                 MEMSPARE2 IS CONNECTED TO A UPAL3 INPUT PIN.            }

        { I/O PINS USED ONLY AS INPUTS }
        A29EN             = 0 ;
        A30EN             = 0 ;
        A31EN             = 0 ;
        MEMSPARE0EN       = 0 ;
        MEMSPARE1EN       = 0 ;
        MEMSPARE2EN       = 0 ;
        A29           NOT = 0 ;
        A30           NOT = 0 ;
        A31           NOT = 0 ;
        MEMSPARE0     NOT = 0 ;
        MEMSPARE1     NOT = 0 ;
        MEMSPARE2     NOT = 0 ;

        { RESET AND PRESET ARE NOT USED IN THIS PAL. }
        RESETEN  = 0 ;
        PRESETEN = 0 ;


        RAMCSNEN          = !MEMSPARE2 ;
        RAMCSN NOT        = !MEMSPARE0 AND
                                !A31 AND !A30 AND !A29 AND !A28
                             AND !A27 AND !A26 AND !A25 AND !A24
```

```
                              AND !A23 AND !A22 AND !A21 AND !A20
                              AND !A19 AND !A18 AND !A17
        ;


        EPROMCSNEN        = !MEMSPARE2 ;
        EPROMCSN NOT      = !MEMSPARE0 AND
                                 !A31 AND !A30 AND !A29 AND  A28
                              AND  A27 AND  A26 AND  A25 AND  A24
                              AND  A23 AND  A22 AND !A21 AND !A20
                              AND !A19 AND !A18 AND !A17
        ;


        UARTCSNEN         = !MEMSPARE2 ;
        UARTCSN NOT       = !MEMSPARE0 AND
                                 !A31 AND !A30 AND !A29 AND  A28
                              AND  A27 AND  A26 AND  A25 AND  A24
                              AND  A23 AND  A22 AND  A21 AND !A20
                              AND !A19 AND !A18 AND !A17
        ;


        TIMERCSNEN        = !MEMSPARE2 ;
        TIMERCSN NOT      = !MEMSPARE0 AND
                                 !A31 AND !A30 AND !A29 AND  A28
                              AND  A27 AND  A26 AND  A25 AND  A24
                              AND  A23 AND !A22 AND !A21 AND !A20
                              AND !A19 AND !A18 AND !A17
        ;


END;
END UPAL1.
```

```
{ TITLE   : MEMCONT.LPLC
            UPAL2 MEMORY CONTROLLER PAL FOR THE R305X BEHAVIORAL BUS EMULATOR
            MEMORY EVALUATION BOARD
  PURPOSE: PRODUCES READ, WRITE, AND BUS ERROR ACKNOWLEDGE CONTROLS (RDCENN,
            ACKN, BUSERRORN) BASED ON A 4 OR 5 BIT COUNTER AND CYCLE END
            STALL CYCLE (WAIT STATE) EQUATIONS.
  LANG    : LPLC – TM OF CAPILANO COMPUTING SYSTEMS
  AUTHOR : ANDY NG, IDT INC.
  UPDATES: C4B76 03-18-91 AP NOTE SECOND RELEASE
}

MODULE UPAL2      ;
TITLE  UPAL2      ;
TYPE   AMD 22V10 ;

INPUTS ;
        { REGULAR INPUT PINS }
        SYSCLK          NODE[PIN1]  ;          { UN-INVERTED SYSTEM CLOCK      }
        RESETN          NODE[PIN2]  ;          { MASTER RESET                  }
        RDN             NODE[PIN3]  ;          { READ                          }
        WRN             NODE[PIN4]  ;          { WRITE                         }
        BURSTN          NODE[PIN5]  ;          { BURST READ | WRITE NEAR       }
        RAMCSN          NODE[PIN6]  ;          { RAM CHIP SELECT               }
        EPROMCSN        NODE[PIN7]  ;          { EPROM CHIP SELECT             }
        UARTCSN         NODE[PIN8]  ;          { UART CHIP SELECT              }
        TIMERCSN        NODE[PIN9]  ;          { TIMER CHIP SELECT             }
        MEMSPARE0       NODE[PIN10] ;          {                               }
        MEMSPARE2       NODE[PIN11] ;          {                               }
        TESTEN          NODE[PIN13] ;          { TEST PIN TO Z-STATE OUTPUTS   }

        { REGISTER FEEDBACK PINS }
        C WIDTH[5]      NODE[PIN15,PIN14,PIN21,PIN22,PIN23] ;
        ENSTARTN        NODE[PIN16] ;
        CYCENDN         NODE[PIN18] ;
        RDCENN          NODE[PIN19] ;
        ACKN            NODE[PIN20] ;
        BUSERRORN       NODE[PIN17] ;


OUTPUTS ; { ATTRIBUTES C – COMBINATIONAL, R – REGISTERED, H – HIGH, L – LOW  }

        { REGISTERED OUTPUT PINS }
        { BINARY UP COUNTER INPUTS MSB TO LSB C4, C3, C2, C1, C0 }
        C WIDTH[5]      NODE[PIN15,PIN14,PIN21,PIN22,PIN23] ATTR[RL] ;
        ENSTARTN        NODE[PIN16] ATTR[RL] ;  { READ/WRITE OUTPUT ENABLE START }
        CYCENDN         NODE[PIN18] ATTR[RL] ;  { CYCLE END (COMPOSITE ACK)      }
        RDCENN          NODE[PIN19] ATTR[RL] ;  { R305X READ BUFFER CLOCK ENABLE }
        ACKN            NODE[PIN20] ATTR[RL] ;  { R3050X ACKNOWLEDGE             }
        BUSERRORN       NODE[PIN17] ATTR[RL] ;  { R305X BUS ERROR                }

        { OUTPUT ENABLES }
        CEN WIDTH[5]    NODE[PIN15EN,PIN14EN,PIN21EN,PIN22EN,PIN23EN] ;
        ENSTARTNEN      NODE[PIN16EN] ;
        CYCENDNEN       NODE[PIN18EN] ;
        RDCENNEN        NODE[PIN19EN] ;
        ACKNEN          NODE[PIN20EN] ;
        BUSERRORNEN     NODE[PIN17EN] ;
```

```
        { ASYNCHRONOUS RESET AND SYNCHRONOUS PRESET NODES }
        RESETEN         NODE[RESET]  ;
        PRESETEN        NODE[PRESET] ;


TABLE ;

        { RESET AND PRESET ARE NOT BEING USED.                                  }
        RESETEN  = 0 ;
        PRESETEN = 0 ;

        { PURPOSE: PROVIDES REGISTERED VERSION OF RDN AND WRN.

          NOTE:    QRDN AND QWRN ARE KEPT LOW ONE EXTRA CLOCK BY CYCENDN.
                   THIS IS BECAUSE THE RISING EDGE OF RDN OR WRN MAY NOT
                   HAVE ENOUGH HOLD TIME FROM THE RISING EDGE OF
                   (BUFFERRED) SYSCLK.
          NOTE:    QRDN AND QWRN DO NOT NECESSARILY TRANSITION BACK HIGH
                   BETWEEN CONSECUTIVE MEMORY CYCLES, E.G., WRITE FOLLOWED
                   BY A WRITE.                                                  }

        { QRDN NOT      :=   RESETN AND (!RDN OR (!QRDN AND !CYCENDN)) ; }
        { QWRN NOT      :=   RESETN AND (!WRN OR (!QWRN AND !CYCENDN)) ; }


        { PURPOSE: C[4]-C[0] PROVIDES A 5-BIT BINARY UP COUNTER.  IT IS RESET
                   ANYTIME RESETN IS ASSERTED AND AT THE END
                   OF EVERY MEMORY CYCLE AFTER CYCENDN IS ASSERTED.
                   IT BEGINS COUNTING UP WHEN A READ OR WRITE CYCLE IS
                   INITIATED.
          NOTE:    CYCENDN IS ASSUMED TO ASSERT WITH THE LAST RDCENN
                   ON READS AND WITH ACKN ON WRITES.  THUS CYCENDN WILL CLEAR
                   THE COUNTER WHETHER OR NOT RDN OR WRN HIGH TRANSITION
                   MEETS THE REGISTER SETUP AND HOLD TIME REQUIREMENTS.         }

        { NOTE:    TO ADD A GENERAL PURPOSE READY (A.K.A. BUSYN AND WAITN)
                   INPUT, CHANGE EACH OF THE COUNTER C[4:0] EQUATIONS SO
                   THAT THEIR VALUE CAN BE HELD WITH AN ADDITIONAL TERM, E.G.:
                   C[0]    :=   RESETN AND CYCENDN AND (!RDN OR !WRN)
                                  AND (    (C[0] XOR 1)
                                        OR (C[0] AND !READY) ) ;
                   A READY INPUT CAN BE USED FOR DUAL-PORT MEMORY INTERFACING,
                   EEPROM WRITE INTERFACING, ETC.
        }

        CEN[0]   =   !TESTEN ;
        CEN[1]   =   !TESTEN ;
        CEN[2]   =   !TESTEN ;
        CEN[3]   =   !TESTEN ;
        CEN[4]   =   !TESTEN ;

        C[0]    :=   RESETN AND CYCENDN AND (!RDN OR !WRN)
                        AND (C[0] XOR 1) ;
        C[1]    :=   RESETN AND CYCENDN AND (!RDN OR !WRN)
                        AND (C[1] XOR C[0]) ;
        C[2]    :=   RESETN AND CYCENDN AND (!RDN OR !WRN)
                        AND (C[2] XOR (C[1] AND C[0])) ;
```

```
C[3]     :=   RESETN AND CYCENDN AND (!RDN OR !WRN)
                 AND (C[3] XOR (C[2] AND C[1] AND C[0])) ;
C[4]     :=   RESETN AND CYCENDN AND (!RDN OR !WRN)
                 AND (C[4] XOR (C[3] AND C[2] AND C[1] AND C[0])) ;


{ PURPOSE: ENSTARTN OUTPUT PROVIDES THE TIMING FOR THE LEADING
           EDGE OF OEN AND WEN STROBES SO THAT 1. THE ADDRESS LINES HAVE
           TIME TO BE DECODED AND 2. OE/DATA PINS HAVE TIME TO Z-STATE
           FROM READS ON THE PRECEDING CYCLE.  THE CYCENDN TERM IS
           NEEDED TO HOLD OFF A CONSECUTIVE MEMORY CYCLE, E.G., WHEN
           WRITE DEASSERTS AND REASSERTS WITHIN THE SAME CLOCK.
           ENSTARTN SHOULD NOT BE USED TO END WRITE TRANSCEIVER
           ENABLES AS IT DEASSERTS WITH THE WRITE LINE INSTEAD OF
           HOLDING FOR ONE MORE 1/2 CLOCK.                                  }

ENSTARTNEN      = !TESTEN ;
ENSTARTN  NOT := !MEMSPARE0 AND RESETN AND (C >= 1) AND CYCENDN ;


{ PURPOSE: CYCLE END GOES LOW (SYNCHRONOUSLY) DURING THE LAST RDCENN ON
           READS AND DURING ACKN ON WRITES.  IT RETURNS HIGH
           SYNCHRONOUSLY BY INTERLOCKING ON THE COUNTER OUTPUTS
           WHICH COUNT ONE GREATER THAN THE ASKED FOR VALUE BEFORE
           RESETTING BACK TO ZERO (VIA CYCENDN).  THUS CYCENDN WILL
           DEASSERT ON THE SAME CLOCK AS THE RDN, WRN, OR BURSTN RISING
           EDGES REGARDLESS OF WHETHER OR NOT THOSE RISING EDGES MEET
           THE REGISTER'S SETUP AND HOLD TIMES.                             }
{ NOTE:    TO FIT CYCENDN INTO A 16V8, TWO OUTPUTS MAY BE NEEDED.           }

CYCENDNEN     = !TESTEN ;
CYCENDN NOT := RESETN AND CYCENDN AND (
                   (!RAMCSN    AND (C == 02H) AND !RDN AND  BURSTN)
               OR (!RAMCSN     AND (C == 08H) AND !RDN AND !BURSTN)
               OR (!RAMCSN     AND (C == 03H) AND !WRN             )
               OR (!EPROMCSN AND (C == 03H) AND !RDN AND  BURSTN)
               OR (!EPROMCSN AND (C == 0CH) AND !RDN AND !BURSTN)
               OR (!UARTCSN   AND (C == 06H)                       )
               OR (!TIMERCSN AND (C == 06H)                        )
               OR ( {!BUSERRORN} (C == 1FH)                        )
);


{ NOTE: IN THIS EXPERIMENT MEMSPARE0 IS PULLED LOW AND CAN BE
        USED TO DISABLE THIS CONTROLLER'S RDCENN, ACKN, AND BUSERRORN.
        SINCE MEMSPARE0 IS ATTACHED TO THE MEMDEC.LPLC PAL, THE
        MEMDEC PAL COULD COMBINE THE CSN'S SO THAT THESE SIGNALS
        ARE ONLY DRIVEN WHEN NEEDED.                                        }

{ NOTE: ANOTHER POSSIBILITY IS TO USE MEMSPARE0 AS AN EXTRA CHIP
        SELECT.                                                             }


{ PURPOSE: READ BUFFER CLOCK ENABLE IS USED BY THE R305X TO STROBE
           DATA INTO ITS INTERNAL READ BUFFERS.                             }
{ NOTE:    IT IS ASSUMED THAT THE UART AND TIMER ARE
           IN UNCACHABLE MEMORY SPACE AND WILL NOT BE BURST READ.
           IF THEY ARE BURST READ, THE STATE MACHINE LOOPS 4 TIMES.         }
```

```
RDCENNEN      = !MEMSPARE0 ;
RDCENN NOT   := RESETN AND CYCENDN AND (
                  (!RAMCSN AND !RDN
                     AND (                  (C == 02H)
                           OR (!BURSTN AND (C == 04H))
                           OR (!BURSTN AND (C == 06H))
                           OR (!BURSTN AND (C == 08H))
                         )
                  )
               OR (!EPROMCSN AND !RDN
                     AND (                  (C == 03H)
                           OR (!BURSTN AND (C == 06H))
                           OR (!BURSTN AND (C == 09H))
                           OR (!BURSTN AND (C == 0CH))
                         )
                  )
               OR (!UARTCSN AND !RDN
                     AND (                  (C == 06H)
                         )
                  )
               OR (!TIMERCSN AND !RDN
                     AND (                  (C == 06H)
                         )
                  )
);


{ PURPOSE: ACKNOWLEDGE IS PRIMARILY USED TO END WRITE CYCLES.  IT
           SHOULD BE PULSED ONE (HALF) CLOCK CYCLE BEFORE THE WRITE
           STROBE IS NEEDED.  ON READ CYCLES, ACKNOWLEDGE WILL
           IMPLICITLY BE GENERATED BY THE R305X, HOWEVER, IF OPTIMAL
           TIMING IS DESIRED, ACK SHOULD BE DRIVEN NO SOONER THAN 1
           CLOCK BEFORE THE END OF A SINGLE READ AND FOR BURSTS NO
           SOONER THAN 4 CLOCKS BEFORE THE END OF THE LAST READ.        }

ACKNEN        = !MEMSPARE0 ;
ACKN NOT     := RESETN AND CYCENDN AND (
                  (!RAMCSN AND !WRN                          { WRITE CYCLE }
                     AND (                  (C == 03H)
                         )
                  )
                OR (!RAMCSN AND !RDN AND !BURSTN             { READ  CYCLE }
                     AND (                  (C == 05H)
                         )
                  )
                OR (!EPROMCSN AND !RDN AND !BURSTN           { READ  CYCLE }
                     AND (                  (C == 09H)
                         )
                  )
                OR (!UARTCSN AND !WRN                        { WRITE CYCLE }
                     AND (                  (C == 06H)
                         )
                  )
```

```
                OR (!TIMERCSN AND !WRN                          { WRITE CYCLE }
                      AND (                 (C == 06H)
                           )
                   )
);


{ PURPOSE: BUSERRORN SIMPLY ENDS A WAYWARD UNDECODED BUS CYCLE.  ON
           READS IT CAUSES AN EXCEPTION.  ON WRITES IT DOES NOT CAUSE
           AN EXCEPTION CONDITION FOR THE PROCESSOR.  TO DO THAT, LATCH
           BUSERRORN AND FEED IT TO AN INTERRUPT PIN OR A BRANCH
           CONDITION PIN.                                              }

BUSERRORNEN     = !MEMSPARE0 ;
BUSERRORN NOT := RESETN AND CYCENDN AND (
                                        (C == 1FH)
```

```
{ TITLE   : MEMEN.LPLC
            UPAL3 MEMORY READ AND WRITE ENABLE PAL FOR THE R305X BEHAVIORAL BUS
            EMULATOR MEMORY EVALUATION BOARD
  PURPOSE : GENERATES READ AND WRITE ENABLES FOR MEMORY CONTROLS.
  LANG    : LPLC - TM OF CAPILANO COMPUTING SYSTEMS
  AUTHOR  : ANDY NG, IDT INC.
  UPDATES : C7C4F 03-18-91 AP NOTE FIRST RELEASE
}

MODULE UPAL3     ;
TITLE  UPAL3     ;
TYPE   AMD 22V10 ;

INPUTS ;
        {  DEMULTIPLEXED MEMORY ADDRESS LINES }
        SYSCLK          NODE[PIN1]  ;          { INVERTED SYSCLKN            }
        POWRESETN       NODE[PIN2]  ;          { POWER UP RESET              }
        RDN             NODE[PIN3]  ;          { READ LINE                   }
        WRN             NODE[PIN4]  ;          { WRITE LINE                  }
        ENSTARTN        NODE[PIN5]  ;          { ENABLE START                }
        CYCENDN         NODE[PIN6]  ;          { CYCLE END                   }
        BEN0            NODE[PIN7]  ;          { BYTE ENABLE 0               }
        BEN1            NODE[PIN8]  ;          { BYTE ENABLE 1               }
        BEN2            NODE[PIN9]  ;          { BYTE ENABLE 2               }
        BEN3            NODE[PIN10] ;          { BYTE ENABLE 3               }
        UARTCSN         NODE[PIN11] ;          { UART CHIP SELECT            }
        MEMSPARE2       NODE[PIN13] ;          { SPARE INPUT                 }

        { OUTPUT FEEDBACK NODES (NEEDED FOR LPLC'ISM) }
        RESETN          NODE[PIN23] ;
        WRENN           NODE[PIN18] ;
        WRDATAEN        NODE[PIN17] ;

OUTPUTS ; { ATTRIBUTES C - COMBINATIONAL, R - REGISTERED, H - HIGH, L - LOW }

        { WRITE ENABLES }
        WRENNA          NODE[PIN22] ATTR[RL] ;  { WRITE ENABLE FOR BYTE 0     }
        WRENNB          NODE[PIN21] ATTR[RL] ;  { WRITE ENABLE FOR BYTE 1     }
        WRENNC          NODE[PIN20] ATTR[RL] ;  { WRITE ENABLE FOR BYTE 2     }
        WRENND          NODE[PIN19] ATTR[RL] ;  { WRITE ENABLE FOR BYTE 3     }
        WRENN           NODE[PIN18] ATTR[RL] ;  { WRITE ENABLE MOTO-TYPE I/O  }
        WRDATAEN        NODE[PIN17] ATTR[RL] ;  { WRITE DATA XCEIVER ENABLE   }

        { READ ENABLES }
        RDENN           NODE[PIN16] ATTR[RL] ;  { READ OUTPUT ENABLE (FOR WORDS)}
        RDDATAENN       NODE[PIN15] ATTR[RL] ;  { READ DATA XCEIVER ENABLE    }

        { MISCELLANEOUS CONTROLS }
        RESETN          NODE[PIN23] ATTR[RL] ;  { SYNCHRONIZED RESET          }
        GUARTCSN        NODE[PIN14] ATTR[RL] ;  { GATED/GUARDED UART CHIP SELECT}

        { I/O PINS USED AS INPUTS }
        { NONE }

        { OUTPUT ENABLES }
        WRENNAEN        NODE[PIN22EN] ;
        WRENNBEN        NODE[PIN21EN] ;
```

```
        WRENNCEN        NODE[PIN20EN] ;
        WRENNDEN        NODE[PIN19EN] ;
        WRENNEN         NODE[PIN18EN] ;
        WRDATAENEN      NODE[PIN17EN] ;
        RDENNEN         NODE[PIN16EN] ;
        RDDATAENNEN     NODE[PIN15EN] ;
        RESETNEN        NODE[PIN23EN] ;
        GUARTCSNEN      NODE[PIN14EN] ;

        { ASYNCHRONOUS RESET AND SYNCHRONOUS PRESET NODES }
        RESETEN         NODE[RESET]   ;
        PRESETEN        NODE[PRESET]  ;

TABLE ;

        { RESET AND PRESET ARE NOT USED IN THIS PAL. }
        RESETEN  = 0 ;
        PRESETEN = 0 ;


        { PURPOSE: WRITE BYTE ENABLES AND WRITE WORD ENABLE ALLOW
                   SUFFICIENT TIME FOR THE ADDRESS TO DECODE AND
                   FOR A VALID CHIP SELECT BEFORE ENABLING THE
                   WRITE STROBE FOR A SPECIFIC BYTE BANK.
          NOTE:    BANK A IS THE BIG ENDIAN'S LSB BYTE3 OR THE LITTLE
                   ENDIAN'S LSB BYTE0.  IT ALWAYS HOLDS D(7:0).
                   BANK D IS THE BIG ENDIAN'S MSB BYTE0 OR THE BIG
                   ENDIAN'S MSB BYTE3.  IT ALWAYS HOLDS D(31:23).
                                                                    }

        WRENNAEN          = !MEMSPARE2 ;
        WRENNA       NOT := RESETN AND (
                               !WRN AND !BEN0 AND !ENSTARTN AND CYCENDN
        );

        WRENNBEN          = !MEMSPARE2 ;
        WRENNB       NOT := RESETN AND (
                               !WRN AND !BEN1 AND !ENSTARTN AND CYCENDN
        );

        WRENNCEN          = !MEMSPARE2 ;
        WRENNC       NOT := RESETN AND (
                               !WRN AND !BEN2 AND !ENSTARTN AND CYCENDN
        );

        WRENNDEN          = !MEMSPARE2 ;
        WRENND       NOT := RESETN AND (
                               !WRN AND !BEN3 AND !ENSTARTN AND CYCENDN
        );

        { PURPOSE: WRENN IS USED TO PROVIDE A WRITE LINE THAT HOLDS
                   LOW FOR AN EXTRA CYCLE, SO THAT IT CAN BE USED FOR
                   MOTOROLA-TYPE I/O DEVICES ON THEIR MULTIPLEXED
                   READ/WRITE LINE.                                 }

        WRENNEN           = !MEMSPARE2 ;
        WRENN        NOT := RESETN AND (
```

```
                    (!WRN AND CYCENDN)
                OR (!WRENN AND !CYCENDN)
);

{ PURPOSE: WRDATAEN AND RDDATAENN DRIVE THE OUTPUT ENABLE
           CONTROLS ON A FCT623T TRANSCEIVER BANK FOR THE
           DATA BUS.  THE CONTROLS CAN BE USED FOR ANY
           DUAL-OUTPUT ENABLE TRANSCEIVER (1 FOR EACH
           DIRECTION.  OUTPUT ENABLE/DIRECTION CONTROLLED
           TRANSCEIVERS (FCT245) REQUIRE MORE INTERFACING
           IF OUTPUT CONTENTION IS TO BE AVOIDED BY
           ONLY CHANGING THE DIRECTION WHEN THE OUTPUTS ARE
           DISABLED.                                        }

{ NOTE:    WRITE DATA ENABLE DEASSERTS ONE CLOCK AFTER
           WRN DOES TO PROVIDE SUFFICIENT HOLD TIME FOR THE
           WRITE DATA INTO THE MEMORY (SEE UPAL2 QWRN FOR A
           MORE DETAILED EXPLANATION).
  NOTE:    WRDATAEN IS ACTIVE HIGH FOR THE FCT623T OUTPUT ENABLE
           CONTROL.  FOR THE FCT861 OUTPUT ENABLES, USE ACTIVE
           LOW.
  NOTES:   THE FIRST OR-TERM ASSERTS WRDATAEN WHILE THE SECOND
           OR-TERM DEASSERTS WRDATAEN.                      }

WRDATAENEN       = !MEMSPARE2 ;
WRDATAEN        := RESETN AND (
                       (!WRN AND !ENSTARTN)
                   OR (WRDATAEN AND (!ENSTARTN OR !CYCENDN))
);

RDENNEN          = !MEMSPARE2 ;
RDENN       NOT := RESETN AND (
                       !RDN AND !ENSTARTN AND CYCENDN
);

{ PURPOSE: RDDATAENN IS CONNECTED TO THE MEMORY BOARD'S
           DATA TRANSCEIVER OUTPUT ENABLE (FCT623T OR FCT861)
           AND ONLY ENABLES FOR THIS BOARD'S CHIP SELECTS.
           IF THE MEMORY CONTROLLER IS USED FOR ANOTHER
           BOARD'S MEMORY, THEN THE TRANSCEIVER OUTPUT ENABLE
           SHOULD BE DISABLED FOR THOSE CHIP SELECTS (VIA
           MEMSPARE2.                                       }

{ NOTE:    IN MOST SYSTEMS, R305X'S DATAENN OUTPUT CAN BE
           CONNECTED DIRECTLY TO THE TRANSCEIVER ENABLE PIN
           INSTEAD OF USING A SYNTHESIZED RDDATAENN.        }

RDDATAENNEN      = !MEMSPARE2 ;
RDDATAENN   NOT := RESETN AND (
                       !RDN AND !ENSTARTN AND CYCENDN
);


{ PURPOSE: RESET SYNCHRONIZES THE POWER UP RESET FOR THE
           MEMORY CONTROLLER STATE MACHINES AND FOR THE R305X.  }

RESETNEN         = !MEMSPARE2 ;
RESETN      NOT := !POWRESETN ;
```

```
        { PURPOSE: GUARDED/GATED UART CHIP SELECT, GUARTCSN GATES
                   UARTCSN BECAUSE THE UART BEING USED HAS A MOTOROLA-
                   TYPE I/O DEVICE INTERFACE WHICH MULTIPLEXES ITS
                   READ/WRITE INPUT PIN SUCH THAT THE CHIP SELECT MUST
                   STROBE IN OR OUT DATA.  THIS IS IN CONTRAST TO AN
                   INTEL-TYPE I/O DEVICE INTERFACE WHICH WOULD HAVE A
                   SEPARATE READ STROBE AND WRITE STROBE AS WELL AS A
                   CHIP SELECT.  IT IS IMPORTANT NOT TO HAVE A
                   GLITCH (FROM ADDRESS DECODING THE CHIP SELECT) ON
                   READS IN ORDER TO ALLOW THE I/O DEVICE TO UPDATE
                   FIFO POINTERS, ETC.  THUS GUARTCSN STARTS LATE AND
                   ENDS EARLY, SO THAT READ/WRITE IS HELD VALID
                   THROUGHOUT THE CHIP SELECT.                          }

        GUARTCSNEN       = !MEMSPARE2 ;
        GUARTCSN    NOT := RESETN AND (
                               !UARTCSN AND !ENSTARTN AND CYCENDN
        );

END;
END UPAL3.
```

```
{ TITLE   : MEMINT.LPLC
            UPAL4 MEMORY I/O INTERRUPT CONTROLLER PAL FOR THE R305X BEHAVIORAL
            BUS EMULATOR MEMORY EVALUATION BOARD
  PURPOSE: REPLICATES THE TIMER/UART INTERRUPT CONTROLLER ON THE 7RS382 BOARD.
            ADDITIONAL FUSE BITS ADDED FOR 16V8 COMPATIBILITY.
  LANG    : LPLC – TM OF CAPILANO COMPUTING SYSTEMS
  AUTHOR : IDT INC.
  UPDATES: C3F98 01-04-91 16V8 PCB VERSION FIRST RELEASE A.N.
}


{ U24A_382 INTERRUPT PAL}
{ 1-2-90,12-14-89 }
{JEDEC file's CHECKSUM = 379E } { NOTE: 01-04-91 – NOT APPLICABLE TO 16V8 }

{     CONTROL PAL FOR 8254 TIMER'S AND UART INTERRUPT
      USED FOR EVALUATION BOARD 382 }

MODULE U24A_382;
TITLE U24A_382;
TYPE MMI 16R8;

{ FUSE BITS FOR 16V8 FAMILY ATTRIBUTES USED AS A 16R8 }
FUSE 2048..2079 00000000000000000000000000000000 ;
FUSE 2080..2111 00000000000000000000000000000000 ;
FUSE 2112..2143 00000000000000001111111111111111 ;
FUSE 2144..2175 11111111111111111111111111111111 ;
FUSE 2176..2193 111111111111111101               ;

INPUTS;

MRES/       NODE[PIN2];
UARTINT/    NODE[PIN3];
PMRD/       NODE[PIN4];
CSTIM/      NODE[PIN5];
EA02        NODE[PIN6];
EA04        NODE[PIN7];
OUT1        NODE[PIN8]; {input from Timer output OUT1}
OUT0        NODE[PIN9]; {input from Timer output OUT0}

DT0A/       NODE[PIN14]; {feedback}
DT0B/       NODE[PIN15]; {feedback}
T0INT/      NODE[PIN16]; {feedback}

DT1A/       NODE[PIN17]; {feedback}
DT1B/       NODE[PIN18]; {feedback}
T1INT/      NODE[PIN19]; {feedback}

OUTPUTS;

UINT5/      NODE[PIN13];
DT0A/       NODE[PIN14];
DT0B/       NODE[PIN15];
T0INT/      NODE[PIN16]; { goes to R3000's UINT3}

DT1A/       NODE[PIN17];
DT1B/       NODE[PIN18];
T1INT/      NODE[PIN19]; { goes to R3000's UINT4}
```

```
TABLE;
{     8254 TIMER generates 2 square-wave outputs OUT0 and OUT1.
      When OUT0 goes from high to low, this PAL asserts interrupt
      TOINT/, which will interrupt R3000 through UINT3.
      Same scheme applies to OUT1, T1INT/ and UINT4.
      Reading physical addresses 1F80 0010 and 1F80 0014 (which are
      virtual addresses BF80 0010 and BF80 0014 in this 382 board)
      will clear interrupt UINT3 and UINT4, respectively.

      This PAL also synchronizes UART interrupt signal }


DT0A/       :=     OUT0;       {delay TIMER's OUT0 through a register}
DT0B/       :=     DT0A/;      {delay again}
T0INT/ NOT  :=     MRES/ AND
                   ((NOT DT0A/ AND DT0B/) OR
                   (NOT T0INT/ AND (NOT EA04 OR EA02 OR CSTIM/ OR PMRD/)));

DT1A/       :=     OUT1;
DT1B/       :=     DT1A/;
T1INT/ NOT  :=     MRES/ AND
                   ((NOT DT1A/ AND DT1B/) OR
                   (NOT T1INT/ AND (NOT EA04 OR NOT EA02 OR CSTIM/ OR PMRD/)));

UINT5/      :=     UARTINT/ OR NOT MRES/ ;
                   {put UART's interrupt through a register to synchronize
                   it with R3000 clock }
END;
END U24A_382.
```

Integrated Device Technology, Inc.

R3051™ FAMILY PERFORMANCE IN EMBEDDED APPLICATIONS

APPLICATION NOTE AN-89

By V. S. Ramaprasad

INTRODUCTION

The IDTR3051™ is a family of RISC controllers specially suited for embedded applications. Instruction and data caches are integrated on the chip to yield cache hit rates of over 90% for a wide range of typical embedded applications. These RISC controllers also provide the designer with a simple interface to the rest of the system through built in read/write buffers, a multiplexed address/data bus and a small set of control signals. This simple interface enables the designer to select an optimal price/performance memory and I/O system.

In this application note the performance of a 33MHz R3051-based system is presented. Standard integer benchmarks are run on the software model of the R3051 DRAM-based sytem, and the results obtained are compared with the published results for 33MHz i960 and 33MHz 29k RISC processor-based systems. The performance of R3051-based systems can be attributed to the raw horse power of R3000A core coupled with the highly-desired optimal integration provided on the chip.

SYSTEM DESCRIPTION

The 33MHz R3051-based system modelled is made up of 80ns DRAMs with a page mode access time of 50ns. The refill sizes for both the caches is four. The processors burst mode of access is utilized for refilling both the caches on cache misses. This implies that after the initial latency cycles the 2-way interleaved main memory is capable of supplying the subsequent instructions or data at the processor speed. The instructions are streamed into the processor along with the on-chip cache refill.

The 33MHz R3051 system is modelled with a software simulation tool called Cache305x. This software is based on the Cache2000,™ which is part of the Systems Programmers Package developed by MIPS® Computer Systems, Inc. Cache2000 is used to model R3000/R3001-based systems with more than 98% accuracy of simulation.

To accurately model R3051-based systems, the existing Cache2000 is modified. Besides setting the cache sizes, the block refill sizes, the write buffer depth etc., sections are added to the Cache2000 program to simulate the bus priority scheme adopted by the R3051 family for processing the main memory transactions and to implement the read/write protocols. Memory transactions are listed here with descending order of priorities. DMA activity is assumed not to be present in these simulations.

1. Current transaction completes without pre-emption.
2. Instruction cache misses are processed.
3. Data residing in the four-deep write buffer is retired to the main memory.
4. Data cache misses are carried out next.

The read/write operations follow the priority scheme. The initiation of either of these transactions depends on the pending memory transaction requests. The built-in bus arbitration logic resolves the conflict for the memory bus following the above mentioned priority scheme. The arbitration unit operates in parallel with the execution core. The core could be executing instructions from the caches, while the bus arbitration unit is retiring the writes currently residing in the write buffer.

For instruction cache misses, in the best case where there is no write in progress, a read signal to the external memory is initiated one cycle after the core missed in the instruction cache. This extra cycle is for the arbitration unit to generate the read signal request. On top of this arbitration cycle, if a write is currently in progress, the processor stalls till the write operation is terminated. In this case, after a write operation a DRAM-based system needs to be precharged before the read operation. The first instruction is read into the processor after the initial read latency of the memory system. The remaining three instructions are read in three consecutive cycles. After the reads, the DRAM precharge cycles are added to the total cycle count.

For data cache misses, there is an extra penalty of flushing the contents of the write buffer besides the extra one cycle for the arbitration. The number of cycles it takes to flush the write buffer depends on the number of words that are resident and also whether they could be retired as idle writes, or page writes, or non page writes. In the current system that is modelled, four words of data is brought in on a cache miss. The first word is read into the processor after the initial read latency of the memory system. The remaining three words are read in the following three consecutive cycles. After the reads, the DRAM precharge cycles are added to the total cycle count.

The write buffer interface decouples the core processor from the external slow memory system. Writes are retired in parallel with the processor executing out of the caches. In this state of execution, write operations always win the arbitration, and continuously retires the writes. This parallel mode of operation gets terminated only when the write buffer is full and a store is pending or when the processor can no longer execute out the caches. Keeping in mind that our interest in these simulations is the total cycle count for the complete execution of the program, write operations contribute to the total cycle count only when the processor needs to read from the external memory or when the processor can not proceed

with the execution of a store instruction because the write buffer is currently full.

The penalty cycles due to writes delaying the processor external reads on cache misses are accounted for during the read transactions. When the write buffer is full and the processor is executing a store instruction, penalty cycles that would vacate a single entry in the write buffer is added. This is not the same as retiring a single write, but it is equivalent to four cycles. This is due to the availability of an extra data register that captures the data being vacated from the write buffer. If another store follows in this situation where the four entries of the write buffer are full and the extra data buffer that drives the bus is loaded, the penalty is that of retiring a write to the memory.

DRAM PARAMETERS

The memory system considered in this R3051 design is made up of 80ns DRAMs with page mode access time of 50ns. The other parameters of the DRAM that affect the access time in different modes of DRAM are the initial read latency cycles, number of cycles to perform a write operation when the DRAM is in idle mode, number of cycles to perform a read/write operation when the DRAM is in page mode, number of cycles to perform a write operation when the DRAM is not in page mode. The parameters are set to fixed values to model a DRAM system that works with R3051 running at 33MHz.

The initial read latency cycles at 33MHz is the summation of the cycles to win the internal arbitration (1 cycle), cycles for the DRAM controller to generate RAS/CAS signals and perform a random read from the DRAM (6 cycles). The first word of instruction/data is read in the fixup cycle (1 cycle). The remaining words are read in the three following cycles. It should be noted that the DRAM precharge cycles are part of the 6-cycle random read latency mentioned above.

The idle write latency is the number of cycles to retire a write when the DRAM is in idle mode. Using 80ns DRAMS this can be accomplished in 6 cycles. The page mode read or write operations can be completed in 3 cycles, while the non-page writes can be carried out in 6 cycles. In the current system that is modelled with Cache305x, DRAM RAS precharge cycles are added when a read follows a write operation.

DRAM Parameters @ 33MHz	
Read latency	7 Cycles
RAS precharge	3 Cycles
Idle write	6 Cycles
Page write	3 Cycles
Non-page write	6 Cycles

COMPETITION

In this application note two other RISC systems, namely the i960 and the 29k, are compared with the R3051 DRAM system.

The Intel i960CA system is the ASV960CA board running at 33MHz with 0 wait-state memory for instructions and 3 wait-state memory for the data. The memory is implemented with 15ns SRAM. Internally the i960CA has 1kB of instruction cache memory. The benchmarks are compiled with 1.35 GCC/960 (results obtained from Intel).

The 29k system is the YARCs card running at 33MHz using RevD AM29000. It has a 2 MB of instruction memory, and a 512kB of data memory. The memories are implemented with 35ns Static RAMs (results obtained from AMD).

STANDARD INTEGER BENCHMARKS

Several standard integer benchmarks are run on the 33MHz R3051-based system using the Cache305x. They are Quicksort, Bubblesort, Pi500, Anneal, Matmult, and Dhrystone1.1. (0) The suite was selected by Intel, these benchmarks are selected because of (1) the availability of results for two other RISC processors, namely the i960 and the 29k, and (2) though being small, they still provide an insight into the capability of the processor in embedded environments.

Quicksort performs sorting of 5000 elements of an integer array using a recursive algorithm.

Bubblesort manipulates and sorts an array of 500 elements after reading a file.

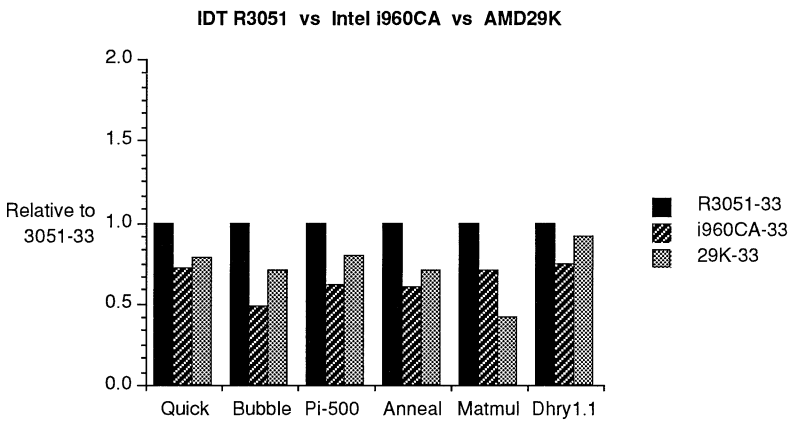
Pi500 computes the value of the mathematical constant 'Pi' upto 500 decimal points. This program does not use any floating point math, but more than 50% of the cycles are spent in integer multiplications and integer divisions.

Anneal program solves the travelling-salesman problem by the method of simulated annealing.

Matmult is a program that loops for 100 times, and in each loop it performs the multiplication of two 8 x 8 integer arrays. The result is stored in another 8 x 8 array.

Dhrystone 1.1 benchmark demonstrates the integer number crunching power of the processor, although it is susceptible to compiler optimizations. Dhrystone 1.1 is reported here for the R3051 system instead of Dhrystone 2.0 for lack of data for the i960 and the 29k.

All the above mentioned integer benchmarks are compiled with a C compiler version 2.0 on an M/120 system running RISC/os 4.0. Except for Dhrystone benchmark, all the other benchmarks are compiled with the highest level of optimization O4. This includes optimization techniques such as global register allocation, optimal calling sequences, common subexpression elimination, procedure merging/inlining etc. For Dhrystone, O3 level of optimization is used. This level of optimization does not include procedure merging as it is against the spirit of Dhrystone benchmarking.



Benchmark	IDT R3051-33	I960CA-33	29K-33
QUICKSORT (ms)	**36**	50	46
BUBBLESORT (ms)	**41**	85	59
PI-500 (ms)	**1,023**	1,624	1,282
ANNEAL (ms)	**5,056**	8,388	7,205
MATMULT (µs)	**19,148**	26,898	44,578
DHRYSTONE 1.1	**55,236**	41,030	50,301

* R3051 system is 80ns DRAM based system.
* i960CA-33 system is ASV960CA with 0ws for code and 3ws for data.
* 29k-33 system is YARC card with 35ns SRAMs.

The results for R3051 are listed below along with the results published for 33MHz i960 and 29K. The execution times for above mentioned programs are shown in the table (smaller values are better except for Dhrystone 1.1).

CONCLUSIONS

The standard integer benchmarks, even though they do not represent any real applications, provide an insight into the inherent performance of a processor when running typical embedded applications. The R3051 system considered here is a DRAM-based system, and still delivers more performance compared to the fastest i960CA and 29k-based designs. It can easily be deduced from the above data that the i960CA 33MHz system is actually equivalent to a 21.2MHz R3051-based system, and the 29k 33MHz system is equivalent to a 23.1MHz R3051-based system. Still faster R305x systems are feasible when designed with Static RAMs and it is reasonable to expect further gains in performance.

Integrated Device Technology, Inc.

DESIGNING A DISCRETE DRAM CONTROLLER FOR THE R3051 RISController™ FAMILY

APPLICATION NOTE AN-90

By Bob Napaa

INTRODUCTION

The IDT R3051™ RISController™ family utilizes a high-performance computing core to achieve high performance across a variety of applications. Further, the amount of cache incorporated in the R3051 family allow these CPUs to achieve very high performance even with simple, low-speed, low-cost memory subsystems.

The R3051 RISController CPU family includes a full R3000A core RISC processor, and thus is fully software compatible with the standard MIPS processor. In order to provide high-bandwidth to the CPU core, the family also incorporates on-chip up to 8kB of instruction cache and 2kB of data cache. The external memory interface from the R3051 family is very flexible, and allows a wide variety of implementations according to the price/performance goals of the application. For a detailed reference to the system interface of the R3051 family, the reader is advised to refer to the "R3051 Family Hardware User's Manual".

This applications note is a design example on the interface to a non-interleaved DRAM memory subsystem. The goals of this subsystem are to provide a simple, extensible memory interface using off-the-shelf components, and to illustrate basic design techniques for systems using an R3051 family CPU.

GENERAL DESCRIPTION OF THE DRAM SYSTEM

Figure 1 illustrates a typical system based on the R3051 RISController family. The R3051 family uses a double-frequency input clock for its internal operation and provides a nominal frequency reference clock output for the external system. This output clock, $\overline{\text{SysClk}}$, synchronizes the external memory subsystems to the R3051.

Memory transactions from the R3051 use a single, time multiplexed 32-bit address and data bus and a simple set of control signals. External logic then performs address demultiplexing and decoding, memory control, interface timing, and data path control.

The system shown in Figure 1 runs at 25MHz (2x clock = 50MHz). The R3051 interfaces to a DRAM system as the main

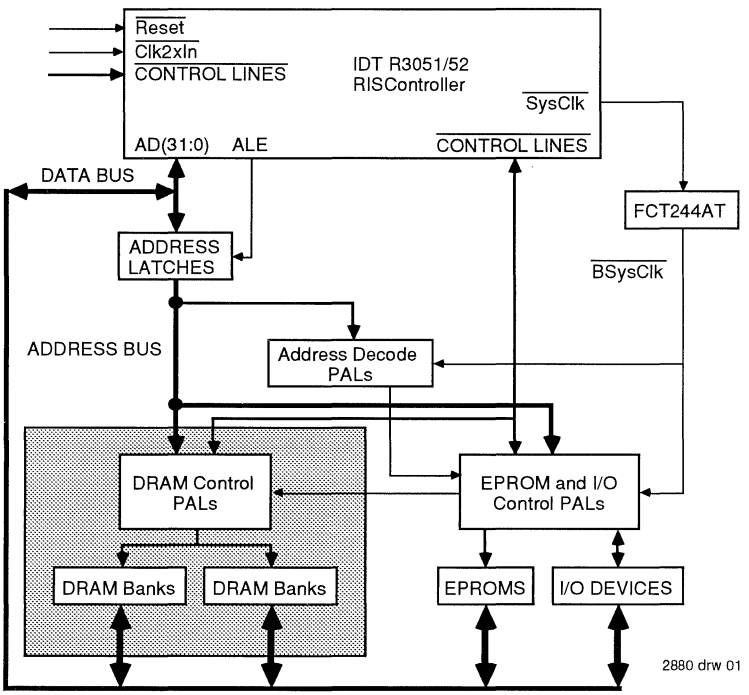


Figure 1. R3051 RISController Family Based System

memory, to an EPROM system and to various I/O devices and controllers. Address latches decouple the address bus from the data bus. Address decoders select among the various external modules. The output clock from the R3051 ($\overline{SysClk}$) is buffered ($\overline{BSysClk}$) to reduce the loading effect and to provide clock drive capability with minimum clock skew for the system. This applications note will focus on the DRAM control and data path subsystem.

The main DRAM memory system is based on 1 to 4 banks of non-interleaved DRAMs with 80ns of access time (t_{RAC} = 80ns). The density of the DRAMs used is 256K x 4 to provide a maximum memory space of 4MB. The DRAM memory space occupies the lower 4MB of the physical memory space (A21:A0). Figure 2 illustrates the architecture of the main DRAM memory system.

Table 1 illustrates the decoding scheme used in accessing the DRAM memory space. To simplify address decoding, software will insure that all references to the DRAM memory occur with address bit A(22) LOW, and thus only that bit will be used in the decoding. Address bits A(21:20) will select among the four banks, and the $\overline{Rd}$ and $\overline{Wr}$ outputs from the R3051 differentiate between read and write accesses.

Each 1MB bank of DRAMs is individually controlled by separate $\overline{RAS}$ and $\overline{CAS}$ control signals. Thus, each bank may be independently selected. The banks are arranged so that each bank represents a single, contiguous range of 1MB (as opposed to an interleaved memory structure).

Data buffers isolate the DRAM banks from the R3051 data bus to reduce the loading effect and to prevent any bus contentions between the R3051 and the DRAMs from occurring. Note that this also alleviates concerns about the relatively slow tri-state times associated with DRAM devices. The data buffers selected are actually bidirectional latching transceivers; the use of a latching transceiver greatly simplified the timing control of the DRAM accesses, as will be described later.

DRAM addresses are provided by multiplexing the latched R3051 address bus, using IDT FBT2827B memory drivers. This device type was chosen based on its ability to drive large capacitive loads, such as that found when driving 32 DRAMs. A single FBT output has sufficient drive to drive all four banks of the DRAM subsystem.

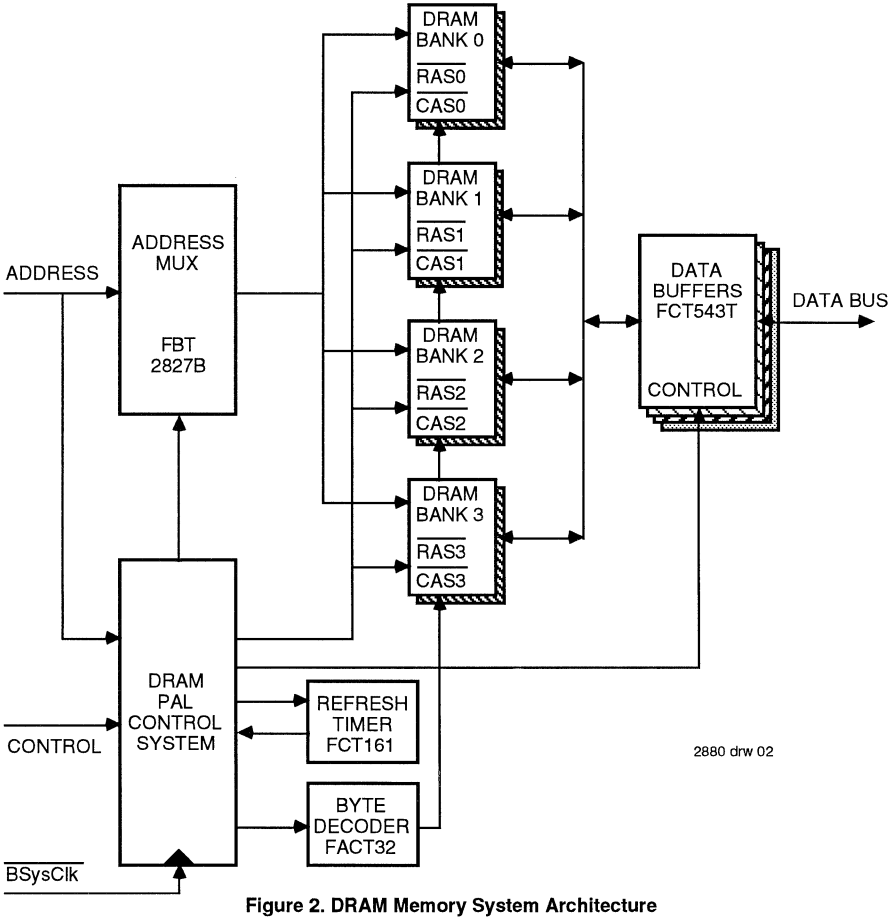


Figure 2. DRAM Memory System Architecture

Table 1. DRAM Memory Space Decoding

A22	0	0	0	0	0	0	0	0	1	1	X
A21	0	0	1	0	0	0	1	1	X	X	X
A20	0	1	0	1	0	1	0	0	X	X	X
$\overline{WR}$	1	1	1	1	0	0	0	0	1	0	1
$\overline{RD}$	0	0	0	0	1	1	1	1	0	1	1
SELECTION	READ BANK 0	READ BANK 1	READ BANK 2	READ BANK 3	WRITE BANK 0	WRITE BANK 1	WRITE BANK 2	WRITE BANK 3	READ OUTSIDE DRAM SPACE	WRITE OUTSIDE DRAM SPACE	NO ACCESS

In an R3051 system, it is possible to perform a 32-bit read access even when smaller data elements are requested. However, on writes, it is important to enable only those bytes which are actually being written by the CPU. The R3051 bus interface provides four individual byte enables to indicate which byte lanes are involved in a particular transfer. The DRAM subsystem uses a byte decoder (OR gate) to individually select from 1 to 4 bytes for write accesses. Each write byte enable is connected to those DRAMs which reside on that particular byte lane (across the multiple banks)

An 8-bit refresh timer requests the refreshing of the DRAMs every 9.6µs. Although this is more frequent than is actually required by the DRAMs, the use of this value simplified the control logic associated with page mode write. DRAMs require that $\overline{RAS}$ be maintained low no longer than 10µs; by choosing a refresh value smaller than this maximum time, the system is assured that maximum $\overline{RAS}$ low time will not be violated. The operation of the DRAM memory system is synchronized by $\overline{BSysClk}$.

STATE MACHINE IMPLEMENTATION

A simple state machine is used to perform the major aspects of DRAM control. The state machine uses a simple four-bit counter (C(3:0)) to dictate the timing for the DRAM control and CPU response, and is sequenced using $\overline{BSysClk}$. There are nine major states to the state machine, as illustrated in Figure 3; these states are dictated by the type of transfer requested and the state the DRAM control logic was left in by the prior transfer. Three PALs are required to implement the entire DRAM control logic.

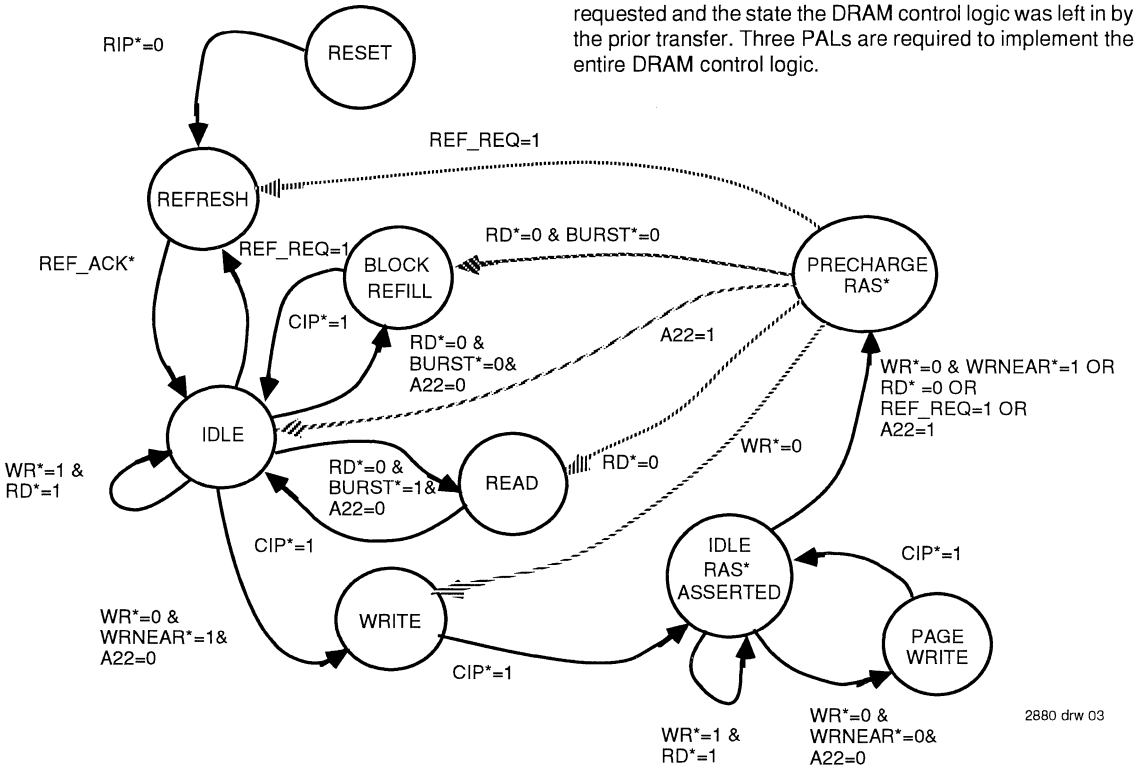


Figure 3. State Machine

The state machine uses the Reset pulse to reset its internal states and to synchronize its operation to the R3051. During the RESET state, it also performs one refresh cycle before entering the IDLE state.

In the IDLE state, the state machine arbitrates between a refresh cycle and a bus access. A DRAM bus access is started whenever Rd or Wr are asserted and A22 is LOW. A refresh request is detected using the REF_REQ (Refresh_Request) pulse from the refresh timer.

The state machine supports four types of bus accesses: "Block refill read", "Single read", "Single write" and "Page write", according to the types of transfers which the R3051 may request.

After a "Single write" or a "Page write" access, the machine enters the IDLE RAS ASSERTED state. This state is very much analogous to the IDLE state, except that the RAS control signal to the DRAMs remains asserted. This state allows subsequent "near" writes to be retired using page mode accesses, which are much quicker than standard accesses. When the IDLE RAS ASSERTED state must be exited (i.e. an action other than near write is requested) the RAS signal must be pre-charged prior to another DRAM transaction.

THE DRAM MEMORY SYSTEM IMPLEMENTATION DETAIL

The DRAM memory system consists of the control system, the address path and the data path as illustrated earlier in Figure 2.

PAL System

The state machine and control PAL system consists of three standard speed PALs: PAL 1 (PAL22V10-10), PAL 2 (PAL20R8-10) and PAL 3 (PAL16R8-10). Figure 4 illustrates the control system and the address path. The PAL equations are included in the appendix to this applications note.

PAL 1 is driven by SysClk directly. This allows the CIP line to detect transitions on the Rd and Wr signals from the R3051. Signals generated by PAL 1 include:

- 4 RAS signals (one per DRAM bank)
- The DRAM_ACK and DRAM_RDCEN response signals to the R3051 family CPU.
  These signals are used to provide termination response to the processor.
- The CIP (Cycle_In_Progress) indicates to the rest of the control system that a bus access is being performed.
- The DRAM_WN (DRAM_WrNear) signal indicates that the RAS signals are kept asserted after a "Single write" or a "Page write" access.

PAL 2 is also driven by SysClk directly. PAL 2 generates:

- 4 CAS signals (one per DRAM bank)
- DRAM_LE (DRAM_Latch_Enable), which latches the read data into the data buffers.
- The S (Select) controls the memory drivers selection.
- The T/R (Transmit/Receive) controls the data buffers during read accesses.
- The DRAM_WR (DRAM_Write), used during write accesses.

PAL 3 uses the buffered CIP signal (BCIP) which is delayed with respect to CIP by the buffer propagation delay. This is important to ensure the proper operation of PAL 3, which is driven by the buffered SysClk (BSysClk). PAL 3 generates the master 4-bit counter. It also generates:

- The RIP (Reset_In_Progress), which indicates that a reset cycle is being performed.
- The REF_ACK (Refresh_Acknowledge) signals that a refresh cycle is being performed.
- The GATE_COUNTER controls the operation of the counter when transitioning between bus accesses and refresh accesses.

Refresh Timer

The refresh timer consists of two "74FCT161" counters cascaded together as shown in Figure 4. The refresh timer issues a REF_REQ pulse every 9.6µs. The refresh timer is loaded with the value b00001111 after each refresh. It is incremented by one for every clock cycle. At value b11111111, it will issue the REF_REQ pulse. This amounts to a total count of 240 which at 25MHz reflects a 9.6µs refresh period.

The refresh period is set to be shorter than the maximum 15.5µs refresh period that most DRAM require. The refresh interval has been set to 9.6µs in order not to violate the RAS maximum pulse width of 10µs (t_{RAS} = 10µs max). In an IDLE RAS ASSERTED state, the RAS signals are left asserted while the CAS signals are de-asserted.

Byte Decoding

The byte decoding uses a "74FACT32" OR gate to OR the BE signals from the R3051 with the DRAM_WR signal to produce the write-byte signals WB(3:0). The DRAM_WR signal ensures that the WB(3:0) are only asserted during DRAM write accesses and that the WB(3:0) meet the "write command hold time" (t_{WCH} = 20ns) of the DRAMs. It also ensure that the WB(3:0) are asserted before the CAS signals for "Early Write" accesses. Every WB signal enables one byte of the DRAM banks and of the data buffers during write accesses to allow for partial word write operations. The WB(3:0) are always issued one clock cycle before the CAS signals are asserted, in order to meet the timing requirements for a DRAM "Early Write" cycle.

Address Path

The DRAM address path consists of 2 "74FBT2827B" memory drivers to multiplex the row and column address of the DRAMs. The "FBT2827" have a 25Ω series resistance incorporated in the output buffers and are used to drive multiple memory banks with large capacitive loading. The S bit from PAL 2 selects between the row address and the column address that drive all the DRAM banks. Figure 4 illustrates the address path architecture. The address to the DRAMs is always set one clock cycle before the assertion of either the RAS or the CAS signals, in order to guarantee proper address set-up time to the DRAMs.

Data Path

The data path consists of the DRAM banks and four 74FCT543 latched transceivers. Figure 5 illustrates the architecture of the data path and of the data buffers. Latching

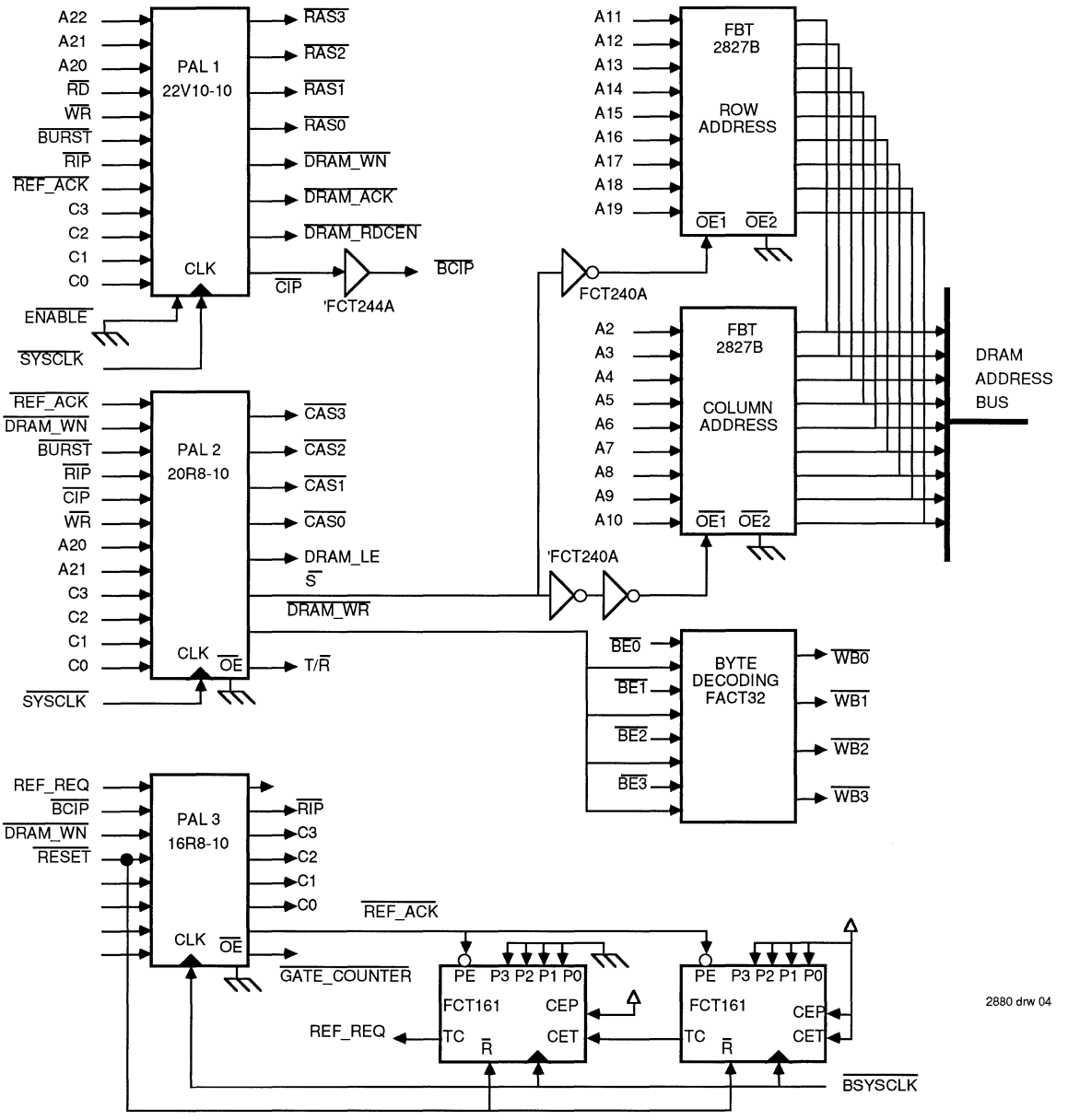


Figure 4. Control System and Address Path

transceivers are used to allow more access time to the DRAMs; the data is captured by the latches one-half cycle before they are needed by the CPU. During this half-cycle, the data propagates through the buffer; if traditional buffering transceivers had been used, the buffer propagation delay would have occurred at the expense of the DRAM access time.

Up to four banks of DRAMs are used, with each bank having its own set of $\overline{RAS}$ and $\overline{CAS}$ signals to minimize the loading impact of multiple DRAM devices. Address bits A21 and A20 determine the bank selection.

The latched transceivers serve three roles in the DRAM subsystem: they isolate the DRAMs from the A/D bus of the R3051 to minimize loading; they latch the data from the DRAMs on reads to allow a better timing model; and they are used to prevent bus contention from occurring at the end of a read (as the processor begins another transaction). The

R3051 is connected to the A bus of the transceivers, and the DRAM system is connected to the B bus.

In a processor write access, the R3051 drives both the address and the data. In this case the latches are left transparent to pass the processor data through directly to the DRAMs. Only those transceivers whose byte lanes are involved in the write are output enabled, since only those DRAMs will be written into. DRAMs not accessed in this write will output the current contents of their memory at that location, since the $\overline{OE}$ of the DRAMs is asserted. $\overline{DRAM\_WR}$ controls the $\overline{LEAB}$, leaving the latch transparent throughout the write. $\overline{WB}(3:0)$ controls the $\overline{OEAB}$ of the latches, thus enabling only those bytes that are written.

In a processor read access, the DRAM system drives the data bus. The DRAM system is synchronized to the rising edge of $\overline{BSysClk}$, and the R3051 samples the input data on the falling edge of $\overline{SysClk}$ before terminating the access. Thus, the DRAM control design, which drives the $\overline{RAS}$ and $\overline{CAS}$ signals on the rising edge of $\overline{SysClk}$, actually removes $\overline{CAS}$ one-half cycle before the data is sampled by the CPU. Thus, data output by the DRAMs is actually latched by the transceivers, and remains valid when the CPU samples the A/D bus one-half clock cycle later.

The DRAM_LE from the DRAM controller is connected to the $\overline{LEBA}$ pin, which latches the data into the transceivers. The T/$\overline{R}$ signal connected to the $\overline{CEBA}$ pin, which controls the direction of the bidirectional transceiver. The $\overline{DataEn}$ signal from the R3051 is connected directly to the $\overline{OEBA}$ pin to control the timing of the output enable onto the A/D bus. This ensures that the output buffers are tri-stated before the next R3051 access starts and prevents any bus contention.

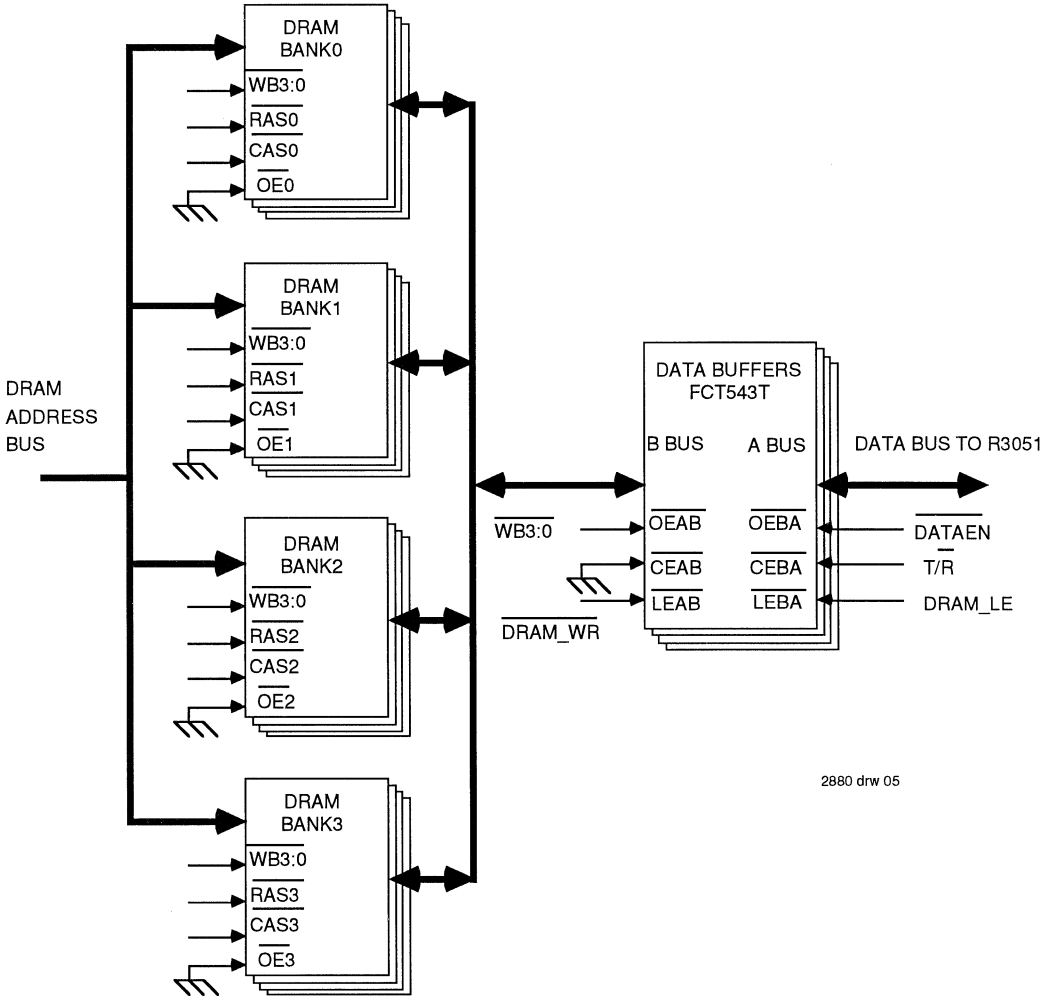


Figure 5. DRAM Banks and Data Buffers

THE DRAM MEMORY SYSTEM TIMING

The R3051 system interface allows this DRAM interface to be simply constructed. Features of the R3051 which are used in this DRAM system include:

- On-chip four-deep read and a four-deep write buffers. These buffers decouple the system interface speed from the speed of the execution engine on-chip.
- Single word reads and four-word refills. Block refills amortize the relatively long latency of DRAMs over multiple words, taking advantage of high-bandwidth capabilities (e.g. Page Mode) offered by DRAMs.
- The $\overline{\text{WrNear}}$ signal, which informs the external DRAM subsystem that two consecutive writes have the same upper 22 address bits (equivalent to a local page of 256 words), and can be written using a Page Mode access.

For the system running at 25MHz, the clock period is 40ns. DRAMs with 80ns of access time require 160ns (t_{RC} =160ns) to complete one read access (as per DRAM data sheet). A 5 clock cycles (200ns) read access time allows an acceptable margin for address decoding, control signal propagation, and bus interface.

For a four-word block refill read, the initial latency (time to read the first word) is the same as for a single-word read access (200ns). For the next three consecutive words, the DRAM memory system provides a word every 2 clock cycles (every 80ns). A block refill access can be completed in 11 clock cycles (440ns), which is an average of 110ns per word. Thus, block refill, with this simple scheme, provides a significant improvement in the average access time per word (over 2 clock cycles-per-word savings).

The state machine to manage write operations takes advantage of two features of the R3051:

- On a write cycle, the write data from the processor is held one full clock cycle after the clock edge where the processor samples its ACK input. Thus, the DRAM system can give an early acknowledge, and still rely on the CPU to continue driving data.
- The $\overline{\text{WrNear}}$ output from the CPU, which indicates that this write may be retired using a Page Mode write. This reduces the number of cycles required to perform write-intensive operations, such as building the program stack or flushing the write buffer.

The state machine for single word writes is optimized to allow subsequent near writes to be retired using page mode accesses. The DRAM memory system takes advantage of the $\overline{\text{WrNear}}$ signal from the R3051 by defaulting to the case that any single write to the DRAM system will be followed by another write with the same upper 22 address bits (within the local page of 256 words). Given this assumption, the $\overline{\text{RAS}}$ signals must be kept asserted after every write access to remain in the page mode of the DRAMs.

Thus, an initial single write can be performed in 4 clock cycles (160ns) since the $\overline{\text{RAS}}$ signals are not de-asserted and the $\overline{\text{RAS}}$ precharge time (t_{RP} = 70ns) will be deferred until the end of the page write mode. Note that this is faster than a single read; the state machine takes advantage of the fact that the processor will drive data a full clock cycle after acknowledge is given.

A consecutive write to the same DRAM page can be performed in 3 clock cycles (120ns) since the $\overline{\text{RAS}}$ signal is already asserted and doesn't need to be precharged. When this state is exited (when a write outside of page or a different type of access occurs) the $\overline{\text{RAS}}$ signal needs to be precharged for 2 clock cycles (80ns) before responding to the pending access.

Single Write Cycle/Page Write Cycle

Figure 6 illustrates the timing diagrams for a single write access followed by a page write. The R3051 initiates a single DRAM write access by the assertion of $\overline{\text{Wr}}$ and with A22 LOW. Since the state machine is in the IDLE state, $\overline{\text{RAS}}$ is de-asserted and the ROW addresses are flowing through the address multiplexer. The $\overline{\text{CIP}}$ is issued on the next clock edge to inform the rest of the machine that the write is being processed, thus preventing the commitment of any other state (e.g. refresh). The appropriate $\overline{\text{RAS}}$ signal is issued on the same edge as the $\overline{\text{CIP}}$. The $\overline{\text{DRAM\_ACK}}$ is issued on the following edge and the $\overline{\text{CAS}}$ signal on the 4th edge to terminate the write access. At the end of the access, the $\overline{\text{CIP}}$ is removed while the $\overline{\text{RAS}}$ signal is kept asserted in anticipation of a consecutive write access within the same page. At the end of an initial write access, the $\overline{\text{DRAM\_WN}}$ signal remains asserted. This signal informs the rest of the state machine that the $\overline{\text{RAS}}$ signals are kept asserted.

Idle, $\overline{\text{RAS}}$ Asserted State

At the end of a write access the state machine enters this state where a $\overline{\text{RAS}}$ signal is kept asserted while the state machine awaits a subsequent transaction. If the next access is a local write ($\overline{\text{WrNear}}$ from the R3051 is asserted) the state machine enters the page write mode. If a different access type occurs (read, block refill, not local write) or a refresh is pending, the state machine exits this state.

Upon exiting this state, the machine precharges the $\overline{\text{RAS}}$ signal before responding to the pending access. For the ease of discussion, any access that requires the $\overline{\text{RAS}}$ signals to be precharged before the access is processed will be referred to as "delayed" access. If an access outside the DRAM space is detected ($\overline{\text{Wr}}$ or $\overline{\text{Rd}}$ asserted while A22=1) the $\overline{\text{RAS}}$ signals are immediately de-asserted and the machine goes into the IDLE state. This is an important condition; an intervening write to another memory location causes the R3051 to report subsequent writes as "near" to that other memory location, and thus the DRAM controller should not process these writes as near writes.

Page Write Cycle

A page write cycle is a write access to the DRAM following another write with the same upper 22 address bits. Figure 6 illustrates the timing diagram for a page write access. The R3051 initiates a page write cycle by the assertion of $\overline{\text{Wr}}$, $\overline{\text{WrNear}}$ and A22 = 0. On the following clock edge $\overline{\text{CIP}}$ and $\overline{\text{DRAM\_ACK}}$ are issued, and on the 3rd clock edge $\overline{\text{CAS}}$ is asserted and the access is terminated ($\overline{\text{CIP}}$ is negated). The

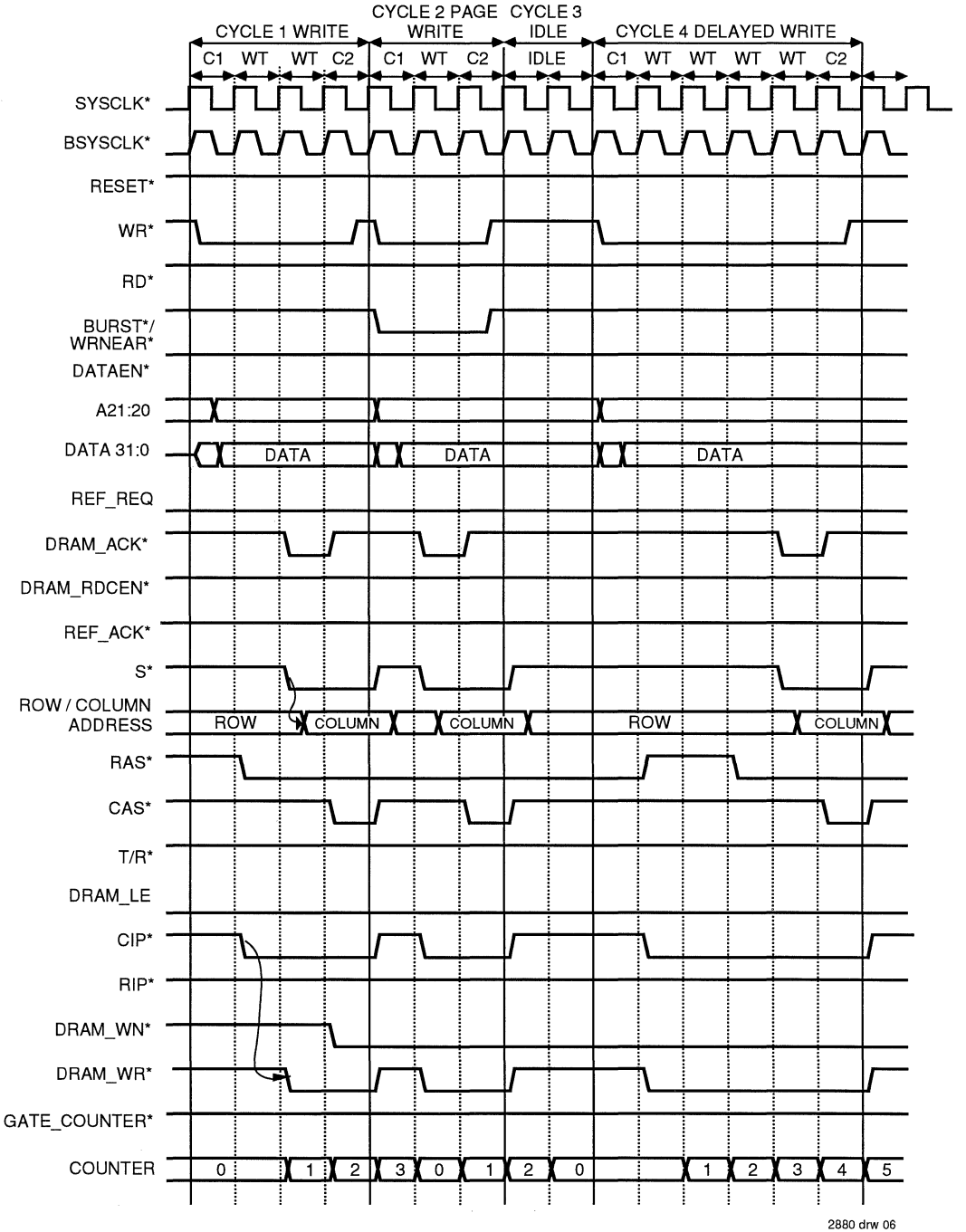


Figure 6. Single Write, Page Write and Delayed Write Timing Diagrams

RAS and DRAM_WN signals are kept asserted, allowing subsequent page writes to be rapidly processed. The state machine exits this state into the IDLE RAS ASSERTED state to await subsequent page mode writes.

Delayed Write Cycle

The delayed write cycle has exactly the same sequence as a single write but is delayed by two clock cycles. A delayed write is a "non-near" write detected in the IDLE RAS ASSERTED state. Figure 6 illustrates the timing diagrams for a delayed write access.

The R3051 initiates a delayed write access by the assertion of Wr and A22 = 0 while RAS and DRAM_WN are asserted. On the next clock edge RAS is de-asserted while the DRAM_WN is kept asserted. The precharging of the RAS signal takes two clock cycles. The DRAM_WN signal is kept asserted to inform the state machine that the control signals for this access have to be delayed by two clock cycles. This is true for all the delayed accesses.

Single Read Cycle

A single read cycle is a read access to the DRAM following an IDLE state in which the RAS and the DRAM_WN are not asserted. Figure 7 illustrates the timing diagrams for a read access. The R3051 initiates a single read access by the assertion of Rd with A22 LOW while the state machine is IDLE and all RAS outputs are de-asserted. The CIP is issued on the next clock edge to inform the rest of the machine that a cycle is ongoing, thus preventing the commitment of any other state. The appropriate RAS signal is issued on the same edge as the CIP. Two clock cycles later, the CAS, DRAM_RDCEN and the DRAM_ACK are issued to terminate the cycle.

For a read access both the DRAM_ACK and the DRAM_RDCEN are required to end the cycle. The processor will not actually sample RdCEn until one-clock after the clock edge used to generate DRAM_RDCEN, and thus will not sample the data until one and one-half clock cycles after the edge used to generate DRAM_RDCEN. From the timing diagrams it is clear that the CAS and the RAS signals are removed half a clock cycle before the falling edge of the clock when the R3051 samples the data. DRAM_LE latches the DRAM data into the transceivers and holds it for one clock cycle. At the end of the access the CIP is removed.

Delayed Read Cycle

The timings of a delayed read are exactly the same as for a single read but shifted by two clock cycles to accommodate RAS pre-charge time. A delayed read cycle is a read access to the DRAM following an IDLE RAS ASSERTED state in which the RAS and the DRAM_WN are still asserted. Figure 8 illustrates the timing diagrams for a delayed read access. Once a read access is detected, the RAS signal is de-asserted while the DRAM_WN is kept asserted. The RAS signal is precharged for two clock cycles. At the end of a delayed read, the DRAM_WN and the CIP are removed and the machine enters the IDLE state.

Block Refill Cycle

A block refill cycle is a four-word read access to the DRAM following an IDLE state. Figure 7 illustrates the timing diagrams for four-word block refill access. The R3051 indicates a block refill read access by the assertion of Rd and Burst with A22 LOW. The DRAM control subsystem handles block refill accesses using the Throttled Block Refill mode of the R3051. In a throttled read, RdCEn is used to control the data rate of memory back to the CPU. The Ack input is not provided back to the processor until the transfer has sufficiently progressed such that the last word of the transfer is clocked into the on-chip read buffer before the processor core requires it.

In the block refill access the first word read takes the same time as a single read while the three subsequent words are read into the read buffer at the rate of one word every two clock cycles. The DRAM_RDCEN is issued with every word being read to cause the R3051 to latch the data into the read buffer. The DRAM_ACK is issued between the second and the third word read. This ensures that for four subsequent falling edges of SysClk the read buffer can provide data to the R3000A core at the rate of a word every clock cycle.

Block refill uses the Page Mode characteristics of the DRAM to obtain subsequent words at a high data rate. In this access, the RAS signal is kept asserted while the CAS signal is toggled four times to produce four data words. Every word from the DRAM system is latched into the transceivers as for a single read operation, using the DRAM_LE to clock the latched transceivers. At the end of the access RAS and CIP are de-asserted, and the state machine returns to the IDLE state.

In the block refill access, address lines Addr(3:2) from the R3051 act as a two-bit counter to provide the address of four consecutive words. These two lines are incremented on the falling edge of SysClk. This timing could prove critical at high-frequencies: this is only half a clock margin (20ns) before the CAS signals are asserted, in which address set-up time to CAS must be provided. These two lines are part of the address path and are driving large capacitive loads. To minimize additional delay due to loading, two sets or more of memory address drivers could then be used to minimize the effect of the capacitive loads and to ensure proper operation.

Delayed Block Refill Cycle

A delayed block refill cycle is a block refill access to the DRAM following an IDLE RAS ASSERTED state in which the RAS and the DRAM_WN are asserted. Figure 9 illustrates the timing diagrams for a delayed block refill access. A delayed block refill is exactly the same as a block refill with the exception that the access is shifted by two clock cycles to accommodate RAS precharge requirements. The DRAM_WN signals to the machine that the access has a delayed timing. At the end of the access, the DRAM_WN and the CIP are de-asserted.

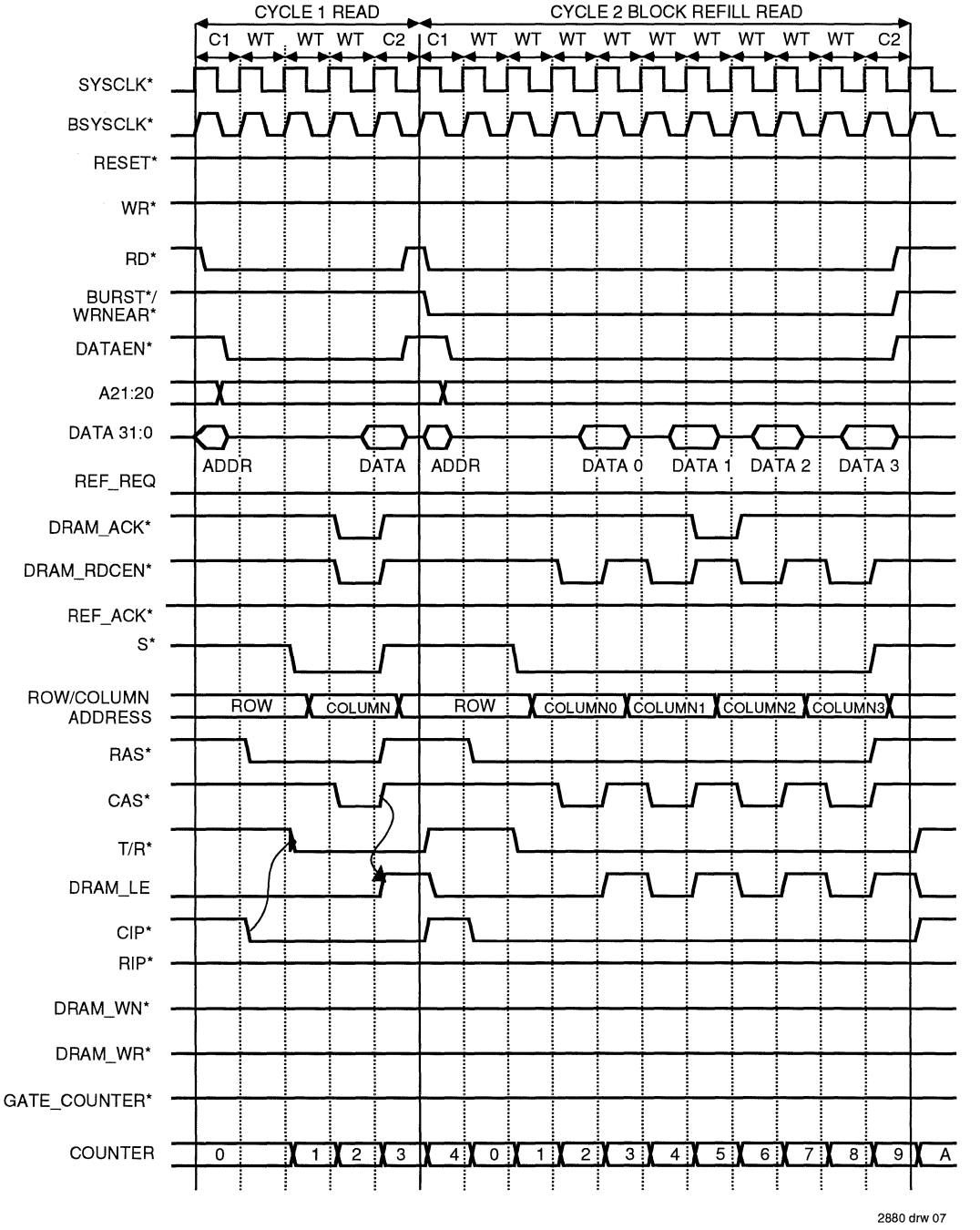


Figure 7. Single Read and Block Refill Read Timing Diagrams

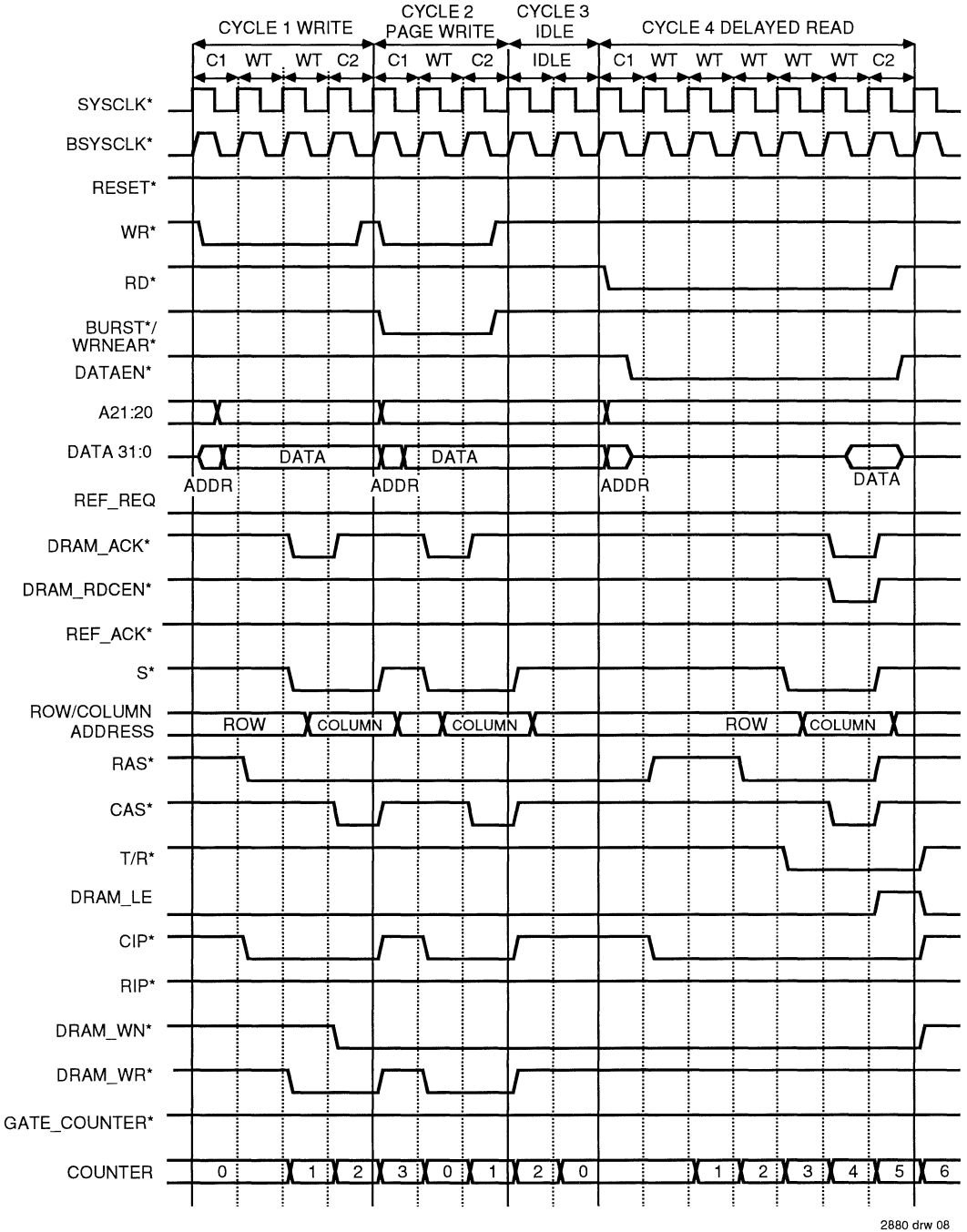
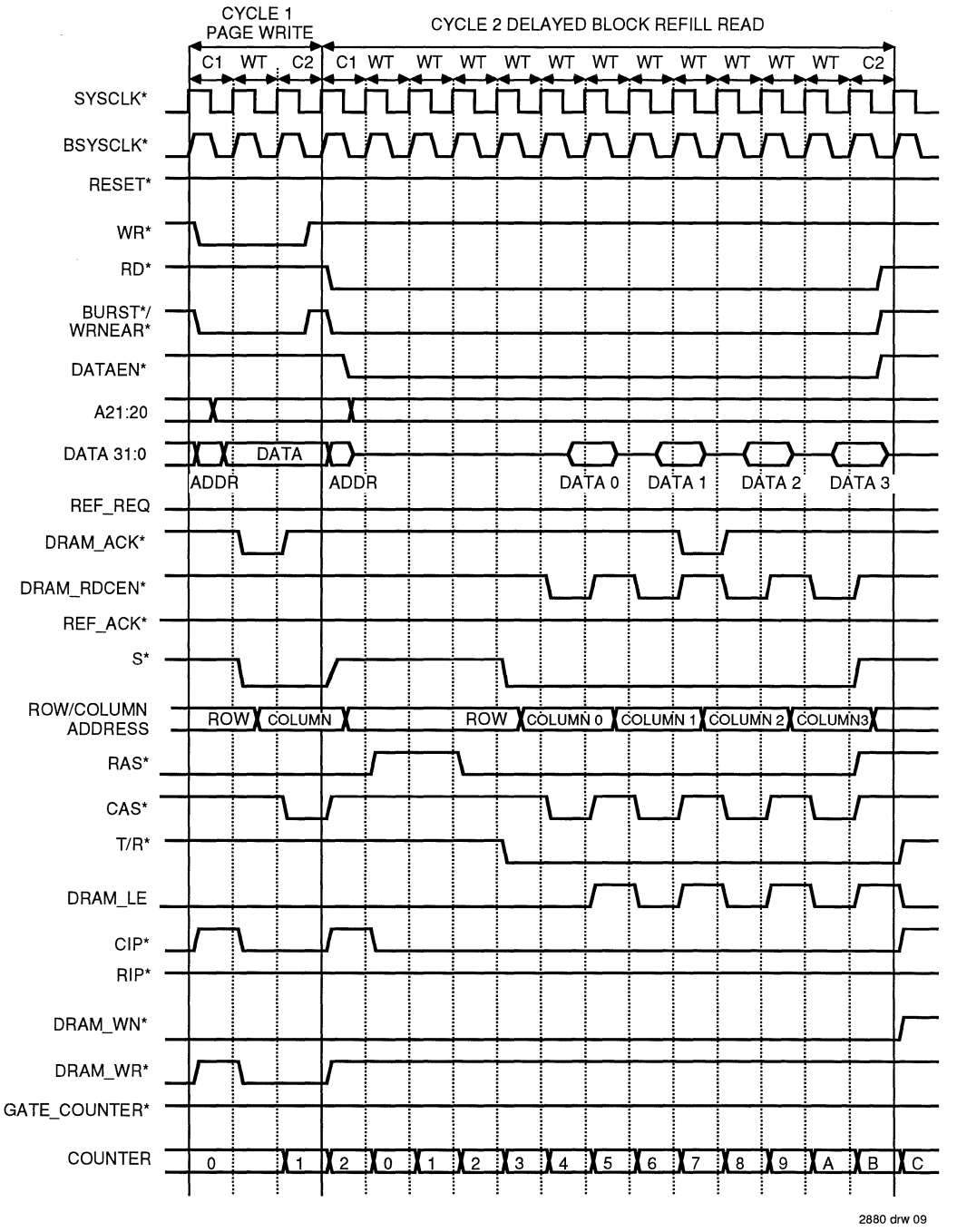


Figure 8. Delayed Single Read Timing Diagrams



2880 drw 09

Figure 9. Delayed Block Refill Read Timing Diagrams

Refresh Cycle

A refresh cycle is initiated every time a REF_REQ pulse is detected. The state machine responds immediately by asserting the $\overline{\text{REF\_ACK}}$ signal on the following clock edge. This disables the refresh timer until the refresh access is completed. Figure 10 illustrates the timing diagrams for a refresh arbitration and the actual refresh access.

If a REF_REQ occurs during an access or at the same time as an access, the refresh is delayed until the access is terminated (signaled by $\overline{\text{CIP}}$ de-asserted). Asserting $\overline{\text{REF\_ACK}}$ at the detection of REF_REQ ensures that the following access will be a refresh access and prevents the commitment of any other state. Delaying a refresh request until the end of a bus access doesn't affect the DRAM operation, since the refresh period selected is much less than the maximum refresh period of a DRAM row. The refresh period is every 9.6µs and the longest access is the delayed block refill with 14 clock cycles (until $\overline{\text{CIP}}$ is removed) which is 0.56µs. Thus, the refresh will be serviced at a maximum of 10.16µs, which is substantially below the maximum 15.5µs refresh requirement of the DRAMs. By the same reasoning, if the granted access is a delayed access, the $\overline{\text{RAS}}$ signal will be precharged prior to the 10µs $\overline{\text{RAS}}$ pulse width maximum requirements. If a Page Mode Write is granted, it will be retired in three cycles, or 0.12µs, and thus $\overline{\text{RAS}}$ will be precharged for the refresh no longer than 9.72µs after it was asserted.

The refresh access is a $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh in which all four $\overline{\text{CAS}}$ and $\overline{\text{RAS}}$ signals are issued. The $\overline{\text{CAS}}$ signal is issued one clock cycle before the $\overline{\text{RAS}}$ signal. A refresh access takes 10 clock cycles. This time is long enough to allow the $\overline{\text{RAS}}$ signals to be precharged if needed (delayed refresh). A delayed refresh has then the same timing as a refresh access.

Figure 11 shows the timing diagrams for the delayed refresh cycles. $\overline{\text{GATE\_COUNTER}}$ controls the operation of the 4-bit counter when transitioning between bus accesses and refresh accesses. It is mainly used in the arbitration phase when a bus access and refresh access are requested at the same time.

Reset Cycle

A reset cycle is initiated by the assertion of the $\overline{\text{Reset}}$ signal. This is a hardware reset and is used to initialize the PALs to the correct IDLE state. The $\overline{\text{RIP}}$ signal is asserted on the following clock edge to inform the machine that a reset cycle is in progress. After the $\overline{\text{Reset}}$ signal is de-asserted, the $\overline{\text{RIP}}$ stays asserted and one refresh access is initiated. At the end of this refresh access, the $\overline{\text{RIP}}$ is removed and the state machine enters the IDLE state. Figure 12 illustrates the timing diagrams of the reset operation.

Most DRAMs require at least 8 $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh accesses prior to a regular access, to insure proper initialization. The actual state machine provides only one refresh access. It is the responsibility of the software to ensure that no DRAM access is made prior to the elapsing of 8 refresh periods from the refresh timer. This can typically be insured by normal operation of the boot PROM; however, software could "spin-lock" for a pre-determined number of loops to insure that sufficient time has elapsed.

Idle State

The IDLE state is the state in which the machine is not performing any bus access or a refresh access but is constantly monitoring the bus for any access request. All the signals are de-asserted and the 4-bit counter operation is halted.

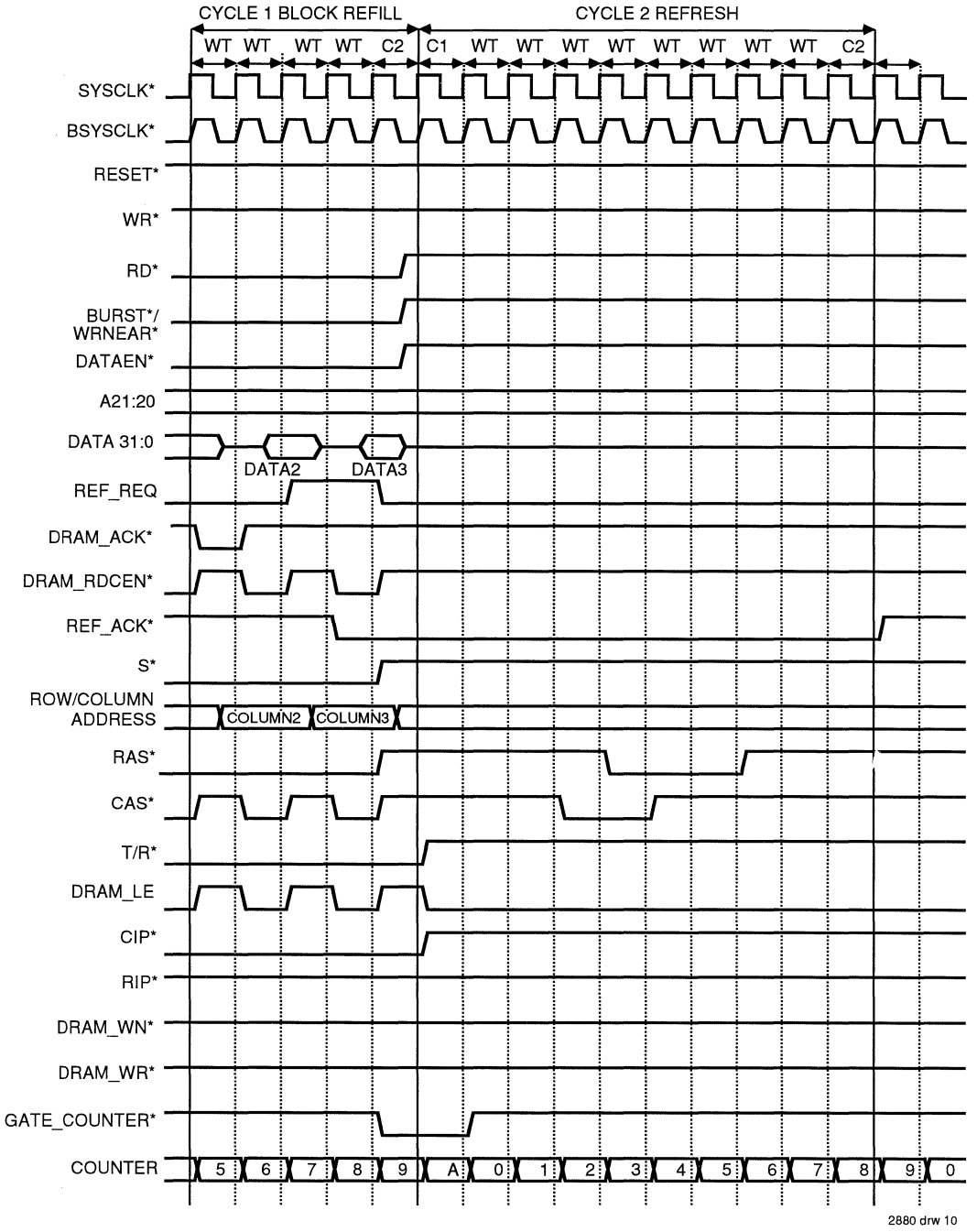


Figure 10. Refresh Arbitration and Refresh Timing Diagrams

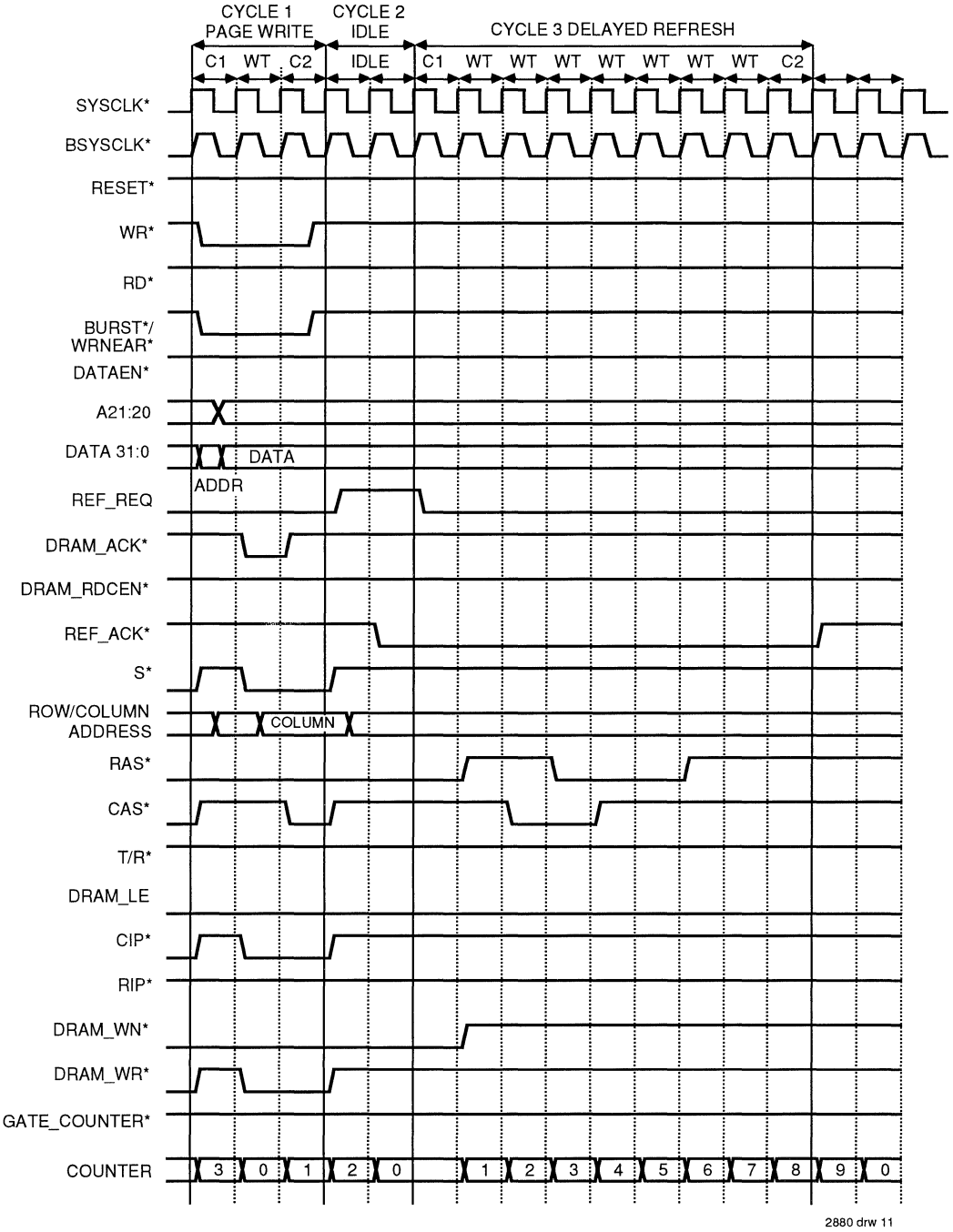


Figure 11. Delayed Refresh Timing Diagrams

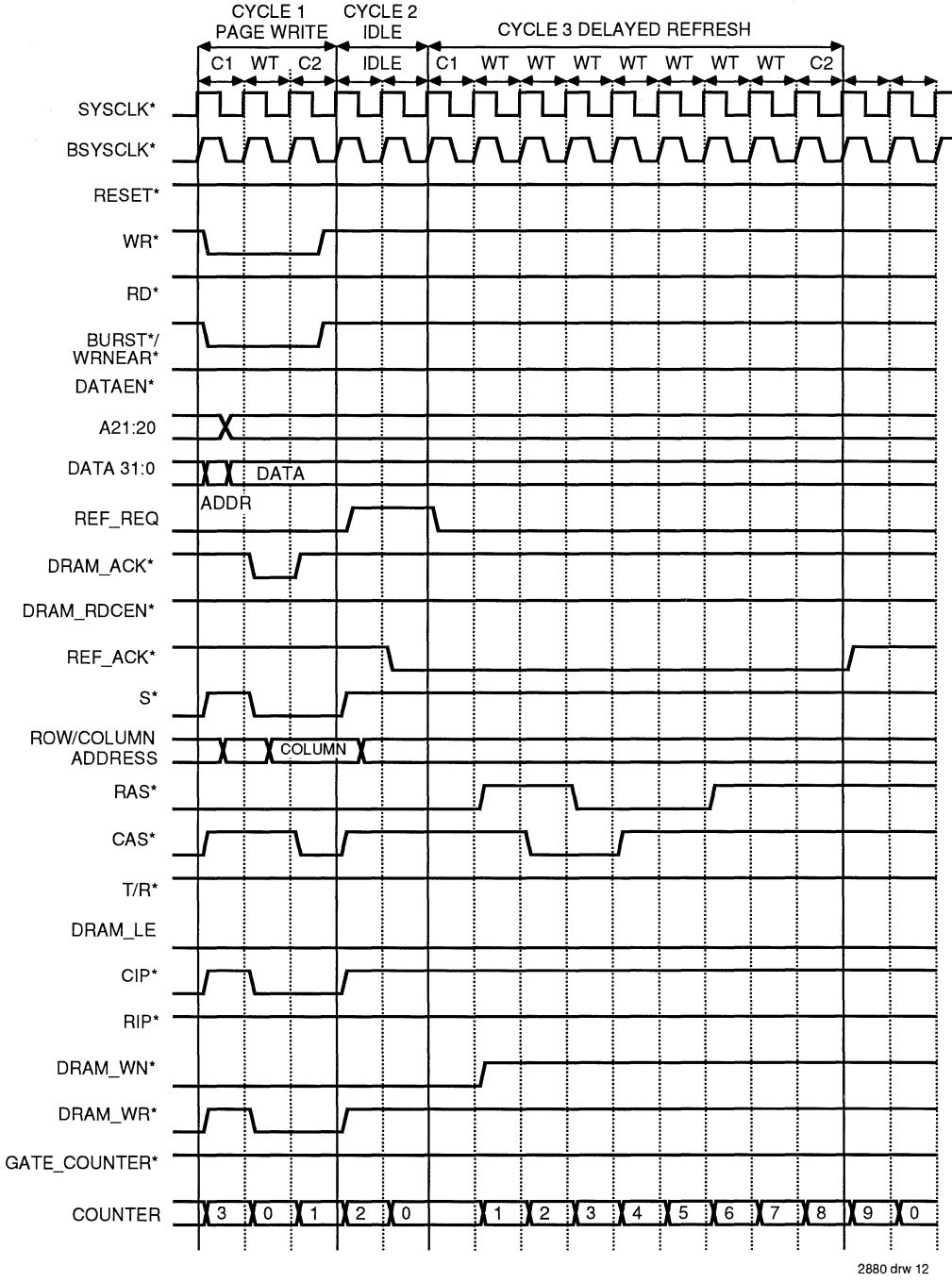


Figure 12. Reset Timing Diagrams

CRITICAL TIMING CALCULATIONS

The following is a timing analysis of some of the critical paths in the DRAM system.

DRAM Data for a Read or Block Refill Access

As illustrated in all the timing diagrams, the $\overline{CAS}$ signal is asserted for only one clock cycle for a read or a write access. For a write access there is no critical timing since the DRAM latches the data in at the $\overline{CAS}$ leading edge, and the processor insures sufficient data hold time by holding data for one cycle after $\overline{ACK}$ is detected.

For a read or a block refill access the DRAMs provide the data to the R3051 and the maximum delays must be considered. Figure 13 illustrates the detailed timing for a portion of a block refill access which is also true for a read access. The R3051 uses the $\overline{SysClk}$ for its reference with a period Tclk of 40ns. The $\overline{CAS}$ and the DRAM_LE signals are delayed with respect to $\overline{SysClk}$ by the PAL 2 propagation delay T1. The data is available from the DRAM after T2 (t_{cac} = 25ns max). The critical path requires that the DRAM data be available and meet the setup time of the transceivers before the DRAM_LE is asserted. The timing calculation for this data path is as follows:

Tclk	= 40.0ns	
- T1 max	= 8.0	
	= 32.0	
- T2 max	= 25.0	
	= 7.0	
- T setup	= 3.0	FCT543T data set-up time.
	= 4.0	

The available margin is 4.0ns. Some 80ns DRAMs have T2 (t_{cac} = 20ns) which could offer more margin.

Transceivers Turn Off Time

For a read or a block refill access, the DRAMs provide the data to the R3051 through the latched transceivers. As illustrated in Figure 7, the R3051 reads the data from the bus half a clock cycle before it starts a new access in which it can drive address on the bus. This information is explained in detail in the R3051 User Manual.

The critical path requires that the transceivers be tri-stated before the R3051 starts driving the bus in the next clock cycle. The $\overline{DataEn}$ signal directly from the R3051 enables the B to A output buffers of the transceivers (FCT543T). The $\overline{DataEn}$ is delayed by T3 from the falling edge of $\overline{SysClk}$ at which the R3051 samples the data (as per R3051 data sheet). The transceivers disable the output buffers within T4. Figure 14 illustrates the timing for this path.

Tclk / 2	= 20.0ns
- T3 max	= 6.0
	= 14.0
- T4 max	= 9.0
T margin	= 5.0ns

This margin of 5ns is long enough to accommodate for any $\overline{SysClk}$ skews.

$\overline{DRAM\_ACK}$ and $\overline{DRAM\_RDCEN}$ Timings

The $\overline{DRAM\_ACK}$ and the $\overline{DRAM\_RDCEN}$ are issued for one clock cycle only as illustrated in the timing diagrams. They are removed by the clock edge which the R3051 uses to sample them. The R3051 requires that these two signals be held constant for a minimum of 4ns after the clock edge. These two signals are usually combined with similar signals from other memory subsystems (e.g. EPROM) to form one set that is routed to the R3051. This extra delay, plus the PAL 1 minimum propagation delay are long enough to meet the R3051 required hold time.

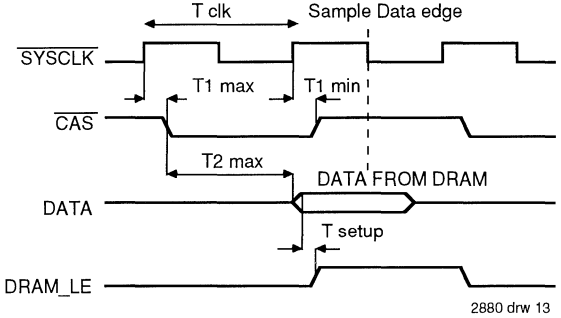


Figure 13. Read or Block Refill Access

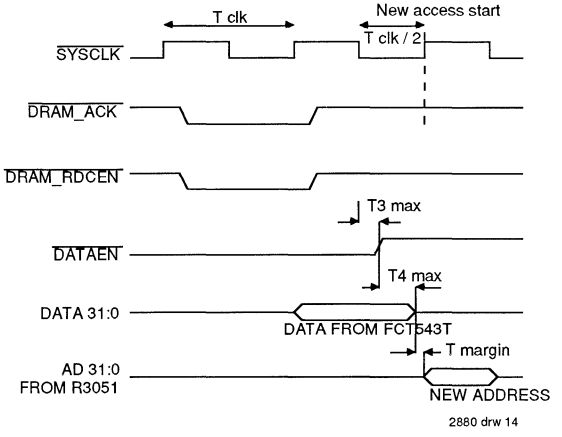


Figure 14. Termination of a Read or Block Refill Access

PERFORMANCE

The performance of the different types of R3051 bus accesses to the DRAM memory is usually measured by the number of clock cycles it takes to send the $\overline{Ack}$ back to the R3051. This time is computed from the beginning of the external access. The performance of the DRAM system can be summarized as follows:

- single read: 4 clock cycles.
- block refill: 7 clock cycles.
- first write: 3 clock cycles.
- page write: 2 clock cycles.

The above numbers (with the exception of page write) will be increased by 2 in the case of delayed accesses.

Thus, relatively high memory performance is obtained with minimal external logic parts count, and low-cost commodity DRAM. More aggressive designs could utilize faster DRAMs, and techniques such as memory interleaving, to achieve still higher levels of performance.

CONCLUSION

The R3051 RISController family bus interface was designed to allow memory systems of differing complexity and performance to be implemented. Even a relatively simple DRAM system, as the one described here, offers very high performance. With simple modifications, this approach is applicable to higher frequencies (33 and 40MHz) and to interleaved memory systems yielding even higher performance.

```
{
       TITLE:              PAL1
       PURPOSE:            RAS
       AUTHOR:             BOB NAPAA,  IDT  INC.
       DATE:               4/5/91
}

MODULE  PAL1;
TITLE  PAL1;
TYPE  AMD  22V10;

INPUTS;

       SYSCLKB                    NODE[PIN1];
       ENABLEB                    NODE[PIN2];
       RDB                        NODE[PIN3];
       WRB                        NODE[PIN4];
       BURSTB                     NODE[PIN5];
       RIPB                       NODE[PIN6];
       REFACKB                    NODE[PIN7];
       A22                        NODE[PIN8];
       A21                        NODE[PIN9];
       A20                        NODE[PIN10];
       C3                         NODE[PIN11];
       C2                         NODE[PIN13];
       C1                         NODE[PIN14];
       C0                         NODE[PIN15];

       {FEED  BACK  PINS}
       CIPB                       NODE[PIN16];
       RAS3B                      NODE[PIN17];
       RAS2B                      NODE[PIN18];
       RAS1B                      NODE[PIN19];
       RAS0B                      NODE[PIN20];
       DRAMWNB                    NODE[PIN21];
       DRAMACKB                   NODE[PIN22];
       DRAMRDCENB                 NODE[PIN23];


OUTPUTS;
       CIPB                       NODE[PIN16]  ATTR[RL];
       RAS3B                      NODE[PIN17]  ATTR[RL];
       RAS2B                      NODE[PIN18]  ATTR[RL];
       RAS1B                      NODE[PIN19]  ATTR[RL];
       RAS0B                      NODE[PIN20]  ATTR[RL];
       DRAMWNB                    NODE[PIN21]  ATTR[RL];
       DRAMACKB                   NODE[PIN22]  ATTR[RL];
       DRAMRDCENB                 NODE[PIN23]  ATTR[RL];

       {OUTPUT  ENABLES}
       CIPBEN                     NODE[PIN16EN];
       RAS3BEN                    NODE[PIN17EN];
       RAS2BEN                    NODE[PIN18EN];
       RAS1BEN                    NODE[PIN19EN];
       RAS0BEN                    NODE[PIN20EN];
       DRAMWNBEN                  NODE[PIN21EN];
       DRAMACKBNODE[PIN22EN];
       DRAMRDCENBEN               NODE[PIN23EN];
```

```
TERMS;

        RAS3BEN             =       ENABLEB;
        RAS3B  NOT          :=      RAS3B AND REFACKB AND RIPB AND DRAMWNB AND !RDB AND
                                    !A22 AND A21 AND A20 {read/block refill}
                            OR      !RAS3B AND !CIPB AND !RDB AND DRAMACKB AND DRAMRDCENB
                                    {keep for read/delayed read}
                            OR      RAS3B AND !CIPB AND RIPB AND !DRAMWNB AND !RDB AND
                                    !A22 AND A21 AND A20 AND !C3 AND !C2 AND !C1 AND C0
                                    {delayed read/delayed block refill}
                            OR      !RAS3B AND !CIPB AND !RDB AND !BURSTB AND !C3 {keep block refill}
                            OR      !RAS3B AND !CIPB AND !RDB AND !BURSTB AND !DRAMWNB AND
                                    !C1 {keep delayed block refill}
                            OR      RAS3B AND REFACKB AND RIPB AND DRAMWNB AND !WRB AND
                                    !A22 AND A21 AND A20 {write}
                            OR      RAS3B AND REFACKB AND RIPB AND !DRAMWNB AND !WRB AND
                                    !A22 AND A21 AND A20 AND !C3 AND !C2 AND !C1 AND C0
                                    {delayed write}
                            OR      !RAS3B AND !WRB AND !CIPB {keep for write}
                            OR      !RAS3B AND !DRAMWNB AND REFACKB AND RIPB AND RDB AND
                                    WRB AND BURSTB {no access pending}
                            OR      !RAS3B AND !DRAMWNB AND REFACKB AND RIPB AND !WRB AND
                                    !BURSTB AND !A22 AND A21 AND A20 {keep for page write}
                            OR      !REFACKB AND CIPB AND !RAS3B AND !DRAMWNB AND C0
                                    {remove in refresh}
                            OR      RAS3B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND !C2
                                    AND C1 AND C0 {issue for refresh}
                            OR      !RAS3B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                                    AND !C1 AND !C0 {keep for refresh}
                            OR      !RAS3B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                                    AND !C1 AND C0; {keep for refresh}

        RAS2BEN             =       ENABLEB;
        RAS2B  NOT          :=      RAS2B AND REFACKB AND RIPB AND DRAMWNB AND !RDB AND
                                    !A22 AND A21 AND !A20 {read/block refill}
                            OR      !RAS2B AND !CIPB AND !RDB AND DRAMACKB AND DRAMRDCENB
                                    {keep for read/delayed read}
                            OR      RAS2B AND !CIPB  AND RIPB AND !DRAMWNB AND !RDB AND
                                    !A22 AND A21 AND !A20 AND !C3 AND !C2 AND !C1 AND C0
                                    {delayed read/delayed block refill}
                            OR      !RAS2B AND !CIPB AND !RDB AND !BURSTB AND !C3 {keep block refill}
                            OR      !RAS2B AND !CIPB AND !RDB AND !BURSTB AND !DRAMWNB AND
                                    !C1{keep delayed block refill}
                            OR      RAS2B AND REFACKB AND RIPB AND DRAMWNB AND !WRB AND
                                    !A22 AND A21 AND !A20 {write}
                            OR      RAS2B AND REFACKB AND RIPB AND !DRAMWNB AND !WRB AND
                                    !A22 AND A21 AND !A20 AND !C3 AND !C2 AND !C1 AND C0
                                    {delayed write}
                            OR      !RAS2B AND !WRB AND !CIPB {keep for write}
                            OR      !RAS2B AND !DRAMWNB AND REFACKB AND RIPB AND RDB AND
                                    WRB AND BURSTB {no access pending}
                            OR      !RAS2B AND !DRAMWNB AND REFACKB AND RIPB AND !WRB AND
                                    !BURSTB AND !A22 AND A21 AND !A20 {keep for page write}
                            OR      !REFACKB AND CIPB AND !RAS2B AND !DRAMWNB AND C0
                                    {remove in refresh}
                            OR      RAS2B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND !C2
                                    AND C1 AND C0 {issue for refresh}
                            OR      !RAS2B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                                    AND !C1 AND !C0 {keep for refresh}
                            OR      !RAS2B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                                    AND !C1 AND C0; {keep for refresh}
```

```
RAS1BEN          =     ENABLEB;
RAS1B  NOT       :=    RAS1B AND REFACKB AND RIPB AND DRAMWNB AND !RDB AND
                       !A22 AND !A21 AND A20 {read/block refill}
                 OR    !RAS1B AND !CIPB AND !RDB AND DRAMACKB AND DRAMRDCENB
                       {keep for read/delayed read}
                 OR    RAS1B AND !CIPB AND RIPB AND !DRAMWNB AND !RDB AND
                       !A22 AND !A21 AND A20 AND !C3 AND !C2 AND !C1 AND C0
                       {delayed read/delayed block refill}
                 OR    !RAS1B AND !CIPB AND !RDB AND !BURSTB AND !C3 {keep block refill}
                 OR    !RAS1B AND !CIPB AND !RDB AND !BURSTB AND !DRAMWNB AND
                       !C1{keep delayed block refill}
                 OR    RAS1B AND REFACKB AND RIPB AND DRAMWNB AND !WRB AND
                       !A22 AND !A21 AND A20 {write}
                 OR    RAS1B AND REFACKB AND RIPB AND !DRAMWNB AND !WRB AND
                       !A22 AND !A21 AND A20 AND !C3 AND !C2 AND !C1 AND C0
                        {delayed write}
                 OR    !RAS1B AND !WRB AND !CIPB {keep for write}
                 OR    !RAS1B AND !DRAMWNB AND REFACKB AND RIPB AND RDB AND
                       WRB AND BURSTB {no access pending}
                 OR    !RAS1B AND !DRAMWNB AND REFACKB AND RIPB AND !WRB AND
                       !BURSTB AND !A22 AND !A21 AND A20 {keep for page write}
                 OR    !REFACKB AND CIPB AND !RAS1B AND !DRAMWNB AND C0
                       {remove in refresh}
                 OR    RAS1B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND !C2
                       AND C1 AND C0 {issue for refresh}
                 OR    !RAS1B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                       AND !C1 AND !C0 {keep for refresh}
                 OR    !RAS1B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                       AND !C1 AND C0; {keep for refresh}

RAS0BEN          =     ENABLEB;
RAS0B  NOT       :=    RAS0B AND REFACKB AND RIPB AND DRAMWNB AND !RDB AND
                       !A22 AND !A21 AND !A20 {read/block refill}
                 OR    !RAS0B AND !CIPB AND !RDB AND DRAMACKB AND DRAMRDCENB
                       {keep for read/delayed read}
                 OR    RAS0B AND !CIPB AND RIPB AND !DRAMWNB AND !RDB AND
                       !A22 AND !A21 AND !A20 AND !C3 AND !C2 AND !C1 AND C0
                       {delayed read/delayed block refill}
                 OR    !RAS0B AND !CIPB AND !RDB AND !BURSTB AND !C3 {keep block refill}
                 OR    !RAS0B AND !CIPB AND !RDB AND !BURSTB AND !DRAMWNB AND
                       !C1 {keep delayed block refill}
                 OR    RAS0B AND REFACKB AND RIPB AND DRAMWNB AND !WRB AND
                       !A22 AND !A21 AND !A20 {write}
                 OR    RAS0B AND REFACKB AND RIPB AND !DRAMWNB AND !WRB AND
                       !A22 AND !A21 AND !A20 AND !C3 AND !C2 AND !C1 AND C0
                       {delayed write}
                 OR    !RAS0B AND !WRB AND !CIPB {keep for write}
                 OR    !RAS0B AND !DRAMWNB AND REFACKB AND RIPB AND RDB AND
                       WRB AND BURSTB {no access pending}
                 OR    !RAS0B AND !DRAMWNB AND REFACKB AND RIPB AND !WRB AND
                       !BURSTB AND !A22 AND !A21 AND !A20 {keep for page write}
                 OR    !REFACKB AND CIPB AND !RAS0B AND !DRAMWNB AND C0
                       {remove in refresh}
                 OR    RAS0B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND !C2
                       AND C1 AND C0 {issue for refresh}
                 OR    !RAS0B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                       AND !C1 AND !C0 {keep for refresh}
                 OR    !RAS0B AND !REFACKB AND CIPB AND DRAMWNB AND !C3 AND C2
                       AND !C1 AND C0; {keep for refresh}
```

```
        DRAMWNBEN           =       ENABLEB;
        DRAMWNB  NOT        :=      DRAMWNB AND !CIPB AND RIPB AND !WRB  AND !C3  AND !C2 AND
                                    !C1 AND C0 {write}
                            OR      !DRAMWNB AND !REFACKB AND CIPB AND RIPB AND !C3 AND !C2
                                    AND !C1 AND !C0{remove at refresh}
                            OR      !DRAMWNB AND RIPB AND !RAS3B  {keep asserted if any RAS}
                            OR      !DRAMWNB AND RIPB AND !RAS2B
                            OR      !DRAMWNB AND RIPB AND !RAS1B
                            OR      !DRAMWNB AND RIPB AND !RAS0B
                            OR      !DRAMWNB AND RIPB AND !RDB AND !CIPB {keep for read}
                            OR      !DRAMWNB AND RIPB AND !WRB AND !CIPB;   {keep for write}

        DRAMACKBEN          =       ENABLEB;
        DRAMACKB  NOT       :=      !CIPB AND !RDB AND DRAMWNB AND BURSTB AND !C3 AND !C2 AND
                                    !C1 AND C0 {read}
                            OR      !CIPB AND !RDB AND !DRAMWNB AND BURSTB AND !C3 AND !C2
                                    AND C1 AND C0 {delayed read}
                            OR      !CIPB AND !RDB AND DRAMWNB AND !BURSTB AND !C3 AND C2 AND
                                    !C1 AND !C0 {block refill}
                            OR      !CIPB AND !RDB AND !DRAMWNB AND !BURSTB AND !C3 AND C2
                                    AND C1 AND !C0 {delayed block refill}
                            OR      !CIPB AND !WRB AND DRAMWNB  AND !C3 AND !C2  AND !C1 AND !C0
                                    {write}
                            OR      !CIPB AND !WRB AND !DRAMWNB AND BURSTB AND !C3 AND !C2
                                    AND C1 AND !C0 {delayed write}
                            OR      !WRB AND !BURSTB AND !DRAMWNB AND REFACKB AND RIPB AND
                                    CIPB AND !A22 AND !RAS3B {page write}
                            OR      !WRB AND !BURSTB AND !DRAMWNB AND REFACKB AND RIPB AND
                                    CIPB AND !A22 AND !RAS2B {page write}
                            OR      !WRB AND !BURSTB AND !DRAMWNB AND REFACKB AND RIPB AND
                                    CIPB AND !A22 AND !RAS1B {page write}
                            OR      !WRB AND !BURSTB AND !DRAMWNB AND REFACKB AND RIPB AND
                                    CIPB AND !A22 AND !RAS0B ;{page write}

        DRAMRDCENBEN        =       ENABLEB;
        DRAMRDCENB  NOT     :=      !CIPB AND !RDB AND DRAMWNB AND BURSTB AND !C3 AND !C2 AND
                                    !C1 AND C0 {read}
                            OR      !CIPB AND !RDB AND !DRAMWNB AND BURSTB AND !C3 AND !C2
                                    AND C1 AND C0 {delayed  read}
                            OR      !CIPB AND !RDB AND DRAMWNB AND !BURSTB AND !C3 AND C0
                                    {block  refill}
                            OR      !CIPB AND !RDB AND !DRAMWNB AND !BURSTB AND !C3 AND !C2
                                    AND C1 AND C0 {delayed block refill}
                            OR      !CIPB AND !RDB AND !DRAMWNB AND !BURSTB AND !C3 AND C2
                                    AND C0 {delayed block refill}
                            OR      !CIPB AND !RDB AND !DRAMWNB AND !BURSTB AND C3 AND !C2
                                    AND !C1 AND C0;   {delayed block refill}

        CIPBEN              =       ENABLEB;
        CIPB  NOT           :=      CIPB AND REFACKB AND RIPB AND !RDB AND !A22 {read}
                            OR      CIPB AND REFACKB AND RIPB AND !WRB AND !A22 {write}
                            OR      !CIPB AND !RDB {keep for read}
                            OR      !CIPB AND !WRB ;{keep for write}

END;
END  PAL1.
```

```
{
        TITLE:              PAL2
        PURPOSE:            CAS
        AUTHOR:             BOB NAPAA, IDT INC.
        DATE:               4/5/91
}

MODULE  PAL2;
TITLE  PAL2;
TYPE MMI 20R8;

INPUTS;

        {SYSCLKB                    NODE[PIN1];  }
        REFACKB                     NODE[PIN2];
        DRAMWNB                     NODE[PIN3];
        BURSTB                      NODE[PIN4];
        RIPB                        NODE[PIN5];
        CIPB                        NODE[PIN6];
        WRB                         NODE[PIN7];
        A21                         NODE[PIN8];
        A20                         NODE[PIN9];
        C3                          NODE[PIN10];
        C2                          NODE[PIN11];
        {OUTENABLEB                 NODE[PIN13];  }
        C1                          NODE[PIN14];
        C0                          NODE[PIN23];

        {FEED BACK PINS}
        CAS3B                       NODE[PIN22];
        CAS2B                       NODE[PIN21];
        CAS1B                       NODE[PIN20];
        CAS0B                       NODE[PIN19];
        DRAMLE                      NODE[PIN18];
        DRAMWRB                     NODE[PIN17];
        SB                          NODE[PIN16];
        TRB                         NODE[PIN15];

OUTPUTS;

        CAS3B                       NODE[PIN22];
        CAS2B                       NODE[PIN21];
        CAS1B                       NODE[PIN20];
        CAS0B                       NODE[PIN19];
        DRAMLE                      NODE[PIN18];
        DRAMWRB                     NODE[PIN17];
        SB                          NODE[PIN16];
        TRB                         NODE[PIN15];

TABLE;

        CAS3B NOT           :=      CAS3B AND RIPB AND !CIPB  AND DRAMWNB AND (A21 AND A20
                                    AND !C3 AND !C2 AND !C1 AND C0)  {read or write}
                            OR      CAS3B AND RIPB AND !CIPB  AND !DRAMWNB AND (A21 AND A20
                                    AND !C3 AND !C2 AND C1 AND C0)  {delayed read/write}
                            OR      CAS3B AND RIPB AND !CIPB AND !BURSTB AND DRAMWNB AND
                                    WRB AND !SB AND (A21 AND A20 AND !C3 AND C0)  {block refill}
                            OR      CAS3B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                                    WRB AND !SB AND (A21 AND A20 AND C0)  {delayed block refill}
                            OR      CAS3B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                                    !WRB AND !SB AND (A21 AND A20 AND !C3 AND !C2 AND !C1 AND
                                    !C0)  {page write}
```

```
                    OR      CIPB AND DRAMWNB AND !REFACKB AND CAS3B AND (!C3 AND !C2
                            AND C1 AND !C0)   {refresh}
                    OR      CIPB AND DRAMWNB AND !REFACKB AND !CAS3B AND (!C3 AND !C2
                            AND C1 AND C0); {refresh}

CAS2B  NOT          :=      CAS2B AND RIPB AND !CIPB  AND DRAMWNB AND (A21 AND !A20
                            AND !C3 AND !C2 AND !C1 AND C0)   {read or write}
                    OR      CAS2B AND RIPB AND !CIPB  AND !DRAMWNB AND (A21 AND !A20
                            AND !C3 AND !C2 AND C1 AND C0)   {delayed read/write}
                    OR      CAS2B AND RIPB AND !CIPB AND !BURSTB AND DRAMWNB AND
                            WRB AND !SB AND (A21 AND !A20 AND !C3 AND C0)   {block refill}
                    OR      CAS2B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                            WRB AND !SB  AND (A21 AND !A20 AND C0)   {delayed block refill}
                    OR      CAS2B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                            !WRB AND !SB AND (A21 AND !A20 AND !C3 AND !C2 AND !C1 AND
                            !C0)   {page write}
                    OR      CIPB AND DRAMWNB AND !REFACKB AND CAS2B AND
                            (!C3 AND !C2 AND C1 AND !C0) {refresh}
                    OR      CIPB AND DRAMWNB AND !REFACKB AND !CAS2B AND
                            (!C3 AND !C2 AND C1 AND C0); {refresh}

CAS1B  NOT          :=      CAS1B AND RIPB AND !CIPB AND DRAMWNB AND (!A21 AND A20
                            AND !C3 AND !C2 AND!C1 AND C0)    {read or write}
                    OR      CAS1B AND RIPB AND !CIPB  AND !DRAMWNB AND (!A21 AND A20
                            AND !C3 AND !C2 AND C1 AND C0)    {delayed read/write}
                    OR      CAS1B AND RIPB AND !CIPB AND !BURSTB AND DRAMWNB AND
                            WRB AND !SB AND (!A21 AND A20 AND !C3 AND C0)    {block refill}
                    OR      CAS1B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                            WRB AND !SB  AND (!A21 AND A20 AND C0)   {delayed block refill}
                    OR      CAS1B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                            !WRB AND !SB AND (!A21 AND A20 AND !C3 AND !C2 AND !C1 AND
                            !C0)   {page write}
                    OR      CIPB AND DRAMWNB AND !REFACKB AND CAS1B AND (!C3 AND !C2
                            AND C1 AND !C0) {refresh}
                    OR      CIPB AND DRAMWNB AND !REFACKB AND !CAS1B AND (!C3 AND !C2
                            AND C1 AND C0); {refresh}

CAS0B  NOT          :=      CAS0B AND RIPB AND !CIPB AND DRAMWNB AND (!A21 AND !A20
                            AND !C3 AND !C2 AND !C1 AND C0)   AND CAS0B {read or write}
                    OR      CAS0B AND RIPB AND !CIPB  AND !DRAMWNB AND (!A21 AND !A20
                            AND !C3 AND !C2 AND C1 AND C0)   AND CAS0B {delayed read/write}
                    OR      CAS0B AND RIPB AND !CIPB AND !BURSTB AND DRAMWNB AND
                            WRB AND  !SB AND (!A21 AND !A20 AND !C3 AND C0)    {block refill}
                    OR      CAS0B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                            WRB AND !SB  AND (!A21 AND !A20 AND C0)   {delayed block refill}
                    OR      CAS0B AND RIPB AND !CIPB AND !BURSTB AND !DRAMWNB AND
                            !WRB AND !SB AND (!A21 AND !A20 AND !C3 AND !C2 AND !C1 AND
                            !C0)    {page write}
                    OR      CIPB AND DRAMWNB AND !REFACKB AND CAS0B AND (!C3 AND !C2
                            AND C1 AND !C0) {refresh}
                    OR      CIPB AND DRAMWNB AND !REFACKB AND !CAS0B AND (!C3 AND !C2
                            AND C1 AND C0); {refresh}

DRAMLE  NOT         :=      TRB AND CAS3B AND CAS2B AND CAS1B AND !CAS0B {issue after}
                    OR      TRB AND !CAS3B AND CAS2B AND CAS1B AND CAS0B {any CAS if}
                    OR      TRB AND CAS3B AND !CAS2B AND CAS1B AND CAS0B {read cycle}
                    OR      TRB AND CAS3B AND CAS2B AND !CAS1B AND CAS0B
                    OR      CAS3B AND CAS2B AND CAS1B AND CAS0B;
```

```
        DRAMWRB  NOT        :=    !CIPB AND RIPB AND !WRB AND DRAMWRB {issue for write}
                            OR    !WRB AND !BURSTB AND !DRAMWNB AND DRAMWRB AND RIPB
                                  AND REFACKB {issue for page write}
                            OR    !CIPB AND !DRAMWRB AND CAS3B AND CAS2B AND CAS1B
                                  AND CAS0B AND RIPB; {keep until end of write}

        SB NOT              :=    SB AND !CIPB AND DRAMWNB AND (!C3 AND !C2 AND !C1
                                  AND !C0) {read/write/block refill}
                            OR    !SB AND !CIPB AND !BURSTB AND WRB AND !C3 {keep for block refill}
                            OR    SB AND !CIPB AND !DRAMWNB AND (!C3 AND !C2 AND C1
                                  AND !C0) {delayed read/write/block refill}
                            OR    !SB AND !CIPB AND !DRAMWNB AND !BURSTB AND WRB AND
                                  !C1 {delayed block refill}
                            OR    !SB AND !CIPB AND BURSTB AND WRB AND C0 AND CAS3B AND
                                  CAS2B AND CAS1B AND CAS0B {read and delayed read}
                            OR    !SB AND !CIPB AND !WRB  AND CAS3B AND CAS2B AND CAS1B AND
                                  CAS0B {keep for write}
                            OR    !WRB AND !BURSTB AND !DRAMWNB AND SB AND REFACKB; {page write}

        TRB NOT             :=    TRB AND !CIPB AND WRB AND DRAMWNB AND (!C3 AND !C2
                                  AND !C1 AND !C0) {read/block refill}
                            OR    TRB AND !CIPB AND WRB AND !DRAMWNB AND SB AND (!C3
                                  AND !C2 AND C1 AND !C0) {delayed read/block refill}
                            OR    !TRB AND !CIPB AND !SB; {keep asserted for read/block refill}

END;
END PAL2.
```

```
{
       TITLE:              PAL3
       PURPOSE:            COUNTER
       AUTHOR:             BOB NAPAA, IDT INC.
       DATE:               4/5/91
}

MODULE  PAL3;
TITLE  PAL3;
TYPE MMI 16R8;

INPUTS;

       {BSYSCLKB                NODE[PIN1];  }
       RESETB                   NODE[PIN2];
       REFREQ                   NODE[PIN3];
       BCIPB                    NODE[PIN4];
       DRAMWNB                  NODE[PIN5];
       {OUTENABLEB              NODE[PIN11];  }

       {FEED BACK PINS}
       RIPB                     NODE[PIN18];
       C3                       NODE[PIN17];
       C2                       NODE[PIN16];
       C1                       NODE[PIN15];
       C0                       NODE[PIN14];
       REFACKB                  NODE[PIN13];
       GATECOUNTERB             NODE[PIN12];

OUTPUTS;

       RIPB                     NODE[PIN18];
       C3                       NODE[PIN17];
       C2                       NODE[PIN16];
       C1                       NODE[PIN15];
       C0                       NODE[PIN14];
       REFACKB                  NODE[PIN13];
       GATECOUNTERB             NODE[PIN12];

TABLE;

       RIPB NOT                 :=    !RESETB       {reset}
                                OR    !RIPB AND !RESETB   {keep for reset}
                                OR    !RIPB AND REFACKB   {keep for refresh}
                                OR    !RIPB AND !REFACKB AND !C3; {keep until end of refresh}

       C3 NOT                   :=    !GATECOUNTERB AND !BCIPB AND REFACKB
                                OR    !GATECOUNTERB AND BCIPB
                                OR    GATECOUNTERB AND BCIPB AND REFACKB
                                OR    !C3 AND !C2
                                OR    !C3 AND C2 AND !C1
                                OR    !C3 AND C2 AND C1 AND !C0
                                OR    C3 AND C2 AND C1 AND C0;

       C2 NOT                   :=    !GATECOUNTERB AND !BCIPB AND REFACKB
                                OR    !GATECOUNTERB AND BCIPB
                                OR    GATECOUNTERB AND BCIPB AND REFACKB
                                OR    !C2 AND !C1
                                OR    !C2 AND C1 AND !C0
                                OR    C2 AND C1 AND C0;
```

```
C1 NOT                    :=    !GATECOUNTERB AND !BCIPB AND REFACKB
                          OR    !GATECOUNTERB AND BCIPB
                          OR    GATECOUNTERB AND BCIPB AND REFACKB
                          OR    !C1 AND !C0
                          OR    C1 AND C0;

C0 NOT                    :=    !GATECOUNTERB AND !BCIPB AND REFACKB
                          OR    !GATECOUNTERB AND BCIPB
                          OR    GATECOUNTERB AND BCIPB AND REFACKB
                          OR    C0;

REFACKB NOT          :=   REFACKB AND REFREQ AND RESETB    {for refreq}
                          OR    !REFACKB AND !BCIPB AND RESETB   {as long as cipb low}
                          OR    !REFACKB AND !C3 AND RESETB AND GATECOUNTERB
                     {keep asserted}
                          OR    REFACKB AND RESETB AND !RIPB {reset}
                          OR    !REFACKB AND !GATECOUNTERB; {keep for reset}

GATECOUNTERB NOT := GATECOUNTERB AND !REFACKB   AND !BCIPB AND RIPB
                                        {issue for both refack and cipb}
                          OR    !GATECOUNTERB AND !BCIPB AND RIPB
                                     {keep as long as cipb}
                          OR    !GATECOUNTERB AND !REFACKB AND RIPB AND C3
                          OR    !GATECOUNTERB AND !REFACKB AND RIPB AND C2
                          OR    !GATECOUNTERB AND !REFACKB AND RIPB AND C1
                          OR    !GATECOUNTERB AND !REFACKB AND RIPB AND C0;

END;
END PAL3.
```

Integrated Device Technology, Inc.

IDT79R3051™ MAIN MEMORY AND SYSTEM I/O INTERFACING

APPLICATION NOTE AN-92

By Andrew Ng

INTRODUCTION

The IDT79R3051™ RISController™ family provides a simple, flexible external bus interface to directly support main memory and system I/O resources. The bus interface is straightforward in that it uses a single, multiplexed 32-bit address and data bus and a small number of supporting control signals. The bus interface is adaptable in that it can handle different types and speeds of memory including DRAM, SRAM, and EPROM and different kinds of I/O resources. Thus the simple, flexible R3051 bus interface allows designers to make optimal trade-offs between system speed and cost issues.

MAIN MEMORY DESIGN

The R3051 normally accesses its internal instruction and data cache memories as in Figure 1, while using external main memory as a secondary source of memory as in Figure 5. Since the R3051 contains its own internal instruction and data caches, the complexity of the cache timing and interfacing is kept on-chip, which allows the external interface to be dedicated to main memory and system I/O interfacing. The system interface is decoupled from cache memory by the use of an internal 4-deep read buffer and an internal 4-deep write buffer.

The instruction and data cache allow the R3051 to access 1 instruction and 1 data word on each clock cycle. On reads, when a cache miss or an uncachable reference occurs, the R3051 begins an external read cycle which buffers 1 word on non-burst reads and 4 words at a time on burst reads from system I/O and main memory. On writes, the R3051 maintains a write-through cache update policy which simultaneously updates both the data cache and main memory. With the use of its 4-deep write buffer, the R3051 can continue to execute instructions from its instruction cache while the main memory retires up to 4 words from the write buffer.

Read and Write Cycle Protocols

The simple read interface allows a wide range of memories and I/O to be used with the R3051, from slow I/O peripherals to high-speed burst accessed DRAM and SRAM. As shown in Figure 2 and 3, the read interface supports both single datum accesses and 4-word burst accesses simply by providing a $\overline{\text{Burst}}$ output signal and by providing dedicated LSB address line outputs Addr(3:2) which are used as a word counter. System I/O or main memory is only required to acknowledge each of the 4 words with the $\overline{\text{RdCEn}}$ input which is used as a read clock enable to latch each word into the 4-deep read

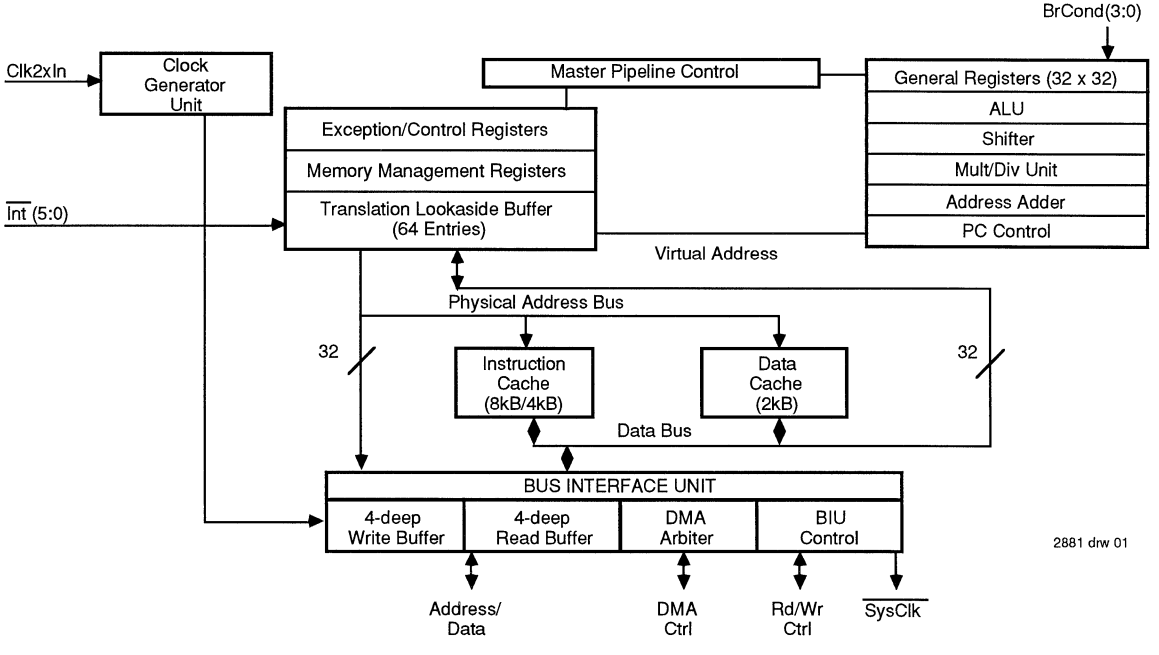


Figure 1. R3051 RISController Internal Architecture

buffer. Read interfacing also has the option of using the $\overline{Ack}$ acknowledge input signal to optimally control when the R3051 core restarts its pipeline on burst read cycles.

The simple write interface allows a wide range of memories and I/O to be used with the R3051 by buffering writes from the R3051 core which are done at cache speeds. This allows main memory and I/O to retire write cycles at their own rate of speed by returning $\overline{Ack}$, to acknowledge that the word has been received as shown in Figure 4.

Basic System Functional Blocks

The following sections will describe the functional blocks that are typical of R3051 main memory and system I/O interfacing. As shown in Figure 5 these blocks include:

- Address Demultiplexing
- Address Decoding and Chip Selection
- Data Transceivers
- Wait-State Controller and Interface Handshaking
- Read/Write Enables and Strobes

The discussion concentrates on the general interface blocks involved when using the following modules:

- SRAM Interfacing
- DRAM Interfacing
- EPROM Interfacing
- I/O Interfacing
- DMA Interfacing

Specific information on using the different memory and I/O types is presented in detail in other application notes.

ADDRESS DEMULTIPLEXER AND DECODER

The R3051 uses a multiplexed A/D(31:0) bus to output its address and to send and receive data. Thus main memory must de-multiplex the address by using the R3051's Address Latch Enable control signal, ALE, before decoding the address to select chip enables.

Latching A/D(31:0)

Transparent latches such as the IDT54/74FCT373 and the IDT54/74FCT841 pass inputs straight through to the outputs when their Latch Enable input is HIGH. When their Latch Enable input is LOW, the data in the latches are held constant. The R3051 provides the ALE output for direct connection to the transparent latches' Latch Enable pins. Transparent latches are typically used to allow address decoding to take place when ALE is HIGH and the address begins to become valid, instead of waiting until the latch closes.

The Address Latch Enable, ALE, is designed to clock the address into a transparent latch such as the FCT373. ALE is also designed to meet the address hold time of latches. As with all high-speed processors, ALE should be considered a critical signal. Thus Printed Circuit Board routing should minimize ALE's trace length and crosstalk susceptibility.

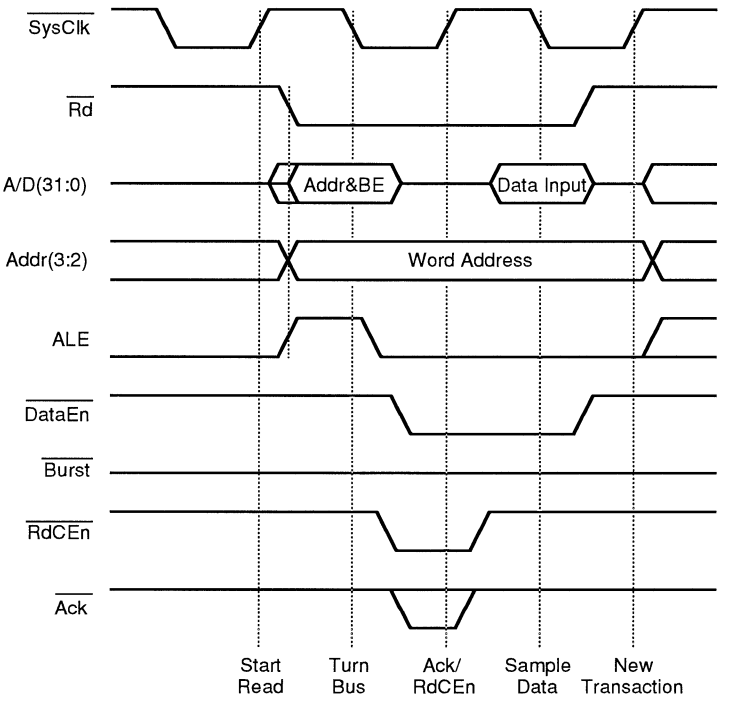


Figure 2. R3051 Single Word Read

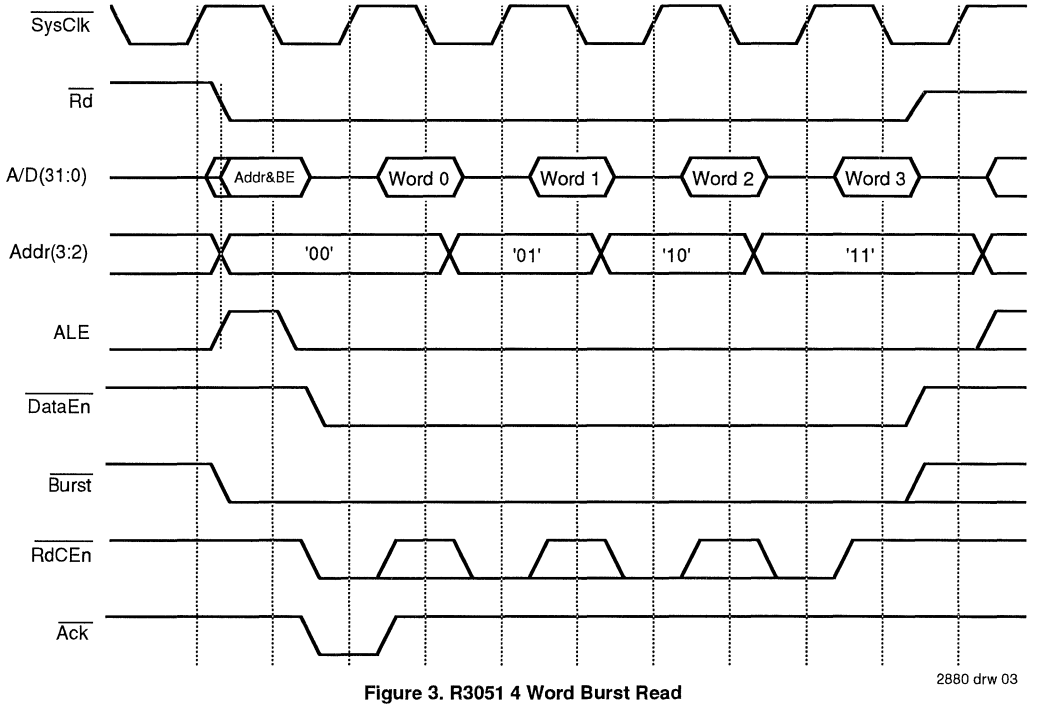


Figure 3. R3051 4 Word Burst Read

Decoding A(31:0)

Address decoding, which selects between the various memory and I/O banks in the system, can be done with IDT54/74FCT138/139 decoders as shown in Figure 6.

The time for the main memory chip selects to become valid in such a scheme is:

$$tDecode = \max(t3051ALEProp + t373LEtoO, t3051AddrProp + t373DtoO) + t138AtoO + tCap$$

Systems that require the chip selects to not have decoding glitches while the address drives to a valid value can register the decoder outputs by using SysClk as the clock and a CycleStart signal as the clock enable. The CycleStart signal is derived from the Rd and Wr control lines so that it asserts at the beginning of every memory cycle.

Decoding Byte Enables with Chip Selects

During the address phase, the R3051 uses the lower 4 bits of the multiplexed A/D(31:0) bus to output BE(3:0). Byte enables are used to determine which bytes of each word are being read or written to support partial word accesses. Because BE(3:0) are used throughout the memory cycle, they are latched by ALE along with the other A/D bits.

In general, it is permissible to process all reads as 32-bit reads—the processor will only take the data it requested from the bus. However, in write operations, the system must insure that only the specified bytes are written. Thus, the byte enable outputs are used to control this.

There are two ways in which the byte enables may be used:

- Gate the byte enables with the memory chip selects. Thus, only those bytes of memories which will be written are selected. A single write enable can then be presented to all banks of that memory subsystem. This solution requires that each memory subsystem further decode the chip-selects, and thus one decoder per memory subsystem is required.
- Gate the byte enables with the memory chips read/write enables/strobes. Thus, although all of the devices in that bank of memory are "selected", only those bytes to be written are enabled for the writes. This is a common strategy in DRAM subsystems. Note that the individual byte strobes may be broadcast to all memory systems, and the address decoder will insure that only one subsystem is "Selected". Thus, a single decoder for byte enables can serve the entire memory system.

If the memories being used are 1-bit to 8-bits wide, gating the byte enables with the chip selects can be done. Because the byte enables are predetermined within the R3051 by using the LSB address bits, the endianness of the system, and the type of load or store instruction, the byte enables have the same timing as the rest of the A/D lines during the address phase when ALE is asserted. This allows a memory decoder to have individual chip selects for each byte of each bank with no timing penalty. An example is shown in Figure 7.

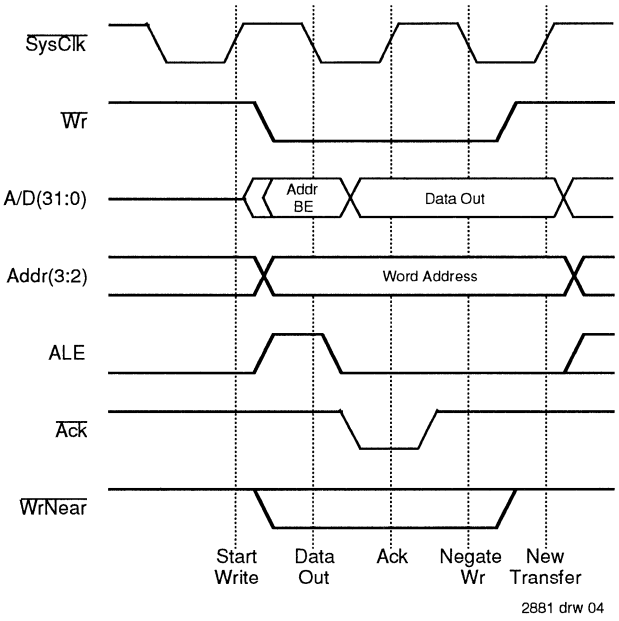


Figure 4. R3051 Single Word Write

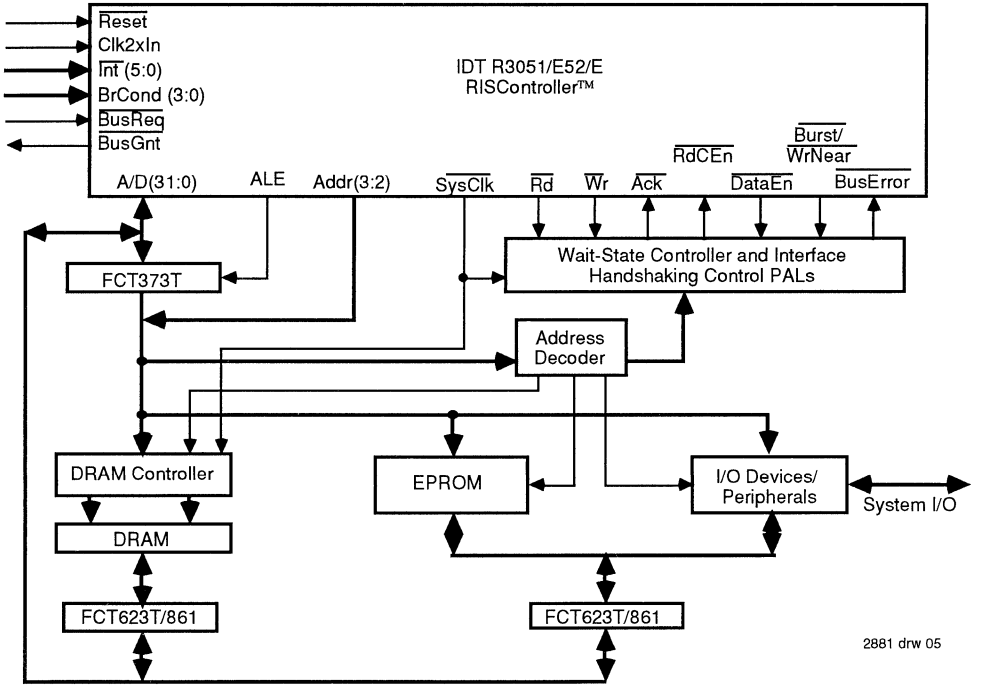


Figure 5. R3051 with Main Memory

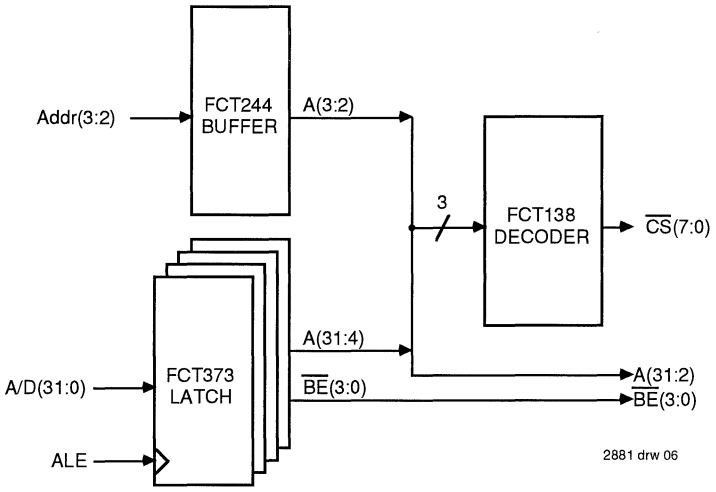


Figure 6. Address Demultiplexer and Decoder

As gating the byte enables with the chip selects usually takes more output pins than gating the byte enables with the read and write enables, the latter is usually preferred. The use of byte enables with read/write enables will be discussed in the read/write enable/strobe section.

Using Addr(3:2)

Since the lower 4 A/D bits are used for byte enables during the R3051's address phase, the R3051 provides the information for addressing words through its Addr(3:2) output pins. The R3051 uses 4 bytes-per-word and pre-decodes the byte enables instead of providing the 2 LSB address lines. Addr(3:2) are driven throughout external bus cycles and do not require latching. During non-burst read cycles and all write cycles, Addr(3:2) contains the instruction cache miss address. The advantage of dedicating output pins for Addr(3:2) is that during burst read cycles, Addr(3:2) are incremented from 0 to 3 by the R3051 $\overline{\text{RdCEn}}$ protocol so that the system memory system does not have to provide a counter for this function.

Since each memory chip requires Addr(3:2), large memory systems that use Addr(3:2) extensively may want to use buffers. A common strategy may be to provide a buffered version of Addr(3:2) to non-time critical areas of memory (e.g. the boot prom), or to areas which do not perform burst accesses (I/O devices), and directly use the outputs of the R3051 in time-critical areas such as the DRAM control.

The crossover point where buffering is appropriate can be determined by determining if the delay through an IDT54/74FCT244 buffer and the capacitive derating from all the Addr(3:2) inputs driven by the buffer (Addr(3:2) can be buffered for separate branches of memory banks) would be less than the delay from the capacitive derating from all the Addr(3:2) inputs driven directly from the R3051. In addition, the crossover doesn't occur until Addr(3:2) is delayed past when rest of the A(31:4) lines reach their inputs.

$$t3051Addr(3:2) + t244 + t244Cap \leq \max(t3051Addr(3:2) + t3051Cap, tA(31:4))$$

where:

$$t244Cap = (\text{sum}(CInput/Output) + C244 + tTrace - 50)/33\ pf/nsec$$
$$t3051Cap = (\text{sum}(CInput/Output) + C3051 + tTrace - 25)/25\ pf/nsec$$

Using Diag(1:0)

Some systems may need to know whether a read cycle is cachable or uncachable and whether a cachable read cycle is an instruction or a data fetch. In Figure 8, this information is provided by latching the diagnostic pins, Diag(1:0) with the same latch controls as the address lines. These signals are useful during reads for:

- Decoding whether a read in the lowest half GB of physical memory is from kseg0 or kseg1.
- Tracing processor execution by knowing which address caused the I-Cache miss.

DATA TRANSCEIVERS

The R3051 uses a multiplexed A/D(31:0) bus to output its address and to send and receive data. Thus main memory must drive or receive data after the R3051 has tri-stated its address. Further, to support high-performance memory systems, the R3051 family is capable of initiating a new bus transaction one-half clock cycle after data is sampled for a read operation.

Determining if Data Transceivers are Needed

Multiplexed CPU busses often use data transceivers to separate the memory system from the processor bus. Read cycles require the memory system to stop driving data on the A/D bus before the processor drives the next memory cycle's address. Slow memories with relatively long output disable times cannot meet this limitation without data transceivers.

However, some memories, such as the IDT71B256 BiCMOS 32k x 8 Static RAM, have very short access time and output disable time which makes it possible to consider attaching memory device data I/O pins directly to the multiplexed A/D(31:0) bus. Alternatively, in low-frequency systems, the amount of time provided by the R3051 may be sufficient for the memory devices attached to the bus.

The key parameter is the memory output disable time, tOZ, which has to be less than 1/2 clock to disable before the next memory's address is driven. In addition the address and data driven from the R3051 is delayed because of the extra capacitance of the memory data I/O pins.

$$tOZ \leq tSysClk/2 - tDisableControl + \min(t3051Addr)$$

Data Transceivers also serve to isolate memory banks from each other. In systems with varying speeds of memory, transceiver banks can be used to separate chips with relatively long output disable times from those with relatively quick output disable times. Thus in many systems, fast scratch-pad SRAMs may have their own set of transceivers, while slower EPROMs and I/O peripherals might have a separate set of transceivers.

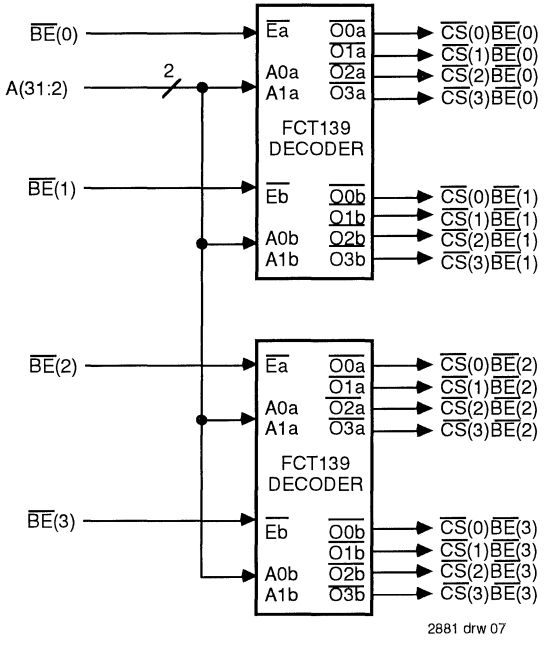


Figure 7. Gating Byte Enables with Chip Selects

Using IDT54/74FCT861s and IDT54/74FCT245s for Data Transceivers

Most systems will use slower memories and thus require data buffering through a transceiver interface. There are two basic families of transceiver interfaces:

1. IDT54/74FCT861 with separate enable pins for each direction.
2. IDT54/74FCT245 with a direction pin and an enable pin.

Using IDT54/74FCT861s for Data Transceivers

The 10-bit transceiver FCT861 approach functionally combines two 10-bit tri-statable FCT827 buffers internally. The 8-bit FCT623T transceiver is similar to the FCT861 except that one of its output enables is Active-HIGH. On read cycles, if there is only one transceiver bank, then $\overline{DataEn}$ can be used directly to control the read direction output enable. Otherwise, combinational logic such as an FCT157/257 multiplexer can be used to combine $\overline{DataEn}$ with the chip selects of the bank whose transceivers need to be enabled (see Figure 16 for a similar common input OR gate circuit). Alternatively, some transceivers, such as the 9-bit IDT54/74FCT863 and the 8-bit IDT54/74FCT543 have two logically AND'ed output enables for each direction so that $\overline{DataEn}$ and the bank chip select can be hooked up directly to the transceiver. State machines using an inverted $\overline{SysClk}$ can also use a $\overline{Rd}$ derived signal to synchronously assert and de-assert the read direction output enable.

The write direction output enable can use a signal derived from Wr which asserts at the beginning of the cycle and waits until after the data has been strobed into the memory or I/O device before de-asserting to provide sufficient data setup and hold time. For systems with 1 wait-state or more, the derived write direction enable signal should ideally assert after the A/D bus finishes driving its address phase to reduce switching noise.

The transceiver control's critical timing path is the transition from a read cycle to a write cycle. After a read cycle, slower memory chips take a relatively long time to disable from the data bus. If the next memory cycle is a write, the transceivers will drive data onto the same bus. Such systems can use the second memory cycle's wait-states to delay the assertion of

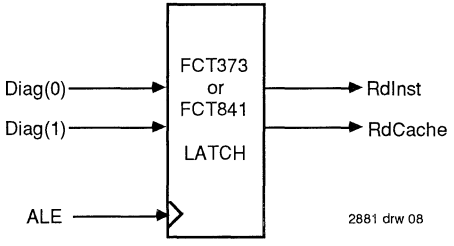


Figure 8. Latching Diag(1:0)

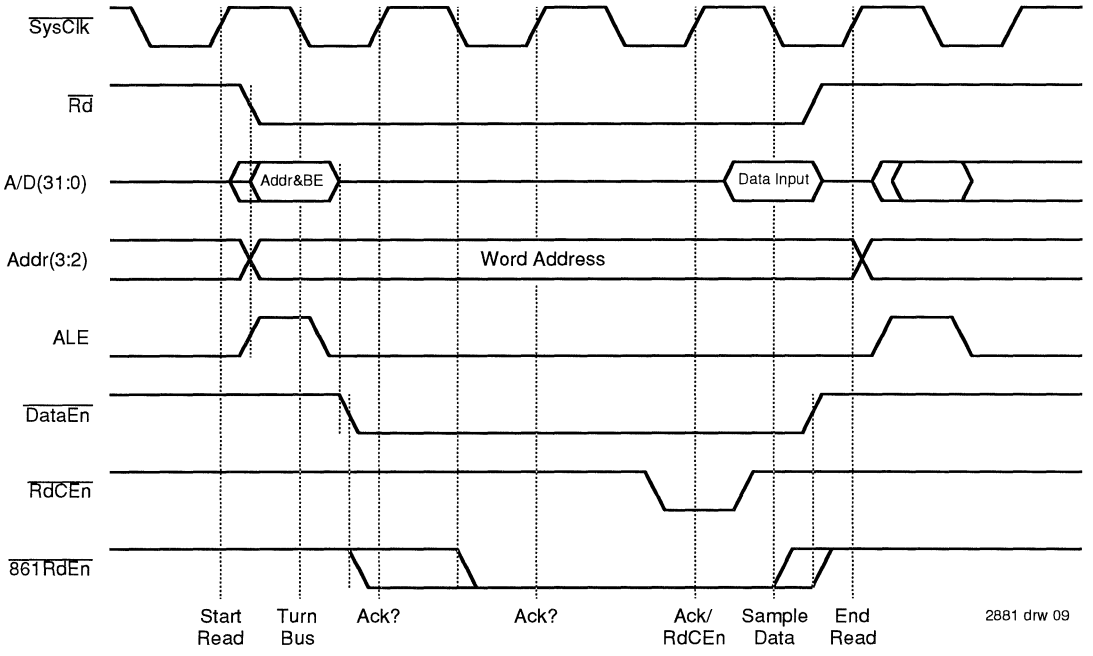


Figure 9a. Timing Diagram of FCT861 Read Direction Enable

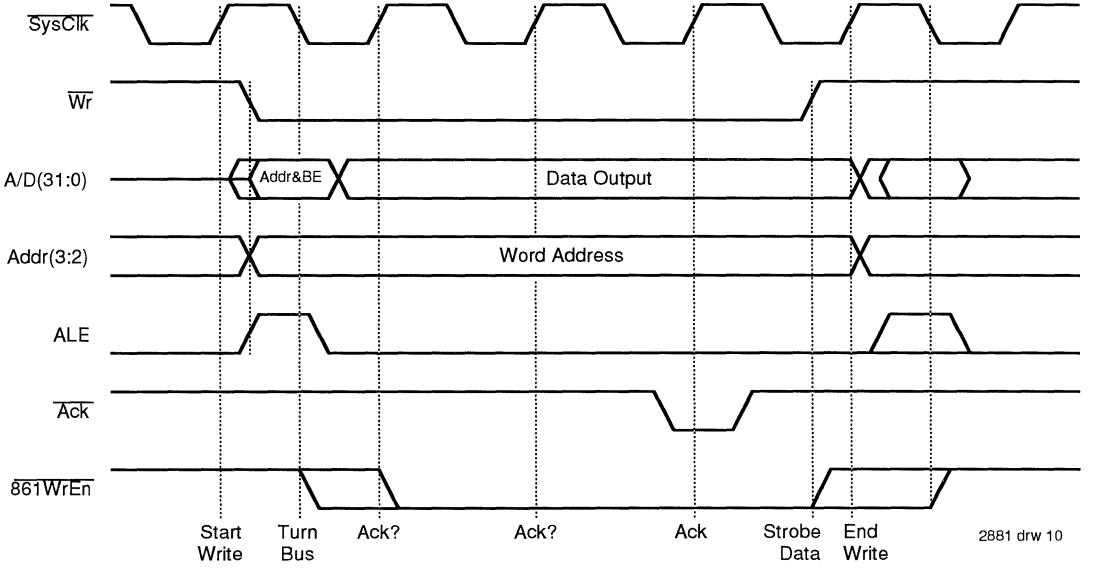


Figure 9b. Timing Diagram of FCT861 Write Direction Enable

the transceiver's write direction output enable until the first memory cycle's memory has fully disabled. The cutoff for determining if the memory output disable time is small enough to require no wait-states is:

tSysClk >= tDisableControl + tMemReadDisable - tWriteData

Systems that use memory chips without an output enable pin (i.e., a read is implied for every chip select with no write enable) require special transceiver interfacing in order to support partial word writes. During partial word writes, where only some of the bytes are selected for writing, bytes which are not being written may actually output onto their byte lanes, and thus conflict with the transceiver write direction outputs. In such memory subsystems, there are two options: only chip select those devices actually being written into; or, only enable those transceivers whose byte lanes are used in this write transfer. Either of these solutions will insure that no bus conflict occurs.

Using IDT54/74FCT245s for Data Transceivers

The 8-bit FCT245 transceiver approach ideally requires that the direction control only be changed when the outputs are disabled to prevent bus contention. Although such systems are easy to design, this general discussion uses the following assumptions:

1. Either a $\overline{\text{SysClk}}$ or SysClk-based state machine is used.
2. The memories require at least 1 wait-state.

The output enable of an FCT245 needs to be determined by finding the start and end of the memory cycle, which can be determined by logically ANDing $\overline{\text{Rd}}$ and $\overline{\text{Wr}}$. The assertion of the output enable can be easily delayed to occur well after the transfer, depending on the number of wait-states in the memory controller. That is, the transceiver only needs to be enabled in time to allow the data to propagate through to the CPU as the read data response is finally returned to the processor. In read cycles, the output may be disabled using the same clock edge as is used by the CPU to negate $\overline{\text{Rd}}$. On write transactions, the transceiver must be enabled until the data set-up and hold time requirements of the memory being written are met, which may extend until the next falling edge of $\overline{\text{SysClk}}$ (note for the R3051, the processor guarantees that valid data will remain for one-half clock cycle after the negation of $\overline{\text{Wr}}$).

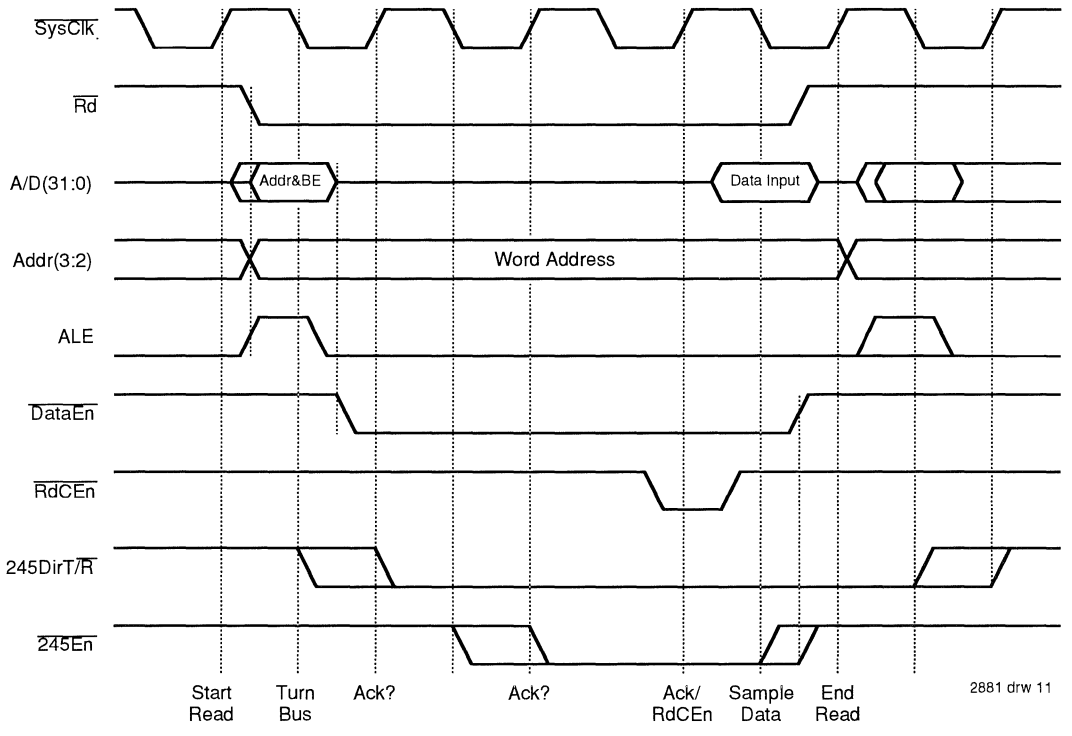


Figure 10a. Timing Diagram of FCT245 Enable and T/$\overline{\text{R}}$ Direction Controls for a Read

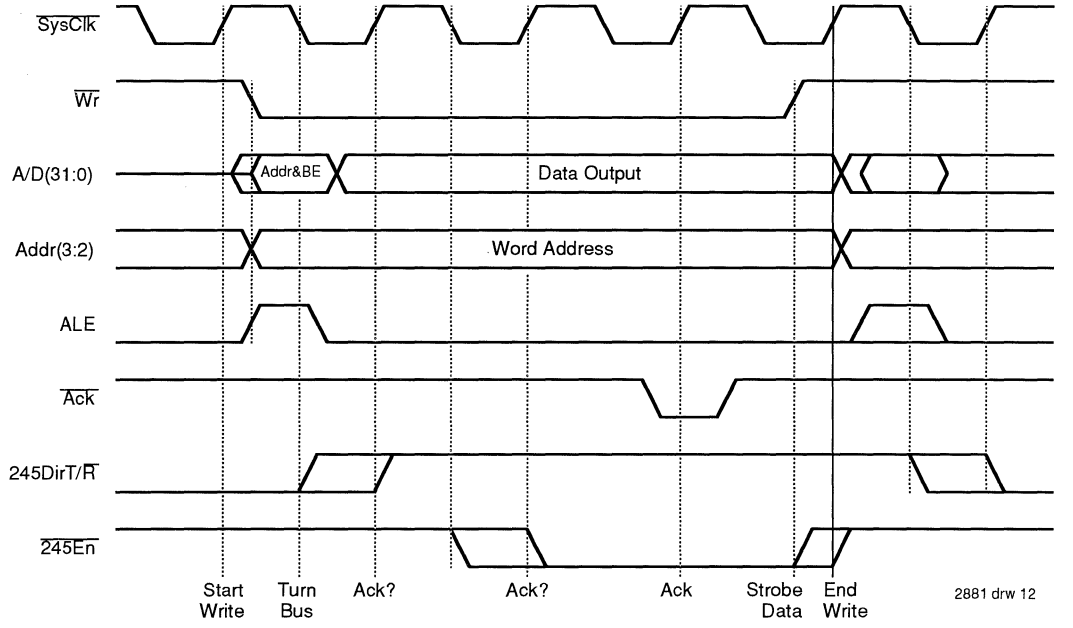


Figure 10b. Timing Diagram of FCT245 Enable and T/R Direction Controls for a Write

The T/R direction pin of the FCT245 should be asserted before the output enable asserts, which can be achieved by using a Rd or Wr derived signal. The direction should be held until the next clock edge after Rd or Wr de-asserts; that is, until after the output enable is de-asserted.

Systems that use memories without a dedicated output enable pin require separate byte output enables in the data path, as discussed above.

PULL-DOWN/UP RESISTORS ON R3051 OUTPUTS

The R3051 tri-states its outputs under three conditions:

1. If no external read or write memory cycles are being executed, the A/D bus will tri-state. Control signal outputs will be driven to negated states.
2. If a DMA bus grant is given, all bus interface outputs will tri-state.
3. If the Tri-State reset mode has been invoked, all outputs except SysClk will be tri-stated.

The following paragraphs detail which outputs are affected when the R3051 is in a tri-stated condition.

Pull-down/up Resistors on the A/D Bus

The R3051 tri-states the A/D bus when it finishes a write (or read) cycle and there is not another pending memory cycle that it needs to execute. This situation occurs when the R3051 is getting instructions from its internal instruction cache and it executes a sequence without store instructions. Since the A/D bus can be tri-stated for these periods, it is desirable for the input pins of the address latches and data transceivers to maintain the A/D bus with defined, valid logic values by using pull-up/pull-down resistors. The use of pull-up or pull-down resistors also has the benefit of easing Automatic Test Equipment programming on board-level and in-circuit tests.

Pull-down/up Resistors on Control Lines for DMA

The R3051 has an on-chip Direct Memory Access (DMA) arbiter that allows outside processors and controllers to take control of the external memory systems, and perform transactions. It does this by indicating a request to the R3051, which then tri-states its bus interface to allow it to be driven by the external agent.

During DMA, the R3051 will execute instructions from its internal caches until it has a cache miss, makes an uncacheable reference, or its write buffer becomes full.

An external agent requests bus mastership by asserting the R3051 BusReq input. If BusReq is asserted by the DMA device, the R3051 tri-states its outputs and asserts BusGnt to signal to the DMA device so that it can begin to drive its own memory cycles. During DMA, the R3051 tri-states all outputs except SysClk and BusGnt. During the time that the R3051 and the DMA controller transfer control back and forth, neither one drives the control line outputs (to avoid bus conflicts). In order to properly transfer control, the R3051 control outputs should be kept in their de-asserted state. If the transfer time is relatively short, the system designer may choose to rely on bus capacitance to hold these signals in their negated positions. Alternatively, a more conservative strategy is to hold the bus in a negated position with pull-down or pull-up resistors. Thus Rd, Wr, Burst/WrNear, and DataEn should use pull-up resistors and ALE should use a pull-down resistor.

Pull-down/up Resistors on Control Lines for Tri-State

The R3051 has a reset mode vector which allows the chip to tri-state all its outputs, except SysClk. This mode is attained by asserting Tri-State via SInt(1) while Reset is asserted. In addition to the control lines above, BusGnt is tri-stated. Thus for Automatic Test Equipment programming on board-level and in-circuit testing, a pull-up resistor for BusGnt can be used.

WAIT-STATE CONTROLLER LOGIC

Wait-states are used to extend the number of clocks within a memory transfer to provide sufficient memory access and data setup time for the particular type of memory being accessed. Such control can be provided with a wait-state controller state machine. In general, a wait-state machine has four steps:

1. Detect the beginning of a memory cycle.
2. Determine the type of cycle:
   a. Which chip select (address decode)
   b. Read or write
   c. Single word or burst, write near or non-page write.
3. Count out cycles until memory is ready and assert R3051 handshaking signals.
4. Acknowledge the end the cycle.

Thus, the basic control strategy is to use a counter which is held at zero until a cycle is started, and which then increments every clock cycle until the transfer is completed. This master counter then provides the reference by which control outputs to the memory, data path, and CPU are provided.

R3051's Use of Both Clock Edges

The R3051 uses both edges of the clock to assert and de-assert its control signals. This is to ameliorate the fixup time between memory cycles, which for most processors, takes 1 full clock cycle. The R3051 is able to do the fixup in 1/2 clock cycle. This would seem to complicate the design of state machines which must latch these signals synchronously to one edge or the other. However, as will be shown in the following sections, a traditional state machine that follows a small number of simple design rules can still use a single edge clock.

The R3051 uses an input clock, Clk2xIn, that runs at twice the frequency of the processor. The R3051 provides an output clock, SysClk, that runs at the same frequency as the processor and can be used to clock external state machines. The polarity of SysClk was chosen intentionally so that either an unbuffered SysClk or an inverted version of SysClk, (referred to here as SysClk) can be used. Because all the R3051 control outputs have very short propagation delays (less than 1/2 clock), a state machine can use either edge of SysClk.

In developing the set of constraints brought on by the use of both the rising and falling clock edges, some observations can be made:

1. All clockable control line outputs, except DataEn assert off the rising edge of SysClk.
2. All clockable control line outputs de-assert off the falling edge of SysClk.
3. All control line inputs required by the R3051 are sampled on the rising edge of SysClk.

Observations 1 and 2 can be specifically applied to two of the primary control signals, Rd and Wr.

1. Rd and Wr both assert off the rising edge of SysClk.
2. Rd and Wr both de-assert off the falling edge of SysClk.

The similarity of edge assertions for Rd and Wr can be used to simplify the wait-state controller.

Detecting the Beginning of a Memory Cycle

State machines looking for the beginning of a memory cycle can look for one of two things:

1. Rd or Wr asserting.
2. ALE asserting.

In general, state machines have to choose between using SysClk and SysClk. State machines such as those implemented in ASICs can use both clock edges, however, to simplify the discussion it will be assumed that only one or the other clocks is being used. If SysClk is used, certain registers must use SysClk directly from the processor to provide sufficient hold time from the processor. Only a negative edge clocked register can synchronously clock ALE under worst case timing, since ALE is only HIGH surrounding the falling SysClk edge which requires a negative edge triggered flip-flop. SysClk cannot be used because its inverter delay will put it past when ALE could fall.

Machines which use SysClk (the inverted SysClk) will have a delay from inverting SysClk. All state machines can use Rd and Wr to determine the beginning of a cycle. SysClk machines are able to do this easily with wide margins on setup and hold times to its registers. SysClk machines must use SysClk directly from the processor and use registers with 0 hold time and also have a guaranteed minimum clock to output delay to meet the R3051's input hold time.

Determining the Type of Memory Cycle

The type of memory cycle usually depends on the following variables:

1. Type of memory.
2. Read or write cycle.
3. Burst or non-burst, write near or non-page write.

These three variables are usually logically ANDed together to form equations for determining the number of wait-states before asserting RdCEn, Ack, or BusError as well as any transceiver controls. The chip selects from the memory decoder can be used to determine the type of memory to count the correct number of wait-states. By using the R3051's Rd and Wr lines, the transceiver controls can be defined. On read cycles, the R3051's Burst/WrNear line determines if 1 word or 4 words are to be returned. On write cycles, Burst/WrNear determines if a consecutive write is on the same 256 word page as its predecessor. An example of a state transition diagram that uses the read/write and burst/non-burst variables for one memory type is shown in Figure 11. Each memory type in the system also has a state diagram.

Further variables that affect the type of memory cycle are implied by the mode initialization vector which is supplied

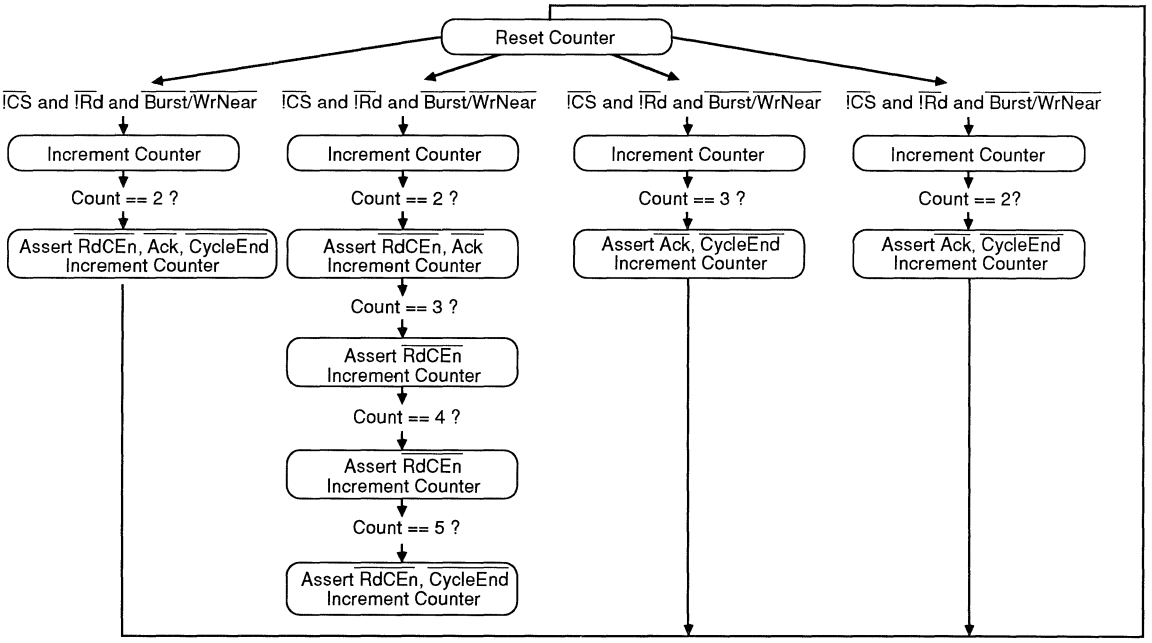


Figure 11. State Diagram of an Example Wait-State Controller for a Single Memory Type

during processor reset initialization. The variables determine whether the data byte ordering is Big or Little Endian and whether data cache miss refills are handled one word at a time or as 4 word block refill reads. BigEndian and DBRefill are set by multiplexing the interrupt lines on the de-assertion of reset, an example of which is shown in Figure 12.

The mode vector of the R3051 was chosen to allow it to be supplied by just using pull-up resistors on the appropriate interrupt inputs. For example, the multiplexer shown in Figure 12 could be eliminated, and the pull-up resistors tied directly to the SInt(2:0) pins.

Note that to maintain compatibility with future versions of the R3051 family, $\overline{\text{Int}}$(5:3) should be HIGH when $\overline{\text{Reset}}$ is de-asserted. This also can be performed using pull-up resistors.

Memory Interface Handshaking

The R3051 uses two inputs, $\overline{\text{RdCEn}}$ and $\overline{\text{Ack}}$,to indicate that the memory system is ready to receive or return data. On read cycles, $\overline{\text{RdCEn}}$ is sampled on the rising edge of $\overline{\text{SysClk}}$ by the R3051 so that it can enable its internal read buffer clock on the next falling edge of $\overline{\text{SysClk}}$. Thus on single word reads, a single $\overline{\text{RdCEn}}$ is asserted as the memory becomes ready as shown in Figures 2 and 11. On 4 word burst reads, $\overline{\text{RdCEn}}$ is asserted for each of the 4 words. Thus on burst reads, the wait-state controller can optionally "throttle" each word into the R3051 by delaying the return of each word by a varying number of clocks. $\overline{\text{RdCEn}}$ can be generated by gating the memory type and the count:

```
RdCEn not := Reset and CycleEnd and BusError and (
                (!RamCS and !Rd
                        and ( (Counter == 02H)
                                or (!Burst/WrNear and (Counter == 03H))
                                or (!Burst/WrNear and (Counter == 04H))
                                or (!Burst/WrNear and (Counter == 05H))
                        )
                )
);
```

The acknowledge input, $\overline{\text{Ack}}$, has two uses. On reads, $\overline{\text{Ack}}$ can be used to optimize the processor execution engine restart. On writes, $\overline{\text{Ack}}$ is used to signal the end of the cycle, as will be explained later. The R3051 throttles burst reads into its internal read buffer at the rate of the memory system; however, it reads data from the read buffer on every clock cycle. Therefore, the R3051 will either wait until the last $\overline{\text{RdCEn}}$ has occurred to begin reading the internal read buffer, or until the memory system signals $\overline{\text{Ack}}$ to the processor. Asserting $\overline{\text{Ack}}$ on a read cycle causes the R3051 to start reading words from the read buffer in the next cycle; thus, the memory system times the assertion of $\overline{\text{Ack}}$ so that the last word can be presented by the memory system just before it is read from the read buffer. Thus for optimal speed burst reads, $\overline{\text{Ack}}$ should be asserted 3 clocks before the last $\overline{\text{RdCEn}}$ occurs, as shown in Figure 3. For optimal single datum reads, $\overline{\text{Ack}}$ should be asserted at the same time as $\overline{\text{RdCEn}}$.

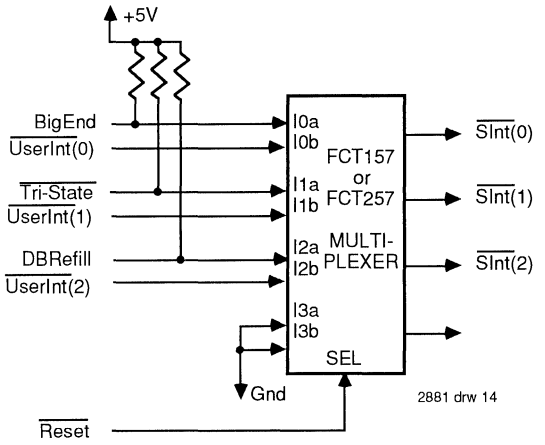


Figure 12. Reset Vector Circuit

On write cycles, Ack is sampled on the rising edge of SysClk by the R3051 so that the cycle ends on the next falling edge of SysClk as shown in Figure 4. Ack is used by the wait-state controller on write cycles to acknowledge that data is being strobed into memory. Ack can be generated by gating the memory type and count.

Note that in writes, the WrNear output from the processor may also affect the write timing. For example, when writing to Page Mode DRAMs, it will be possible to retire near writes faster than non-near writes.

An example of generating Ack from gating the memory type and count is:

```
Ack not := Reset and CycleEnd and BusError and (
                (!RamCS and !Wr
                        and ( ( Burst/WrNear and (Counter == 03H))
                                or (!Burst/WrNear and (Counter == 02H))
                        )
                )
        or      (!RamCS and !Rd
                        and (Counter == 02H)
                )
);
```

Stopping the Counting

Four common ways to end the memory cycle and stop the counter include:

1. Use a SysClk state machine and look for the de-asserting edge of Rd or Wr.
2. Use a SysClk state machine and gate the type of cycle into the counter to reset it independently of the de-asserting edge of Rd and Wr (predict the end of the cycle).
3. Use registers with asynchronous resets and gate Rd and Wr into the reset.
4. Interlock a SysClk register looking for the asserting edge of Rd or Wr with a SysClk register looking for the de-asserting edge of Rd or Wr.

In method 1, the SysClk registering of Rd or Wr is straightforward. However, if the counting is based on SysClk, the state machine will not be able to bring Ack or RdCEn LOW during the first possible clock cycle that they are sampled for by the R3051. This is, because the state machine will not detect the assertion of Rd or Wr in time. This implies that a SysClk-based state machine will have a minimum of one or more wait-states.

In method 2, SysClk-based state machines must determine when to stop counting independent of the de-assertion of Rd or Wr. In general they cannot use Rd or Wr to terminate the cycle because Rd or Wr may de-assert within the buffered (inverter delayed) SysClk register's setup or hold time. Thus SysClk-based state machines should use its counter to determine when the cycle will end, e.g., with CycleEnd. CycleEnd or a similar signal uses the chip selects and a counter to determine the end of the memory cycle, without using the de-asserting edges of Rd and Wr. Logic equations for CycleEnd and the LSB of an N-bit binary up counter look like:

```
CycleEnd not := Reset and CycleEnd and (
                (!RamCS and (Counter == 02H) and !Rd and Burst)
                (!RamCS and (Counter == 05H) and !Rd and !Burst)
                (!RamCS and (Counter == 03H) and !Wr and Burst)
                (!RamCS and (Counter == 02H) and !Wr and !Burst)
                ({Bus Error Timeout} (Counter == 0FH))
);

Counter(0) := Reset and CycleEnd and BusError and (!Rd or !Wr)
                and (Counter(0) xor 1)
;
```

A Timing Diagram of CycleEnd showing how CycleEnd asserting at the end of the memory cycle will reset the wait-state counter independently of Rd and Wr is shown in Figure 13.

Counters using CycleEnd use the type of cycle to determine when the wait-state counter should stop and reset independent of the de-asserting edge of Rd or Wr.

Wait-state machines implemented in ASICs can consider using method 4 which involves interlocking SysClk and SysClk-based registers as shown in Figure 15. ASICs can also selectively combine two independent SysClk and SysClk state machines to avoid 1/2 cycle interlock timing constraints.

Bus Errors

Bus errors can be handled by timing out with the wait-state controller counter as it is about to overflow. For all types of memory cycles, the R3051 de-asserts its control edges, e.g., Rd or Wr, on the clock following the assertion of BusError. SysClk-based state machines can look for the de-asserting edge of Rd or Wr in order to reset the wait-state machine's counter. In SysClk-based state machines, BusError can directly reset the wait-state machine's counter or the overflow count can be used to assert CycleEnd which will then reset the counter.

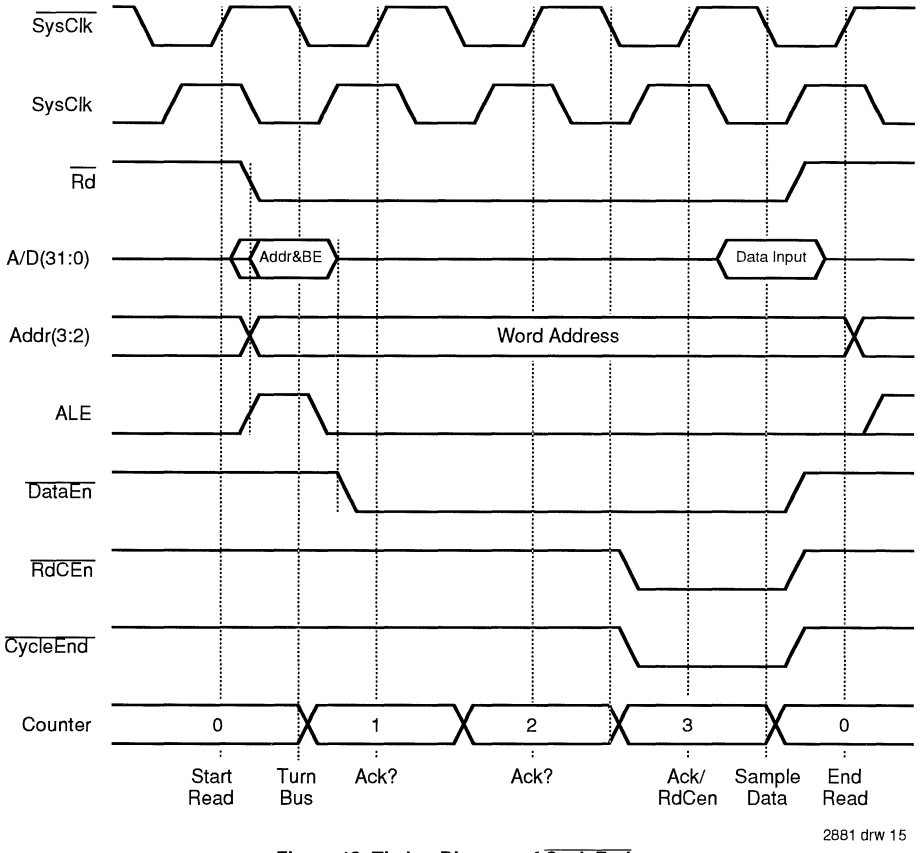


Figure 13. Timing Diagram of $\overline{\text{CycleEnd}}$

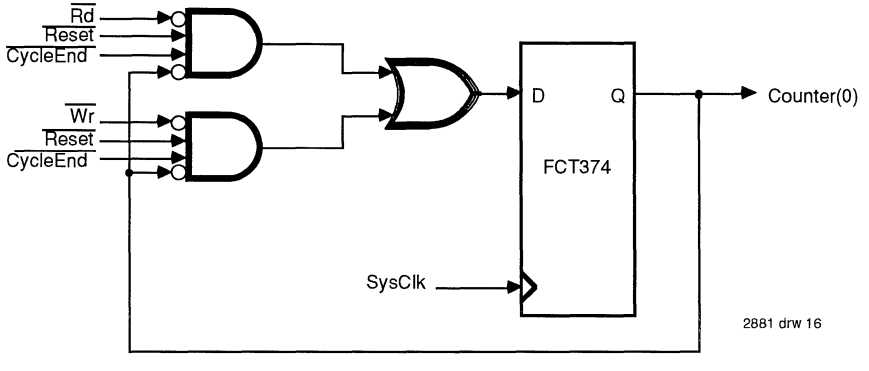


Figure 14. Using $\overline{\text{CycleEnd}}$ in a SysClk Based Counter

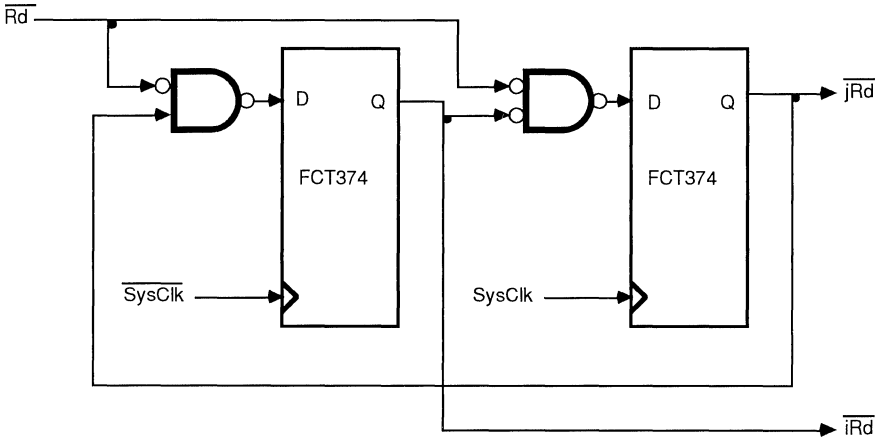


Figure 15. Using Interlocked Registers

Bus errors signal an exception to the R3051 only if it is a read cycle. If exceptions need to be noted for write or DMA cycles, BusError should be gated into an interrupt line. The interrupt must be held until the R3051 can acknowledge it, since the R3051 re-registers its interrupt inputs on each clock cycle in which it is executing instructions in its run or fixup state.

READ ENABLES AND WRITE ENABLES

Memories and I/O devices have a combination of chip selects, read enables, and write enables to drive data out of the device and to strobe data into the device. Because the exact timing and functions of the selects, enables, and strobes differ for DRAM, SRAM, and I/O, this section discusses read and write enables and their relationship to the byte enables.

Read Enables

In general, a memory or I/O device has an output enable pin to enable its data outputs on a read cycle. Typical designs will address all 8-bit and 16-bit I/O devices using 32-bit word addressed, (i.e., use Addr(3:2) as their LSBs). Even though the R3051 produces byte enables on read cycles, it is rare to require use of the byte enables for reads as the R3051 will internally mask the bytes not being used. The output enable for the device can be derived from Rd or from DataEn.

If more than one memory device uses a single transceiver, it may be necessary to generate device Output Enables using a delayed version of DataEn. If one of the memory or I/O devices has a long output disable to tri-state time, then extra time must be allowed for that device to tri-state before another device is enabled. An equation determining if the read enables should be delayed on a back to back read cycle is:

tSysClk >= tDisableControl + tOldMemoryDisable - tNewMemoryData + tCap

The output enable control should be asserted at least until the clock cycle that Rd and DataEn de-assert to provide sufficient data hold time to the R3051.

Gating Write Enables and Byte Enables

Memory and I/O devices have a write enable pin or a similar protocol to strobe data into the device. A special case occurs for partial word stores, where only the pertinent bytes of a word have their byte enables asserted. Partial word stores occur when a store byte, store half-word, or store tri-byte instruction is executed. Because of the efficiency and optimization capabilities of modern compilers, such as the MIPS® and IDT Compilers for the R3000™ family, the hardware must always assume that the software will make use of the partial word store instructions. Thus the write enables (or as shown earlier the chip selects) of each byte of a word must be gated with their respective byte enables. Gating the byte enables into the write enables can be done with an FCT157/257 multiplexer by configuring it as a set of four OR gates with a common input term as shown in Figure 16. The write enable signal can be derived from Wr.

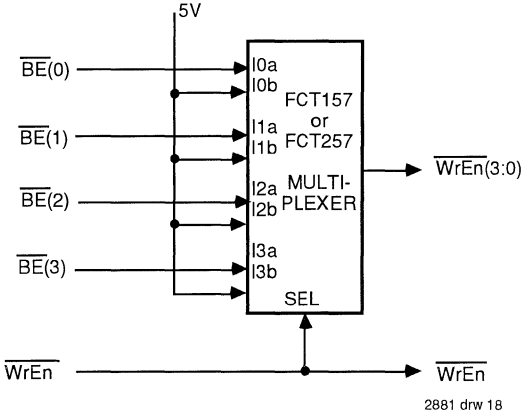


Figure 16. Gating Byte Enables into the Write Enables

SUMMARY

The main memory interface of the R3051 is conventional and simple. Basic blocks include address de-multiplexing, address decoding, data transceivers, wait-state controller, as well as the memory and I/O modules themselves. The R3051's uses both edges of the clock for control signals to reduce inter-cycle latency. Thus conventional wait-state controller algorithms can be used if the following guidelines are followed:

1. In $\overline{\text{SysClk}}$-based wait-state controllers, the input clock should be unbuffered from the processor's $\overline{\text{SysClk}}$ output. $\overline{\text{SysClk}}$ controllers will have a minimum of 1 or more wait-states. $\overline{\text{SysClk}}$ registers require small hold time and a minimum clock to output propagation delay to meet the R3051 input hold time.
2. In SysClk (inverted version of processor $\overline{\text{SysClk}}$ output) based wait-state controllers, the master reference counter must be reset independently of the de-asserting edges of $\overline{\text{Rd}}$ or $\overline{\text{Wr}}$. This can be done by gating the memory type and cycle type into a $\overline{\text{CycleEnd}}$ output which deterministically resets the counter.

The R3051's integration of an instruction cache, a data cache, read buffers, and write buffers allows simple main memory interfacing which can be implemented using a small amount of external logic. Thus the R3051 reduces the cost and board size of RISC processing, while maintaining very high throughput.

Integrated Device Technology, Inc.

USING THE IDT79R3051™ WITH THE HP16500 LOGIC ANALYZER

APPLICATION NOTE AN-93

By Andrew Ng

INTRODUCTION

The IDT79R3051™ RISController™ is a highly-integrated, high-performance MIPS® R3000™ instruction set compatible CPU that minimizes system cost and power consumption across a wide variety of embedded applications. The R3051 includes 4kB–8kB of instruction cache, 2kB of data cache, 4-deep read and write buffers, on-chip DMA arbitration, a simple external bus interface, as well as the core R3000A execution engine—all in a single chip 84-pin package. However, in today's marketplace, the technical features of a microprocessor are not enough to guarantee a successful product. A new CPU such as the R3051 must also have a large base of software applications, and very importantly, adequate hardware and software development and debug tools. The R3051 family already has a large base of software applications and a large set of development tools because of its R3000A instruction set compatibility and also because of its widespread market acceptance. The use of just one of these tools, the IDT7RS364 Disassembler for the HP16500 Logic Analyzer will be explained here.

THE IDT7RS364 DISASSEMBLER AND THE HP16500 LOGIC ANALYZER

The IDT7RS364 Disassembler for the HP16500 Logic Analyzer is a useful tool meant to ease the task of debugging software run on R3000-based Target System Boards. Logic analyzers are inexpensive, general purpose debug tools which do not have the power of in-circuit emulators to actively control and simulate target system CPU and memory behavior. However, logic analyzers do provide a useful subset of in-circuit emulator debug capabilities by allowing an engineer to observe and analyze the digital circuit behavior of the target system.

The IDT7RS364 Disassembler consists of a software package that when loaded into the HP16500, pre-processes and formats the state trace listings of the Logic Analyzer. As shown in Figure 1, the HP16500 allows the engineer to capture the CPU's executed hex/binary machine opcodes in a typical Logic Analyzer State Trace Listing format. The user can set multilevel trace traps to capture the area of interest. As shown in Figure 2, with the addition of the IDT7RS364 Disassembler, the hex machine opcodes are automatically decoded and displayed in R3000 assembly code level mnemonic format. Thus the readability and usefulness of the state trace list display screen of the Logic Analyzer are greatly improved.

State/Timing E | Listing 1 | Invasm | Print | Run

Markers Off

Label> / Base>	ADDR Hex	DATA Hex	STAT Hex	Time Absolute
-6	1FC00000	0BF00088	0010	0 s
-5	1FC00004	00000000	0010	760 ns
-4	1FC00220	3C020010	0010	1.52 us
-3	1FC00224	40826000	0010	2.24 us
-2	1FC00228	40806800	0010	3.00 us
-1	1FC0022C	3C02A000	0010	3.76 us
0	1FC00230	3C08AAAA	0010	4.52 us
1	1FC00234	35085555	0010	5.24 us
2	1FC00238	AC480000	0010	6.00 us
3	1FC0023C	AC400004	0010	6.76 us
4	00000000	AAAA5555	0000	7.40 us
5	1FC00240	8C490000	0010	7.88 us
6	00000004	00000000	0000	8.52 us
7	1FC00244	00000000	0010	9.00 us
8	00000000	AAAA5555	0010	9.64 us
9	1FC00248	11280003	0010	10.32 us

2883 drw 01

Figure 1. R3051 Address/Data Trace List on a Logic Analyzer

State/Timing E | Listing 1 | Invasm | Print | Run

Markers Off

Label> / Base>	ADDR Hex	R3000 Mnemonic hex	STAT Hex	Time Absolute
-6	1FC00000	J 0x1FC00220	0010	0 s
-5	1FC00004	NOP	0010	760 ns
-4	1FC00220	LUI v0,0x0010	0010	1.52 us
-3	1FC00224	MTC0 v0,$12	0010	2.24 us
-2	1FC00228	MTC0 zero,$13	0010	3.00 us
-1	1FC0022C	LUI v0,0xA000	0010	3.76 us
0	1FC00230	LUI t0,0xAAAA	0010	4.52 us
1	1FC00234	ORI t0,t0,0x5555	0010	5.24 us
2	1FC00238	SW t0,0x0000(v0)	0010	6.00 us
3	1FC0023C	SW zero,0x0004(v0)	0010	6.76 us
4	00000000	STORE DATA 0xAAAA5555	0000	7.40 us
5	1FC00240	LW t1,0x0000(v0)	0010	7.88 us
6	00000004	STORE DATA 0x00000000	0000	8.52 us
7	1FC00244	NOP	0010	9.00 us
8	00000000	LOAD DATA 0xAAAA5555	0010	9.64 us
9	1FC00248	B 0x1FC00258	0010	10.32 us

2883 drw 02

Figure 2. R3051 Instruction Disassembly on the HP16500 Logic Analyzer

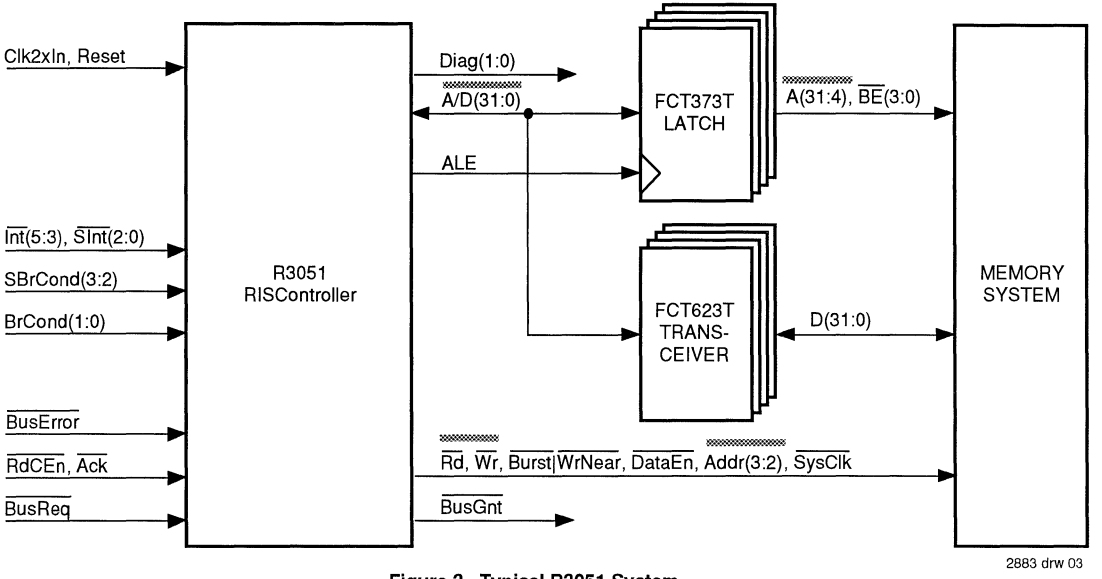


Figure 3. Typical R3051 System

Connecting the R3051 to the HP16500 Pod Sets

Before the Disassembler can be used, the correct connections between the R3051 and the HP16500 must be made. The Disassembler requires five 16-channel probe pod sets. The Disassembler expects that the Pod Probe connections follow its interface protocol so that the pre-processing can correctly interpret the address, data, and status lines. The Disassembler typically uses 32 Address lines, 32 Data lines, a Read line, and a Write line.

In the typical R3051 system as shown in Figure 3, the R3051's $\overline{Rd}$ output is used as the read line and the R3051's $\overline{Wr}$ output is used as the write line. The Disassembler uses the read and write signals as clocks to strobe the address and data into the Logic Analyzer. Since the top speed of the State

traces on the HP16500 is 35MHz and the fastest possible memory cycle is 2 clocks, the Disassembler can easily support 40MHz R3051 CPUs and has a theoretical limitation of 70MHz.

The Address lines can be gathered from the Address Latch outputs and Addr(3:2). Not all 32 address lines need to be attached, as the user can format the address line's MSB channel probes to not show up in the state trace listing if desired. In such a case, the user can use the extra channel probes for other purposes.

In general, Data lines can be gathered from the A/D bus. Some systems, with only one set of Data Transceivers, can gather the data from the memory side of the Data Transceivers in order to reduce A/D bus loading. The R3051 connections to the five HP16500 Channel Probe Pod sets are listed in Table 1.

The Disassembler has three status lines, Write, AccTyp(2) and AccTyp(0). The R3051's $\overline{Wr}$ output can be used as the write line so that the Disassembler can distinguish between a read and a write cycle. AccTyp(2) and AccTyp(0) are optional connections for cached code and in general should be grounded or at least left unconnected. The optional use of AccTyp(2) and AccTyp(0) will be explained in more detail in the Cached Code/Data section. The 16-channel status pod has 13 unused channels that can be used to display other signals, e.g., the Byte Enables.

To a limited extent, the default ordering of the channel probe connections can be changed by the user. The relative ordering of the bits must still occur from MSB to LSB for the address/data/status bus labels such that the Pod Number and Channel Numbers go from MSB to LSB. An example of reformatting the Pod interface is shown in Table 2 and Figure 4. The example in Table 2 and Figure 4 also demonstrates the use of the HP16500's demultiplexed clock feature. When using the demultiplexed clock, the address and data lines can use the same probes. This allows both the address and data to be taken from the multiplexed A/D(31:0) bus. The address is slave-clocked with ALE and the data is master-clocked with $\overline{Wr}$ or $\overline{Rd}$. When using two clocks, only the 8 LSB probes on each pod can be used since the channels are internally multiplexed by the HP16500. Demultiplexed clocking is limited to 50ns master to slave clock recovery, which limits its use to 25MHz CPU systems.

The HP16500 allows an extensive number of multi-level traps and triggers so that the code trace for the area of interest can be found. Care should be taken when setting up trigger conditions. Sometimes when in the trace/trigger menu, the

Table 1. R3051 Default Pod Connections on the HP16500 Logic Analyzer

POD chan	5 sig	POD chan	4 sig	POD chan	3 sig	POD chan	2 sig	POD chan	1 sig
15	X	15	A/D(31)	15	A/D(15)	15	A(31)	15	A(15)
14	X	14	A/D(30)	14	A/D(14)	14	A(30)	14	A(14)
13	X	13	A/D(29)	13	A/D(13)	13	A(29)	13	A(13)
12	Gnd	12	A/D(28)	12	A/D(12)	12	A(28)	12	A(12)
11	X	11	A/D(27)	11	A/D(11)	11	A(27)	11	A(11)
10	Note 2	10	A/D(26)	10	A/D(10)	10	A(26)	10	A(10)
9	X	9	A/D(25)	9	A/D(9)	9	A(25)	9	A(9)
8	X	8	A/D(24)	8	A/D(8)	8	A(24)	8	A(8)
7	X	7	A/D(23)	7	A/D(7)	7	A(23)	7	A(7)
6	X	6	A/D(22)	6	A/D(6)	6	A(22)	6	A(6)
5	X	5	A/D(21)	5	A/D(5)	5	A(21)	5	A(5)
4	$\overline{Wr}$	4	A/D(20)	4	A/D(4)	4	A(20)	4	A(4)
3	X	3	A/D(19)	3	A/D(3)	3	A(19)	3	Addr(3)
2	X	2	A/D(18)	2	A/D(2)	2	A(18)	2	Addr(2)
1	X	1	A/D(17)	1	A/D(1)	1	A(17)	1	Gnd
0	X	0	A/D(16)	0	A/D(0)	0	A(16)	0	Gnd
NClk		MClk	$\overline{Rd}$	LClk		KClk		JClk	$\overline{Wr}$

2883 tbl 01

NOTES:

1. Master Clock Format: J↑ + M↑
2. POD5(12) is AccTyp(2) and POD5(10) is AccTyp(0). If AccTyp(2) is grounded then AccTyp(0) is not used by the Disassembler and can be used for other purposes. See text for further explanation.
3. A(31:4) are connected to the Address Latch outputs. The rest of the signals are connected to R3051 outputs. X's denote unused probes that can be assigned by the user.

Disassembler format in the data field trigger condition can conceal a trap condition. Changing the Disassembler format temporarily to hex format while in the trigger menu can prevent such confusion.

When Running with Cached Code/Data

All Logic Analyzers and Disassemblers can only capture external CPU memory accesses. Since the R3051 is capable of running code and accessing data in its internal caches, such accesses are not seen by the external memory system. Thus in order for the Disassembler to accurately reflect the complete instruction/data flow, the R3051 must be run uncached.

As the target system becomes more and more functional, it becomes necessary to begin running cached code and data. Running cached code/data will affect the Disassembler's accuracy in the following ways:

Cached Instructions

1. Instruction fetch i-cache hits are not seen.
2. Only the last word of a cachable 4-word burst instruction i-cache miss will be seen.

Table 2. Example of Reformatted Pod Connections

POD 5 chan	5 sig	POD 4 chan	4 sig	POD 3 chan	3 sig	POD 2 chan	2 sig	POD 1 chan	1 sig
15		15		15		15		15	X
14		14		14		14		14	X
13		13		13		13		13	X
12		12		12		12		12	Gnd
11		11		11		11		11	X
10		10		10		10		10	Note 3
9		9		9		9		9	X
8		8		8		8		8	X
7	A/D(31)	7	A/D(23)	7	A/D(15)	7	A/D(7)	7	X
6	A/D(30)	6	A/D(22)	6	A/D(14)	6	A/D(6)	6	X
5	A/D(29)	5	A/D(21)	5	A/D(13)	5	A/D(5)	5	X
4	A/D(28)	4	A/D(20)	4	A/D(12)	4	A/D(4)	4	Wr
3	A/D(27)	3	A/D(19)	3	A/D(11)	3	A/D(3)	3	Addr(3)
2	A/D(26)	2	A/D(18)	2	A/D(10)	2	A/D(2)	2	Addr(2)
1	A/D(25)	1	A/D(17)	1	A/D(9)	1	A/D(1)	1	Gnd
0	A/D(24)	0	A/D(16)	0	A/D(8)	0	A/D(0)	0	Gnd
NClk		MClk	$\overline{Rd}$	LClk		KClk	ALE	JClk	$\overline{Wr}$

2883 tbl 02

NOTES:

1. Master Clock Format: J↑+M↑
2. Slave Clock Format: KØ
3. POD5(12) is AccTyp(2) and POD5(10) is AccTyp(0). If AccTyp(2) is grounded then AccTyp(0) is not used by the Disassembler and can be used for other purposes. See text for further explanation.
4. On Master/Slave Pods, only the 8 LSB probes are actually connected. E.g., A/D(23:16) is connected to Pod4(7:0).
5. X's denote unused probes that can be assigned by the user.

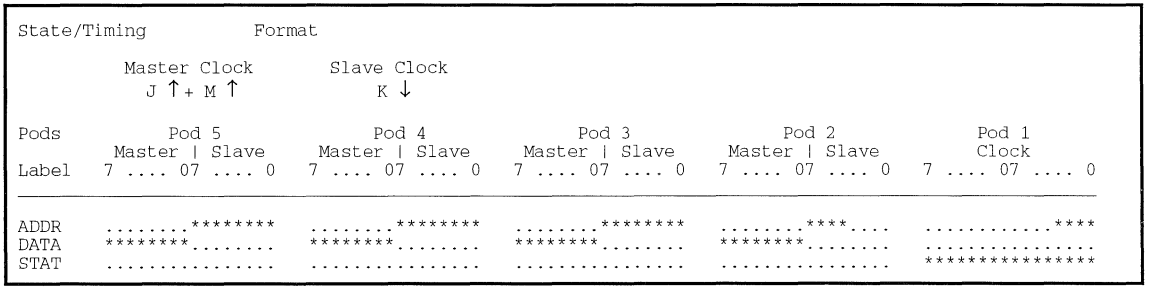


Figure 4. Example of Reformatted Pod Format

2883 drw 04

Cached Data Loads
1. Data load d-cache hits are not seen.
2. Only the last word of a cachable 4-word data block refill d-cache miss will be seen.
3. If the load instruction was an i-cache hit (not seen) then the associated data fetch if seen will be listed as an instruction. The data fetch is assumed to be the second (due to pipelining) read cycle after the load instruction.

Cached Data Stores
1. Data stores are handled correctly, since the R3051 maintains a write-through cache policy which ALWAYS updates main memory as well as the d-cache.
2. Because the R3051 has a 4-word deep write buffer, a data store may or may not occur on the second (due to pipelining) memory cycle following its instruction fetch. Multiple stores are always handled in the proper FIFO order, but each store may be interspersed with later instruction fetches.

Other than running the software uncached, the following less intrusive methods may be used to help interpret cached code/data:
1. Use the R3051's testability mode to invoke the Force I-Cache Miss Mode. This will put all instruction fetches onto the external main memory interface so that the logic analyzer can see all of them. However, forced i-cache misses may or may not be 4-word burst reads.

   In general, 4-word burst reads can be displayed properly if a more complex read strobe is formatted:
   J clock: $\overline{Ack}$ == LOW
   M clock: $\overline{RdCEn}$ == LOW
   N clock: $\overline{SysClk}$ == positive edge-triggered

   The HP16500 ORs level conditions together, OR's edge conditions together and ANDs level conditions with edge conditions. Thus the above strobe clocks the state when:

   ($\overline{SysClk}$ == ≠) AND [($\overline{Ack}$ == 0) OR ($\overline{RdCEn}$ == 0)]

   This example clock set-up is only applicable to systems that happen to bring Ack LOW at the same time RdCEn is LOW on 4-word burst reads or don't bring Ack LOW on 4-word burst reads. Also 1/2 clock margin on the memory read access time is necessary in this example. Thus depending on the particular system design, variants of RdCEn, Ack, and SysClk can be combined or temporarily modified to create a 4-word read strobe and a write strobe.
2. Latch the R3051's Diag(1:0) outputs with ALE. On external main memory reads, if LatchedDiag(1) == 1 then the fetch is cachable and can be used as an indication that the state trace entry should be interpreted judiciously. When LatchedDiag(1) == 1, LatchedDiag(0) == 1 indicates a cachable instruction fetch and LatchedDiag(0) == 0 indicates a cachable data load.

   LatchedDiag(1:0) are the R3051's equivalents of the R3000's AccTyp(2) and AccTyp(0). As such they can be connected to the Disassembler's AccTyp(2) and AccTyp(0) probes. This allows the Disassembler to differentiate between cached instructions and data so that they can be displayed properly. However, AccTyp(2) and Diag(1) are undefined for writes, e.g., when the write buffer is full or on partial word stores. So if the AccTyp(2) probe is used, in order for the Disassembler to interpret write cycles correctly, LatchedDiag(1) needs to be AND'ed with $\overline{Wr}$ as shown in Figure 5, so that it is always LOW during write cycles.
3. Use the Reset Mode Vector to set the R3051 to use single word data refills instead of 4-word data block refills. This will allow all 4 words on a data load d-cache misses to be seen.

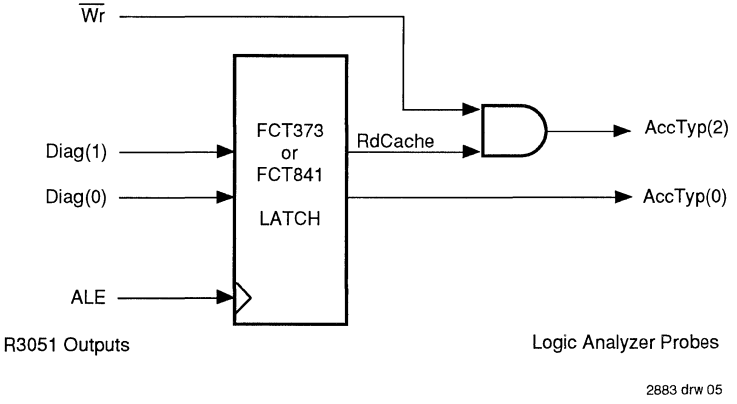


Figure 5. Using Diag(1:0) with the Disassembler

Using State Trace Listings and Timing Waveforms

The IDT7RS364 Disassembler is a good tool for easing the use of a Logic Analyzer when debugging a target system. However, sometimes, even lower level detail is needed to examine clock by clock behavior of particular bus cycles. The HP16500 performs this function in its State Analyzer mode by sampling with the CPU's system clock as shown in Figure 6. Because the state analyzer mode has a maximum speed of 35MHz, certain restrictions apply. Ideally because the R3051 uses both edges of its $\overline{\text{SysClk}}$ output to generate control lines, it is preferable to use Clk2xIn or to clock on both edges of either $\overline{\text{SysClk}}$ or its buffered/inverted version SysClk. On the HP16500, high-speed clocks should always use their ground shield on the probe to reference the input properly so that the probe does not sense signal overdrive. The edge of the reference clock should be chosen carefully so that it ideally clocks just before ALE de-asserts as shown in Figure 7. This allows the address to be seen along with the data on the multiplexed A/D bus so that dedicated address lines probes are not required. When choosing a clock, keep in mind that the HP16500 has 10ns set-up time and 1ns hold time relative to the clock. In addition, the HP16500's Time Tagging feature if used is limited to 16.67MHz.

State/Timing E | Listing 1 | Invasm | Print | Run

Markers Off

Label>	DATA	ADDR	CLKN	BAWRRA	ALE	WRNRDN
Base>	Hex	Hex	Hex	Binary	Binary	Binary
274	8C490000	4	1	111110	0	11
275	8C490000	0	0	111110	0	11
276	00000000	4	1	110111	1	01
277	00000000	4	0	110110	0	01
278	00000000	4	1	110110	0	01
279	00000000	4	0	110110	0	01
280	00000000	4	1	110110	0	01
281	00000000	4	0	110110	0	01
282	00000000	4	1	110110	0	01
283	00000000	4	0	110110	0	01
284	00000000	4	1	110110	0	01
285	00000000	4	0	100110	0	01
286	00000000	4	1	100110	0	01
287	00000000	4	0	111110	0	11
288	1FC00240	4	1	111101	1	10
289	1FC00240	4	0	111100	0	10

2883 drw 06

Figure 6. R3051 State Trace Listing Using Clk2xIn

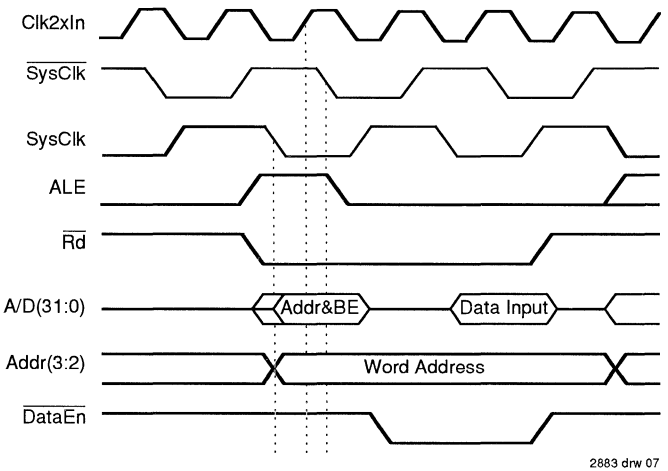


Figure 7. Choosing a Clock Edge

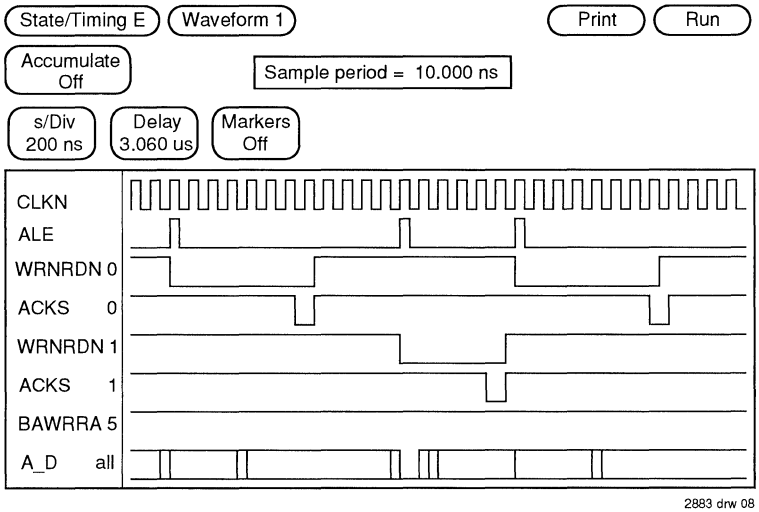


Figure 8. R3051 Timing Mode Waveform

Systems running with a Clk2xIn over 35MHz (17.5MHz CPU) can either clock the State Analyzer mode less frequently or use the Timing Analyzer mode. When clocking less frequently, care must be taken to chose a clock edge that adequately strobes ALE during its HIGH period so that the address can be determined. Because the R3051 only has a 1/2 clock intercycle memory latency, $\overline{Rd}$ and $\overline{Wr}$ and other control lines may not be seen to de-assert between memory cycles when clocked at the $\overline{SysClk}$ frequency.

The HP16500 Logic Analyzer's Timing mode displays signals in waveform format as shown in Figure 8 and is capable of internally generating a 100MHz (10ns) sample clock. To maintain all the functional timing relationships relative to the Clk2xIn, the timing mode allows asynchronous sampling up to 50MHz CPU speed. The disadvantage of using the Timing mode is that the value of busses is hard to decipher when shown in waveform format. If necessary, HP16500 can be set up in its mixed mode display to display both state and timing modes on the same screen.

SUMMARY

The use of the HP16500 and the IDT7RS364 Disassembler is but one example of the availability and compatibility of R3000 tools and software that can be used on the R3051. The Disassembler formats logic analyzer state traces into assembly level mnemonics to allow easier user interpretation. Similarly, other R3000 software, compilers, as well as other development tools such as the IDT7RS901 IDT/sim™ ROMable Kernel/Boot Monitor can also be used on R3051 systems with little or no modification.

Integrated Device Technology, Inc.

INTERFACING THE IDT79R3051™ TO THE SONIC™

APPLICATION NOTE AN-95

By Danh Le Ngoc (Integrated Device Technology, Inc.) and Paul Cheng and Bill Harmon (National Semiconductor)

OVERVIEW

The IDTR3051™ family is a series of high-performance 32-bit microprocessors featuring a high-level integration and high-performance. The R3051 family integrates the MIPS® R3000A™ RISC CPU, along with 8kB of instruction cache and 2kB of data cache. The R3051 family uses a simple time-multiplexed 32-bit address and data bus to provide a low cost system interface (and to minimize the cost of ASIC devices designed to interface with the processor). In order to minimize the impact of a time-multiplexed bus, the R3051 family incorporates a 4-deep read buffer and 4-deep write buffer into the interface, allowing relatively slow memory systems to be mated to a high-speed processor. The R3051 family is able to offer 35mips of integer performance at 40MHz without requiring external SRAM or caches.

The R3051 family is designed to bring the high-performance inherent in the MIPS RISC architecture into low cost simplified embedded applications such as laser printers, X-Window terminals and network bridges and routers. Figure 1 illustrates the simplified block diagram of the R3051-based X-Window terminal.

The focus of this application note to describe the interface between the R3051 and National Semiconductor's System Oriented Network Interface Controller (SONIC).

The SONIC™ is National Semiconductor's System Oriented Network Interface Controller (DP83932). This Ethernet

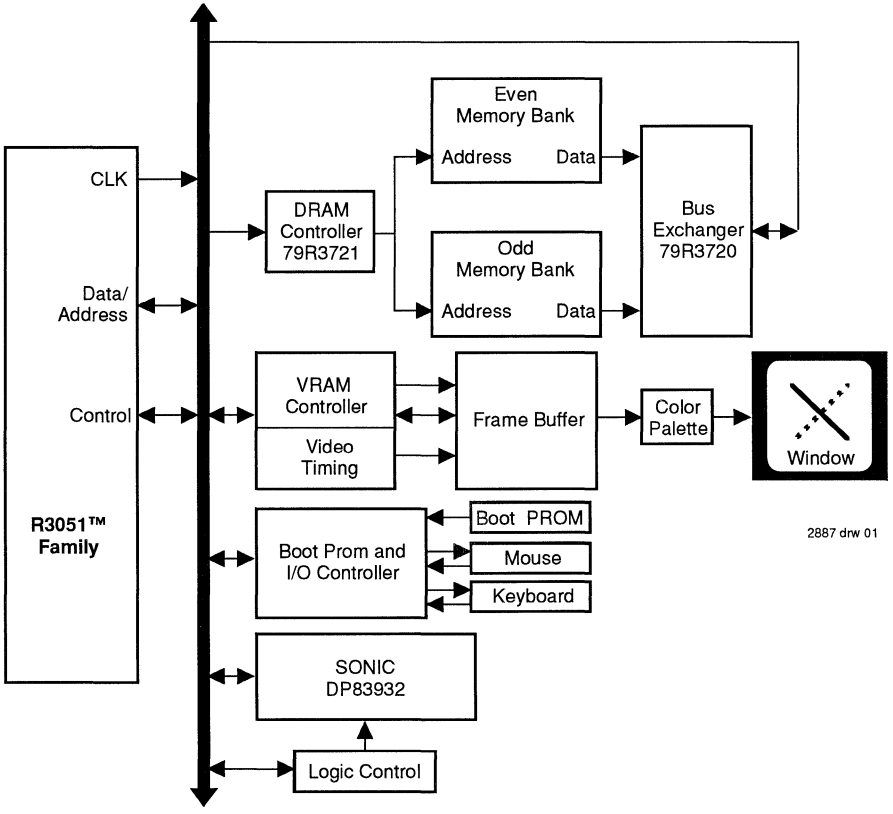


Figure 1. X-Window Terminal

controller is intended to provide a high performance 32 or 16-bit Ethernet connection for systems that require efficient, high-throughput, low-power network connectivity. The SONIC can be employed in an R3051-based system, in order to tightly couple the system's CPU and main memory to the network. Figure 2 depicts this interface.

The SONIC is ideally suited to embedded processing applications such as X-Terminals, due to its unique feature set. The SONIC completely supports all the required specifications set forth in the IEEE 802.3 standard, including the Media Access Control (MAC) requirements contained in the IEEE 802.3 layer management specification. Additionally, SONIC's high performance DMA channels allow it to use a very small percentage of the bus bandwidth, while its efficient linked list buffer management scheme limits the number of descriptor and data fetches required. It is also important to note that the SONIC utilizes internal content addressable memory (CAM) to provide a 100% perfect address filter for both multicast and physical address packets. This alleviates the need to waste bus bandwidth, memory space, and CPU time on unwanted packets. Finally, the SONIC contains an integrated Manchester encoder/decoder, which is required in all Ethernet applications. This provides a savings in board space, as well as improved reliability.

FUNCTIONAL OVERVIEW

System Interface

The R3051 has a multiplexed 32-bit address and data bus. Since the SONIC's address and data buses are demultiplexed, it is necessary to employ a set of external latches to connect the SONIC to the processor's address and data buses. In many applications, these latches may also be used to demultiplex the R3051 bus to other parts of the system memory and I/O.

In order to allow the R3051 to have access to the SONIC's internal registers, as well as allow the SONIC to gain control of the system bus and perform DMA operations, the SONIC is interfaced to the system bus as both a slave and a master. As a slave, the SONIC appears as a block of 256 bytes, consist-

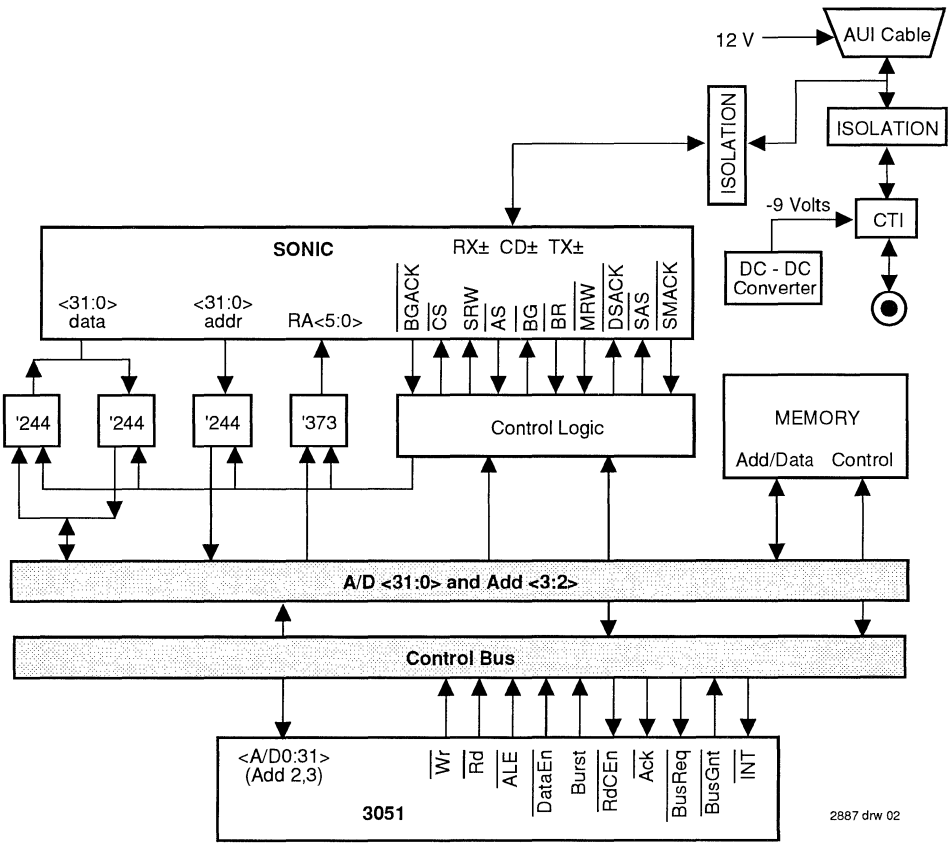


Figure 2. SONIC Interface to the R3051

ing of sixty-four 3- bit words. The SONIC can be mapped into any location of memory and will typically provide for a 7-cycle register access. In R3051 applications, the SONIC will typically be mapped into the processor kseg1, which is an unmapped, uncached address space typically used for processor I/O resources.

As a master, the SONIC will arbitrate with the R3051 for ownership of the bus and proceed to operate as a 32-bit DMA engine between the network and the system memory. While operating on the bus, the SONIC is capable of performing 32-bit/3 cycle DMA operations. It is important to note that the ability to place the SONIC on the same bus as the R3051 and the system memory is critical: this eliminates the need for the Ethernet controller to have a local buffer, which the CPU must spend time and bandwidth to transfer to main memory. The ability of the SONIC to place data directly in main memory and communicate with the CPU through linked list descriptors, as well as register accesses, makes the SONIC/R3051 interface CPU and bandwidth efficient.

Network Interface

With respect to the physical layer design, both AUI drop cable Ethernet and thin wire Ethernet are supported. The block diagram in Figure 2 contains a 15-pin AUI drop cable connector for standard drop cable Ethernet implementations, as well as a thin-wire Ethernet connection via the National Semiconductor coaxial transceiver interface (CTI, DP8392). Either of these network connections can be chosen through the use of a single jumper between the 5V supply and the 5V to –9V DC-to-DC converter. In either case, the AUI signals (RX±, TX±, and CD±) are sent back to the SONIC. These signals are interfaced to the ENDEC portion of the SONIC, which provides for communication between the AUI interface and the non-return to zero (NRZ) signals (RXD,TXD, and COL) of the Media Access Control (MAC) module of the SONIC. It should be noted that the integrated ENDEC module of the SONIC alleviates the need for an external Ethernet Manchester encoder/decoder, such as National's CMOS Serial Network Interface (CMOS SNI, DP83910).

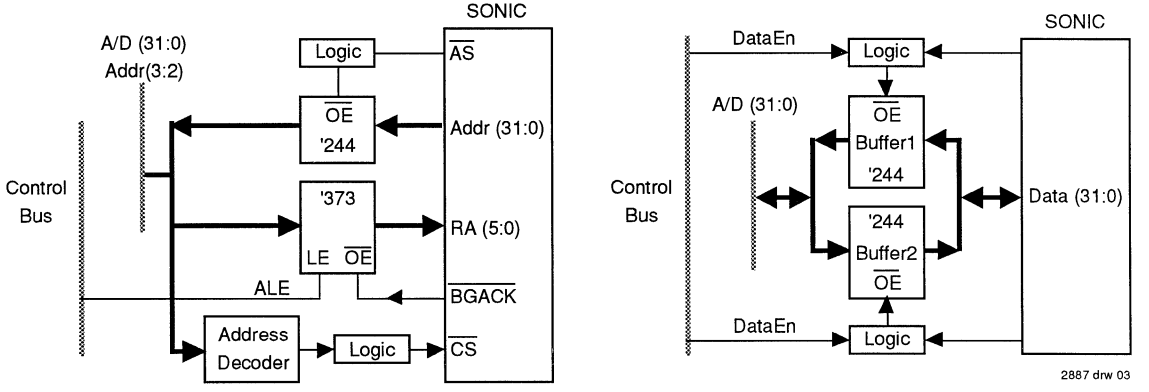
ARCHITECTURE AND DESIGN

Bus Interface

The SONIC's bus interface can be externally configured to operate in one of two modes. If the SONIC's BMODE pin is tied to ground, the SONIC will operate on the bus exactly like an 80386 microprocessor. If the SONIC's BMODE pin is tied to 5V, the SONIC will operate on the bus exactly like a 68030 microprocessor. In this design, the most appropriate mode of operation was achieved by connecting BMODE to 5V.

The bus interface, as depicted in Figure 3, consists of two parts. There is an address bus interface and a data bus interface. Since the R3051's address and data buses are multiplexed, it is necessary to utilize a set of '244 buffers and '373 latches to multiplex the SONIC busses onto the CPU bus. The '244 buffers are required to tri-state the SONIC's address lines from the system bus during the data portion of master transfers, while the '373 is required to latch the register addresses being sent to the SONIC during slave operations. The output enable signal of the '244 is asserted when the SONIC is the master of the bus and both the SONIC's address strobe ($\overline{AS}$) is asserted and the master logic's address latch enable (ALE) signal is asserted. The '373 should latch the address when the R3051 is the bus master and it asserts its ALE signal.

The data bus interface requires the use of two sets of '244 buffers. The first set of buffers (Buffer 1) prevent the SONIC from placing data onto the system's multiplexed address and data bus prematurely. In the slave mode of operation, the output buffer is enabled once the address output drivers are tri-stated. This is signaled by the assertion of the DataEn signal. In the case of a master operation, the buffers are enabled once the address buffers external to the SONIC are



ADDRESS BUS INTERFACE DATA BUS INTERFACE

Figure 3. Address and Data Bus Interface

tri-stated, which takes place upon the de-assertion of the ALE signal.

The second set of buffers is enabled when the SONIC's registers are being written by the R3051 and data is being presented on the multiplexed system address/data bus, or when the SONIC is reading system memory and the memory is placing data on the multiplexed address/data bus. The assertion of the DataEn signal by the system signals that data is now able to be placed on the bus. The actual logic representation for the bus interface can be found in the bus interface logic segment of the Control Logic section of this application note.

Slave Operation

The timing diagram for a slave access of the SONIC is shown in Figure 4. The falling edge of the R3051's ALE signal latches the output of an address decoder and the address lines being passed to the register address lines of the SONIC. If the address decode selects the SONIC, a signal called "AdrDec" will be asserted. The logic for generating this signal is shown in Figure 5. The value of this signal is passed to the chip select ($\overline{CS}$) and slave address strobe ($\overline{SAS}$) signals of the SONIC on the rising edge of the bus clock. The acknowledge signals back to the R3051 ($\overline{ACK}$ for a write and $\overline{RdCEn}$ for a read) are asserted 2 clocks after the SONIC generates its slave acknowledge signal ($\overline{SMACK}$). These signals remain asserted to the R3051 for a clock cycle, after which they are removed. The $\overline{ACK}$ and $\overline{RdCEn}$ signals inform the R3051 that the data has been latched or is valid, respectively. The de-assertion of these signals results in the de-assertion of $\overline{CS}$ and $\overline{SAS}$ to the SONIC. The logic for implementing this part of the design can be found in the slave logic segment of the Control Logic section.

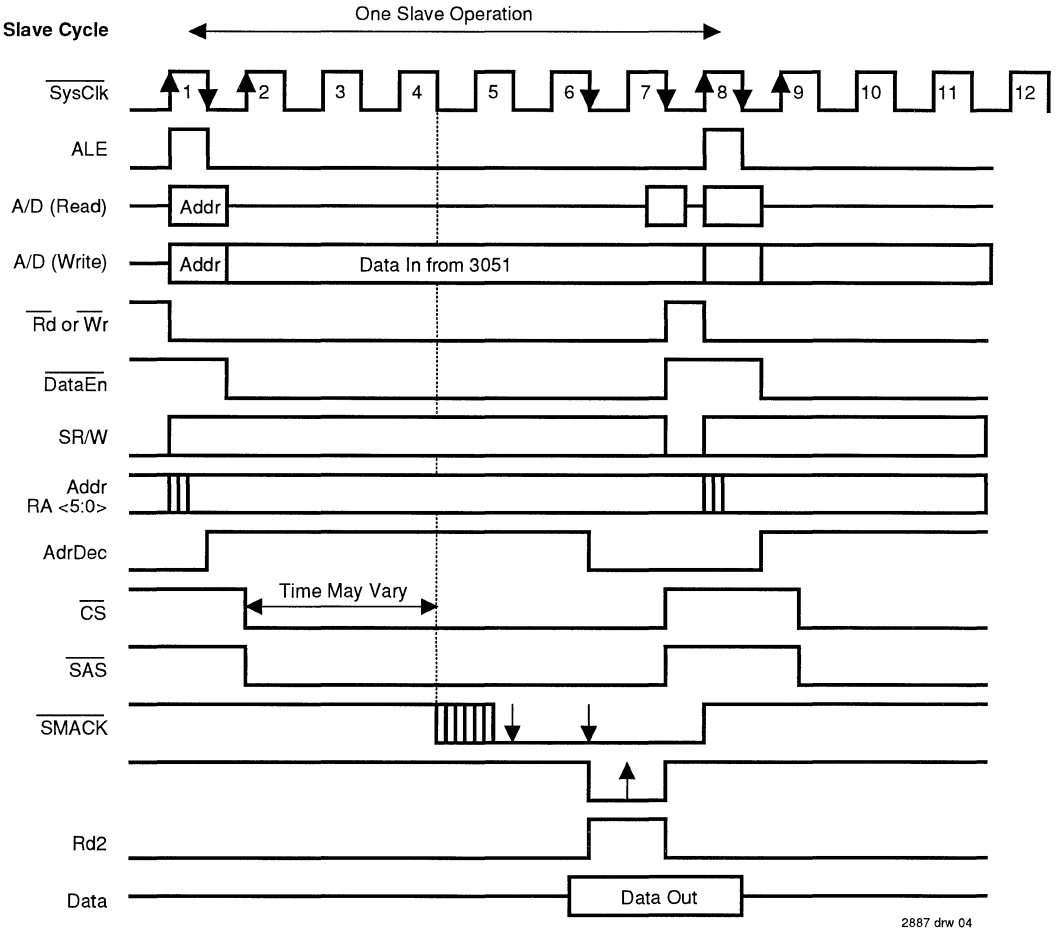


Figure 4. Slave Access Timing Diagram

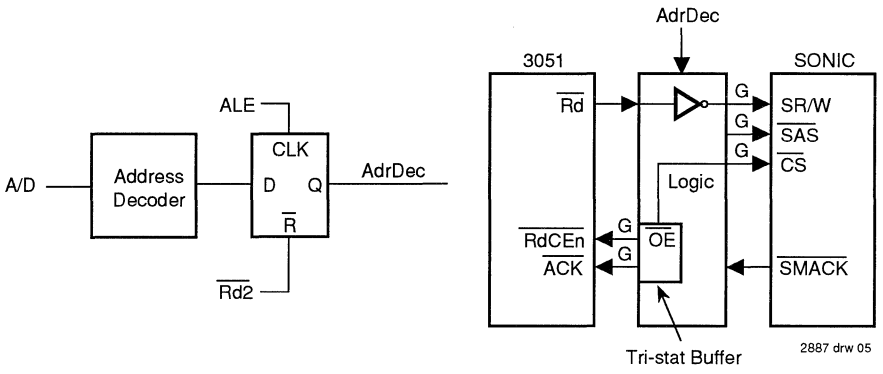


Figure 5. Slave Interface Block Diagram

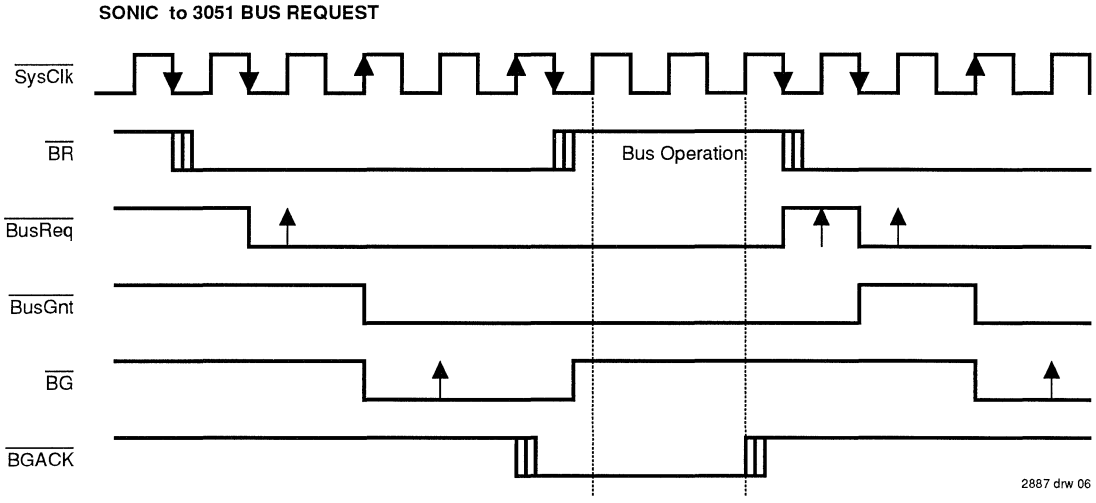


Figure 6. Bus Request Timing Diagram

Master Operation

The first step in designing the master interface is implementing the bus request logic. The timing diagram for this is shown in Figure 6. The bus request ($\overline{BR}$) signal of the SONIC is passed to the R3051's bus request ($\overline{BusReq}$) on the falling edge of the bus clock. The SONIC then waits for the bus grant ($\overline{BusGnt}$) from the R3051, which is passed directly to the SONIC's bus grant ($\overline{BG}$) signal. The assertion of $\overline{BG}$ causes the SONIC to assert bus grant acknowledge ($\overline{BGACK}$) and begin its master DMA operations. It is important to note that the assertion of $\overline{BGACK}$ causes the SONIC to de-assert $\overline{BR}$, which would cause the bus request logic to de-assert $\overline{BG}$ to the SONIC. Thus, the BusReq signal to the R3051 should be the logical "OR" of the SONIC BR and BGAck outputs. A block diagram of the bus request logic appears in Figure 7, while the actual illustration of the logic is found in the bus request logic segment of the Control Logic section.

Once the SONIC has gained control of the bus, it will begin to perform master DMA operations, as illustrated in the Figure 9 timing diagram. Ideally, if the memory is fast enough, the SONIC will be able to perform 3-cycle DMA. At 25MHz, less than 3.75% of the bus' bandwidth will be consumed by the network interface.

There are two very important points to note. First, the R3051's $\overline{ACK}$ signal is basically equivalent to the SONIC's $\overline{DSACK}$ signals, but the SONIC's $\overline{DSACK}$ signals require that the memory system provide a total of 8ns hold time from the rising edge of the clock, while the R3051 requires only 4ns. Second, the ALE signal generated from the SONIC's control signals will be de-asserted 3ns later than the R3051's would be. However, this should not be a significant factor, since the address set-up and hold time provided to the memory system's latches is consistent with the R3051's specification.

When interfacing to the multiplexed bus, it is necessary for the master logic to generate an ALE signal for the system bus. The ALE signal is asserted on the rising edge of the second cycle in the SONIC's memory access. It is necessary to assert the ALE in this cycle, in order to guarantee that the latch will be provided with an adequate amount of set-up time for the address. The ALE signal is then removed on the falling edge

of the same clock cycle. The de-assertion of ALE triggers the assertion of $\overline{\text{DataEn}}$ on a read operation, in order to inform the memory that the bus' address drivers are tri-stated and data can now be driven. The $\overline{\text{DataEn}}$ signal is actually arrived at by delaying the the ALE signal through a buffer or PAL, since the ALE signal is also responsible for disabling the output buffers of the address drivers.

The final piece of interface logic is used to make the SONIC's read and write (MR/W) strobe compatible with the R3051's read ($\overline{\text{Rd}}$) and write ($\overline{\text{Wr}}$) signals. The SONIC's read/write signal is passed to the appropriate read or write strobe of the system bus, on the falling edge of $\overline{\text{AS}}$. The $\overline{\text{Rd}}$ or $\overline{\text{Wr}}$ signal is then de-asserted on the falling edge of the last clock cycle. The block diagram for the master interface is found in Figure 9, while the logical implementation is shown in the master interface logic segment of the Control Logic section.

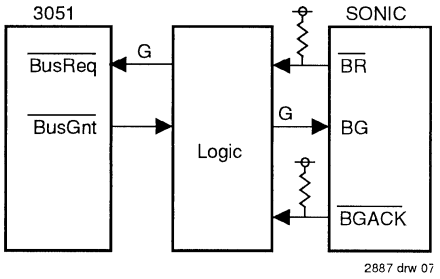


Figure 7. Bus Request Interface Block Diagram

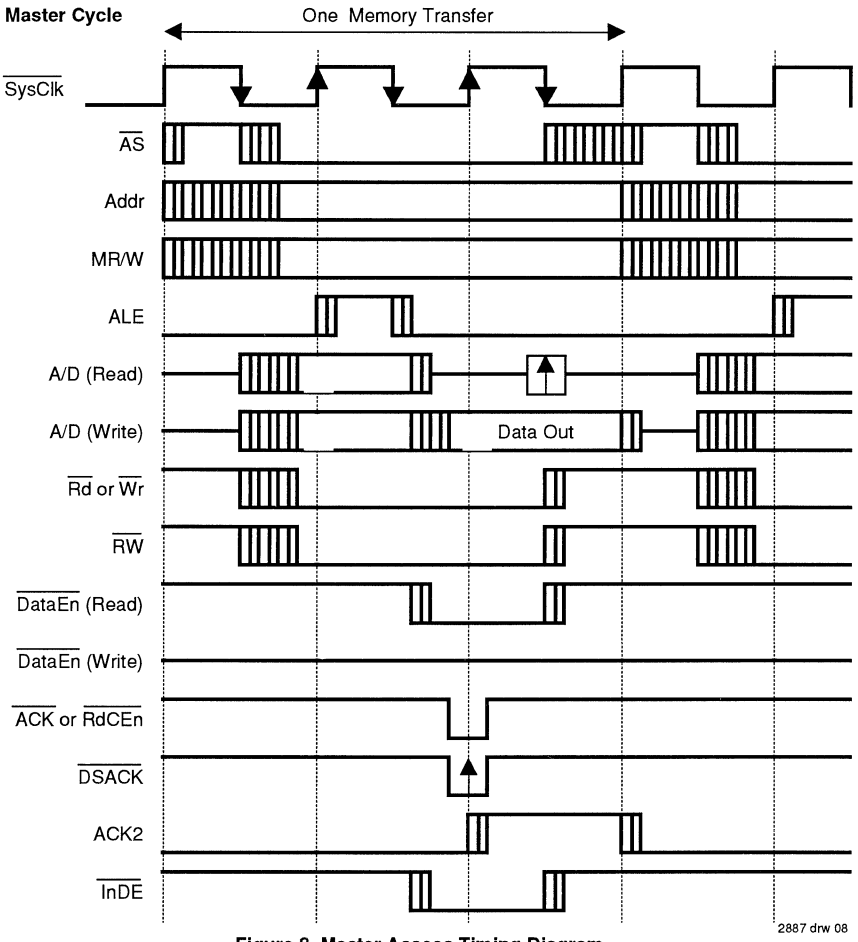


Figure 8. Master Access Timing Diagram

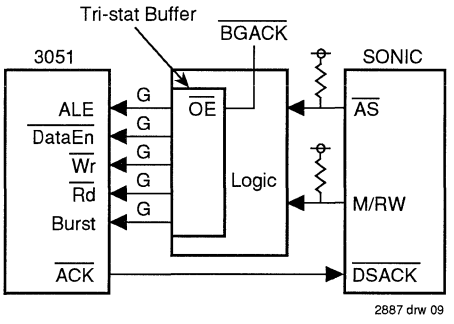


Figure 9. Master Interface Block Diagram

Physical Layer

Figure 10 contains a block diagram of the physical layer interface, while a schematic of the physical layer design is located on the last page of this application note. This design can be used in either a thin wire or standard drop cable Ethernet environment. When the design is used in a thin wire Ethernet application, the 5V supply must be connected to the DC-to-DC converter, so that the necessary –9V output can be supplied to National Semiconductor's Coaxial Transceiver Interface (CTI, DP8392). The CTI provides an interface between the 10MHz Manchester encoded coax cable and the 10MHz Manchester encoded differential signals of the SONIC's ENDEC. In the case of a standard drop cable Ethernet application, the 5V supply is left unconnected, so that the CTI will not receive power. This allows the signals of the SONIC's ENDEC to pass directly to the AUI cable, via the 15-pin AUI connector. In examining the schematic of the physical layer design, it can be seen that there is a pulse transformer at the AUI side of the CTI. This is placed here to isolate the CTI from the SONIC's ENDEC signals, when the AUI drop cable connection is being employed. This transformer also provides the IEEE 802.3 specified isolation between the coax and the differential AUI signals, when thin wire Ethernet is being used. It is also necessary to provide a termination for the 78Ω AUI cable's differential receive and collision pair (RX± and CD ±). This is the reason for the 39Ω –1% resistors and .01µF capacitors that are shown in Figure 10.

Additionally, there are two more significant considerations. First, each one of the transmit pairs (TX+ and TX–) requires a 270Ω non-precision pull-down resistor to complete the internal source follower amplifiers that drive these signals. Second, there is an isolation transformer placed between the differential signals of the SONIC's ENDEC and the AUI cable. This isolation is necessary to guarantee that the SONIC meets the IEEE 802.3 fail-safe specification of a 16V DC level appearing on the AUI cable's differential signals. This external isolation is necessary, because in the powered down state the CMOS process, in which the SONIC is manufactured, may not be able to withstand this voltage.

Control Logic

This application note was developed with the intention of displaying the necessary requirements for interfacing the SONIC to the R3051 system bus. Therefore, the actual implementation of the control logic will be graphically depicted in state machine form, as opposed to being partitioned into actual PAL devices. This leaves the freedom for the designer to incorporate this logic into his/her system in PALs, ASICs, FPGAs, etc.

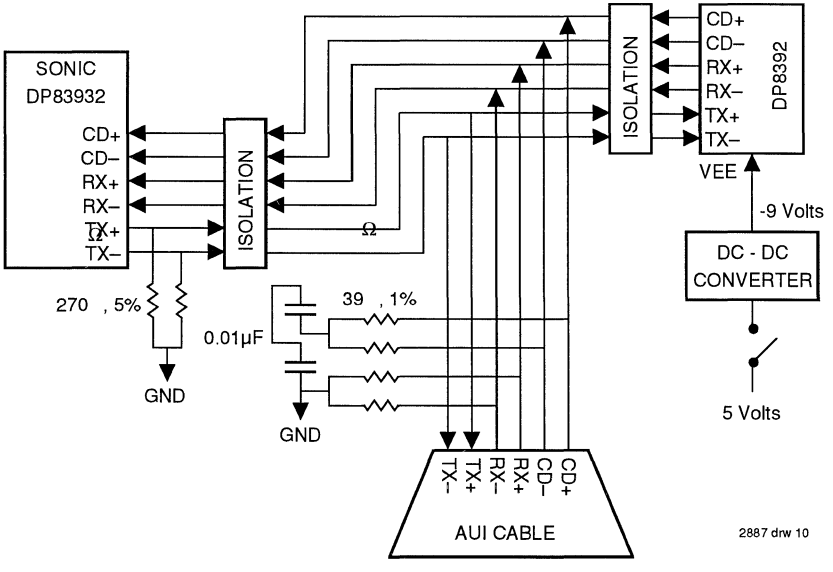
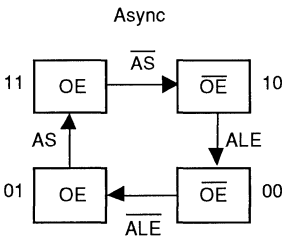


Figure 10. Physical Layer Interface Block Diagram

BUS INTERFACE LOGIC

Address



$Q1^ = Q1 * Q2 + Q1 * \overline{ALE} + Q2 * AS$

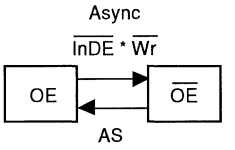
$Q2^ = \overline{Q1} * Q2 + \overline{Q1} * \overline{ALE} + Q2 * AS$

$OE = Q2^$

Data

$$OE = \overline{(DataEn + CS)} + (BGACK + DataEn)$$

1st case 2nd case



$OE1 = DataEn + CS$

$OE2 = Q^ = AS * InDE + AS * \overline{Q} + Wr * Q + InDE * Q$

$$OE = DataEn + CS + AS * InDE + AS * OE2 + Wr * \overline{OE2} + InDE * OE2$$

case 1 case 2

2887 drw 11

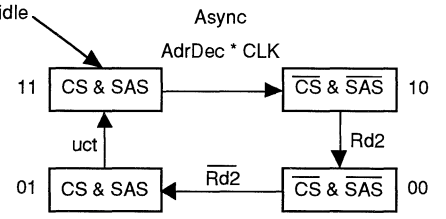
Note:
1. Q1^ refers to the first state machine bit and Q2^ refers to the second state machine bit (10: Q1^=1 & Q2^ = 0)

SLAVE INTERFACE LOGIC

$Q1^ = Q2 + Q1 * \overline{Q2} * \overline{Rd2}$

$Q2^ = Q1 * Q2 * \overline{AdrDec} + \overline{Q1} * \overline{Q2} * \overline{Rd2} + \overline{Q1} * Q2$

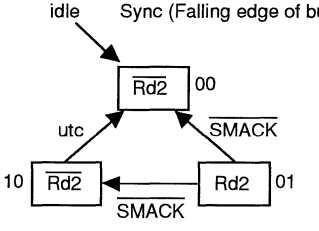
$CS = SAS = Q2^$

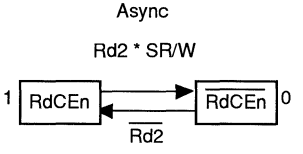


$Q1^ = \overline{SMACK} * Q2$

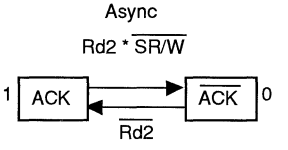
$Q2^ = \overline{Q1} * \overline{Q2} * \overline{SMACK} + Q2 * SMACK$

$Rd2 = Q2^$





$RdCEn = Q^ = \overline{Rd2} + Q * \overline{SR/W}$



$ACK = Q^ = \overline{Rd2} + Q * SR/W$

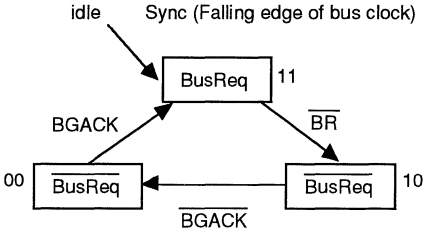
2887 drw 12

BUS REQUEST INTERFACE LOGIC

$Q1^ = BGACK + Q1 * Q2$

$Q2^ = BR * Q2 + BGACK * \overline{Q1}$

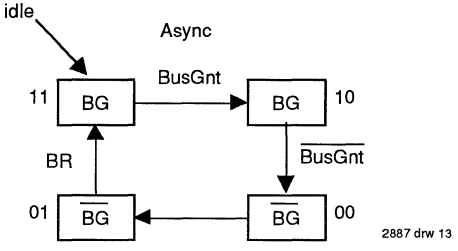
$BusReq = Q1^ * Q2^$



$Q1^ = Q1 * Q2 + Q2 * BR$

$Q2^ = \overline{Q1} + Q2 * \overline{BusGnt}$

$BG = Q1^$



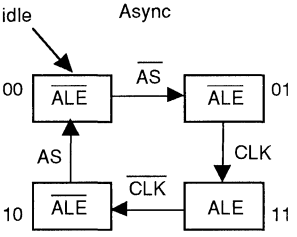
2887 drw 13

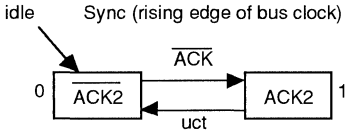
MASTER INTERFACE LOGIC

Q1^ = Q1 * $\overline{AS}$ + Q2 * CLK + Q1* Q2

Q2^ = $\overline{Q1}$* $\overline{AS}$ + $\overline{Q1}$* Q2 + Q2* CLK

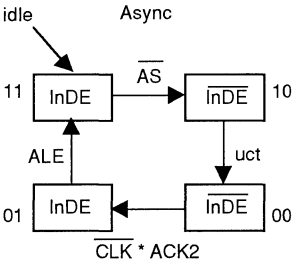
ALE = Q1^ * Q2^

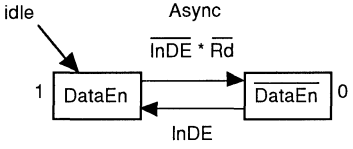




Q^ = $\overline{Q}$ * $\overline{ACK}$

ACK2 = Q^





Q^ = InDE + Rd * Q

DataEn = Q^

Q1^ = Q1 * Q2 + $\overline{Q1}$ * Q2 * ALE

Q2^ = $\overline{Q1}$ * Q2 + $\overline{Q1}$ * $\overline{Q2}$ * $\overline{CLK}$ * ACK2

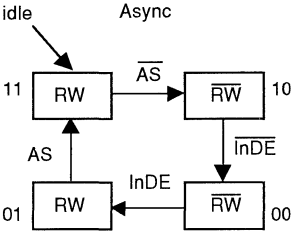
+ Q1 * Q2 * ALE

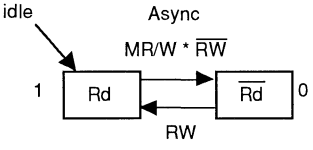
InDE = Q2^

Q1^ = Q1 * Q2 + Q1 * InDE + AS * Q2

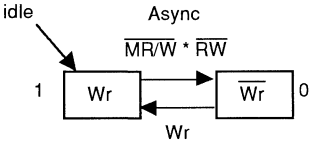
Q2 ^ = InDE * $\overline{Q1}$ + $\overline{Q1}$ * Q2 + Q2 * AS

RW = Q2^





Rd = Q^ = RW + Q * $\overline{MR/W}$



Wr = Q^ = RW + Q * MR/W

2887 drw 14

Integrated Device Technology, Inc.

IDT79R3051™ ADDRESS/DATA BUS TURN-AROUND BEHAVIOR

APPLICATION NOTE AN-97

by Andrew Ng

INTRODUCTION

This application note describes the behavior of the IDTR3051's multiplexed Address/Data, "A/D" bus and presents the issues of a particular topic called "Bus Turn-Around." Bus Turn-Around will be defined, design issues will be presented, and design solutions will be given for conventional R3051 systems, as well as a "DMA $\overline{\text{BusReq}}$" design solution for very low-speed and very high-speed systems.

Definition of the R3051

The IDT79R3051™ RISController™ is a highly integrated MIPS® R3000™ instruction set compatible microprocessor that minimizes system cost and power consumption. The R3051 includes 4kB to 8kB of instruction cache, 2kB of data cache, an optional on-chip TLB memory management unit, 4-deep read and write buffers, on-chip DMA arbitration, a simple external bus interface, as well as the R3000A CPU execution engine — all in a single compact plastic 84-pin package.

Definition of the A/D Bus

One of the key features of the R3051 is its low pin count. The low pin count is largely a result of its simple control interface and its use of a multiplexed Address and Data bus, called A/D(31:0). As shown in Figures 1 and 2, the multiplexed A/D bus drives its address during the first phase of a read or write memory cycle. In the second phase of a read memory cycle, the CPU expects the external memory system to drive the bus and return the data. In the second phase of a write memory cycle, the CPU drives the data out to the memory system. Thus in a typical R3051 system, the address can be latched using a bank of transparent latches such as with the 54/74FCT373T or 54/74FCT841T as shown in Figures 4 and 5 so that the address is de-multiplexed from the data lines.

In systems using an ASIC, such as for a DRAM or DMA Controller or as an Integrated I/O Subsystem/Controller with on-chip programmable registers, the multiplexed A/D bus has an advantage over separate Address and Data busses in that the ASIC requires substantially fewer pins. The ASIC can latch the 32 Address bits internally, using the Address Latch Enable output from the CPU called "ALE", and then use the same input pins to provide data. In addition, the CPU has less noise from simultaneous switching of the 32 A/D lines than if it had to switch 64 separate Address and Data lines. Thus R3051 systems can often save cost and space by using inexpensive and low pin count ASICs.

Although a multiplexed bus may be thought of as a disadvantage in terms of system performance, this is rarely the case in R3051 systems. An analysis of memory behavior and the bus shows that in conventional memory systems (those that do not use exclusively high-speed, single-cycle SRAMs for

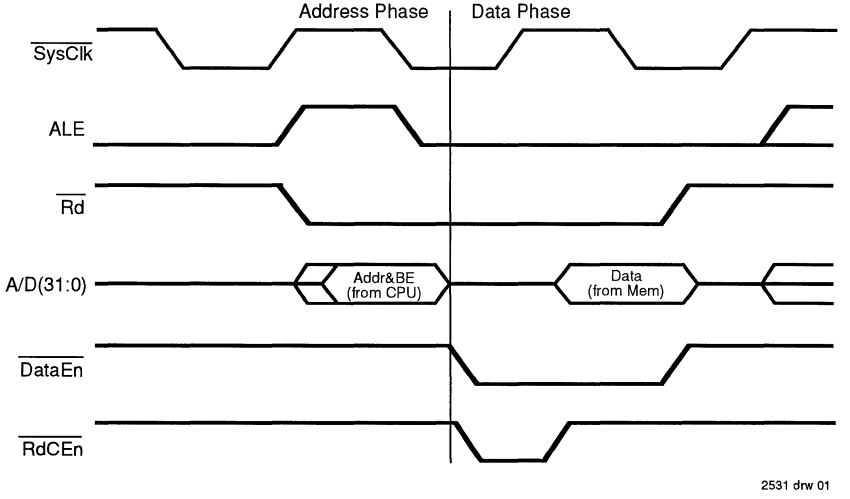


Figure 1. R3051 Read Cycle

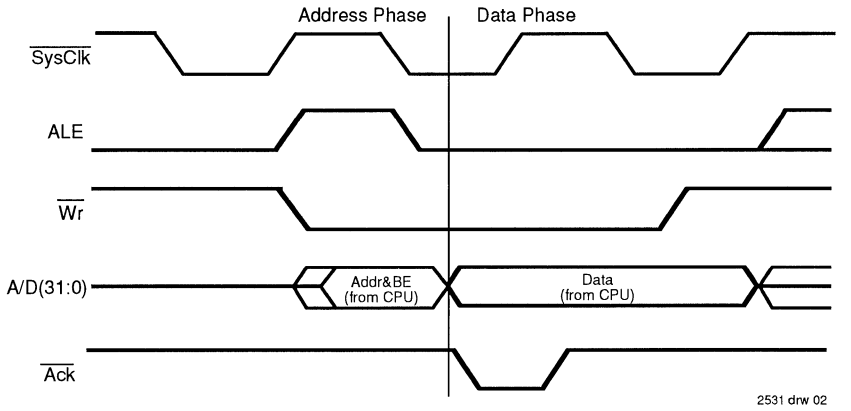


Figure 2. R3051 Write Cycle

the entire memory system), the R3051 bus structure causes no real performance loss.

For example, conventional memory systems use the address before the data is generated on read cycles or needed by write cycles. On read cycles, the address is always needed before the data array can be accessed. The multiplexed R3051 bus provides the address as early as a non-multiplexed bus would; thus, the read access is not delayed. Since memory read performance is described as "Address and Chip-Select valid to Data Available", the multiplexed bus causes no performance loss on reads.

Similarly, on write cycles, most memories (except for self-timed memories) require the address before the data in order to properly coordinate the write strobe with the correct internal row and column address decode/selects. The R3051 bus provides the write target address for one-half cycle, and then immediately presents the write data. That half cycle is required to perform address decoding, and to provide a Chip-Select to the memory device. Thus, once the address and Chip-Select are available to the memory, the data is also available.

Further, the R3051 decouples the system bus performance from processor performance based on the integration of on-chip resources. Specifically, the large on-chip caches minimize the number of main memory reads, thus making system read performance less criticial. The on-chip 4-deep write buffer isolates the processor from the memory system write speed, allowing it to continue execution while store operations are actually updated into the memory. Thus, R3051 performance, while somewhat dependant on memory system performance, is largely isolated from the memory system. Thus, high-performance systems using relatively slow EPROM and DRAM devices can be easily realized.

Definition of Bus Turn-Around

The other consequence of a multiplexed bus arises from the fact that during a particular transaction, as well as from one transaction to the next, transitions between sources of the bus can occur. For example, a read transaction begins with the processor driving the address on the bus, and ends with the memory driving the data on the bus. Similarly, at the end of a read, the next transaction on the bus will begin again with the CPU driving an address on the bus.

Note that similar concerns are present even for non-multiplexed busses. For example, a read followed by a write results in the data bus first being driven by the memory, and then being driven by the CPU. Thus, bus turn-around is also a consideration in non-multiplexed bus systems.

Bus Turn-Around behavior is the action that the CPU takes when its address/data bus transitions between the CPU and the memory, particularly when it changes direction from being a driver to being a non-driver or vice-versa. The actions that the CPU can take are:

1. Drive the address.
2. Drive the data.
3. Tri-state.

There are two basic times when the A/D bus will transition:

1. Intra-Cycle—Within a memory cycle as the address phase transitions into the data phase.
2. Intercycle—Between two memory cycles when the data phase transitions into the address phase of the next memory cycle.

Intra-Cycle Bus Turn-Around

A typical case of an address to data transition happens during a read cycle. As shown in Figure 1, when the Address Latch Enable (ALE) is negated, the address is externally

latched and the CPU turns the bus around by tri-stating the A/D bus, so that the external memory system can begin to drive the expected data back to the CPU. The second case occurs during write cycles when the CPU finishes driving the address, it begins driving the data to the memory system. Since the CPU drives both the address and data during write cycles, bus turn-around is not a significant issue during write cycles. The two intra-cycle transition cases are listed in Table 1, which shows the state of the CPU A/D output buffers during the address and data phases of the transaction.

Note that the processor provides an output, $\overline{\text{DataEn}}$, to indicate that this transaction has occurred. During the addressing phase, $\overline{\text{DataEn}}$ is negated, indicating the CPU is driving the A/D bus. During the Data Phase, $\overline{\text{DataEn}}$ is asserted, indicating that the bus is to be driven by the external memory system. During write cycles, and during idle cycles, $\overline{\text{DataEn}}$ is guaranteed to be negated, indicating that the external memory system should not be driving the A/D bus.

Table 1. R3051 Address to Data Bus Transitional Behavior Within Memory Cycles

READ	A, Z
WRITE	A, D

Note: A = Address, D = Data, Z = Tri-State

Intercycle Bus Turn-Around

A typical case of the transition between two memory cycles occurs on a read cycle that is immediately followed by a write cycle as shown in Figure 3. In this case, the memory system is required to turn the bus around by tri-stating the bus before the next write cycle begins to drive its address onto the A/D lines. Table 2 lists the R3051's behavior on each of the cases of intercycle memory transitions. The table lists the state of the CPU output buffers at the end of the first transaction, followed by the state of the buffers at the beginning of the next transaction. Note that if a read or write cycle occurs while the CPU is executing instructions from its internal cache, the next external memory cycle might not occur until many clocks later, in which case the A/D bus is tri-stated since it is idle. Also, many of the cases, such as the transitions after writes have both the data and address driven by the CPU. Thus bus turn-around is not a significant issue after write cycles. Other transitions may not actually be possible. For example, it is impossible to have a read followed by a read. At least two idle cycles are required, to accomodate the read buffer latch and the internal fix-up cycle required by the processor (see the R3051 Hardware User's Manual for more detail).

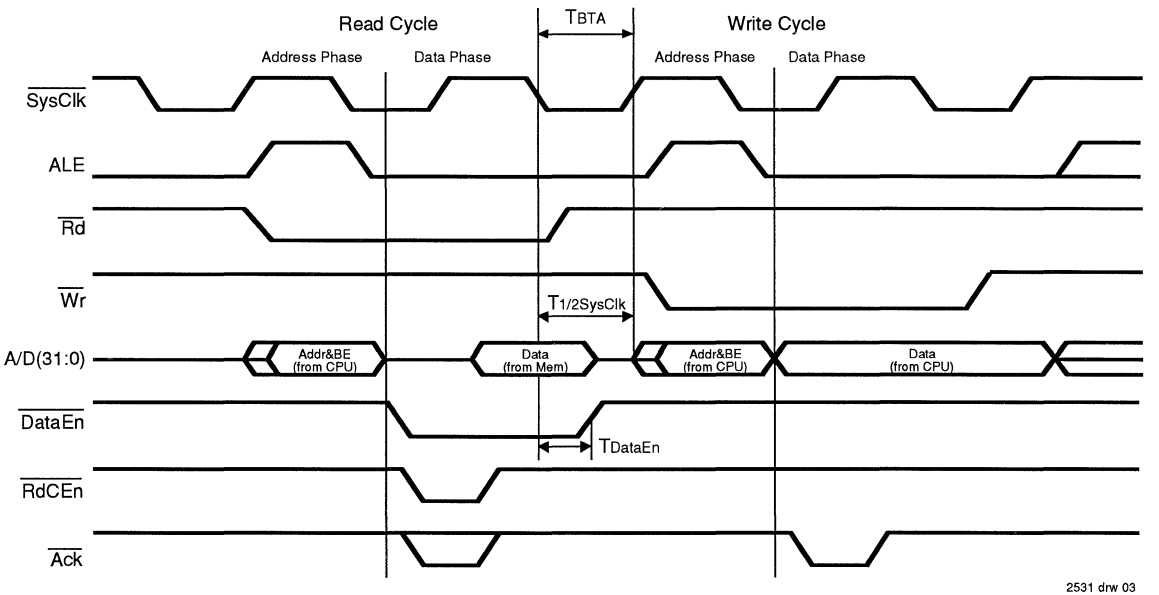


Figure 3. R3051 Read Cycle Followed by a Write Cycle

Table 2. R3051 Data to Address Bus Transitional Behavior Between Memory Transactions

From To	READ	WRITE	DMA	IDLE
READ	Z, A	Z, A	Z, Z	Z, Z
WRITE	D, A	D, A	D, Z	D, Z
DMA	Z, A	Z, A	Z, Z	Z, Z
IDLE	Z, A	Z, A	Z, Z	Z, Z

Note: A = Address, D = Data, Z = Tri-State

TYPICAL SYSTEMS AND BUS TURN-AROUND

To handle the timing associated with the bus turn-around within a memory cycle, the Data Enable output, $\overline{\text{DataEn}}$ is provided by the R3051. As shown in Figure 1, on read cycles, $\overline{\text{DataEn}}$ gives an indication when the CPU has tri-stated the A/D bus. Thus after $\overline{\text{DataEn}}$ asserts, the memory system can begin driving data onto the A/D bus. The system designer can also look for the rising clock edge of $\overline{\text{SysClk}}$ after $\overline{\text{Rd}}$ asserts before allowing the memory system to drive data.

To handle the timing associated with the bus turn-around between two memory transactions, consider the case of a read cycle immediately followed by a write cycle. The read cycle output enable control of the memory system must be such that the output drivers of the memory system turn off within 1/2 clock before the next address is driven by the write cycle. If the memory devices have an output disable to tri-state time (T_{OEZ}) of more than 1/2 clock, then they can be isolated from the A/D bus with a bank of data transceivers such as the 54/74FCT245T, 54/74FCT861, or 54/74FCT623T or with latched data transceivers such as the 54/74FCT543T or 54/74FCT646T as shown in Figure 4. All of these transceivers have very fast output disable times.

VERY FAST $\overline{\text{SYSCLK}}$ OR VERY SLOW T_{OEZ} AND BUS TURN-AROUND

The majority of systems will use evenly matched memories relative to the system clock speed or use transceivers. However, two exceptions may occur:

1. Very Fast $\overline{\text{SysClk}}$ — Even with the highest speed transceivers, their output disable times (T_{OEZ}) are around 5–8ns. Thus at 40MHz, if $\overline{\text{DataEn}}$ is used, it has a clock to de-assert time of 4ns. (Assume that the transceiver has two internally Anded output enable inputs. For example, as shown in Figure 4, the FCT543T transceiver bank can use $\overline{\text{DataEn}}$ and the bank select for inputs to the output enables). If 1ns is allowed for clock skew, this just meets the worst case timing criterion of:

   $T_{1/2SysClk}$ (12.5) ≥ T_{DataEn} + T_{OEZ} + $T_{ClkSkew}$ + T_{Cap} (4+6.5+1+0)

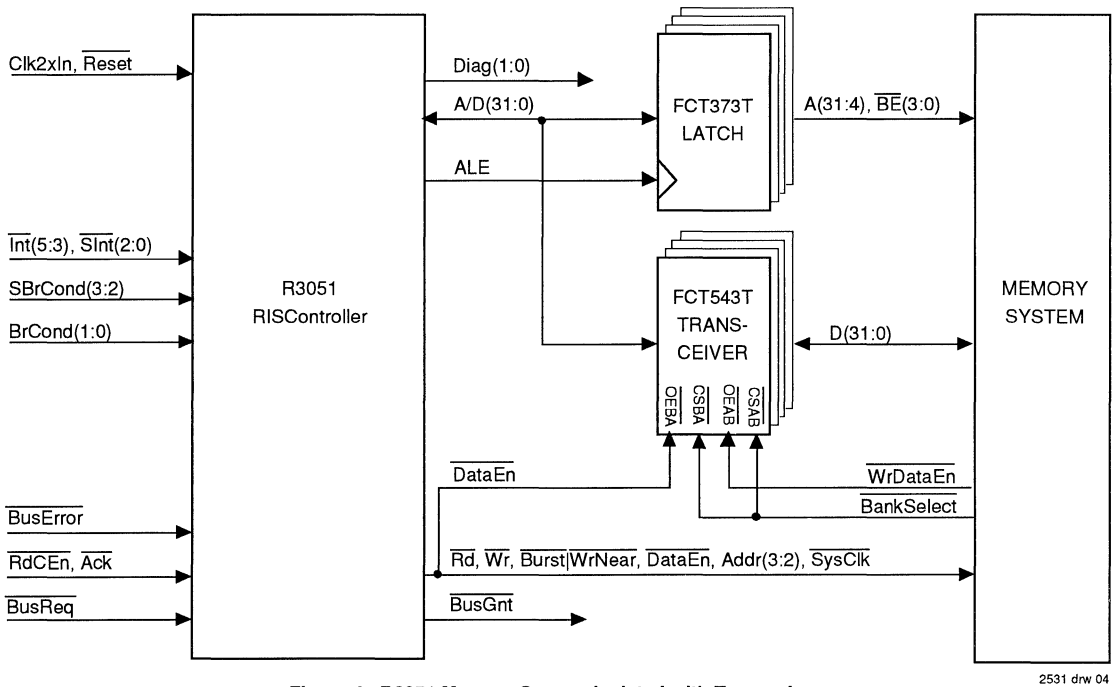


Figure 4. R3051 Memory System Isolated with Transceivers

Some choices of transceiver and PLA-based output enable control combinations may need more time than is allowed by the above equation. Solutions to this problem will be given in the section below, "Using DMA BusReq to Match CPU and Memory Speeds."

2. Very Slow Memories — The second case occurs when relatively slow TOEZ memories are attached directly to the A/D bus as shown in Figure 5. Such systems require these memories to turn off within 1/2 clock. A 20MHz R3051 has a TDataEn for the de-asserting edge of DataEn of 7ns. Assume that additional output enable control circuitry adds an additional delay of 10ns. 1ns is allowed for clock skew. For an inexpensive, slow 120ns EPROM, the output disable time is about 50ns, which seems to limit the clock speed to about 7MHz:

   $T_{1/2SysClk}$ (71.4) ≥ T_{DataEn} + $T_{OutputEnableControl}$ + T_{OEZ} + $T_{ClkSkew}$ + T_{Cap} (7+10+50+1+0)

However, as will be explained below in the section called, "Using DMA BusReq to Match CPU and Memory Speeds," the overall CPU speed does not have to be slowed down just because a slow TOEZ memory is attached directly to the A/D bus.

USING DMA BUSREQ TO MATCH CPU AND MEMORY SPEEDS

For systems with very fast SysClk or very slow memories, a solution exists to the bus turn-around timing constraints by using the Direct Memory Access (DMA) interface on the R3051. The R3051 DMA interface consists of two pins called BusReq and BusGnt as shown in Figure 6. Normally these pins are used for giving an external device control of the CPU bus instead of giving control of the bus to the R3051. In the R3051, when BusReq is asserted, DMA always has the highest priority immediately after the current memory cycle completes. The BusReq input is always sampled on the rising edge of SysClk. After the BusGnt is given, all of the CPU control line outputs, except SysClk and BusGnt are tri-stated. When the DMA device is finished with the bus, it de-asserts BusReq which then causes the CPU to de-assert BusGnt. The BusGnt output is always asserted on the rising edge of SysClk and de-asserted on the falling edge of SysClk.

Because a BusReq always has the highest priority, in a very fast SysClk system or a very slow memory system, asserting BusReq during the read cycle insures that the DMA request will always be granted at the end of the read cycle. After this happens, the BusReq pin can be de-asserted after the desired

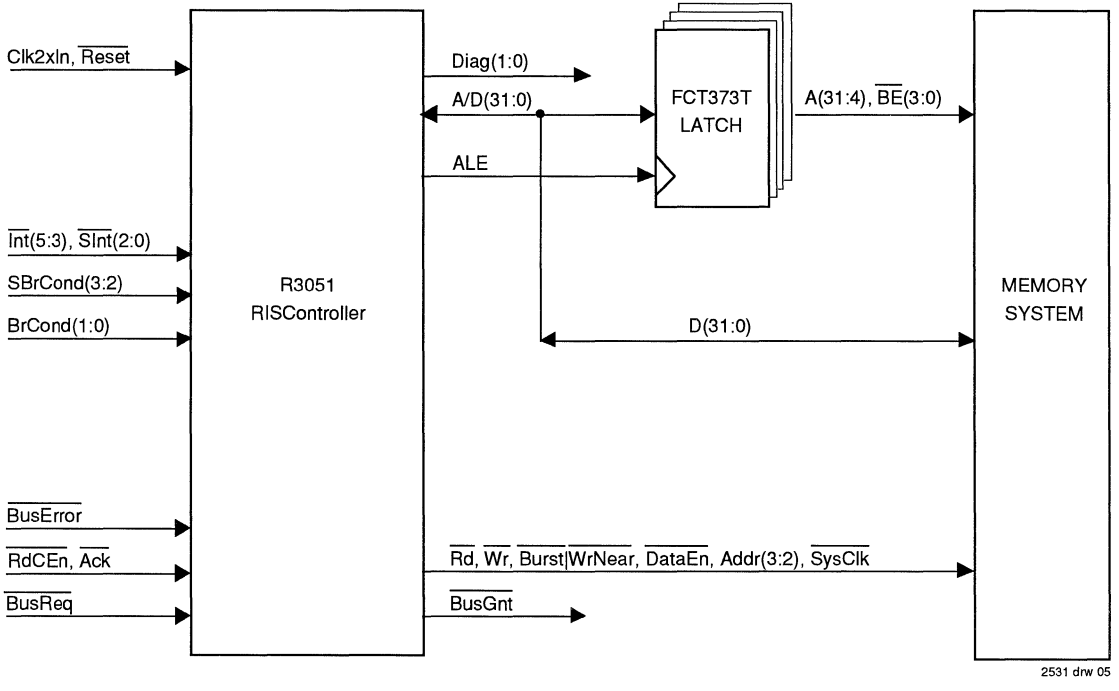


Figure 5. R3051 Memory System Connected Directly to the A/D Bus

number of intercycle wait-states have been inserted. For example, as shown in Figure 7, by attaching the buffered read line, $\overline{Rd}$ to $\overline{BusReq}$, the R3051 will grant the $\overline{BusReq}$ and immediately release it. Note that $\overline{Rd}$ needs to be buffered to meet the hold time of the $\overline{BusReq}$ input. Examine Figure 3, where a write cycle normally can follow a read cycle after 0.5 clocks and then compare it with Figure 7. In Figure 7, by using $\overline{BusReq}$, it can be seen that a minimum of 1.5 clocks is guaranteed before the next memory cycle is started by the CPU.

Note that when using DMA, the system may choose to resistively pull-up or down its control signals since the DMA

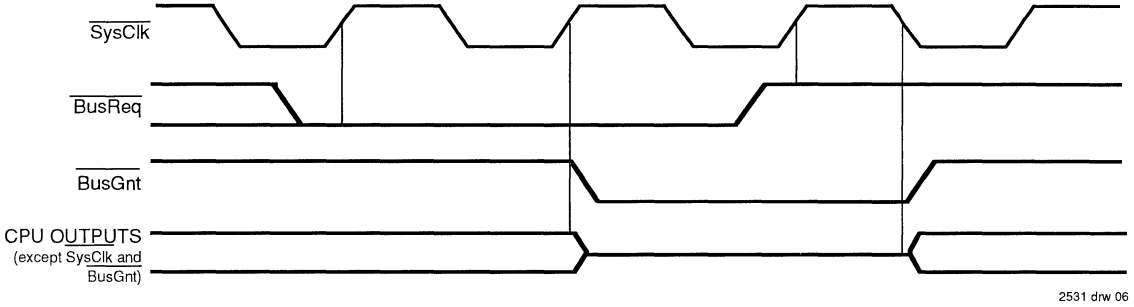


Figure 6. R3051 DMA $\overline{BusReq}$ and $\overline{BusGnt}$ Timing

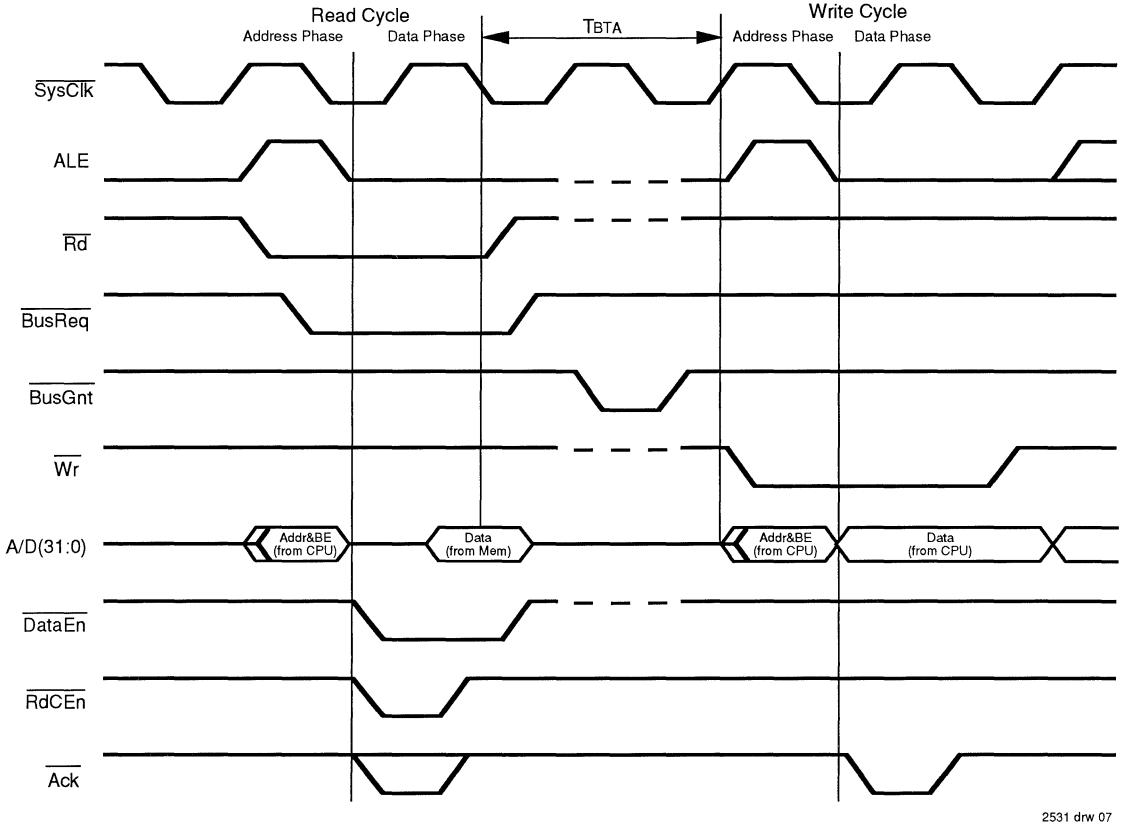


Figure 7. Using $\overline{BusReq}$ to Add More Bus Turn-around Time

when granted will tri-state the CPU control output signals. Thus ALE could use a pull-down, while $\overline{Rd}$, $\overline{Wr}$, $\overline{DataEn}$, and $\overline{Burst}|\overline{WrNear}$ could use pull-ups. The resistor value of the pull-ups and pull-down is not that critical since the R3051 always drives the control signals to their de-asserted states before tri-stating them. Also, if the $\overline{BusReq}$ is needed for conventional DMA, a fixed-priority based arbiter can be used to allow bus turn-around wait-state injection the highest priority and to allow conventional DMA the next priority.

Various improvements can be made to using the $\overline{Rd}$ line for $\overline{BusReq}$. For example, instead of using the buffered $\overline{Rd}$ line, use the decoded chip select of the particular memory (e.g., the EPROM) that has the relatively slow T_{OEZ}. Thus the extra wait-states are only asserted as needed (that is, after the slow memory is accessed).

SUMMARY

The R3051 allows inexpensive systems to be designed with the high throughput R3000 RISC instruction set architecture. The small 84-pin count is achieved with a multiplexed address and data bus, called "A/D". The use of the multiplexed A/D bus allows ASICs and Memory Controllers such as the R3721 DRAM Controller to have fewer interface pins, with no real loss of system performance or real added complexity. However, as for any high-speed bus (either multiplexed or not) care has to be taken to avoid bus clashes as the bus transitions from one device to another. This application note describes these considerations.

As shown in the text, the use of the A/D bus does not inherently limit the overall clock speed of the system, since either transceivers, or the described method of using the DMA $\overline{BusReq}$ input gives a solution for memory/CPU mismatches. Thus any memory or I/O system can use the multiplexed A/D bus and be designed to run at the full CPU clock frequency.

FOR FURTHER INFORMATION:

1. *IDT79R3051 Family Hardware User's Manual*, MAN-RISC-00051, Integrated Device Technology, Inc., Santa Clara, CA, 1991. Describes the H/W features and functionality of the device as well the bus interface.
2. *IDT 1991 RISC Data Book*, DBK-RISC-00021, Integrated Device Technology, Inc., Santa Clara, CA, 1991. Contains the data sheet with packaging, pinout, AC/DC electrical and thermal parameters.
3. G. Kane, *MIPS RISC Architecture*, Prentice Hall, Englewood Cliffs, NJ, 1988. Describes the R3000/R3051 instruction set architecture from a systems and assembly-level programming perspective.
4. *IDT 1991 Logic Data Book*, DBK-LOGIC-00080, Integrated Device Technology, Inc., Santa Clara, CA, 1991. Contains the data sheets for many different high-speed FCT transceivers, latches, and buffers.

Integrated Device Technology, Inc.

IDTR3051™ EMULATION OF REAL8™ LASER PRINTER CONTROLLER USING IDT7RS385 EVALUATION BOARD

APPLICATION NOTE AN-98

By Bob Napaa

INTRODUCTION

To evaluate the performance and system cost of IDT's R3051™ RISController™ family in a laser printer environment, IDT has developed an emulation of the REAL8™ Laser Printer Controller (IDT7RS388), complete with ports of the PeerlessPage™ Imaging Environment, Microsoft® TrueImage™ (PostScript® compatible) PDL™ and PeerlessPrint5™ (HP LaserJet III PCL5™-compatible) languages, using the IDT7RS385 Evaluation Board as the hardware platform. Like REAL8, the 7RS385 board includes a fast Centronics parallel input port and the identical video interface for the Canon LBP-SX™ print engine has been added in the wire-wrap area.

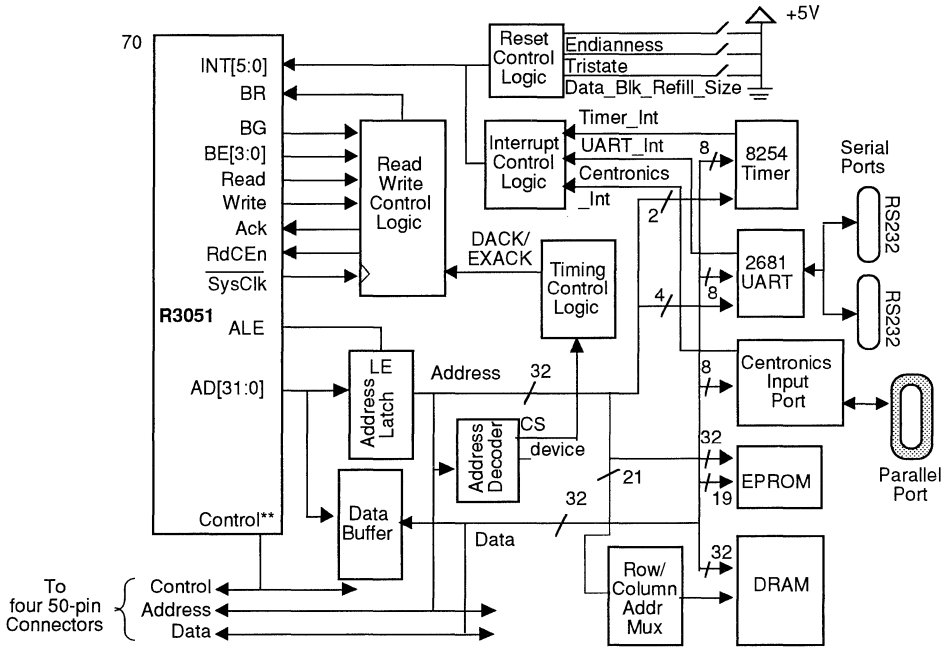
In this configuration, the emulation provides a checklist design model for OEMs wanting to use the same processor family, but adapting it to other I/O configurations, or driving other laser print engines.

IDT7RS385 RISC Evaluation Board

The IDT7RS385 is a complete RISC system self-contained on a single printed circuit board. The 7RS385 is designed around the IDT79R3051 RISController family. All four devices in the family (R3051, R3051E, R3052 and R3052E) are pin and software compatible. As a consequence, any device can be substituted for the R3051. The major features of the 7RS385 include:

- IDT79R3052E RISController
- 1MB DRAM expandable to 4MB
- 128kB EPROM expandable to 2MB
- Programmable DUART (2681) with two serial ports
- Programmable counter/timer (8254)
- Centronics parallel input port with FIFO
- Clock, reset and interrupt generation circuitry
- IDT/sim™– Initialization and monitor debugging software
- HP16500 logic analyzer pod connectors
- Expansion bus connectors
- User wire-wrap area

Figure 1 illustrates the 7RS385 block diagram while Figure 2 illustrates the block diagram of the REAL8 laser printer controller. Additional information on the the IDT79R3051 family, including the CPUs, support chips and development



** These control signals include R3051 and the on-board control logic signals as well.

Figure 1. 7RS385 Block Diagram

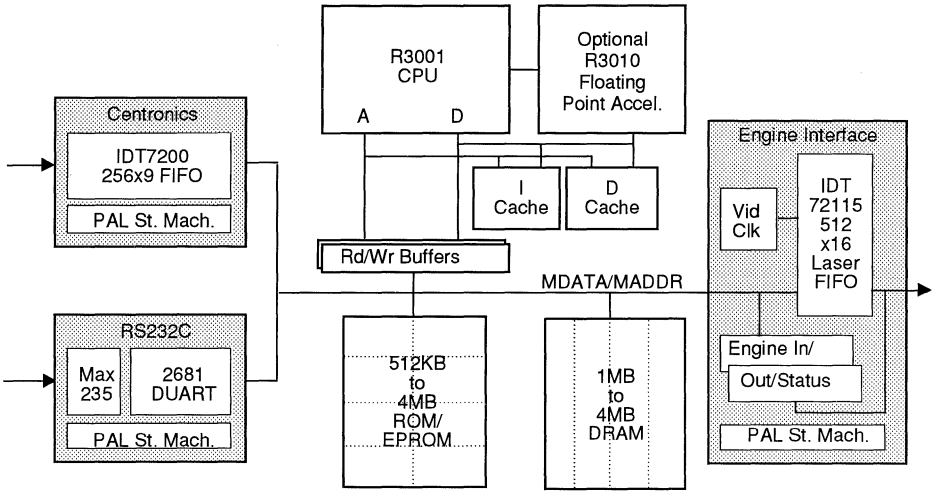


Figure 2. REAL8 Laser Printer Controller Block Diagram

software is available from IDT. The IDT7RS385 User's Manual provides more information on the 7RS385 Evaluation board.

IDT7RS385_REAL8_Emulation

The IDT7RS385_REAL8_Emulation is a modified 7RS385 Evaluation board designed to control a Canon LBP-SX laser printer engine. The 7RS385_REAL8_Emulation is designed to run at 25MHz with 1 bank non-interleaved of 4MB DRAM and 1 bank non-interleaved of 2MB EPROM. The 7RS385_REAL8_Emulation emulates the complete memory mapping, interrupt structure, endianness and video interface implemented on the IDT REAL8 Laser Printer Controller board. The emulation of the hardware of the REAL8 board enables the software from THE PEERLESS GROUP to be run with only minor modifications. The software however does need to be recompiled with IDT/c™ using the floating-point library since the current R3051 family does not support a hardware floating-point accelerator.

The basic configuration of 7RS385 has been modified as follows:

1. Change jumpers and DRAM to 4MB configuration.
2. Change jumpers and EPROM to 2MB configuration.
3. Modify "endianness" to little endian required by the software.
4. Modify memory mapping and interrupt structure to match REAL8.
5. Disabling the 8254 timers not required by REAL8.
6. Adding the engine interface in the wire-wrap area.

Figure 3 illustrates the physical layout of the 7RS385 Evaluation board.

IDT7RS385_REAL8_Emulation Implementation

Several steps of modifications of the original 7RS385 design are required to emulate the complete hardware of the REAL8 board. In addition the video interface must be implemented on the wire-wrap area on the board.

- Endianness

  The original 7RS385 board includes a big endian version of IDT/sim and thus is considered a big endian board. The board has to be changed to a little endian board since the software from The Peerless Group is a little endian one. The endianness to the R3051 can be specified during reset on interrupt pin 0. A level "1" specifies big endian system while a level "0" specifies a little endian system. The board is converted to little endian by shorting pins 1 and 2 of jumper P7.

- Timer

  The original 7RS385 board uses an Intel 8254 timer device which implements two timers. Each of the two timers may be programmed as an independent REAL-time interrupt occurring at regular intervals. The two timer outputs (OUT0 and OUT1) are forwarded to the R3051 via the interrupt PAL as synchronous interrupts. However, this timer device has been disabled (by removing the Vcc and ground pins) since the software does not make use of it. The software implements a timer function using the DUART device. OUT0 and OUT1 of the two timers are replaced by the Timer_interrupt (from the DUART timer) and the Video_Fifo_Empty (from the video interface) signals respectively. However, it is possible to keep the 8254 timers enabled if the application software can make use of them. This requires extra modifications in the interrupt structure and in the address decoding.

- DUART

  The original 7RS385 board uses a Signetics 2681 DUART to control two serial communications ports. The first port is for the CRT terminal while the second is for auxiliary use such as downloading software from a host. The functionality of the DUART is almost unchanged in the modified board (the 7RS385_REAL8_Emulation) with some few exceptions. The software implements a real-time timer using the DUART and uses OP3 as the timer output. This signal is forwarded to the R3051 through the interrupt PAL as the timer interrupt signal. Five of the remaining general purpose output pins (OP2, OP4, OP5, OP6 and OP7) are used for the Centronics interface handshaking. These modifications require cuts and jumps on the OP2 and OP3 pins as demonstrated in the schematics. Also, the input pin TI5 on the Max235 needs to be shorted to ground since this signal was originally connected to OP2.

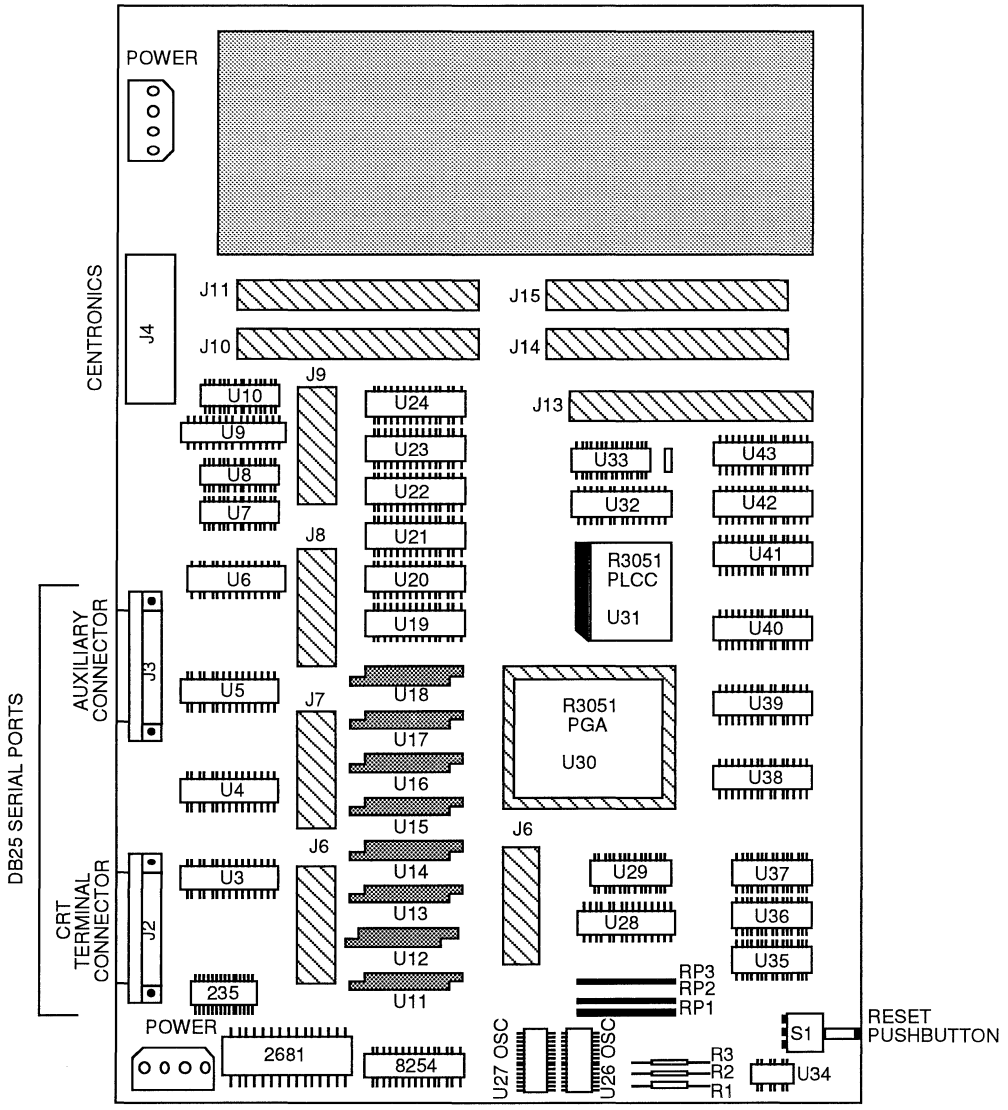


Figure 3. 7RS385 Physical Layout

- Memory Mapping

  Figure 4 illustrates the memory mapping of the original 7RS385 board. The address decoder PAL (U7) uses the upper 12 bits of the address bus to select among different memory segments. This memory mapping does not correspond exactly to the required memory mapping of the REAL8 board and thus modifications of the address decoder PAL are introduced. The new memory mapping is presented in Figure 5 and still uses the upper 12 bits of the address bus. The new memory decoding scheme can support up to 4MB of DRAM space and up to 2MB of EPROM space. For 4MB of DRAM, pins 1 and 2 of jumper P11 need to be cut while pins 2 and 3 need to be shorted. For 2MB of EPROM, pins 1 and 2 of jumpers P1, P2, P3, P4, P5 and P6 need to be cut while pins 2 and 3 need to be shorted.

- Interrupt structure

  The 7RS385 generates four interrupts synchronized to SysClk via the interrupt PAL (U28) and may be connected to any of the available interrupt inputs on the R3051 through the jumper set P12. The interrupts and the default interrupt input into the R3051 are shown below:

Interrupt	Source	Input to R3051
$\overline{\text{CINT0}}$	8254 counter 0	$\overline{\text{SINT0}}$
$\overline{\text{CINT1}}$	8254 counter 1	$\overline{\text{SINT1}}$
$\overline{\text{CENTINT}}$	Centronics FIFO full flag	$\overline{\text{INT3}}$
$\overline{\text{DINT}}$	DUART interrupt	$\overline{\text{INT5}}$

This interrupt structure does not correspond to the requirements of the REAL8 board. Modifications of the

FUNCTION	ADDRESS RANGE (HEX)	ADDRESS BITS (X DESIGNATES UNDECODED BIT)							
		31-28	27-24	23-20	19-16	15-12	11-9	7-4	3-0
DRAM	0000 0000	XX00	0000	1100	XXXX	XXXX	XXXX	XXXX	XXXX
(4 MBYTES)	003F FFFF	XX00	0000	0011	XXXX	XXXX	XXXX	XXXX	XXXX
CENTRONICS	0060 0000	0000	0000	0110	XXXX	XXXX	XXXX	XXXX	XXXX
TIMER	1F80 0000	0001	1111	1000	XXXX	XXXX	XXXX	XXXX	**XX
EPROM	1FC0 0000	0001	1111	1100	XXXX	XXXX	XXXX	XXXX	XXXX
(2 MBYTES)	1FDF FFFF	0001	1111	1101	XXXX	XXXX	XXXX	XXXX	XXXX
DUART	1FE0 0000	0001	1111	1110	XXXX	XXXX	XXXX	XX**	**XX
USER CS	1FA0 0000	0001	1111	1010	XXXX	XXXX	XXXX	XXXX	XXXX

* These bits are decoded at the device

Figure 4. The Memory Mapping of the 7RS385

FUNCTION	ADDED FUNCTION	ADDRESS RANGE (HEX)	ADDRESS BITS (X DESIGNATES UNDECODED BIT)							
			31-28	27-24	23-20	19-16	15-12	11-9	7-4	3-0
DRAM		0000 0000	XX00	0000	1100	XXXX	XXXX	XXXX	XXXX	XXXX
(4 MBYTES)		003F FFFF	XX00	0000	0011	XXXX	XXXX	XXXX	XXXX	XXXX
CENTRONICS		0060 0000	0000	0000	0110	XXXX	XXXX	XXXX	XXXX	XXXX
USER CS	LASER FIFO	0040 0000	0000	0000	0100	XXXX	XXXX	XXXX	XXXX	XXXX
		005F FFFF	0000	0000	0101	XXXX	XXXX	XXXX	XXXX	XXXX
	UNUSED AREA	0080 0000	0000	0000	1000	XXXX	XXXX	XXXX	XXXX	XXXX
	LASER ENGINE	0090 0000	0000	0000	1001	XXXX	XXXX	XXXX	XXXX	XXXX
DUART/TIMER		00A0 0000	0000	0000	1010	XXXX	XXXX	XXXX	XX**	**XX
EPROM		1FC0 0000	0001	1111	1100	XXXX	XXXX	XXXX	XXXX	XXXX
(2 MBYTES)		1FDF FFFF	0001	1111	1101	XXXX	XXXX	XXXX	XXXX	XXXX

* These bits are decoded at the device

Figure 5. The Memory Mapping of the 7RS385_REAL8_Emulation

interrupt PAL and the P12 jumper set are introduced to reflect the new interrupt structure shown below:

Interrupt	Source	Input to R3051
$\overline{\text{VREQINT}}$	Video request	$\overline{\text{SINT0}}$
$\overline{\text{VFEMPINT}}$	Video FIFO empty	$\overline{\text{SINT2}}$
$\overline{\text{CENTINT}}$	Centronics FIFO full flag	$\overline{\text{INT3}}$
$\overline{\text{TMRINT}}$	Timer interrupt	$\overline{\text{INT4}}$
$\overline{\text{DINT}}$	DUART interrupt	$\overline{\text{INT5}}$

To implement the new interrupt structure, $\overline{\text{COUT0}}$, $\overline{\text{COUT1}}$ and $\overline{\text{TIMER}}$ inputs lines to the interrupt PAL are replaced by $\overline{\text{VFEMPT}}$, $\overline{\text{TIMERINT}}$ and $\overline{\text{VREQ}}$ respectively. The jumper set P12 need to be modified as illustrated in the schematics.

- DRAM Memory Latency

  The original 7RS385-25 board design is optimized for non-interleaved 25MHz systems with the following DRAM memory latency for read and write accesses expressed in terms of external bus clock cycles:

single read:	5 clock cycles
quad word read:	5 clock cycles for the first word, 2 clock cycles for the remaining 3 words
single write:	5 clock cycles
page write:	4 clock cycles
$\overline{\text{RAS}}$ precharge time:	2 clock cycles

- EPROM Memory Latency

  The latency of the single, non-interleaved bank of EPROM has been improved for 25MHz, and is expressed in terms of external bus clock cycles as follows:

single read:	5 clock cycles
quad word read:	5 clock cycles for the first word, 4 clock cycles for the remaining 3 words

- Software

  The original 7RS385 board is shipped with a big endian version of the IDT System Integration Manager software (IDT/sim) which is a powerful tool for downloading software and debugging both hardware and software. This version of IDT/sim is tailored to the memory mapping of the existing design. A little endian version of IDT/sim tailored to the memory mapping of the REAL8 board is necessary to boot up the new 7RS385_REAL8_Emulation (a version of IDT/sim with the appropriate DUART address). This version of the software is readily available from IDT and is shipped with the REAL8 board. It only needs to be compiled for a little endian target system. No additional software modifications are required.

- Video Interface

  The 7RS385_REAL8_Emulation is designed to interface to a Canon LBP-SX laser printer controller. The video interface is implemented on the wire-wrap area of the 7RS385. The video interface resides in the User Chip Select segment of the memory as defined by the address decoder PAL (U7). The video interface PAL (U100) uses the $\overline{\text{UserCS}}$ line and address bits A20 to A23 to select between accesses to the engine and accesses to the LaserFifo. The video interface PAL uses a special signal "$\overline{\text{State40}}$" to synchronize its operation to the main state machine of the board and to return the "$\overline{\text{EXACK}}$" (External Ack) in a proper manner. The "$\overline{\text{State40}}$" signal has been specially added to the existing design to inform the external devices that the main state machine has finished all pending accesses and is waiting for a response from the User segment of the memory. This signal has been added to the U42 state machine PAL and prevents any confusion between the added external state machines and the existing one.

  The rest of the video interface design is a copy of the existing design on the 7RS385_REAL8_EMULATION board. An 8-bit register (FCT273) is used to output commands and handshaking signals to the engine while an FCT827 buffers the input status from the engine. An IDT Laser FIFO is used to store the bit map of the image to be delivered to the engine. A clock generator PAL (U102) divides a 14.91MHz clock by 8 to provide the synchronization clock between the controller and the laser printer engine. The second PAL (U103) synchronizes the video data (the bit map) from the LaserFifo to the laser printer engine data requests signals.

  Figure 6 illustrates the physical layout of the 7RS385_REAL8_Emulation board with the video interface added to the wire-wrap area.

CONCLUSION

The 7RS385 Evaluation board is a complete RISC system design intended for use as a stand alone evaluation system for the R3051 family and is flexible enough to be modified to fit the application at hand. It is very simple to design a laser printer controller based on the R3051 family and using the 7RS385 board for basically any type of laser engines (Canon, Sharp....) with minimal additions and modifications. Similarly, the design of the 7RS385 can be tailored to fit an X Terminal application or a data communications application with some external add-on hardware and minimal changes to the existing design.

The Appendices include the schematics of the 7RS385_REAL8_Emulation as well as the PAL equations for the modified PAL and the new added PALs.

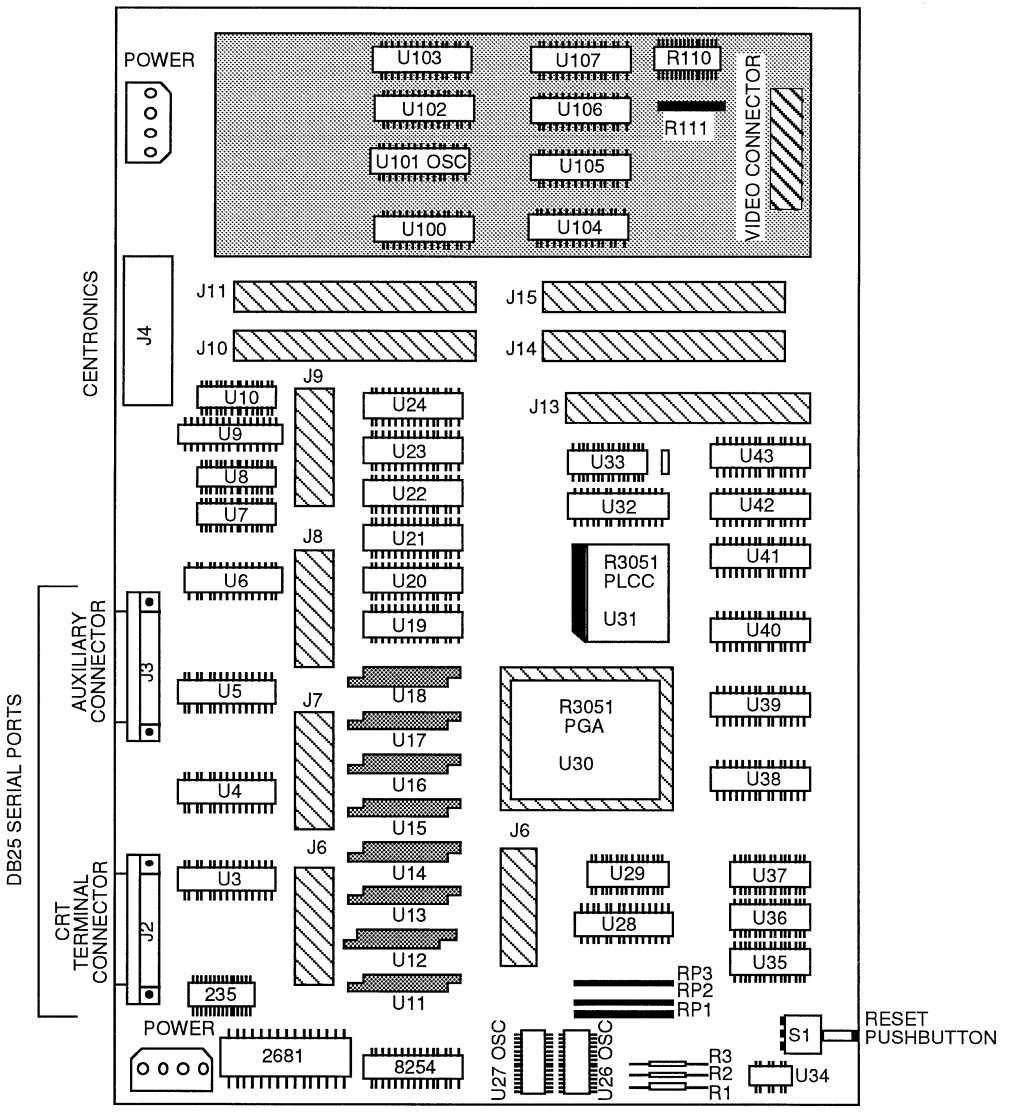


Figure 6. The Physical Layout of the 7RS385_REAL8_Emulation

```
"G. Takushi  03/26/91
"B. Napaa    09/10/91

"EDIT HISTORY:
"Date        Engineer      Checksum Modification
"-----------------------------------------------------------------------
"06/12/91    G. Takushi    8852     Initial release
"07/18/91    G. Takushi    9321     Decode only 1Mb DRAM and 128kb EPROM
"09/10/91    B. Napaa               Modified to emulate 388 memory mapping
"10/22/91    B. Napaa      81A8     Modified the output enable of the timer
"-----------------------------------------------------------------------
MODULE AddrDcdr

TITLE '3051 Evaluation Board: Address Decoder Pal'

U7 DEVICE 'P22V10';

"-----------------------------------------------------------------------

"Inputs
    A31,A30,A29,A28     PIN 13,11,10,9;
    A27,A26,A25,A24     PIN 8,7,6,5;
    A23,A22,A21,A20     PIN 4,3,2,1;

"-----------------------------------------------------------------------

"Outputs
    DEV2        PIN 22;     "Device code bit 2
    DEV1        PIN 21;     "Device code bit 1
    DEV0        PIN 20;     "Device code bit 0
    EPROM       PIN 19;     "EPROM#
    UCS         PIN 18;     "UCS#
    TIMER       PIN 17;     "TIMER#
    DUART       PIN 16;     "DUART#
    DRAM        PIN 15;     "DRAM#

"-----------------------------------------------------------------------

"Constants
    X = .X.;
    Address = [A31..A20, X,X,X,X, X,X,X,X, X,X,X,X, X,X,X,X, X,X,X,X];
    DevCode = [DEV2..DEV0];
    Dram_max = ^H003FFFFF;      "4 Mbyte DRAM
    Eprom_max = ^H1FDFFFFF;     "2 Mbyte EPROM

"-----------------------------------------------------------------------

EQUATIONS
    DevCode     = ((Address & ^H3FFFFFFF) <= Dram_max) & [0,0,0] #          "DRAM# 0

                  (Address == ^H00600000) & [0,0,1] #                       "CENT# 1

                  ((Address >= ^H1FC00000) & (Address <= Eprom_max)) & [1,0,1] #"EPROM# 5

                  (Address == ^H00A00000) & [0,1,0] #                       "DUART# 2

                  (((Address >= ^H00400000) & (Address <= ^H005FFFFF)) #
                  ((Address >= ^H00800000) & (Address <= ^H009FFFFF))) & [1,0,0]; "UCS# 4
```

```
!DRAM = ((Address & ^H3FFFFFFF) <= Dram_max); "A(31:30) are don't cares for DRAM*
                                                "to insure compatibility between
                                                "305x and 305xE.

TIMER.OE    = 0;
!TIMER      = 0;
!EPROM      = (Address >= ^H1FC00000) & (Address <= Eprom_max);
!DUART      = (Address == ^H00A00000);
!UCS        = ((Address >= ^H00400000) & (Address <= ^H005FFFFF)) #
              ((Address >= ^H00800000) & (Address <= ^H009FFFFF));

"_____

END AddrDcdr
```

```
"G. Takushi  04/11/91
"B. Napaa    09/12/91

"EDIT HISTORY:
"Date        Engineer      Checksum Modification
"-----------------------------------------------------------------
"06/12/91    G. Takushi    88EC     Initial release
"09/12/91    B. Napaa      355D     Modified to emulate interrupt structure
"                                   of 388 board.
"-----------------------------------------------------------------

MODULE Interrupt

TITLE '3051 Evaluation Board: Interrupt PAL'

U28 DEVICE 'P22V10';

"-----------------------------------------------------------------

"Inputs
    SYSCLK      pin 1;
    MRES        pin 2;
    DUARTINT    pin 3;
    RD          pin 4;
    VREQ        pin 5;
    A02         pin 6;
    A04         pin 7;
    TIMER       pin 8;
    VFEMPT      pin 9;
    CENTFF      pin 11;
    OutEn       pin 13;

"-----------------------------------------------------------------

"Outputs
    DINT        pin 18; "INT 5
    CENTINT     pin 19; "INT 3
    VFEMPINT    pin 20; "INT 2
    VREQINT     pin 21; "INT 0
    TMRINT      pin 22; "INT 4

"-----------------------------------------------------------------

"Constants
    ON,OFF,T,F,X,C = 1,0,1,0,.X.,.C.;

"-----------------------------------------------------------------

EQUATIONS


    DINT.OE = !OutEn;
    CENTINT.OE = !OutEn;
    VREQINT.OE = !OutEn;
    VFEMPINT.OE = !OutEn;
    TMRINT.OE = !OutEn;

    !CENTINT := MRES & !CENTFF;
```

```
    !VREQINT := MRES & !VREQ;

    !VFEMPINT := MRES & !VFEMPT;

    !TMRINT := MRES & !TIMER;

    !DINT := !DUARTINT;

"-------------------------------------------------------------

END Interrupt
```

```
"G. Takushi  04/11/91
"B. Napaa    09/25/91

"EDIT HISTORY:
"Date        Engineer      Checksum Modification
"-----------  ------------  --------  ------------------------------------
"06/12/91    G. Takushi    A601      Initial release
"09/25/91    B. Napaa                Added State40 output to interface to the video
"11/12/91    B. Napaa      A818      Modified UserCS by removing state 43
"01/13/92    B. Napaa      9197      Modified to reduce EPROM read latency
"-------------------------------------------------------------------------

MODULE StateMach1

FLAG '-r3','-f'

TITLE '3051 Evaluation Board: State Machine A'

U42 DEVICE 'P22V10';

"-------------------------------------------------------------------------

"Inputs
    SYSCLK      pin 1;
    MRES        pin 2;
    RD          pin 3;
    WR          pin 4;
    DRDCEN      pin 5;
    DEV0        pin 6;
    DEV1        pin 7;
    DEV2        pin 8;
    BWN         pin 9;
    BUSGNT      pin 10;
    EXACK       pin 11;
    DACK        pin 13;
    reset       node 25;

"-------------------------------------------------------------------------

"Outputs
    ACK         pin 14;
    STATE40     pin 15;
    CNT5        pin 16;
    CNT3        pin 17;
    CNT2        pin 18;
    CNT0        pin 19;
    CNT1        pin 20;
    CNT4        pin 21;
    RDCEN       pin 23;

"-------------------------------------------------------------------------

"Constants
    ON,OFF,T,F,X,C = 1,0,1,0,.X.,.C.;
    DevCode = [DEV2..DEV0];
    Operation = [MRES,BUSGNT,RD,WR,DevCode];
    Idle        = [  1,    1, 1, 1,        X]; "Idle
    EpromRd     = [  1,    1, 0, 1,     5]; "EPROM read
    Dram        = [  1,    1, X, X,     0]; "DRAM operation
```

```
        DTRd    = [  1,    1, 0, 1,    2]; "DUART/Timer rd
        DTWr    = [  1,    1, 1, 0,    2]; "DUART/Timer  wr
        UserRd  = [  1,    1, 0, 1,    4]; "User rd
        UserWr  = [  1,    1, 1, 0,    4]; "User wr
        CentRd  = [  1,    1, 0, 1,    1]; "Centronics read

        Pstate = [CNT5..CNT0];
        S00 = [1,1,1,1,1,1]; S32 = [0,0,1,1,1,1];
        S01 = [1,1,1,1,1,0]; S33 = [0,0,1,1,1,0];
        S02 = [1,1,1,1,0,0]; S34 = [0,0,1,1,0,0];
        S03 = [1,1,1,1,0,1]; S35 = [0,0,1,1,0,1];
        S04 = [1,1,1,0,0,1]; S36 = [0,0,1,0,0,1];
        S05 = [1,1,1,0,0,0]; S37 = [0,0,1,0,0,0];
        S06 = [1,1,1,0,1,0]; S38 = [0,0,1,0,1,0];
        S07 = [1,1,1,0,1,1]; S39 = [0,0,1,0,1,1];
        S08 = [1,1,0,0,1,1]; S40 = [0,0,0,0,1,1];
        S09 = [1,1,0,0,1,0]; S41 = [0,0,0,0,1,0];
        S10 = [1,1,0,0,0,0]; S42 = [0,0,0,0,0,0];
        S11 = [1,1,0,0,0,1]; S43 = [0,0,0,0,0,1];
        S12 = [1,1,0,1,0,1]; S44 = [0,0,0,1,0,1];
        S13 = [1,1,0,1,0,0]; S45 = [0,0,0,1,0,0];
        S14 = [1,1,0,1,1,0]; S46 = [0,0,0,1,1,0];
        S15 = [1,1,0,1,1,1]; S47 = [0,0,0,1,1,1];
        S16 = [1,0,0,1,1,1]; S48 = [0,1,0,1,1,1];
        S17 = [1,0,0,1,1,0]; S49 = [0,1,0,1,1,0];
        S18 = [1,0,0,1,0,0]; S50 = [0,1,0,1,0,0];
        S19 = [1,0,0,1,0,1]; S51 = [0,1,0,1,0,1];
        S20 = [1,0,0,0,0,1]; S52 = [0,1,0,0,0,1];
        S21 = [1,0,0,0,0,0]; S53 = [0,1,0,0,0,0];
        S22 = [1,0,0,0,1,0]; S54 = [0,1,0,0,1,0];
        S23 = [1,0,0,0,1,1]; S55 = [0,1,0,0,1,1];
        S24 = [1,0,1,0,1,1]; S56 = [0,1,1,0,1,1];
        S25 = [1,0,1,0,1,0]; S57 = [0,1,1,0,1,0];
        S26 = [1,0,1,0,0,0]; S58 = [0,1,1,0,0,0];
        S27 = [1,0,1,0,0,1]; S59 = [0,1,1,0,0,1];
        S28 = [1,0,1,1,0,1]; S60 = [0,1,1,1,0,1];
        S29 = [1,0,1,1,0,0]; S61 = [0,1,1,1,0,0];
        S30 = [1,0,1,1,1,0]; S62 = [0,1,1,1,1,0];
        S31 = [1,0,1,1,1,1]; S63 = [0,1,1,1,1,1];

"_____

EQUATIONS
        reset = !MRES;

        "!STATE40 = !CNT5 & !CNT4 & !CNT3 & !CNT2 & CNT1 & CNT0;

"DMA Request
        ACK.OE = BUSGNT;
        RDCEN.OE = BUSGNT;

"DRAM Operation
        !ACK := !DACK;
        !RDCEN := !DRDCEN;

"_____
```

```
STATE_DIAGRAM Pstate
    state S00:                                          "Idle
        case    Operation == EpromRd: S01;
                Operation == DTRd:    S25;
                Operation == DTWr:    S25;
                Operation == UserRd:  S40;
                Operation == UserWr:  S40;
                Operation == CentRd:  S44;
                Operation == Dram:    S63;
        endcase;

    state S01:      goto S02;                           "Eprom read
    state S02:      !RDCEN := T;
                    goto S03;
    state S03:      goto S04;
    state S04:
        if !BWN then S05 else S00;

    state S05:      goto S06;                           "Eprom burst read
    state S06:      !RDCEN := T;
                    goto S07;
    state S07:      goto S08;
    state S08:      goto S09;
    state S09:      goto S10;
    state S10:      !RDCEN := T;
                    goto S11;
    state S11:      !ACK := T;
                    goto S12;
    state S12:      goto S13;
    state S13:      goto S14;
    state S14:      !RDCEN := T;
                    goto S15;
    state S15:      goto S16;
    state S16:      goto S00;

    state S25:      goto S26;                           "DUART/Timer read/write
    state S26:      goto S27;
    state S27:      goto S28;
    state S28:      goto S29;
    state S29:      goto S30;
    state S30:      goto S31;
    state S31:      !RDCEN := T;
                    !ACK := T;
                    goto S32;
    state S32:      goto S33;
    state S33:      goto S34;
    state S34:      goto S35;
    state S35:      goto S36;
    state S36:      goto S37;
    state S37:      goto S38;
    state S38:      goto S39;
    state S39:      goto S00;

    state S40:      !STATE40 := T;
                    if EXACK then S40 else S41      "User read/write
                        with    !ACK := T;
                                !RDCEN := T;
                        endwith;
```

```
        state S41:      goto S42;
        state S42:      goto S00;
        state S43:      goto S00;

        state S44:      !RDCEN := T;                    "Centronics read
                        goto S45;
        state S45:      goto S46;
        state S46:      goto S00;

        state S63:      if (Operation == Dram) then S63;        "DRAM

"-------------------------------------------------------------------------
END StateMach1
```

```
"G. Takushi 04/11/91

"EDIT HISTORY:
"Date        Engineer      Checksum Modification
"__________  __________    _______  ____________________________________
"06/12/91    G. Takushi    B5E8     Initial release
"01/13/92    B. Napaa      B5E8     modified DBBUSY to reflect new changes
"                                   in EPROM reads
"______________________________________________________________________

MODULE StateMach2

FLAG '-f'

TITLE '3051 Evaluation Board: State Machine B'

U43 DEVICE 'P22V10';

"______________________________________________________________________

"Inputs
        SYSCLK      pin 1;
        MRES        pin 2;
        RD          pin 3;
        WR          pin 4;
        CNT0        pin 5;
        CNT1        pin 6;
        CNT2        pin 7;
        CNT3        pin 8;
        CNT4        pin 9;
        CNT5        pin 10;
        DATAEN      pin 11;
        reset       node 25;

"______________________________________________________________________

"Outputs
        CENTRD      pin 14;
        XWR         pin 16;
        DBUSBSY     pin 20;
        XRD         pin 21;
        DOE         pin 23;
"______________________________________________________________________

"Constants
        ON,OFF,T,F,X,C = 1,0,1,0,.X.,.C.;
        Pstate = [CNT5..CNT0];
        S00 = [1,1,1,1,1,1]; S32 = [0,0,1,1,1,1];
        S01 = [1,1,1,1,1,0]; S33 = [0,0,1,1,1,0];
        S02 = [1,1,1,1,0,0]; S34 = [0,0,1,1,0,0];
        S03 = [1,1,1,1,0,1]; S35 = [0,0,1,1,0,1];
        S04 = [1,1,1,0,0,1]; S36 = [0,0,1,0,0,1];
        S05 = [1,1,1,0,0,0]; S37 = [0,0,1,0,0,0];
        S06 = [1,1,1,0,1,0]; S38 = [0,0,1,0,1,0];
        S07 = [1,1,1,0,1,1]; S39 = [0,0,1,0,1,1];
        S08 = [1,1,0,0,1,1]; S40 = [0,0,0,0,1,1];
        S09 = [1,1,0,0,1,0]; S41 = [0,0,0,0,1,0];
```

```
        S10 = [1,1,0,0,0,0]; S42 = [0,0,0,0,0,0];
        S11 = [1,1,0,0,0,1]; S43 = [0,0,0,0,0,1];
        S12 = [1,1,0,1,0,1]; S44 = [0,0,0,1,0,1];
        S13 = [1,1,0,1,0,0]; S45 = [0,0,0,1,0,0];
        S14 = [1,1,0,1,1,0]; S46 = [0,0,0,1,1,0];
        S15 = [1,1,0,1,1,1]; S47 = [0,0,0,1,1,1];
        S16 = [1,0,0,1,1,1]; S48 = [0,1,0,1,1,1];
        S17 = [1,0,0,1,1,0]; S49 = [0,1,0,1,1,0];
        S18 = [1,0,0,1,0,0]; S50 = [0,1,0,1,0,0];
        S19 = [1,0,0,1,0,1]; S51 = [0,1,0,1,0,1];
        S20 = [1,0,0,0,0,1]; S52 = [0,1,0,0,0,1];
        S21 = [1,0,0,0,0,0]; S53 = [0,1,0,0,0,0];
        S22 = [1,0,0,0,1,0]; S54 = [0,1,0,0,1,0];
        S23 = [1,0,0,0,1,1]; S55 = [0,1,0,0,1,1];
        S24 = [1,0,1,0,1,1]; S56 = [0,1,1,0,1,1];
        S25 = [1,0,1,0,1,0]; S57 = [0,1,1,0,1,0];
        S26 = [1,0,1,0,0,0]; S58 = [0,1,1,0,0,0];
        S27 = [1,0,1,0,0,1]; S59 = [0,1,1,0,0,1];
        S28 = [1,0,1,1,0,1]; S60 = [0,1,1,1,0,1];
        S29 = [1,0,1,1,0,0]; S61 = [0,1,1,1,0,0];
        S30 = [1,0,1,1,1,0]; S62 = [0,1,1,1,1,0];
        S31 = [1,0,1,1,1,1]; S63 = [0,1,1,1,1,1];

"-------------------------------------------------------------------------

EQUATIONS
    reset = !MRES;

    !DOE = !DATAEN # (!WR & DBUSBSY);

    !DBUSBSY := (Pstate == S00) #
                (Pstate == S04) #                    "EPROM toff
                (Pstate == S16) #                    "EPROM burst toff
                (Pstate == S33) # (Pstate == S34) #   "DUART toff
                (Pstate == S42) #  (Pstate == S43);  "User toff

    !XRD := !RD & ((Pstate == S00) # (Pstate == S25) #
                   (Pstate == S26) # (Pstate == S27) #
                   (Pstate == S28) # (Pstate == S29) #
                   (Pstate == S30) # (Pstate == S31) #
                   (Pstate == S32));

    !XWR := !WR & ((Pstate == S00) # (Pstate == S25) #
                   (Pstate == S26) # (Pstate == S27) #
                   (Pstate == S28) # (Pstate == S29) #
                   (Pstate == S30) # (Pstate == S31) #
                   (Pstate == S32));

    !CENTRD := (Pstate == S44) # (Pstate == S45);

"-------------------------------------------------------------------------

END StateMach2
```

```
"B.Napaa    09/17/91

"EDIT HISTORY:
"Date       Engineer     Checksum Modification
"_________  _________    ________ ____________________________________
"09/17/91   B.Napaa               original release
"11/12/91   B.Napaa      CC82     modified the video fifo serial input clock VSOCP
"_____________________________________________________________________

MODULE videostate

FLAG '-R3', '-F'

TITLE '3051 Evaluation Board: Video state machine'

U100 DEVICE 'P22V10';

"_____________________________________________________________________

"Inputs
    SYSCLK      PIN 1;
    MRES        PIN 2;
    USERCS      PIN 3;
    RD          PIN 4;
    WR          PIN 5;
    MA20        PIN 6;
    MA21        PIN 7;
    MA22        PIN 8;
    MA23        PIN 9;
    CLKEN PIN 10;
    VIDCLK      PIN 11;
    STATE40     PIN 13;
    reset       node 25;

"_____________________________________________________________________

"Outputs
    EXACK PIN 14;
    ENGRD       PIN 15;
    ENGWR       PIN 16;
    FIFOWR      PIN 17;
    VC0         PIN 18;
    VC1         PIN 19;
    VC2         PIN 20;
    VC3         PIN 21;
    VSOCP PIN 22;


"_____________________________________________________________________

"Constants
    ON,OFF,T,F,X,C = 1,0,1,0,.X.,.C.;
    Madr = [MA23..MA20];
    Operation = [MRES,STATE40,USERCS,RD,WR,Madr];
    Idle    = [  1,    1,   1, X, X,X];
    Engrd   = [  1,    0,   0, 0, 1,9];
    Engwr   = [  1,    0,   0, 1, 0,9];
    Fifowr  = [  1,    0,   0, 1, 0,4];
```

```
Fifowr1  = [  1,    0,  0, 1, 0,5];
Fiford   = [  1,    0,  0, 0, 1,4];
Fiford1  = [  1,    0,  0, 0, 1,5];

Pstate = [VC3..VC0];
S00 = [1,1,1,1];
S01 = [1,1,1,0];
S02 = [1,1,0,0];
S03 = [1,1,0,1];
S04 = [1,0,0,1];
S05 = [1,0,0,0];
S06 = [1,0,1,0];
S07 = [1,0,1,1];
S08 = [0,0,1,1];
S09 = [0,0,1,0];
S10 = [0,0,0,0];
S11 = [0,0,0,1];
S12 = [0,1,0,1];
S13 = [0,1,0,0];
S14 = [0,1,1,0];
S15 = [0,1,1,1];


"-------------------------------------------------------------------
EQUATIONS
    reset  =     !MRES;

    VSOCP =      CLKEN & !VIDCLK;

"-------------------------------------------------------------------
STATE_DIAGRAM Pstate
    state S00:                                          "Idle
        case   Operation == Engrd:   S01 with
                                                        !EXACK:=T;
                                                        !ENGRD:=T;endwith;
               Operation == Engwr:   S04    with
                                                        !EXACK := T;
                                                        !ENGWR := T;
                                                        endwith;
               Operation == Fifowr:  S07    with !FIFOWR := T;
                                                        endwith;
               Operation == Fifowr1: S07    with !FIFOWR := T;
                                                        endwith;
               Operation == Fiford:  S12    with !EXACK:=T;endwith;
               Operation == Fiford1: S12    with !EXACK:=T;endwith;


        endcase;

    state S01:      !ENGRD:=T;          "engrd
                    goto S02;
    state S02:      !ENGRD:=T;
                    goto S03;
    state S03:      goto S00;

    state S04:      !ENGWR:=T;
                    goto S05;
```

```
state S05:      goto S06;
state S06:      goto S00;               "engwr

state S07:      !FIFOWR:=T;             "fifowr
                goto S08;
state S08:      !FIFOWR:=T;
                !EXACK:=T;
                goto S09;
state    S09:   !FIFOWR:=T;
                goto S10;
state S10:      goto S11;
state S11:      goto S00;

state S12:      goto S13;
state S13:      goto S14;
state S14:      goto S00;

"_____________________________________________________________

END videostate
```

Integrated Device Technology, Inc.

COMPILER TRADE-OFFS IN CODE DEVELOPMENT FOR THE IDT RISController™ FAMILY

APPLICATION NOTE AN-105

by Phil Bourekas

INTRODUCTION

An important part of system development involves the software development tool chain selected. The most appropriate tool chain depends on the end system application and on the desired development environment.

This application note attempts to differentiate between the MIPS® "CC" compiler toolchain, and the IDT/c™ toolchain. A discussion of the differences between these compilers, and appropriate toolchains for various types of applications, are also included.

In addition to the two compilers discussed here, a number of third party compilers applicable to IDT79R3051™ family development have recently been made available. While these compilers are beyond the scope of this applications note, many of the concepts contained here form a valid part of any analysis of those toolsets.

Note that although this applications note does contain some performance information, this information is intended for comparison of relative performance of various toolchain options. These results were not measured on an especially fast benchmark board, nor was any effort made to compare relative performance of different processors or systems.

MIPS COMPILERS: BACKGROUND

The MIPS compiler toolchain is a well-respected optimizing compiler suite, offering the highest levels of RISC performance. Unlike traditional microprocessor compilers, the MIPS compilers were actually originally developed prior to finalization of the microprocessor architecture. Thus, the compilers are able to fully leverage all of the capabilities of the MIPS processor family, and thus achieve the highest levels of optimizations.

A number of studies have shown that the MIPS compilers are the most efficient (across all microprocessor architectures) in obtaining the performance of assembly programming when compiling from high-level languages. The efficiency of the MIPS compilers serves to further the performance advantage of the R3051 family over competing RISC architectures.

Continuing development of these compilers is driven almost exclusively by the needs of the reprogrammable ACE and UNIX marketplaces. Feedback from common programs such as the SPEC benchmark suite serves to drive future enhancements.

Note, however, that reprogrammable systems tend to differ substantially from embedded systems. Specifically, many embedded systems choose to reduce cost by utilizing software (rather than specialized hardware) floating-point. Additional differences are found in the types of libraries required, whether or not the code is typically RAM or ROM resident, and in assumptions about the processor cache and main memory sizes (and memory latencies).

IDT/c: BACKGROUND

In order to directly address the needs of our embedded customers, IDT undertook an effort to develop an alternative toolchain. Rather than focus on competing with the performance of the MIPS compilers, we chose to focus on the various needs of embedded systems designers that were not well satisfied by the MIPS compilers. These needs include:

- Alternative host environments. Originally, the MIPS compilers were only available in native MIPS platforms. Embedded system designers requested a software platform resident on the same systems used for schematic entry, PCB layout, ASIC and PAL development, etc. Thus, a goal for IDT/c was to have it hosted on a wide variety of platforms, both UNIX® and DOS based.
- Efficient software floating-point emulation. The only methods available via the MIPS compiler suite were to either explicitly avoid FPA operations (by modifying the source code to directly call emulation routines), or to utilize a dynamic trap and emulate strategy which introduces significant overhead into the FPA emulation process. IDT/c takes a different approach: a compile time flag is available which will cause the compiler to emit calls to software emulation routines, rather then emit hardware FPA instructions. The result is a substantial performance gain during software FPA emulation, since the overhead of the processor exception mechanism, and the overhead of fully emulating the particular behavior of the R3010A FPA, is eliminated. Thus, IDT/c is a far more efficient compiler for applications using software emulation of floating-point operations.
- Alternative library options. By developing and maintaining our own toolchain, we are better able to integrate various library functions desirable in embedded systems into the toolchain.
- Better control of memory utilization. Since many embedded programs will be ROM resident with data references into a RAM area, IDT/c features linker control which enables binaries to be effectively partitioned according to the needs of the embedded designer.

The basis for our compiler toolchain was the GNU-c compiler, obtained from the Free Software Foundation.

This compiler is an ongoing effort by a number of software engineers at various industrial and academic locations. Thus, it features an effective optimizer as well as a high degree of portability across various UNIX operating systems. We used GNU to perform the front-end lexical analysis, parsing, and

high-level optimizations. We added to this a number of additional optimizations, including a pipeline scheduler suited to the particulars of the R3000 pipeline. In addition, we ported the compiler to non-UNIX platforms such as the PC.

TOOLCHAIN SELECTION CRITERIA

The appropriate toolchain is thus a function of:

- Toolchain host environment
- Development system cost goals
- Floating point content/Hardware or software floating-point
- Libraries/binary development options
- Performance goals

Toolchain Host Environment

The preferences for host environment are frequently dictated by the user's current CAD/CASE environment. It is obviously an advantage to use a software toolchain compatible with the in-place environment.

Note, however, that integrating new toolchain hosts into an existing environment can be relatively simple. For example, MIPS systems (which host the MIPS compilers) can be easily networked into an environment featuring other UNIX systems such as HP, IBM, or Sun. In addition, an X-Windows interface enables a MIPS system to be used as a host for dedicated X-Terminals, or for PCs operating as X-Terminals.

Development System Cost Goals

Similar to the availability of various hosts, the net cost of the development environment can also influence the choice of toolchains. The development system cost goals are influenced by the host availability, the cost of the tools, and the cost of other tools in the development process. For example, if an In-Circuit Emulator is to be used, a toolchain which is best integrated with that tool may be most appopriate, regardless of cost. On the other hand, if minimal development cost is a concern, a toolchain which functions on low-cost PCs may be most appropriate.

Floating Point Content/Hardware or Software Floating-Point

One of the advantages of the R3051 family is the ability to vary price performance in a single footprint. For example, the R3051 and R3052 are both footprint-compatible devices without hardware floating-point but with varied cache sizes; the R3081 adds a hardware floating-point and substantially increases the on-chip caches.

Depending on the application, the system designer may decide to use an integer only device such as an R3051 or R3052. In such a system, if some floating-point operations are required, they must be emulated via integer software.

In this case, IDT/c provides a clear advantage over the MIPS compiler. Table 1 shows the relative performance of a floating-point intensive program under various floating-point options. This table compares three different types of floating-point performance. The values in the table are in units of time.

- The column labeled "R3010" shows the results of a binary which issues hardware FPA instructions. This binary was then run in a system with actual hardware FPA support. The best performance is obviously obtained when an actual hardware floating-point accelerator is included in the system. This binary was generated by a MIPS compiler.
- The column labeled "R3010 Emulation" shows the same binary run in a system with no hardware FPA support. Significantly lesser performance is obtained when the compiler emits R3010A instructions, and the system then traps on these instructions to fully emulate the FPA. This binary was generated using a MIPS compiler.
- The last column shows the results for IDT/c. In this binary, the compiler is instructed to generate integer only code. Thus, whenever a floating-point operation is required, the compiler generates a call to the appropriate library routine to perform the function. Thus, the overhead of the trap exception mechanism, and the overhead of fully emulating all aspects of the R3010A, is bypassed, and higher software floating-point performance is achieved.

While it is unlikely that a user would opt for an integer-only solution to a problem as floating-point intensive as this, this benchmark does serve to illustrate the performance gain from the IDT/c approach to software floating-point emulation. In this benchmark, a significant performance improvement is seen when using the IDT/c approach over the R3010 emulation approach.

Table 1. Floating Point Emulation Performance(1,2,3)

Operation	R3010	R3010 Emulation	IDT/c
add.s	5	1345	25
sub.s	0	1340	25
mul.s	5	1320	25
div.s	10	1830	55
sin.s	55	23490	430
cos.s	55	23650	440
ln.s	40	19045	580
sqrt.s	60	10480	235
add.d	5	2290	35
sub.d	5	2310	40
mul.d	10	2315	50
div.d	10	2485	110
sin.d	55	23295	390
cos.d	55	23535	350
ln.d	45	18930	560
sqrt.d	70	10380	220

NOTES:

1. ".s" means single precision; ".d" means double
2. Results in "centi-seconds" using a 20Hz timer-tick. A "0" result indicates test completed prior to 1st tick.
3. Results indicate amount of time consumed by a particular floating point emulation test. Each test performs 10 floating point calculations per cycle loop. Results are for 10,000 loops.

Library and Binary Development Options

Another consideration has to do with the convenience of generating appropriate binary files for the given application. Specifically:

- Does one compiler offer particular libraries well suited to the application at hand?
- Is one toolchain better suited to mapping a binary into the various memory resources of the target application?

IDT offers a number of libraries, some of which are bundled with the IDT/c compiler. In addition, we have developed IDT/kit™ (kernel integration toolkit) and IDT/sim™ (system integration manager). Both packages contain numerous library routines targeted to runtime support and/or software integration. The data sheets for these products describe their capabilities in more detail.

In addition to these stand alone packages, the floating-point library discussed earlier provides an example of a library package which may influence compiler selection.

Beyond libraries, other considerations include the capabilities of the linker/loader, system download utilities, and symbolic debug capability. While both MIPS and IDT/c offer these capabilities, there are subtle differences in them which may further influence toolchain selection. Differences include:

- The MIPS linker attempts to place all code in one contiguous section of memory, and all data in another. The linker for IDT/c, on the other hand, allows the code and data segments of individual program modules to be separated, resulting in more control over the resulting memory system. This makes it simpler to place some code in ROM and some in DRAM, for example.
- IDT/c offers superior support for partitioning code and data into ROM and RAM areas. This support simplifies generation and download of binaries for the target system and PROM programmer.
- MIPS remote target symbolic debugger obtains more information from the symbol table, including more information on local variables.
- IDT/c symbolic debugger includes a full featured script environment/debug control language, allowing the user to more closely control debug activities such as variable watching and breakpoints.

Again, the user is encouraged to consult the data sheet for IDT/c and the various libraries as part of the toolchain evaluation.

Performance Goals

Depending on the application, one compiler or another may provide better performance. As illustrated earlier, IDT/c offers better performance in systems which use integer software to perform floating-point operations.

On the other hand, in an environment which utilizes hardware floating-point, or an environment which performs no floating-point, the application may realize a performance gain from utilizing the highly efficient MIPS compiler toolchain.

Table 2 illustrates the results of a set of benchmarks compiled by both the IDT/c and MIPS "C" compilers. These benchmarks are commonly referred to as "The Intel Benchmark Suite", since Intel introduced them to measure the performance of various embedded processors began when they announced the i960CA.

Table 3 may be more representative of the range of differences, as the Stanford Benchmark suite tends to exercise more of the processor.

The results indicate a variety of performance differences between IDT/c and MIPS "C" across these benchmarks. Note, however, that these benchmarks may not be fully representative of either compiler, as these benchrmakrs are extremely small, and feature integer only computation.

Note that the performance difference between these compilers is different across different hardware platforms. Specifically, the ability of the benchmark to remain cache resident will influence the performance gain of MIPS techniques such as procedure inlining and loop unrolling. Systems with differing cache sizes and/or memory latency may then show diffferent gains for these techniques.

Mix and Match Strategy

To maximize performance, a system designer could choose to use a "mix and match" strategy in the software toolchain. For example, the bulk of the application could be compiled using the MIPS compiler, while IDT/c is utilized in the floating-point intensive portions of the code.

This approach marries the best of both toolchains. The MIPS compiler extracts maximum performance from the majority of the integer only code, while IDT/c does the best job of performing floating-point operations in software.

Table 2. Compiler Results on Intel Benchmarks(1,2)

Benchmark	MIPS C	IDT/c
Anneal	5200	5340
BubbleSort	448	542
Dhrystone	38,461	35,714
MatMult	1920	2710
PI-500	1140	1540
QuickSort	392	477

NOTES:
1. Results measured on IDT7RS385 board at 33MHz with slow memory.
2. Results in units of time except for Dhrystone.

Table 3. Compiler Results on Stanford Benchmarks(1,2)

Benchmark	MIPS C	IDT/c
Perm	.059	.067
Towers	.061	.066
Queens	.039	.043
IntMatMult	.083	.089
Puzzle	.311	.396
QuickSort	.040	.047
BubbleSort	.044	.054

NOTES:
1. Results measured on IDT7RS385 board at 33MHz with slow memory.
2. Results in units of time.

IDT/c facilitates this approach by allowing IDT/c to use the MIPS backend assembler, thus allowing code generated by IDT/c to be directly linked with code generated by the MIPS compiler. Thus, the programmer can use IDT/c (with the MIPS backend assembler) on the floating-point intensive code, and the MIPS compiler on the rest of the code.

Table 4 illustrates the performance gain achievable when using such a mix and match strategy. In this table, two of the Stanford Benchmarks are shown with an IDT/c only, and with a mix and match strategy. In order to run these benchmarks, the programmer would either have to use the runtime FPA emulation strategy described earlier, or would have to use IDT/c to compile the floating-point portions of the code.

Table 4. Benefits of Mix and Match Compiler Strategy(1,2)

Benchmark	IDT/c	Mix and Match
Mn	.277	.267
FFT	.327	.303

NOTES:
1. Results measured on IDT7RS385 board at 33MHz with slow memory.
2. Results in units of time.

SUMMARY

The compiler toolchain appropriate to a given system development is an extension of the price-performance decisions of the system itself. For the IDT RISController family, various strategies may be appropriate for different systems.

MIPS compiler is shown to remain much more effective in generating efficient code than is the GNU compiler. On the other hand, the IDT toolchain is better at performing software floating-point emulation, is better at allowing memory control when linking, and offers a target debugger with the ability to write debug "scripts". Finally, the availability of the toolchain on the host development environment may also influence the toolchain selection.

This applications note summarizes some of the considerations, and presents some data to help facilitate toolchain evaluation.

Integrated Device Technology, Inc.

CONSIDERATIONS WHEN BENCHMARKING WITH THE IDT7RS385™ EVALUATION BOARD

APPLICATION NOTE AN-107

By Samuel Y. Shen

INTRODUCTION

The IDT7RS385 is a complete RISC system intended for use as a low-cost porting target for applications of the IDT79R3051™ RISController™ family. It is completely self-contained on a single printed circuit board and only requires a simple CRT terminal for operation. An IBM PC, IDT MacStation™, SPARCstation™, or a MIPS® workstation can be connected to the 7RS385 via one of the serial ports and user developed code can be downloaded to the board. In addition, a wire wrap area, expansion connectors and hardware debugging facilities are provided allowing the user to easily prototype additional circuitry. This board serves as a generic example for the construction of an R3051 system.

However, the 7RS385 was never intended to be used as a comparative benchmark vehicle. The design goals for the board required a low-cost platform on which various pieces of software could be developed and debugged. The board does not utilize SRAM, fast DRAM, or techniques such as memory interleaving, which are typically used only to maximize performance.

This applications note describes factors which must be considered if a user attempts to run a benchmark on this board, and also attempts to give some rules of thumb with respect to how a different design may perform. In addition, the applications note presents routines used to manipulate the on-board timer, so that time measurements can be accurately taken.

7RS385 BACKGROUND

The 7RS385 was originally designed as a low-cost platform for R3051 family software. Thus, the design features the following characteristics:

- Easily modified to higher frequency by adding wait states in the memory system. Although this helps satisfy the goals of a test platform, the end result is more synchronization stages than would otherwise be necessary at a given (lower) frequency, such as 25MHz. Thus, the frequency normalized performance (throughput) of the board is lower than would be expected for a design optimized for a given target frequency.
- Monitor program to easily debug software. The board is shipped with IDT/sim™, a program which allows the debug of other applications running on the board. In order to implement debug, IDT/sim makes choices that adversely affect performance. For example, IDT/sim runs uncached, so that it does not interfere with the application being debugged. Similarly, IDT/sim features a large number of indirect references, to allow user supplied exception handlers and I/O routines. The exception handlers have been designed for application debug, and therefore store more state information (in multiple locations) than would otherwise be done. In addition, the exception handlers reside in uncached memory, because the "breakpoint" handler is among the exceptions handled. Thus, using the exception handler provided by the debug monitor simplifies application debug, but has a major impact on various benchmark metrics.

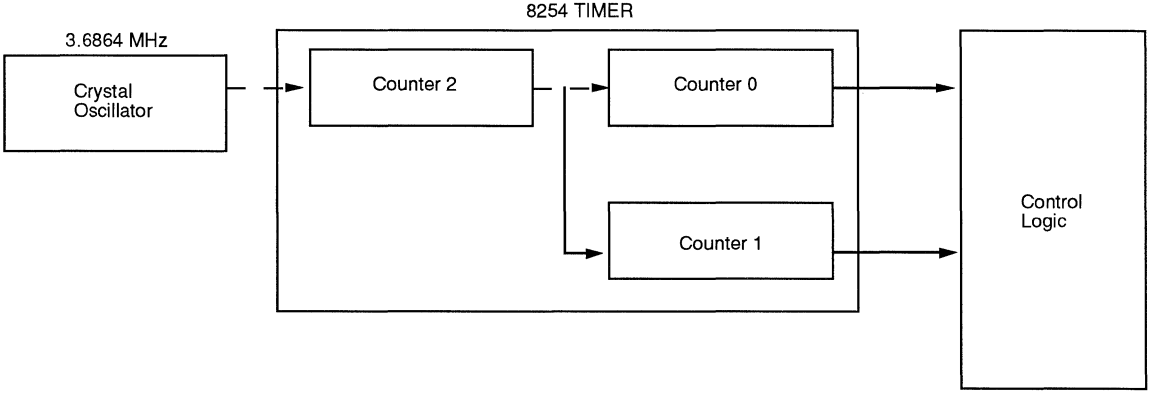


Figure 1. Block Diagram of Timer Function on 7RS385

Hex Addr	Binary Addr A(7:0)	Function	
		Read	Write
1F80 0000	- - - 0 0 0 - -	R counter 0	W counter 0
1F80 0004	- - - 0 0 1 - -	R counter 1	W counter 1
1F80 0008	- - - 0 1 0 - -	R counter 2	W counter 2
1F80 000C	- - - 0 1 1 - -	No OP	W ctrl reg
1F80 0010	- - - 1 0 0 - -	RST cint#0	No OP
1F80 0004	- - - 1 0 1 - -	RST cint#1	No OP

Figure 2. 8254 Addressing on 7RS385

- Low cost. The 7RS385 was designed to be an low-cost software platform. This precludes the use of SRAM, the use of techniques such as memory interleaving, and the use of high-speed memory and logic components. Thus, the frequency normalized performance of the board is lower than what would be typically implemented as an end customer product.

USING THE ON-BOARD TIMER

The 7RS385 includes an 8254 timer on board. This timer can be used to measure the actual execution time of a program running on the board. However, in order to use the timer in this fashion, additional software must be included.

About the 8254

In order to time processes on the '385 board, runtime timer functions need to be accessed to tell the execution time. In the '385, there are two programmable timers/counters. Each of the two timers may be programmed as an independent real-time interrupt occuring at regular intervals. They are implemented in the Intel 8254 timer device on the '385 system board. The arrangement of the timers of the 8254 is outlined in Figure1. Also, the address decode table of the 8254 on the '385 is shown in Figure 2.

There are three 16-bit counters in the 8254, designated as counter 0, 1, and 2. The arrangement used in the '385 is for counter 2 to be a pre-scalar for each of the other counters (counter 0 and counter 1). Counter 2, in turn, is clocked by a crystal oscillator, running at 3.6864MHz (i.e., 271ns per clock cycle). Each counter in the 8254 is capable of handling clock inputs up to 10MHz. Also, each counter has six programmable counter modes. All modes are software programmable. Counters are programmed by writing a control word and then an initial count. The control words are written into the control word register while the initial counts are written into the counters.

The listings for timer functions are shown below. These functions may be compiled separately and linked with the application programs to run out of DRAM on the '385 board.

The first function is an "include" file which sets up the initial values for the separate counters/control registers and assigns name to counters/general registers. The include file expresses these registers as a base address for the register, plus a 3-byte offset ("+3") due to the big-endian nature of the '385. Byte 3 of a word is always the least significant (rightmost) byte. The data path connected to the 8254 timer is on D(7:0), which is the least significant byte; therefore the byte address for on-chip registers will be the word address +3.

```
/***********************************/
/* RS385  (8254 Timer Routine)     */
/* idttimer.h                      */
/***********************************/
#define CONTROL          0xbf80000c+3    /* counter control register */
#define COUNTER1         0xbf800004+3    /* 8-bit counter 0 register */
#define COUNTER2         0xbf800008+3    /* 8-bit counter 2 register */
#define INITIALV         0x000000ff      /* initial value */
#define INITIALV1        0x00000002      /* temporary value */
#define INITIALV2        0x00000000
#define CW_COUNTER1      0x00000070      /* control word for counter 1 */
#define CW_COUNTER2      0x000000b4      /* control word for counter 2 */
#define CL_COUNTER2      0x00000080      /* counter latch for counter 2 */
#define CL_COUNTER1      0x00000040      /* counter latch for counter 1 */
#define TRUE             0x00000001      /* true value */
#define ZERO             $0              /* wired  zero */
#define v0               $2              /* return value */
#define v1               $3
#define t1               $9              /* temporary register */
#define sp               $29             /* stack pointer */
#define ra               $31             /* return address */
```

File 1. Timer Include File

```
/*********************************/
/*  RS385 (8254 Timer Driver)      */
/*  idtTstart.s                    */
/*********************************/
#include "idttimer.h"
        .globl          TimerStart
        .ent            TimerStart
        .set            noreorder
TimerStart:
        subu            sp,24
        sw              ra,20(sp)
        .mask           0x80000000, -4
        .frame          sp,24,ra
        li              t1,CW_COUNTER2  /* 1011 0100 = select counter2, r/w LSB(byte) */
        li              v1,CONTROL      /* then MSB, mode2, binary counter 16-bit */
        sb              t1,0(v1)
        li              t1,INITIALV1    /* 0000 0002 = counter2 LSB */
        li              v1,COUNTER2
        sb              t1,0(v1)
        li              t1,INITIALV2    /* 0000 0000 = counter2 MSB */
        li              v1,COUNTER2
        sb              t1,0(v1)
        li              t1,CW_COUNTER1  /* 0111 0000 = select counter1, r/w LSB(byte) */
        li              v1,CONTROL      /* then MSB, mode 0, binary counter 16-bit */
        sb              t1,0(v1)
        li              t1,INITIALV     /* 1111 1111 = counter LSB */
        li              v1,COUNTER1
        sb              t1,0(v1)
        li              t1,INITIALV     /* 1111 1111 = counter MSB */
        li              v1,COUNTER1
        sb              t1,0(v1)
        li              t1,CW_COUNTER2  /* 1011 0100 = select counter2, r/w LSB (byte) */
        li              v1,CONTROL      /* then MSB, mode2, binary counter 16-bit */
        sb              t1,0(v1)
        li              t1,INITIALV     /* 1111 1111 = counter LSB */
        li              v1,COUNTER2
        sb              t1,0(v1)
        li              t1,INITIALV     /* 1111 1111 = counter MSB */
        li              v1,COUNTER2
        sb              t1,0(v1)
        li              v0,ZERO

        .set            reorder
        addu            sp,24
        j               ra
        .end            TimerStart
```

File 2. Initializing the 8254

File 2 is a timer-start routine example. The 8254 modes used in this example were mode 2 and mode 0. Mode 2 is a "rate generator" mode. It functions like a divide-by-n counter and has been used in counter 2, the pre-scalar counter. Mode 0 is an "interrupt on terminal count" mode, which is typically used for event counting. This was implemented in counter 1. The timer-start routine starts with selecting counter 2, defining it to be a 16-bit binary counter, and setting up r/w as LSB then MSB order. Value 2 and value 0 chosen to be put into LSB and MSB field is because counter 1 (in mode0) needs to be activated by a clock input which happens to be the output of counter 2. Therefore, a small number in counter 2 is required to drive the output low for one cycle. After that, an initial value was put into the counter 1 and counter 2 to start counting. "FFFF" was used as an initial value in this example.

The following programs serve the timer-request functions, which try to read from the 16-bit counter 2 and counter 1. The 8254 "counter latch" command was used. This command allows reading the contents of the counters "on the fly" without affecting counting in progress. Two counter latch commands were used here to latch counter 2 and counter 1. Each counter needs to be latched twice. Again, LSB comes first then MSB.

```
/***********************************/
/*  RS385 (8254 Timer Driver)      */
/*  idtTreq11.s                    */
/***********************************/
#include "idttimer.h"
          .globl          Timer11Req
          .ent            Timer11Req
          .set            noreorder
Timer11Req:
          subu            sp,24
          sw              ra,20(sp)
          .mask           0x80000000,-4
          .frame          sp,24,ra
          li              t1,CL_COUNTER1
          li              v1,CONTROL
          sb              t1,0(v1)
          li              v1,COUNTER1
          lbu             v0,0(v1)
          .set            reorder
          addu            sp,24
          j               ra
          .end            Timer11Req
```

```
/***********************************/
/*  RS385 (8254 Timer Driver)      */
/*  idtTreq21.s                    */
/***********************************/
#include "idttimer.h"
          .globl          Timer21Req
          .ent            Timer21Req
          .set            noreorder
Timer21Req:
          subu            sp,24
          sw              ra,20(sp)
          .mask           0x80000000,-4
          .frame          sp,24,ra
          li              t1,CL_COUNTER2
          li              v1,CONTROL
          sb              t1,0(v1)
          li              v1,COUNTER2
          lbu             v0,0(v1)
          .set            reorder
          addu            sp,24
          j               ra
          .end            Timer21Req
```

```
/***********************************/
/*  RS385 (8254 Timer Driver)      */
/*  idtTreq1h.s                    */
/***********************************/
#include "idttimer.h"
          .globl          Timer1hReq
          .ent            Timer1hReq
          .set            noreorder
Timer1hReq:
          subu            sp,24
          sw              ra,20(sp)
          .mask           0x80000000,-4
          .frame          sp,24,ra
          li              v1,COUNTER1
          lbu             v0,0(v1)
          .set            reorder
          addu            sp,24
          j               ra
          .end            Timer11Req
```

```
/***********************************/
/*  RS385 (8254 Timer Driver)      */
/*  idtTreq2h.s                    */
/***********************************/
#include "idttimer.h"
          .globl          Timer2hReq
          .ent            Timer2hReq
          .set            noreorder
Timer2hReq:
          subu            sp,24
          sw              ra,20(sp)
          .mask           0x80000000,-4
          .frame          sp,24,ra
          li              v1,COUNTER2
          lbu             v0,0(v1)
          .set            reorder
          addu            sp,24
          j               ra
          .end            Timer11Req
```

File 3. Timer Driver Routines

Using the Timer Routines to Time Execution

The C program shown below is an example of calling timer routines from C. First of all, these four timer routines are declared to be the external functions. Then the return value of each functions is assigned to an integer variable. These variables will be used to calculate the execution time.

```
/* Sample "C" Program Using Timer */
extern    TimerStart();
extern    Timer1lReq();
extern    Timer1hReq();
extern    Timer2lReq();
extern    Timer2hReq();
main()
{         int              i, j, k0, k1ls;
          int              k1hs, k2ls, k2hs;
          int              k1le, k1he, k2le;
          int              k2he,result;
          k0 = TimerStart();
          k1ls = Timer1lReq();
          k1hs = Timer1hReq();
          k2ls = Timer2lReq();
          k2hs = Timer2hReq();
          :
          : (main body of this program)
          :
          k1le = Timer1lReq();
          k1he = Timer1hReq();
          k2le = Timer2lReq();
          k2he = Timer2hReq();
          :
          : (print out the timer result)
}
```

File 4. Using the Timer Routines

SOFTWARE ISSUES AFFECTING BENCHMARK RESULTS

The software environment shipped with the 7RS385 was designed to enable the debug of code downloaded and executed on the board. Thus, the 7RS385 includes in its on-board PROMs a version of IDT/sim.

IDT/sim (System Integration Manager) is a set of routines which builds an extensible PROM monitor environment, and which also provides routines for system debug. An analogous piece of software, IDT/kit™, is the software used when the system developer wishes to utilize pre-written library functions in the end product.

However, when running benchmarks on the 7RS385, it is obviously tempting to utilize the library functions included on board, in IDT/sim. Such a decision, however, has a seriously adverse impact on the benchmark results obtained. This section of the applications note describes some common problems encountered when benchmarking on top of the IDT/sim environment. Note that IDT/c™ will automatically default to IDT/sim routines, unless other library routines are explicitly provided.

IDT/sim Cache Utilization

The IDT/sim (system integration manager) provides a range of standard entry point C functions for the IDT79R3051 family. These entry point functions are standard C-type functions. There are formatted printing, string manipulation, standard I/O, conversion routines, etc. These functions are provided to allow the system developer to:

- Include print statements in line with code to print debugging messages.
- Debug key algorithms in advance of completing library development.
- Use existing interrupt handlers during debug.
- Incrementally replace parts of the debug environment with the applications own code, as it is developed.

The IDT/sim routines are shipped in the on-board EPROMs, which are also used to hold the boot code. This results in two benchmarking problems:

- These routines are stored in the EPROMs which are mapped to kseg 1, an uncached space. Thus, using these routines do not take advantage of the large on-chip caches of the R3051 family. This also impacts measurements of exception response.
- Single word reads and the burst reads from the EPROMs are much slower than from the DRAMs. A single word read from the EPROMs takes 8 cycles while a single word read from the DRAMs takes 6 cycles. Quad word reads from the EPROMs takes 26 cycles while quad word reads from the DRAMs takes 15 cycles (refer to 7RS385 user's manual; this data is for a 33MHz 7RS385).

Thus, benchmarks linked to and run with IDT/sim produce results dramatically slower than would be achieved in a real application environment.

An example of this problem can be found from one of the benchmarks in the suite introduced by Intel when they introduced the i960CA. The pi-500 program calculates the value of the mathematical constant pi up to 500 decimal points using iterative integer calculations. As a new value is calculated, its result is printed to the terminal. This benchmark uses a large number of printf function calls; as printf is provided by IDT/sim, a serious (and non-representative) performance degradation occurs. The execution times of the program with printf and without printf, when run on the '385 are quite different: 1084ms versus 1624ms (>50%). To avoid this problem, move entry point functions into the DRAM, and use cached accesses. For pi-500, the resulting execution time is 1086 ms.

Similarly, the Dhrystone benchmark makes heavy use of the strcpy and strcmp functions. A common mistake would be to link to those functions in the IDT/sim in the on-board PROMs. The result may be less than 10,000 dhrystones per second, due to the uncached use of the long latency EPROM memory for much of the execution. Approximately 4–5 times the performance, on the same board and with the same source code, is achieved when the strcpy and strcmp is linked from IDT/kit, and resides in cacheable DRAM memory.

Exception Response Issues

The R3051 family exception handling system efficiently handles machine exceptions, including TLB misses, arith-

metic overflows, I/O interrupts, system calls, breakpoints, reset, and co-processor unusable conditions. Any of these events interrupt the normal execution flow; the R3051 family aborts the instruction causing the exception and flushes the pipeline of subsequent instructions, thus not modifying processor context. The R3051 then performs a direct jump into a designated exception handler routine (bfc0_0000 for reset, and normally 8000_0000 for uTLB and 8000_0080 for general exceptions, if BEV=0. However, uTLB and general exceptions can be moved to bfc0_0000 and bfc0_0180 if BEV=1).

The 7RS385 is designed to have BEV=0, which tries to use cacheable memory for exception handling. However, the exception handlers and decoding of the exception cause still occurs in the IDT/sim EPROMs; the DRAM merely contains a few store instructions (for later debugging information) and a jump into the EPROM (uncacheable) memory space. File 4 illustrates the way in which IDT/sim utilizes the DRAM for exception handling. Notice the branch back to the uncached EPROM space. This structure was included to allow the cache state of a program to be preserved when the "breakpoint" exception is signalled.

If a user chooses to measure exception or interrupt latency by installing his own interrupt handler into IDT/sim, a serious performance degradation will occur. This is due to three factors: the extra overhead of the debug information and branch in the DRAM memory, the longer latency of the EPROM, and the uncached nature of the exception cause decode and interrupt handler software. Thus, installing interrupt handlers into IDT/sim to measure exception response will not give a true reading of the processor's performance.

CACHING THE STACK

In order to obtain representative performance, the user of the 7RS385 must explicitly manage the location of the runtime stack. IDT/sim defaults to an uncached location for the runtime stack, since it may explicitly manipulate the stack at various times. Thus, with the IDT/sim environment of the 7RS385, it is possible to have a program and its associated data reside in cacheable memory, but to have the runtime stack be uncacheable. In certain benchmarks, the performance penalty which results can be significant. This is exacerbated by the recursive nature of many common benchmarks, such as Towers of Hanoi, which will make extensive use of the stack in passing parameters.

To fully understand the performance degradation which can occur, it is important to understand the role of the stack during execution.

The compilers (IDT/c or MIPS/c) classify each routine into one of the following two categories:

- non-leaf routines; that is, routines that call some other routines.
- leaf routines; that is, routines that do not themselves execute any procedure calls. Two types of leaf routines exist: leaf routines that require stack storage for local variables, and leaf routines that do not require stack storage for local variables.

For leaf procedures that use the stack, or for non_leaf procedures which need to preserve registers, stack space must be allocated for the routine's requirements. These requirements include local variables, saved general registers, saved floating point registers, and procedure call argument area (also see Figure 3).

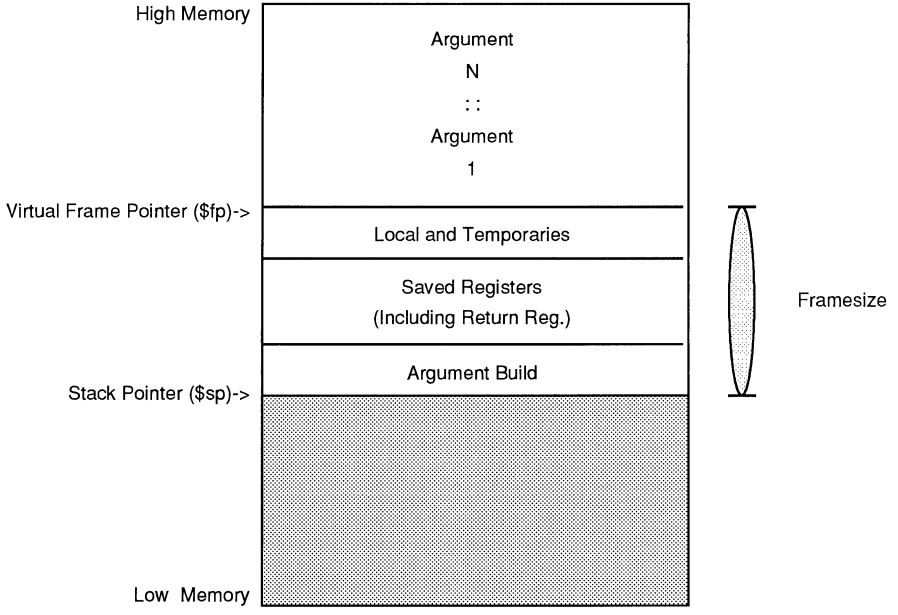


Figure 3. Runtime Stack

```
                (uTLB Miss)                                    (General Miss)

8000 0000:    lui   k0,0xa000                  8000 0080:    lui   k0,0xa000
8000 0004:    ori   k0,k0,0x178                8000 0084:    ori   k0,k0,0x178
8000 0008:    sw    at,0x4(k0)                 8000 0088:    sw    at,0x4(k0)
8000 000c:    sw    gp,0x70(k0)                8000 008c:    sw    gp,0x70(k0)
8000 0010:    li    v0,0x2                     8000 0090:    li    v0,0x1
8000 0014:    lui   at,0xbfc0                  8000 0094:    lui   at,0xbfc0
8000 0018:    ori   at,at,0x620                8000 0098:    ori   at,at,0x620
8000 001c:    jr    at                         8000 009c:    jr     at

/* jr branches to bfc0_0620, uncached EPROM */
```

File 4. Exception Handler Code in 7RS385

On a '385 board, the default address of the stack depends on the runtime environment, rather than on the main program. Unless the user explicitly sets up a stack, IDT/sim will allow the application to use the default stack it set up. Thus, even thought the program and its data area are cacheable (e.g. kseg 0), if the system environment defaults to kseg 1, the runtime stack remains uncached when the benchmark is executed.

This cached/uncached stack issue sometimes causes a big difference in execution time. For example, the anneal program (another benchmark promoted by Intel when they announced the i960CA), calculates the shortest distance between two points. This benchmark shows dramatically different results, depending on whether the runtime stack is cached. Execution times for cached and uncached stacks are 5.14sec and 6.81sec separately (>32%).

This situation can be handled by various methods. One way around this is to issue the IDT/sim 'seg -0' command before the benchmark is executed. This will force the runtime environment to change to kseg 0, a cached space. By doing this, the stack will be automatically cached accessed. Another method is to change the startup code so that the stack will be located in the same segment as the main program's. A sample program is shown in File 5.

MEMORY LATENCY ISSUES

As discussed above, the 7RS385 was never intended as a high-performance board for benchmarking, but rather as a low-cost, flexible design for software porting and testing. Thus, the memory characteristics of the board are not particularly optimized for performance; specifically, the frequency normalized performance of the board is less than would be expected of any dedicated application board, due to extra synchronization stages, the lack of interleaved memory support, and the particular partitioning of memory control

There are two memory areas on the 7RS385. The EPROM memory, described above, is typically referenced as uncacheable memory, and requires relatively long latency when accessed. Techniques such as burst EPROMs were specifically avoided, to avoid the high cost of these Intel-proprietary EPROMs. Similarly, high-speed EPROMs were not used, to reduce end cost.

The main DRAM memory used on the '385 uses 80ns DRAMs, in a non-interleaved, single bank configuration. Two 22v10 and one 16R8 PLDs make up the DRAM controller. This simple controller handles all DRAM accesses as well as refresh requirements, but is not aggressively optimized. Table 1 summarizes all DRAM access times. These times are

```
(initialization)
25      la      t2,_fbss          /* main program's mode as linked, usually cached mode */
26      and     t2,0xf0000000     /* isolate segment */
27      la      t1,6f             /* load label 6's address into t1 */
28      or      t1,t2             /* or with the main program's mode*/
29      j       t1                /* back to the original mode */
30      nop
31 6:   or      v0,t2             /* stack back to original mode */
32      addiu   v0,v0,16          /* overhead */
33      move    sp,v0             /* now replace count w top of memory */
34      move    v1,v0
35      subu    v1,STACK_SIZE
: (clear STACK_SIZE stack)
```

File 5. Start-up Code to Insure Stackable Cache

especially slow when compared to other alternatives such as interleaved memory. In addition, since the design was not particularly optimized for any given frequency, it is slower than would be expected of alternative designs of roughly equivalent complexity.

Table 1. DRAM Characteristics of 7RS385 Board

DRAM R/W	RS385-33	RS385-25
Single read	6 cycles	5 cycles
Quad read	6/3/3/3 cycles	5/2/2/2 cycles
Idle write	5 cycles	5 cycles
Page write	4 cycles	4 cycles

COMPILER OPTIMIZATION ISSUES

Another area for performance increase has to do with the amount of compiler optimization done. A separate applications note discusses some of the trade-offs involved in selecting an appropriate compiler. In addition to those considerations, the amount of optimizations enabled during a particular compilation may effect the results of the benchmark.

Figure 4 illustrates the MIPS compiler system. The compiler processes one procedure at a time. Large procedures offer more opportunities for optimization, since more interrelationships are exposed in terms of constructs and regions.

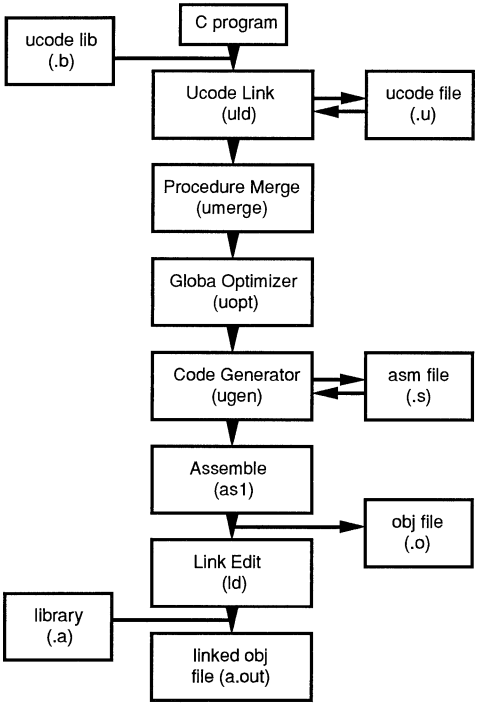


Figure 4. MIPS Optimizing Compiler Flow

The uload and umerge phases of the compiler toolchain permit global optimization among separate units in the same compilation.

Often, programs are divided into separate files, called modules or compilation units, which are compiled separately. For C programs in MIPS system, cc (c compiler) both compiles and link edits. The typical user seldom invokes ld (link editor) directly, unless the user wants to compile modules separately then link them together and download to some specified address. This is what we do where benchmarking on the '385 board. Programs need to be downloaded to user space which is somewhere between 0xa000 46a4 and 0xa00 ffffc (for 1MB DRAM w/ IDT/sim 3.1). Any place above this region will cause a data bus error to be signalled, since this is the top of RAM. Also, any place below this region will have an exception handling problem. This is because the exception handler and BSS for IDT/sim are located in the bottom of the DRAM.

Since the link loader does not have an optimization phase, all the optimization is done in the compiler. If modules are compiled separately, there is no way to do interprocedure register allocation or procedure merging (optimization levels 3 or 4). This means that in order to download programs to '385, ulink and umerge phases have to be bypassed, and only the global optimizer (uopt) phase executes. It performs optimization only within the bounds of individual compilation units (optimization levels 1 and 2).

The inability to use all of the compilation optimizations available can seriously degrade reported performance (–O3 and –O4 optimization levels can improve performance of some benchmarks by 10–20%). These limitations are not applicable when building a real application for a target system, and thus downloading benchmarks to the 7RS385 will underreport the results the designer should really expect.

Table 2 illustrates the variance in performance due to compiler selection and choice of optimization. As can be seen, on some benchmarks, the difference between the MIPS compiler at full optimization vs. the IDT/c compiler can be as much as 50% of performance. Additionally, the difference between the MIPS compiler at "–O2" and "–O4" optimization can be over 10%. Unfortunately, the "Intel benchmarks" may not be fully indicative of the performance gain of aggressive

Table 2. Results from Different Compilers

BENCHMARK	IDT/c "–O"	MIPS "–O2"	MIPS "–O4"
Quicksort (ms)	50.8	38.6	35.5
Bubblesort (ms)	54.3	44.9	40.9
PI–500 (ms)	1355	1039	1015
Anneal (ms)	5,149	5,193	5,115
MatMult (ms)	29,276	19,619	19,599
Dhrystone 1.1 (dhrystone/sec)	35,714	46,356	49,116

NOTES:
MIPS results from Cache-3051 simulator.
PI-500 results with "Printf" disabled.
IDT/kit used for library functions.
Benchmark sources obtained from Intel.

optimization, since most of the applications remain cache resident even with small caches.

PROPER BENCHMARKING

IDT has a software simulator tool, Cache-3051™, which bypasses many of these limitations, and in fact allows benchmarks to be run in a number of proposed target environments. Although it is a simulation, various studies have proven it to be a highly accurate tool. There is a separate applications note which measures the accuracy of the simulator by running benchmarks on the 7RS385, and then simulating the same memory characteristics. These results show that typical simulations are within 2% accuracy.

The Cache-3051 applications note further describes more of the capabilities of the tool, and is available from your local IDT sales representative.

If benchmarking the 7RS385 is desired, the user should check to insure or compensate for the following:

- Is the timer being used properly? Often, a simple program to double check the use of the timer can insure that it is being manipulated properly.
- Are library functions being provided through cacheable references from DRAM? Unless explicit care is taken, the benchmark may be using uncacheable debug routines out of slow EPROM, thus seriously degrading performance.
- Is the exception response model reasonable? If the benchmark builds ontop of IDT/sim, uncacheable exception handlers will be used. In addition, extraneous store information, appropriate for debug but not for benchmarking, will be generated, seriously degrading exception response.
- Is the runtime stack cacheable? Unless explicit steps are taken, the benchmark may run with an uncacheable runtime stack, seriously degrading application performance. There are two methods to insure the stack is cacheable: issue a 'seg -0' command in IDT/c, or have the program explicitly initialize the stack pointer into the cacheable memory area.
- Is the memory latency of the benchmark board representative of the target application? The RS385 features memory latency typically longer than would be expected from a real application board. The Cache-3051 cache simulator can help determine the performance of the benchmark in a more reasonable environment.
- Are the compiler optimizations representative of how the end application would be developed? A common mistake is to use lower levels of optimzation in a benchmark than would otherwise be done. This can result in a 30% performance degradation.
- Is the benchmark truly representative of the application code. Typically, benchmarks are small programs, designed to be simple to develop and port. Such small programs may not fully exercise either the optimization capabilities of the compiler, nor may they fully exploit the advantages of the large caches available in the R3051 family.

CONCLUSION

A number of factors combine to render the 7RS385 an inappropriate choice for benchmarking the capabilities of the 79R3051 family. These factors include the software environment of the board, the limitations imposed on the compiler, and the memory design of the board.

The 7RS385 was originally designed as a software test and porting vehicle, and does a very good job of providing services for this use. However, these requirements in general conflict with the use of this board as a performance evaluation vehicle, and thus the 7RS385 should not be used for this purpose.

IDT has a performance profiling toolchain, Cache-3051, which does a better job of predicting the performance of a R3051 processor on a given piece of software. This simulator has been shown to be highly accurate, and should be the preferred method for benchmarking. In addition, IDT is able to offer support during the benchmark process, to insure that the results obtained are truly representative of the capabilities of the processor.

Integrated Device Technology, Inc.

USING Cache-3051™ FOR SYSTEM PERFORMANCE EVALUATION

APPLICATION NOTE AN-108

By Samuel Y. Shen

INTRODUCTION

IDT offers a performance profiling tool, Cache-3051™, which allows the system designer to accurately measure the various price-performance tradeoffs available. This program allows the system architect to measure the effects of memory latency, cache size, and various memory control strategies before a final hardware design is committed.

This applications note describes the various capabilities of the simulator. In addition, it demonstrates the accuracy of the tool, and describes areas of the system modeled.

Cache-3051 BACKGROUND

The IDT Cache-3051 allows the designer to analyze the performance of the simulated IDT79R3051™ family system by executing a designer's application program on the proposed system. The modeler is derived from an earlier software package, cache2000, which is part of the systems programmers package (SPP) developed by MIPS computer systems,

Cache-3051 models "cacheable" memory references; it does not works for uncacheable references, which are typically only used for I/O and boot code. In order to model system performance, the program analyzes the memory references made by an application program during its execution and generates various statistics about its dynamic behavior. Cache-3051 determines the execution time taken by the user's application program by simulating the latencies involved in accessing the memory; that is, it models the amount of time spent doing memory references.

Note, however, that there are other events within the processor which affect execution time. For example, an address trace analysis will not include the effects of processor stalls which are not due to memory references. Thus, Cache-3051 does not take into account interlock cycles of the CPU or the FPA. These interlock cycles can be determined from the output of Pixstats (a software tool for interlock cycle analysis), so that a final system performance number is determined.

The main memory model simulated is page-mode, and the latencies associated are changeable. By simulating different memory subsystems with Cache-3051, the user can determine the performance of the application program on those systems and can arrive at an optimal solution.

PARAMETERS IN Cache-3051

Cache-3051 models all of the parameters typically under the control of the system designer when implementing an R3051-based system. In addition, the simulator can model the performance differences among the various members of the R3051 family in a given system and for a given application; that is, the user can alternately model the cache sizes of an R3051, an R3052, or an R3081™, along with the various memory speed parameters under his control.

The memory model includes all of the various types of latencies found in a DRAM system: the initial read latency cycles, number of cycles to perform a read/write operation when the DRAM is in page-mode, number of cycles to perform a write operation when the DRAM is not in page-mode. The parameters in Cache-3051 are set to user values to model a DRAM system that works at some user selected frequency.

"read_latency"

The number of cycles between the read signal asserted and the end of the CPU fixup cycle.

"idle_write_time"

The number of cycles to retire a word from the write buffer to an idle memory system (RAS and CAS are inactive).

"page_write_time"

The number of cycles to retire a write when the DRAM is in page-mode. That means the two consecutive writes have the same row address.

"non-page_write_time"

The number of cycles to retire a write when the DRAM is in page-mode, but the write can not be processed as page-mode. That means the two consecutive writes have different row address. Therefore the RAS signal has to be driven HIGH to precharge, prior to the write occurring.

"byte_extra_write_time"

The extra cycles to retire a partial write from write buffer to memory. In R3051/52 systems, this parameter is set to be 0. (This parameter is intended for ECC systems, which process partial writes as page-mode read-modify-write sequences.)

"ras_precharge_time"

The number of cycles to precharge the RAS signal. (This number is not the one defined in DRAM data books.)

"throttled"

A flag for quad-word read: 1 for throttled, 0 for burst. In a burst read, each word is presented to the CPU at its clock rate; in a throttled read, multiple clock cycles per word are required.

"throttled_latency"

The summation of the bus delay cycles between any two consecutive read operation during the throttled read. This parameter is ignored when "throttled" is false (zero).

RS385 BACKGROUND

The IDT evaluation board is a single-board test platform for the R3051 family. The board was initially designed to be a simple test platform for the CPU silicon, and for various software programs.

Given its heritage as a test platform, the board is designed to allow multiple wait-states to be inserted into memory in order to allow the processor to be run at higher frequency. Thus, the board reflects an extremely conservative, low-cost board design philosophy, and is not particularly intended as a benchmarking vehicle. A separate applications note describes the use of this board. However, due to its widespread availability this board was selected to be used as the platform for validating the accuracy of Cache-3051.

A final note of history is appropriate when evaluating the results of benchmarking on this board. Originally, boards shipped to customers used a 33MHz R3052. In order to support this system speed, additional memory wait states were included in the DRAM state machine.

More recently, boards shipped to customers feature a 25MHz processor and memory system. The lower frequency system also features reduced latency to memory (as measured in processor clock cycles) over the original 33MHz board, and thus offers higher frequency normalized performance.

RS385-33 MEMORY LATENCY

Two 22V10s and a 16R8 are used to implement DRAM control on the RS385 board. This chip set handles all DRAM accesses as well as refresh requirements. Page-mode accesses are supported utilizing the Burst/WrNear output from the R3052E. For additional detail on the specifics of the DRAM control implementation of the 7S385, consult the user's manual for the board.

The on-chip instruction and data caches allow the R3052E to access one instruction and one data word in each clock cycle. On reads, when a cache miss or an uncachable reference occurs, the R3052E begins an external read cycle. Figure 2 illustrates the single-datum read and Figure 3 illustrates the quad-word read sequence for the 7RS385 at 33MHz.

On writes, the R3052E maintains a write through strategy which updates the memory as soon as the cache contents are changed. With the use of of the on-chip 4-word deep write buffer, the R3052E can continue to execute instructions from its instruction cache while the main memory retires up four pending stores from the write buffer. The DRAM controller on RS385-33 board supports page-mode write (timing diagram of RS385-33 as in Figure 4) and non-page-mode write (timing diagram of RS385-33 as in Figure 5). On RS385-33, the DRAM controller will go into a "page-mode idle state" after a write which keeps RAS LOW in anticipation of a page-mode write. If the next transaction is not a page-mode write, or a refresh request is received, the controller will bring RAS HIGH, precharging the DRAM, prior to servicing the next access.

MODELING THE RS385 WITH Cache-3051

Creating a Cache-3051 model to simulate the desired memory subsystem can be done either by editing the source file (.c program), or defining the runtime parameters at the UNIX command level. To set the read_latency, a timing diagram of a Single Word Read has to be drawn to decide the number of cycles between the asserting edge of Rd and the end of fixup.

Figure 2 illustrates the timing diagrams for a single word read of RS385-33. Once a read access is detected, the RAS signal is brought to LOW after a cycle. Based on the DRAM controller design of RS385-33, the data is sampled by R3052 three cycles after RAS asserted. Then a refill cycle/fixup cycle is used to bring data out of the read buffer and into the internal processor cache; during this fixup, the processor transitions back into the RUN state. In this design, the RAS signal is precharged for three clock cycles (one in the beginning and two at the end), so no more RAS precharge time needs to be added. In this example, the read_latency is set to be 6 cycles.

To determine the idle_write_time, the timing diagram of a write following a read is drawn. In the case of a write following a read operation, the RAS precharge time has already been counted in the last two cycles of read_latency (Figure 2) and the first cycle of write operation (Figures 4 and 5), and thus no

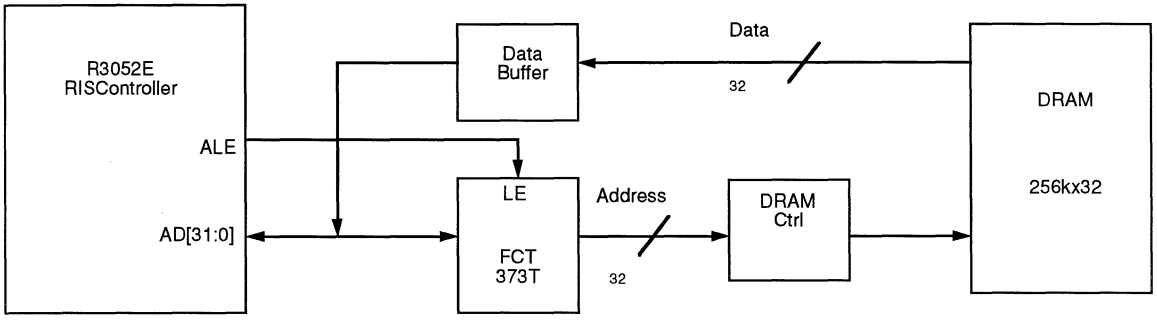


Figure 1. Simplified Block Diagram of 7RS385 DRAM Subsystem

additional cycles are required. The first five cycles of the write operation is defined to be the idle_write_time of RS385-33; that is, it is the time from the assertion of Wr until the negation of Wr. According to the definition of idle_write_time, this is the time of a write operation while the previous operation is not a write.

To determine the page_write_time, the page-mode write timing diagram is drawn. This case occurs when the ongoing write operation shares the same DRAM page as the previous write operation. Page write allows faster data operation within a row address defined page boundary. The current page is determined when RAS is originally asserted; subsequent page-mode cycles occur by selecting a new column address and cycling CAS. These memory cycles are quicker, because CAS precharge time is smaller than RAS precharge, and because RAS and the ROW address have already been presented to the DRAM. Returning RAS HIGH terminates the page-mode write. Figure 4 illustrates a RS385-33 page-mode write (cycle 6 to cycle 9). In this system, 3 cycles are required to retire a page-mode write.

The page-mode write timing diagram is also used to determine the nonpage_write_time setting. a non-page write occurs when the DRAM is anticipating a DRAM page-mode write, but the write which is issued is to a different DRAM page. Therefore not only CAS signal but also RAS signal need to be raised HIGH to strobe-in different column and row addresses. Figure 5 illustrates this write. Note that the RAS needs to be precharged before going LOW again. In this example, the non-page_write_time is 8. It is counted from cycle 6 to cycle 14.

The ras_precharge_time is a little bit different from the RAS precharge defined in DRAM data books. They both imply the RAS signal has to be brought HIGH for precharging. However, the ras_precharge_time in this simulator is only added for the case of a read following a write. The other transactions already implicitly include sufficient RAS precharge cycles, such that no additional time need be allocated. However, in a read following a write, additional precharge time needs to be explicitly added to insure proper operation of the DRAM. Figure 6 explains the RAS precharge time that is included during transitions between bus operations.

The settings of throttled and throttled_latency are strongly related. They both imply burst read operation. Throttled is a flag to indicate how a quad-word read is processed. "Burst" means the first word of the block is returned after an initial read latency, and then each additional word is returned in the immediately subsequent clock cycle. In this case, the "throttled" flag is set false ("0"), and the "throttled_latency" is ignored.

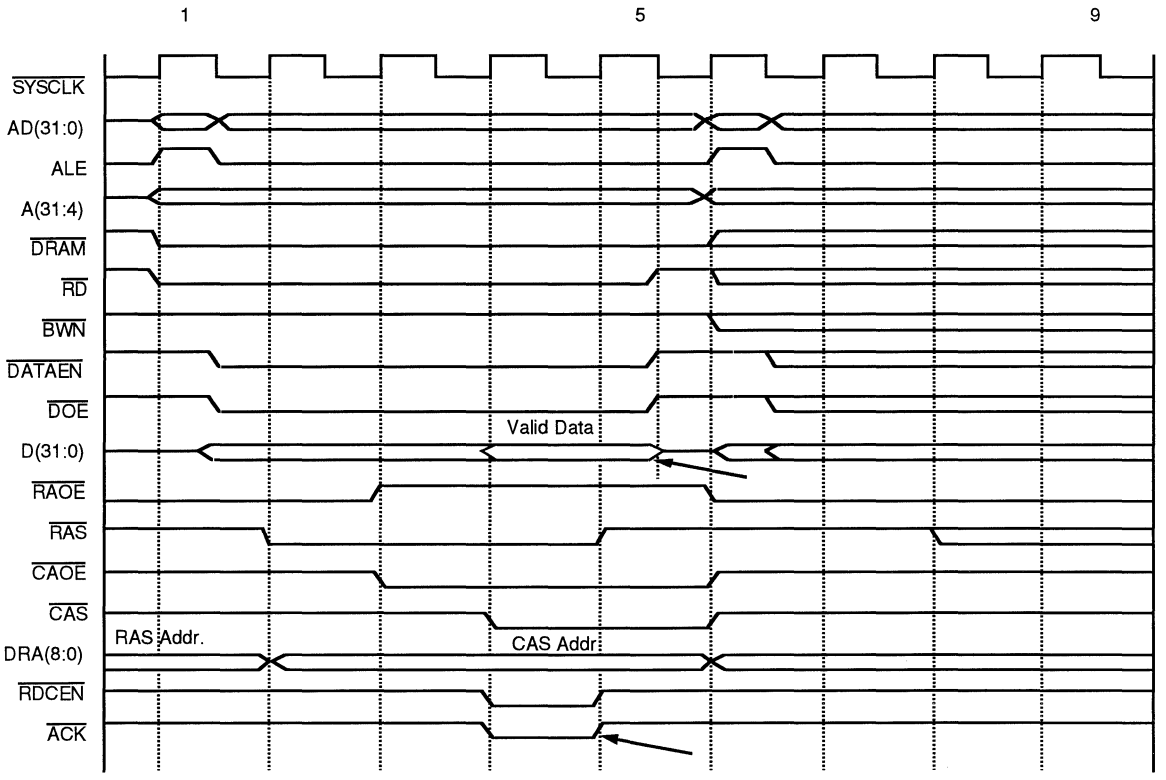


Figure 2. DRAM Single Datum Read

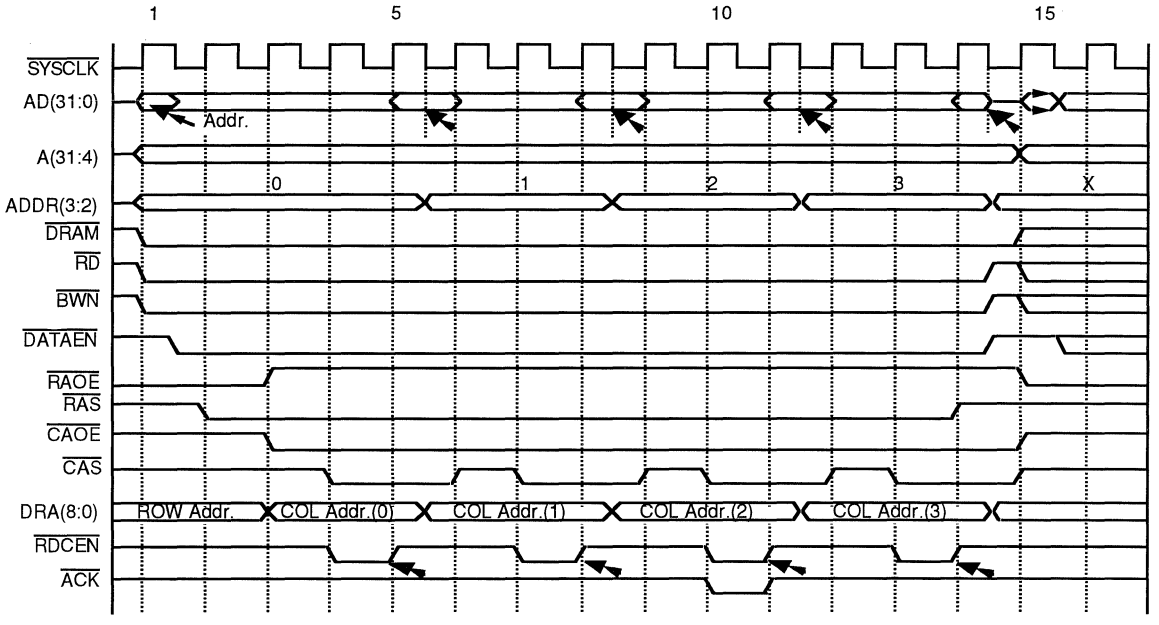


Figure 3. DRAM Quad Word Read

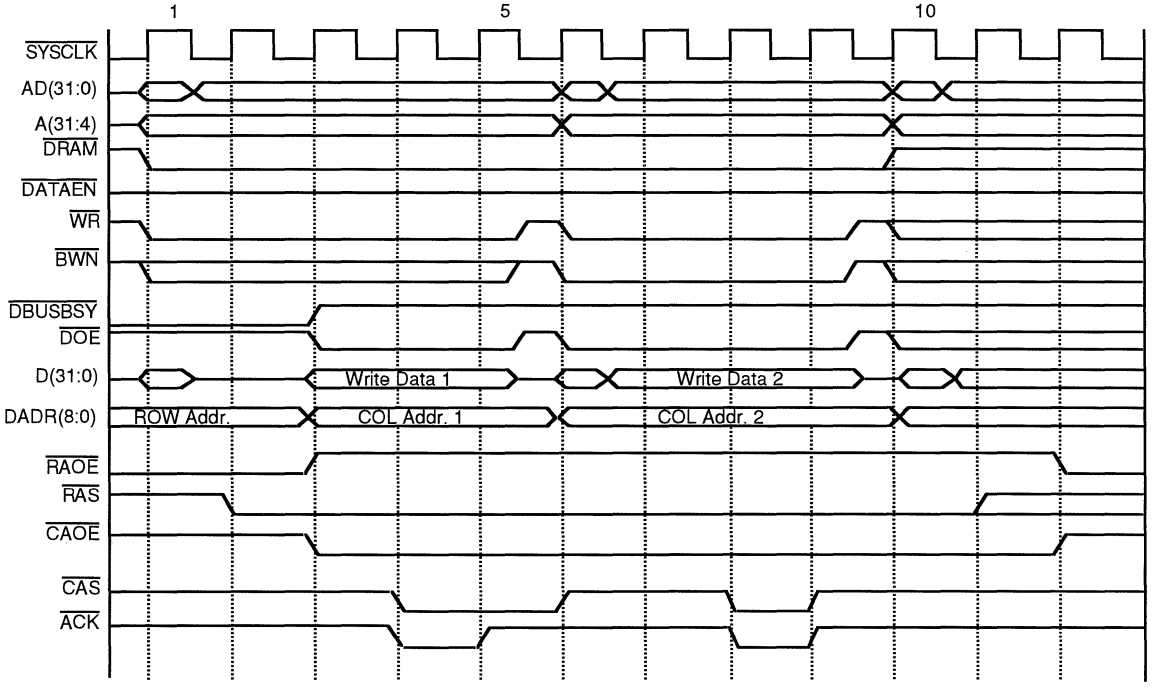


Figure 4. DRAM Page Mode Write

On the other hand, if multiple clock cycles are required between each response word of a quad word read, "throttled" is true ("1"), and the "throttled_latency" paramter is used to indicate the number of idle cycles between words. Throttled_latency is a bus delay cycles between two adjacent words. (i.e., (w1 to w2)+(w2 to w3)+(w3 to w4)).

The summary of DRAM parameters setting based on the RS385 (33Mhz) is shown in Table 1.

Table 1. Summary of 7RS385 Memory System

Parameter	Setting
Read_Latency	6 cycles
Idle_Write_Time	5 cycles
Page_Write_Time	4 cycles
Non-Page_Write_Time	8 cycles
Byte_Extra_Write_Time	0 cycles
RAS_Pre-charge_Time	1 cycle
Throttled	1 (true)
Throttled_Latency	6 cycles

SIMULATION ACCURACY

To validate the relative accuracy of the simulator, a set of benchmarks was run first under the cache simulator and then on the actual board.

The benchmark suite chosen is commonly known as the "Intel Benchmark Suite". Intel chose to use these benchmarks to indicate the performance of the i960CA, when that product was originally introduced.

Although IDT views these benchmarks as inadequate when used to determine actual system performance, they can be used to help determine the relative accuracy of Cache-3051 (for more information on the problems with this benchmark suite, refer to the IDT applications note on this topic).

The Intel integer benchmarks consist of six benchmarks which are: bubblesort, quicksort, pi500, anneal, matmult, and dhrystone1.1. The brief description of each benchmark, and the aspect of system performance Intel feels each measures, is given below:

bubblesort—Sorts a 500 element array in memory using the "bubble sort" algorithm. Performance is heavily dependent on the speed of data access. The benchmark features heavy use of array manipulation.

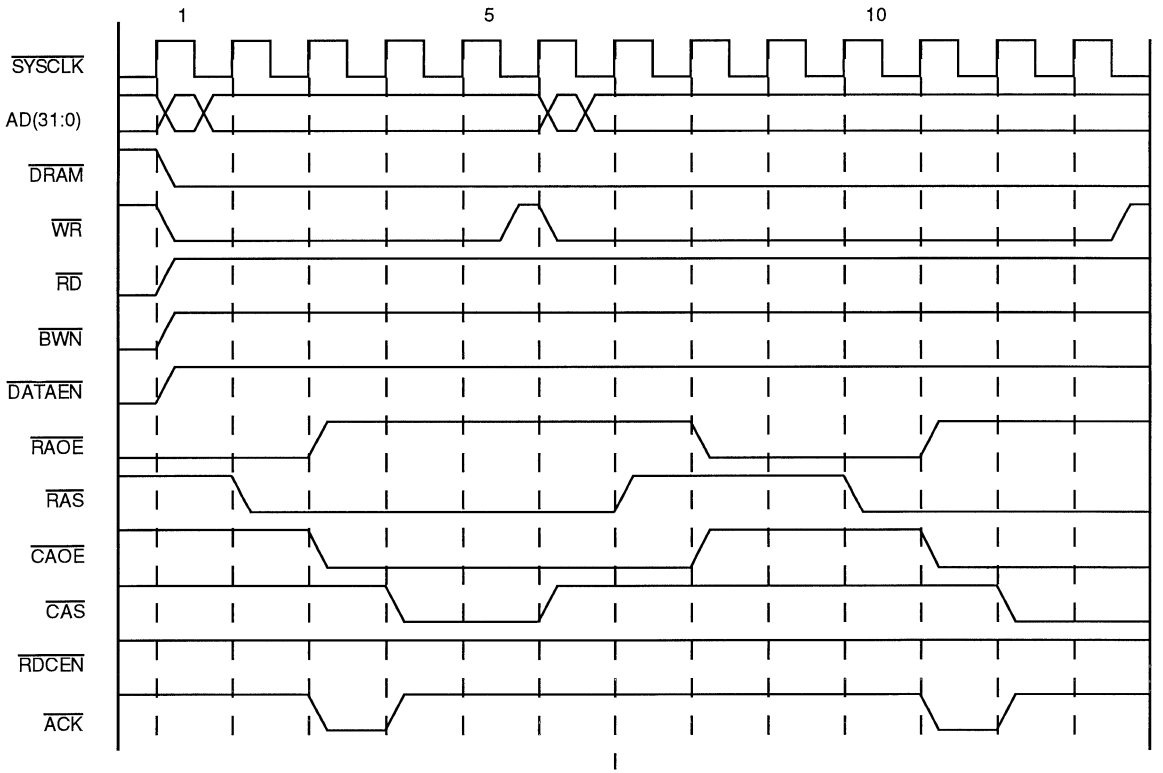


Figure 5. DRAM Non-Page Mode Write

Operation	RAS Precharge Time for RS385's DRAM System
R –> R	2 Clock cycles (at the end of read) + 1 clock cycle (in the beginning of read)
W –> R	0 Clock cycle (at the end of write) + 1 clock cycle (in the beginning of read) + ras_precharge_time
R –> W	2 Clock cycles (at the end of read) + 1 clock cycle (in the beginning of idle write)
W –> W	0 Clock cycle (at the end of write) + 0 clock cycle (for page write) or + 3 (4 –1: for non-page write)

Figure 6. RAS Precharge Between Various Transfers

quicksort—Sorts a 5000 elements array in memory using the "quick sort" algorithm. The benchmark is designed to test recursion and array indexing.

pi500—Calculates the value of the mathematical constant PI up to 500 decimal points using iterative integer calculations.

anneal—Also known as the traveling salesman problem. The benchmark calculates the shortest distance between two points. (20 points were used)

matmult—Multiplication of a matrix of values tests the multiply/divide speed of the processor. Few memory references were used.

dhrystone1.1—Classic integer benchmark measuring relative processor performance for integer instructions. While this benchrmark suppossedly demonstrates the integer number crunching power of the processor, its performance is actually highly dependent on the coding of string library functions.

All the above mentioned integer benchmarks are compiled with a C compiler version 2.11 on an MIPS RC3240 system running RISC/OS 4.51. All the benchmarks are compiled with default optimization level (–O). This level of optimization does not include the full optimization capabilities of the MIPS compiler chain, such as inter-procedual register allocation and procedure merging, due to the method of program generation used for the RS385.

In our example, downloadable benchmarks (.srec files) are generated by compiling individual modules separately, linking together, then downloading to the RS385 board. This process has to bypass the uload and the umerge phases (refer to IDT RISC R3000 Family language Programmer's Guide) which are the –O3 and –O4 phases in the MIPS compiler, and use the link/load phase by linker/loader only. Since the linker/loader does not have an optimization phase, the best optimization can be used in this example is –O.

The results for Cache-3051 and for the RS385-33 board are listed below. The execution times for above mentioned programs are shown in Table 2 (smaller values are better except for Dhrystone1.1).

For the Pi-500 program, two values are given in each column. One is with the "printf" function enabled, and the other is with "printf" disabled. For benchmarking, and for comparing the accuracy of the simulator, disabling "printf" is appropriate for a couple of reasons. First, I/O processing time is system, rather than processor dependent, and depends on the peripheral chosen and the communications channel. Secondly, this function is linked with an entry point supplied by IDT/sim™ (PROM monitor). This monitor program resides on the RS385 in the EPROMs, and executes as an uncached access and with longer memory latency than the DRAM subsystem. Single reads and burst reads from these EPROMs are much slower than from DRAMs.

As shown in Figures 7 and 8, single reads take 8 cycles and burst reads take 6 cycles per subsequent word for accesses to EPROM.

For basically the same reason, the dhrystone benchmark was generated by linking to the IDT/kit™ -llibrary, rather than the IDT/sim functions. Dhrystone performance is heavily dependent on the strcpy and strcmp library functions. In order to avoid accessing the EPROMs for these functions, IDT/kit is used to integrate these library functions into the downloaded benchmark file. Thus, these library routines will execute as cacheable routines, and run with the same memory latency as the rest of the program. (IDT/kit is a set of modules which assists the system developer in interfacing with the R3000/R3051 family of processors. It consists of a micromonitor, the start-up module, kernel integration library, interface into IDT/sim functions, ANSI compatible standard c library, and transcendental math library. More information on IDT/kit and IDT/sim can be obtained from the reference manuals for these products.)

Table 2. Results of Intel Benchmarks on Cache-3051 and on 7RS385 Board

Benchmark	Cache-3052–33 (MIPS-02)	RS385–33 (MIPS-02)	Cache-3052 vs. RS385
QuickSort (ms)	38.6	39.6	(38.6 – 39.6)/39-6 = –2.5%
BubbleSort (ms)	44.9	44.8	(44.9 – 44.8)/44.8 = –0.2%
PI–500 (ms)	1,041 1,039*	1,624 1,084*	(1041 – 1626)/1626 = –35.9% (1039 – 1084)/1084 = – 4.1%
Anneal (ms)	5,193	5,149	(5193 – 5149)/5149 = 0.2%
MatMult (ms)	19,619	19,417	(19619 – 19417)/19417 = 1.0%
Dhrystone 1.1	43,356	41,666	(44822 – 41666)/41666 = 7.5%

NOTES:
* Run with "printf" function disabled.
Dhrystone 1.1 run using string library from IDT/kit.
Compiled using MIPS/c with optimization –02.

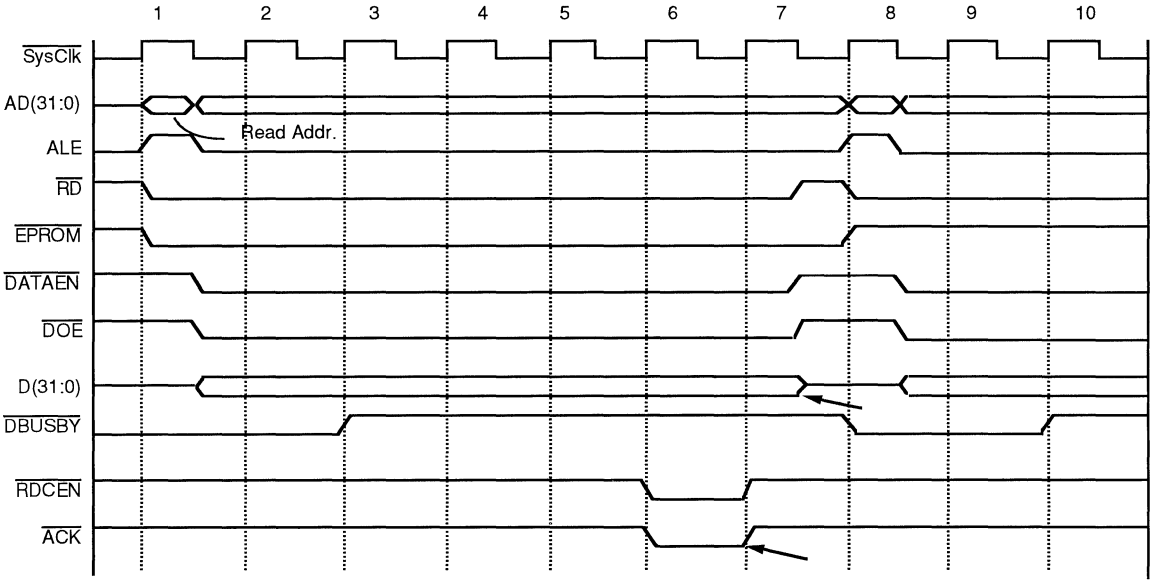


Figure 7. EPROM Single-Word Read on 7RS385 Board

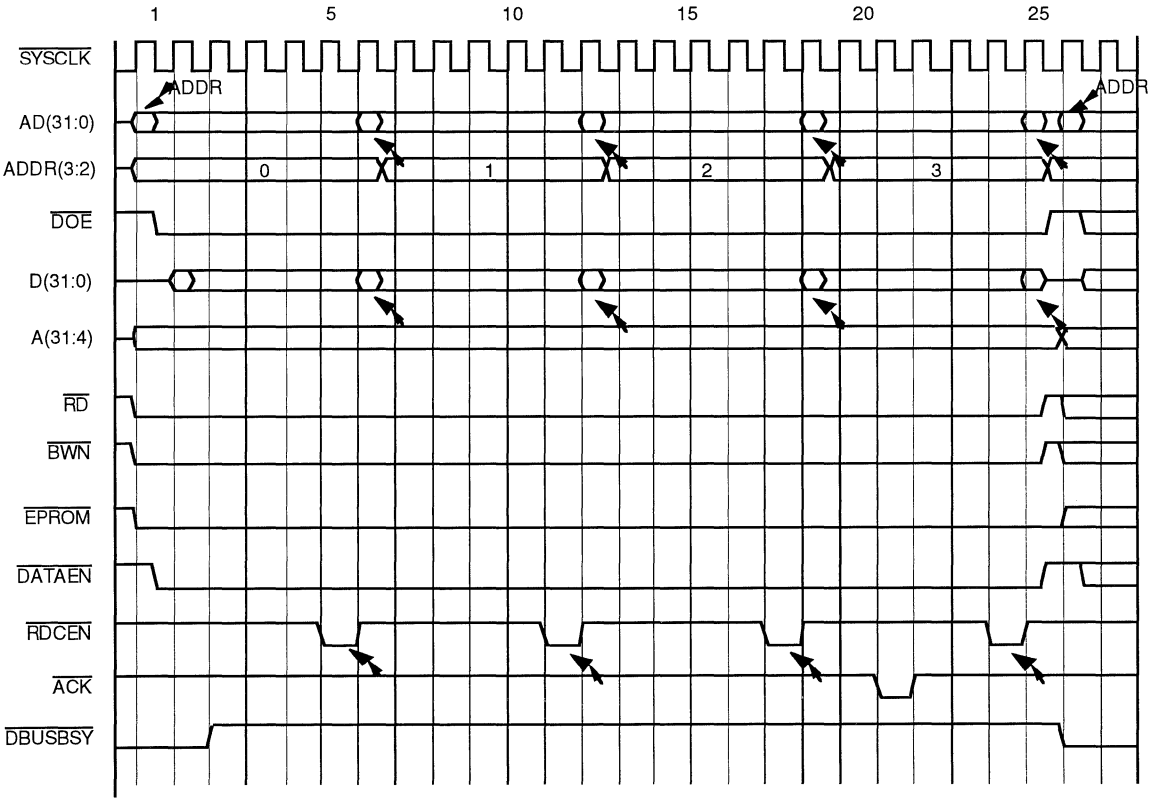


Figure 8. EPROM Quad-Word Read on 7RS385 Board

PERFORMANCE DIFFERENCES BETWEEN SIMULATION AND REAL SYSTEMS

Obviously, some performance differences between a memory simulator and an actual system are inevitable. As can be seen from this application note, the relative inaccuracy is extremely small; in fact, they are due to a number of factors which can not be effectively modeled by such a tool. These factors include:

DRAM refresh. There can be no effective simulation which will include the effects of DRAM refresh. If the processor were operating out of cache during a DRAM refresh, no performance would be lost. On the other hand, DRAM refresh may prevent a subsequent write from being processed as page-mode.

DRAM refresh cannot be modeled typically because its timing is determined by a divide-down counter. Thus, the power up state of the counter versus the reset time (often determined by an RC network) determines when, relative to execution, DRAM refreshes occur. Such an event may in fact vary in a given system each time the system is turned on.

Exception events. Some exceptions, such as arithmetic overflow, could be modeled. Others, such as a periodic time slice counter interrupt, may not be modeled for much the same reasons that DRAM refresh cannot be accurately modeled.

I/O events. Since I/O timing may rely on the response timing of a peripheral (such as a terminal or disk) external to the CPU motherboard, the exact timing of I/O responses cannot be effectively modeled.

Mixed memory systems. The modeler does not really account for systems in which part of the code operates out of a memory with latency characteristics different from the memory for other parts of the program.

Multi-master systems. The R3051 family can operate out of its internal caches during DMA transactions. Thus, it is difficult to derate performance based on bus bandwidth consumed by an external DMA engine, because the processor may operate out of cache during some of those transfers. Similarly, if the processor must arbitrate over a bus or through a multi-port arbiter for a given memory subsystem, the modeler cannot effectively model a memory whose latency varies according to external events.

SUMMARY

The IDT Cache-3051 is a very useful tool, allowing hardware and software designers to project and model the performance of different IDT79R3051-based systems accurately prior to developing actual hardware. This allows effective price-performance tradeoffs to be made early in the design cycle, and allows the resulting software to be effectively tuned to the end system.

When using a system modeler such as Cache-3051, the user must be aware of the limitations of such a tool, and be aware of the relative accuracy of the tool. This application note describes the memory system variables under the user's control, and demonstrates the accuracy of the tool on a benchmark suite. Finally, the application note contains a discussion of certain types of characteristics which cannot be modeled.

Integrated Device Technology, Inc.

USING THE R3081™ IN R3051™-BASED SYSTEMS

APPLICATION NOTE AN-109

By Peter McDonald

INTRODUCTION

The IDT79R3081™ RISController™ is the newest member of IDT's family of high-performance and price-competitive 32-bit microprocessors. Designed to provide the high-performance MIPS® RISC architecture to low-cost and system integration-sensitive solutions, this processor adds to the growing family of RISControllers from IDT. The R3081 RISController is superset and pin compatible with the R3051/52, and includes 20kB of cache, a Floating-Point Accelerator, Hardware Cache Coherency support, and a series of system integration and interface features.

With its larger caches, FPA and interface features, incorporating the R3081 in an existing R3051 design can dramatically increase system performance without adding design complexity. Often upgrading to the R3081 is as simple as placing an R3081 in the R3051 socket. This applications note describes common considerations when upgrading existing R3051 systems with the R3081. As an example, this application note describes how to upgrade the 7RS385 evaluation board from an R3051 processor to an R3081 processor.

NEW FEATURES BROUGHT BY THE R3081

The R3081 is superset pin-compatible with the R3051. That is, in general it is possible to remove an R3051 from a system and replace it with an R3081. The system should run without any hardware or software changes. However, the R3081 adds additional capabilities to the R3051 family; some systems may wish to take explicit steps to take advantage of these new capabilities.

Before discussing system changes needed to implement the superset features of the R3081, a definition of these capabilities is needed. As mentioned above, the R3081 includes larger Instruction and Data Caches, a Floating-Point Accelerator, Hardware Cache Coherency support, and a series of integrated control options. All the hardware options are selected by either the mode initialization vectors (values sampled on the interrupt input lines during reset) or programmed through the new CP0 Configuration register. Below is a summary of the new R3081 features. A more detailed list of these features along with a list of the differences between the R3051 and R3081 are included in the IDT79R3081/3081E Integrated RISController Hardware User's Manual.

- *Larger Instruction and Data Caches*
  The R3081 instruction and data caches total 20kB. The default (reset) configuration is 16kBI and 4kBD, although they are dynamically programmable to 8kB apiece. Both instruction and data caches are parity protected over the data and tag fields. This differs from the R3051, in that both caches are larger than the caches supported by the R3051 or R3052, the cache is configurable and the caches are parity protected.
- *Addition of a Floating-Point Accelerator*
  A full-featured R3010A-compatible floating-point accelerator is incorporated on the R3081 adding single- and double-precision add, multiply, and divide instructions to the instruction set. Which of the six integer unit Interrupts inputs is used for the floating-point interrupt signal is programmable. $\overline{\text{Int3}}$ is the default FP interrupt. Thus, one of the six interrupt inputs of the R3051 is used for the floating-point interrupt and coprocessor 1 instructions will be directly executed by the on-chip floating-point units.
- *Cache Coherency Interface*
  The R3081 has a hardware-based cache coherency interface for multi-master systems. If selected, DMA cycles between memory and I/O can invalidate lines within the R3081 cache, insuring that there is no stale data and avoiding software directed cache flushing. This mechanism can be disabled to achieve full R3051 compatibility; alternately, the system designer can choose to increase the performance of multi-master systems, by performing hardware cache coherency.
- *Power Reduction Mode*
  The R3081 RISController can be dynamically programmed to reduce its operation frequency. In this mode the execution clock, and therefore the output clock, is internally divided by 16. This function allows the power reduction benefits of a lower speed clock to be achieved during idle periods, without requiring external clock shaping logic.
- *Programmable Halt Mode*
  This programmable mode forces the R3081 RISController to stall until either an interrupt or reset is issued. This mode has two effects: it further reduces power consumption; and, it allows software to halt until some external event occurs.
- *Half-Frequency Bus Mode*
  A selectable mode allows the R3081 bus interface to operate at one-half the frequency of the processor core. For example, the execution core can run at 33MHz, and the bus interface at 16MHz. Given the substantial amount of cache on-chip, the slow system interface will not dramatically degrade performance. The end result is a high-performance system with very low system cost.
- *1x or 2x Clock Input*
  The R3081 can operate with either an R3051 compatible double-frequency clock input (2x clock mode), or can operate from a clock at the execution rate (1x clock mode). This capability both simplifies EMI at high frequency, and also

allows for "clock doubling" when used in conjunction with the one-half frequency bus mode.

- *Slow Bus Turnaround*
  A common problem for a high-speed I/O bus is the amount of time available for mastership changes. The R3081 allows software to specify a larger minimum time when transitioning from the memory driving the bus (i.e. read data) and the processor driving the bus (e.g. writes). This reduces the speed requirement of data transceivers, with minimal performance impact.
- *Dynamically programmed data cache refill*
  The R3081 allows software to dynamically select between single word and quad word refill on data cache miss. This allows for additional performance tuning, by enabling the kernel to select the best algorithm for a given section of code. The default refill size is selected at reset time, the same as for the R3051.

POSSIBLE CHANGES

The R3081 hardware options are either mode selectable at reset or programmed through an internal register. Hardware cache coherency support and all clocking modes, half-frequency bus mode and 1x or 2x clock input mode, are selected at reset based on the level of the $\overline{\text{Int}}[5:3]$. In the R3051, $\overline{\text{Int}}[5:3]$ are required to be driven HIGH during reset initialization.

The interrupt inputs, $\overline{\text{SInt}}[2:0]$ are already used by both the R3051 & R3081 to select data cache refill sizes, tri-state test mode, and big or little endian system architectures. The complete table of the R3081 reset mode vectors is listed in Table 1.

A complete description of these modes is provided in the IDT79R3081/3081E Integrated RISController Hardware User's Manual.

Floating-Point Interrupt

The one area where hardware changes may be necessary are with respect to the Floating-Point Accelerator. In the MIPS RISC architecture, the floating-point interrupt is fed into a general purpose interrupt. Interrupts cause the processor to jump to the system's exception handler which then decodes its status to determine the exception cause. One of the six external R3081 interrupts (by default $\overline{\text{Int3}}$) is programmed to be the FPA interrupt. All activity on the external interrupt pin corresponding to the FPA interrupt is ignored.

Although software can use a different interrupt input other than the default, it is still the case that only five external interrupt pins remain available to external peripherals. Therefore, systems that required six external interrupts will need to modify their external interrupt structure, perhaps by causing multiple peripherals to share a single interrupt input. Obviously, software would then need to decode which device on that interrupt actually signalled the exception.

Systems that have defined an interrupt other than $\overline{\text{Int3}}$ for the FPA need to modify their startup code so as not to ignore the assertion of $\overline{\text{Int3}}$.

Table 1. R3081 Mode Selectable Features

Interrupt Pin	Mode Feature
$\overline{\text{Int5}}$	$\overline{\text{CoherentDMAEn}}$
$\overline{\text{Int4}}$	$\overline{\text{1xClockEn}}$
$\overline{\text{Int3}}$	$\overline{\text{Half-frequency Bus}}$
$\overline{\text{SInt2}}$	DBlockRefill
$\overline{\text{SInt1}}$	$\overline{\text{Tri-State}}$
$\overline{\text{SInt0}}$	BigEndian

Some software applications incorporate exception handlers that allow the user to set the FPA interrupt through software. The IDT/sim™ diagnostics uses this method. This adds system flexibility at the cost of the extra performance required to decode the interrupt.

The Config Register

Selecting which interrupt is used by the on-chip FPA, the cache configuration, power reduction mode, current size of data cache refill, halt/stall mode, or slow bus turnaround are all accomplished by writing to the new CP0 configuration register. The Configuration Register data format is shown in Figure 1.

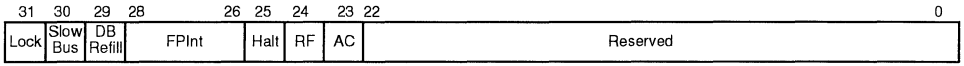
The reset initialization value of the config register depends somewhat on the mode vectors selected at reset. Specifically, the initial values of the Data Block Refill bit, and of the slow bus turnaround bit, are dependent on the reset vectors. At reset, the FPInt field will correspond to $\overline{\text{Int3}}$, and the Lock, Alt. Cache, Halt, and RF bits will be cleared.

Reading and writing all CP0 registers is accomplished by issuing coprocessor load and store instructions. The configuration register is CP0 register 3. An interactive tool to read and write the R3081 configuration register, "the R3081 Configuration Tool", is available as a demo tool through your local sales office, and runs on IDT/sim-based platforms. To insure strict software compatibility with older applications, the Config register can be isolated from subsequent writes by writing a '1' to the configuration register "Lock" field.

Software Compatibility

The R3081 will directly execute applications written for the R3051. The larger on-chip caches will directly benefit existing applications, and thus bring an increase in system performance. Additional gains are possible, depending on the application code, by taking advantage of the hardware FPA on the R3081. Whereas the R3051 must either trap and emulate floating-point instructions, or perform explicit calls to software floating-point libraries, the R3081 can directly execute these operations.

It may be advantageous to generate two distinct binaries from one source; one, which uses software libraries to emulate floating-point operations, and is used with the R3051 or R3052 and another, which uses the on-chip FPA to perform floating point. However, if the prospect of two distinct binaries is too onerous for a particular application, the binary could



Lock: 1 -> Ignore subsequent writes to this register
Slow Bus: 1 -> Extra time for bus turnaround
DB Refill: 1-> 4 word refill
FPInt: Power of two encoding of FPInt <-> CPU Interrupt
Halt: 1 -> Stall CPU until reset or interrupt
RF: 1 -> Divide frequency by 16
AC: 1 -> 8kB per cache configuration
Reserved: Must be written as 0; returns 0 when read

Figure 1. CP0 Configuration Register Data Format

include FPA instructions; with an R3051 processor, a trap will be generated, and software could emulate the operation. Although a single binary suffices for both processors, the cost is reduced performance for the R3051.

Software can dynamically determine whether there is an FPA available, by performing simple FPA diagnostics. Such diagnostics is included in IDT/sim, IDT/c™, and IDT/kit™ startup code. Thus, the boot software could check for the presence of an FPA, and initialize the Coprocessor One useable bit according to the results. This allows a single binary to dynamically determine whether a hardware FPA is available, and can be used to enable the FPA instruction trap mechanism of the R3051 and R3052.

Manipulating the Cache Characteristics

Another possible performance gain may exist by dynamically manipulating the cache characteristics of the R3081. The Config register allows the cache configuration to be dynamically changed from 16kB I-Cache and 4kB D-Cache to 8kB I-Cache and 8kB D-Cache. A kernel may choose to dynamically change the cache organization, depending on the nature of the task about to be executed. The only caveat is that when changing the cache configuration (from 16kB/4kB to 8kB/8kB or vice versa), both the instruction and data caches need to be flushed.

In addition, software could dynamically alter the D-Cache refill size. Changing this bit does not require a cache flush.

Note that to insure compatibility amongst multiple generations of R3051 family members, cache flushing routines that assume a constant cache size are discouraged. The R3081 Hardware User's Manual presents an algorithm where software can determine the cache size available.

UPGRADING THE RS385 BOARD WITH THE R3081

Upgrading the RS385 board with the R3081 RISController is easy to accomplish. Simply remove the R3051 and replace it with the R3081. Both share the same footprint and pinout. The 1xClockEn, Half-frequency bus, and Coherent DMA modes are all disabled in a default 7RS385, thus no further hardware modifications are necessary. $\overline{\text{Int}}[5:3]$ are pulled HIGH during reset disabling these three modes.

The IDT/sim included with the 7RS385 automatically sizes the cache available; thus, the increased cache sizes of the R3081 pose no problem. IDT/sim will not, however, write to the Config register. Thus, the FPU interrupt will default to $\overline{\text{Int3}}$, unless explicit steps are taken.

Currently on the RS385, the R3051 $\overline{\text{Int3}}$ is used for the Centronics port interrupt. If using the Centronics port and the R3081 FPA, the system and/or software must be modified so that the FPA is allowed its own dedicated interrupt. This needs to be done by either re-writing the boot prom to modify the config register or using a different Centronics interrupt and modifying the Centronics driver.

If the 7RS385 has been used as a porting target for another application, the types of software changes needed will be application dependent. Applications developed with IDT/kit and/or IDT/c include startup code that resizse the cache every time they are executed. IDT/sim startup code does not resize the cache at each execution. In addition, it may be desirable to recompile for any floating-point instructions that are implemented with software emulation.

Implementing Additional Reset Modes

When using any of the three reset mode features unique to the R3081, minor modifications to the RS385 board are necessary to implement the interrupt input signal multiplexing during reset. As a general note, the RS385 uses a tri-statable interrupt bus to implement the multiplexing for the $\overline{\text{SInt}}[2:0]$. An asserted MRES# enables the reset mode vector driver. A modification to the RS385 board was made to enable or disable any of the six mode selectable features with jumpers, including the new mode vectors of the R3081. Figure 2 shows the modified R3051/R3081 interface to allow enabling and disabling of the six reset modes. A buffer, U1A, was added to provide the tri-state mux for the three new reset modes.

Other solutions to implement the reset mode selection abound, depending on one's application. All R3051 designs should already pull $\overline{\text{Int}}[5:3]$ HIGH during reset as specified in the IDT79R3051 Family Hardware User's Manual. Therefore, only the new modes being selected need to be added to the current muxing on the RS385. If only one additional mode is needed, jump the one remaining output on the current 74FCT244 reset mode mux (U37 pin 18) to the appropriate interrupt input. The interrupt PAL, U28, can be reprogrammed

to do some of the muxing. (If the PAL can not be easily removed from the board, an additional device can be added to the wire-wrap area.)

An Interesting Upgrade

One of the more interesting upgrades possible is to increase the execution speed while decreasing the bus clock. To do this, select 1x clock mode and half-frequency bus from the new mode reset logic, and replace the R3051 osciallator with a 40MHz oscillator. The result will be a CPU core executing at 40MHz rather than 25MHz, although the bus speed has been reduced to 20MHz.

UPGRADING OTHER R3051 SYSTEMS

Upgrading any R3051-based system with the R3081 RISController is very similar to updating the RS385 board. The one hardware item that may differ has to do with DRAMs and their refresh.

Specifically, if the refresh period is based on counting SysClk cycles, then using the reduced frequency mode of the R3081 may violate the reset period (reduced frequency mode also divides the frequency of the output clock). There are two solutions to this, depending on the application:

- Reprogram the counter to a smaller number of SysClks. This is possible with devices such as the R3721 DRAM controller.
- Use a different reference clock for refresh. Choices include a UART clock, or the clock used to generate the input clock to the processor.

The RS385 board refresh request is generated from a clock independent of $\overline{\text{SysClk}}$. The clock used is derived from the UART clock.

CONCLUSION

Incorporating the high-performance R3081 RISController into existing R3051-based systems is often as simple as merely swapping processors. Little design complexity is added, yet system performance increases due to the larger caches, Floating-Point Accelerator, and other features. Using more of the R3081 features to increase performance even more can be accomplished with minimal hardware and software modifications.

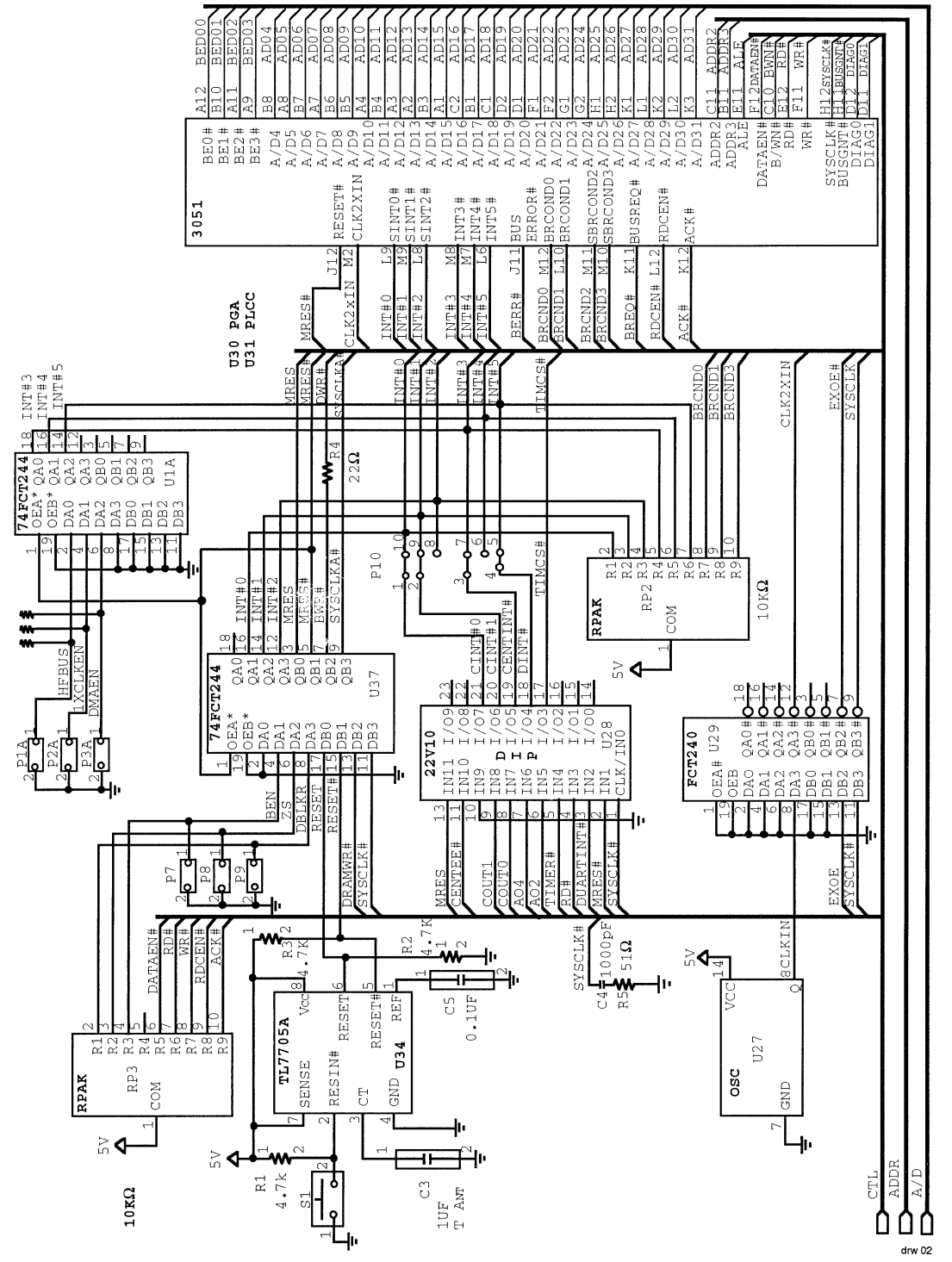


Figure 2. 7RS385 Mode Vector Logic Upgraded for R3081

Integrated Device Technology, Inc.

USING THE IDT79R3051™ AND THE IDT79R3081™ WITH THE HP16500 LOGIC ANALYZER

Supplement to Application Note AN-93

APPLICATION NOTE AN-111

By Gary Szilagyi

INTRODUCTION

In Application Note-93, the use of IDT's 7RS364 disassembler with the HP16500 Logic Analyzer for the IDT79R3051™ RISController™ family of CPUs was discussed in detail. However, the original versions of the disassembler were form-fitted for the R3000 CPU interface of a 32-bit non-multiplexed bus design. In order to accommodate the high level of integration on-board the R3051, including the 4kB–8kB of instruction cache, 2kB of data cache, 4-deep read and write buffers and the R3000A execution engine—all in a single 84-pin package, the 32-bit bus required multiplexing address and data pins. Although the original versions of the disassembler remain compatible with the new family of IDT's RISControllers, an effort was made to simplify the interface between R3051 and the disassembler to accommodate simple triggering schemes, as well as future IDT embedded controllers that continue in the path of the R3051 family.

THE IDT7RS364 DISASSEMBLER AND THE IDTR3051

The IDT7RS364 Disassembler consists of a software package that greatly eases the task of debugging software on the IDTR3051 family of CPUs. The HP16500 allows the capture of executed hex/binary machine opcodes in a typical Logic Analyzer State Trace Listing format with the ability to decode and display the acquisitions in the R3000 assembly code mnemonic format, as seen in Figure 1. Thus, the engineer does not have to resort to look-up tables, and can effectively determine the exact processor state for easy software debugging.

The original versions of the disassembler were form-fitted to the R3000 CPU interface. Although the derivative products of the IDT R3051 family are compatible, the $\overline{RD}$ and $\overline{WR}$ signals used for data acquisitions by the disassembler package causes some confusion during a high-speed burst read. As discussed in Application Note AN-93, the work-around was to create a more complex read strobe in order to capture a four-word burst read by setting up a trigger mechanism on the HP16500 that looks like: [($\overline{SysClk}$ ==↑) AND [($\overline{ACK}$ == 0) OR $\overline{RDCEN}$ == 0)]. However, this is only applicable to systems that bring the $\overline{ACK}$ signal LOW at precisely the same time the $\overline{RDCEN}$ is LOW, or that don't bring it LOW at all during a four word burst read. If, for instance, the $\overline{ACK}$ signal triggered in the phase between two successive $\overline{RDCEN}$s, a duplicated capture would occur. The disassembler was modified a second time to remedy this situation. In a read cycle, the $\overline{RD}$ pin will be asserted LOW for the entire cycle and the $\overline{RDCEN}$ signal toggles to successfully pass each of the four words across the bus. The newest version of the disassembler

State/Timing E | Listing 1 | Invasm | Print | Run

Markers Off

Label> Base>	ADDR Hex	R3000 Mnemonic hex	STAT Hex	Time Absolute
-6	1FC00000	J 0x1FC00220	0010	0 s
-5	1FC00004	NOP	0010	760 ns
-4	1FC00220	LUI v0,0x0010	0010	1.52 us
-3	1FC00224	MTC0 v0,$12	0010	2.24 us
-2	1FC00228	MTC0 zero,$13	0010	3.00 us
-1	1FC0022C	LUI v0,0xA000	0010	3.76 us
0	1FC00230	LUI t0,0xAAAA	0010	4.52 us
1	1FC00234	ORI t0,t0,0x5555	0010	5.24 us
2	1FC00238	SW t0,0x0000(v0)	0010	6.00 us
3	1FC0023C	SW zero,0x0004(v0)	0010	6.76 us
4	00000000	STORE DATA 0xAAAA5555	0000	7.40 us
5	1FC00240	LW t1,0x0000(v0)	0010	7.88 us
6	00000004	STORE DATA 0x00000000	0000	8.52 us
7	1FC00244	NOP	0010	9.00 us
8	00000000	LOAD DATA 0xAAAA5555	0010	9.64 us
9	1FC00248	B 0x1FC00258	0010	10.32 us

Figure 1. R3051 Address/Data Trace List on a Logic Analyzer

begins "LOAD" captures not on $\overline{RD}$, but rather upon the $\overline{RDCEN}$. For interleaved memory systems that do not toggle the $\overline{RDCEN}$ pin, please refer to section "Hazards" for more details. During a write cycle, it triggers upon the rising edge (from LOW-to-HIGH) of the $\overline{WR}$ signal. Thus, the newest revision of the disassembler now expects the $\overline{RDCEN}$ and the $\overline{WR}$ signals as clocks to strobe the address and data into the HP16500, as well as the $\overline{WR}$, DIAG_1 and DIAG_0 to verify and decode the processor status

INTERFACING THE HP16500 TO THE '385 EVALUATION BOARD

In order to insure proper operation of the disassembler, the correct interface between the R305x target system and the HP16500 must be available. The disassembler requires a particular pinout setup on the logic analyzer's five 16-channel probe pod sets. The interface protocol must be followed for correct interpretation of the address, data, and status lines by the pre-processor. Table 1 displays the default pod connections that the HP16500 expects (same setup for the 7RS385 evaluation board). This information is stored on disk in the configuration file "DIS_305x_E". When loaded, this file not only loads the disassembler, but also all the state and timing information, including the default pod connections expected at the system interface.

Application Note-93 discusses in detail the interface between typical R305x based systems and the logic analyzer. Rather than repeat that discussion, the interface between the 7RS385 Evaluation board and the disassembler requires some elaboration. For instance, the '385 Hardware User's Manual shows the connections to be made from the board's five 20-pin logic analyzer sockets and the logic analyzer's five, 16-channel pods. Note however that in section 2–5 of the '385 Hardware User's Manual, the connections on the status pod (pod#5) are incorrect. In order to be consistent with the protocol of the disassembler, some of the pins need to be connected as follows:

- WR (J12 pin #17) needs to be on pod #5 channel #4
- RDCEN (J12 pin #14) needs to be on pod #5 channel #5

The disassembler also requires status lines for determining processor status: $\overline{WR}$, $\overline{RDCEN}$, DIAG_1, and DIAG_0. The $\overline{WR}$ signal distinguishes between read and write cycles. The $\overline{RDCEN}$ pin is used to identify a false trigger for applications that assert the $\overline{RDCEN}$ signal during writes. In order to avoid a duplicate capture, the $\overline{RDCEN}$ signal is polled to determine if it was the cause of the acquisition. If it was, then a trigger-

Table 1. R3051 Default Pod Connections on the HP16500 Logic Analyzer

POD chan	5 sig	POD chan	4 sig	POD chan	3 sig	POD chan	2 sig	POD chan	1 sig
15	X	15	A/D(31)	15	A/D(15)	15	A(31)	15	A(15)
14	X	14	A/D(30)	14	A/D(14)	14	A(30)	14	A(14)
13	X	13	A/D(29)	13	A/D(13)	13	A(29)	13	A(13)
12	Diag_1[2]	12	A/D(28)	12	A/D(12)	12	A(28)	12	A(12)
11	X	11	A/D(27)	11	A/D(11)	11	A(27)	11	A(11)
10	Diag_0	10	A/D(26)	10	A/D(10)	10	A(26)	10	A(10)
9	X	9	A/D(25)	9	A/D(9)	9	A(25)	9	A(9)
8	X	8	A/D(24)	8	A/D(8)	8	A(24)	8	A(8)
7	X	7	A/D(23)	7	A/D(7)	7	A(23)	7	A(7)
6	X	6	A/D(22)	6	A/D(6)	6	A(22)	6	A(6)
5	$\overline{RDCEN}$	5	A/D(21)	5	A/D(5)	5	A(21)	5	A(5)
4	$\overline{WR}$	4	A/D(20)	4	A/D(4)	4	A(20)	4	A(4)
3	X	3	A/D(19)	3	A/D(3)	3	A(19)	3	Addr(3)
2	X	2	A/D(18)	2	A/D(2)	2	A(18)	2	Addr(2)
1	X	1	A/D(17)	1	A/D(1)	1	A(17)	1	$\overline{BEN(1)}$
0	X	0	A/D(16)	0	A/D(0)	0	A(16)	0	$\overline{BEN(2)}$
NClk	$\overline{WR}$	MClk	$\overline{RDCEN}$	LClk		KClk		JClk	

NOTES:

1. Master Clock Format: N↑ + M↑ (default for the 7RS385 Evaluation Board setup)
2. POD5(12) is Diag_1 and POD5(10) is Diag_0 (Diag pins are not latched on the 7RS385 Eval Board). If running uncached, then Diag_1 MUST be grounded (GND), and Diag_0 is not used by disassembler.
3. A(31:4) are connected to the Address Latch outputs. The rest of the signals are connected to R3051 outputs. X's denote unused probes that can be assigned by the user.

error message, "T.E", and the store instruction along with the write data on the bus is displayed (e.g. "T.E. (STORE 0xxxxxxxxx)). The diagnostic pin DIAG_1 distinguishes if the external memory read was cacheable, and if so, determines with DIAG_0 if it was an instruction or data read. Note that for the newest IDT embedded controller, the R3081, DIAG_1 is defined during writes, yielding cache information for "STORE" instructions. A second version of the disassembler, "DIS_3081", exploits this feature for external cache support. By defining the DIAG_1 pin during writes, the CPU will signal whether the data being written was retained in the on-chip data cache. Keep in mind that the DIAG_0 pin remains undefined during write cycles. This information is extremely helpful to the programmer to determine the processor's state when tracing through the software.

The diagnostic pins on the '385 board are **NOT LATCHED**, and therefore are time-multiplexed pins. Thus, the user must either latch these pins with an external latch as seen in Figure 2 or proper decoding of cached code, or connect both diagnostic pins to GND. Although the disassembler is capable of interpreting the bus transactions of cached code, keep in mind that all logic analyzers and disassemblers can only capture external CPU memory accesses. The R3051 has large internal caches, and is capable of running much of its code from within. In order for the disassembler to accurately reflect the entire instruction/data flow, the R3051 must be ran uncached. For more information regarding running cached code and data, please refer to Application Note AN-93 for a complete discussion.

LOADING AND RUNNING THE DISASSEMBLER

Included in the software package are two files. The first is the disassembler application "DIS_305x". The second is the setup file, "DIS_305x_E", containing all the state and timing information required by the disassembler, as well as the assigned pod connections expected by the HP16500 for the R305x target system.

After the HP operating system boots up completely, the system configuration screen as shown in Figure 3 should be displayed. To load the disassembler into the HP16500, the following steps must be taken:

1. Insert the disassembler diskette into the front disk drive.
2. Select the "Configuration" field as shown in Figure 3. A pop-up menu with options will appear. Choose the "Front Disk" under the pop-up menu.
3. A new screen will appear that looks like Figure 4. Select the "Load" and "State/Timing" fields, and load in the configuration file "Dis_305x_E" by selecting "Execute" as shown in Figure 4.

The HP16500 will then load the disassembler, as well as all the state and timing information and the expected pin-configuration as shown in Table 1 previously. Once the disassembler application and setup files are loaded into the HP, the logic analyzer is ready to set trace conditions for data acquisition.

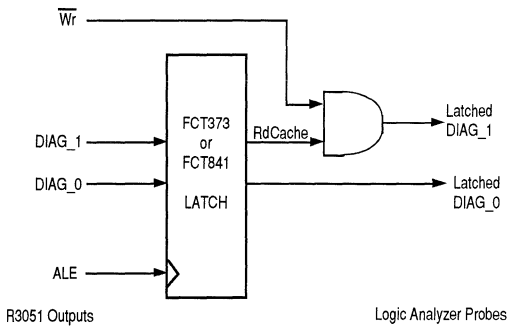


Figure 2. R3051 Address/Data Trace List on a Logic Analyzer

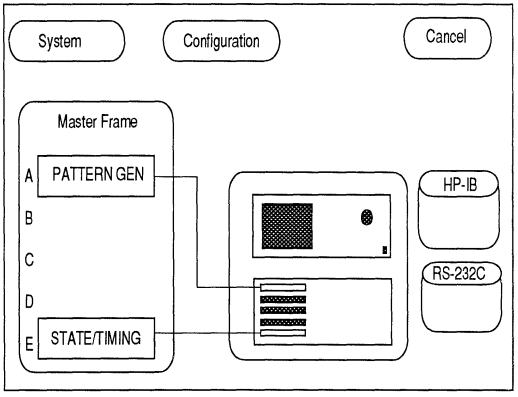


Figure 3. HP16500 Screen Display

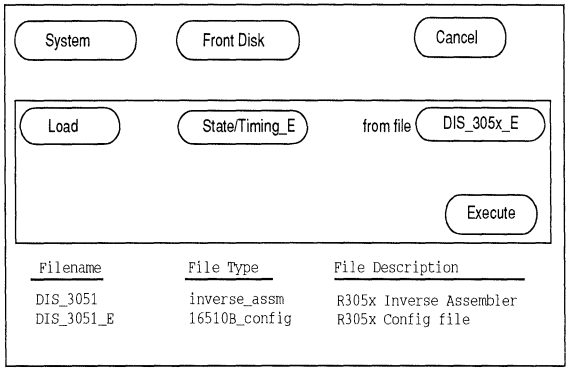


Figure 4. HP16500 Load Screen Display

With the application files loaded, the disassembler is almost ready to be triggered by the target system. Follow the steps below that describe how to run and trigger the disassembler package:

1. Select the "System" field as shown in Figure 4. A pop-up menu will appear with the option of "State/Timing". Choose this field to enter the state and timing mode of acquisition.
2. A new window will appear that is shown in Figure 5. Under the "Configuration" menu lies options that allow the user to set display or change the current configuration of the interface, clocks, and pod connections.
3. Trigger the HP16500.

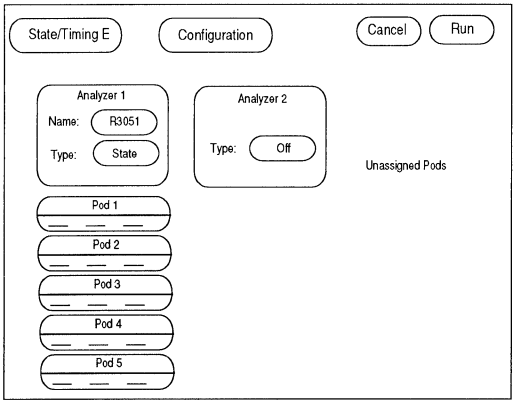


Figure 5. HP16500 State/Timing Mode Display

Once triggered, the logic analyzer will begin its acquisition, and go directly to the "Listing" field. The addresses and disassembled data will be displayed. Note however that the displayed disassembly may be incorrect. This is due to an "unsynchronized" system. The captured data needs to be synchronized with the logic analyzer's display to insure correct disassembly of the bus. The problem of unsynchronized captures arises due to the incomplete status of the processor state for data loads. As a result, when an instruction fetch is scrolled to the top of the screen, and a load data is displayed, but the corresponding load instruction was "cut off" or scrolled off the screen, the disassembler software looses it reference point by which it identifies the load data. As a result, the load data may be decoded incorrectly as an instruction as seen in Figure 6. Notice in this Figure the instruction on line -2. It was disassembled as an instruction instead of as a data load. Also notice the address of the instruction in the sequence of the four word fetch to main memory. This is an unsynchronized display because the corresponding load instruction was scrolled off the top of the display, and due to the way the disassembler interprets and tags the load datas, the reference point was lost. As a result, the load data was interpreted and decoded as an instruction. As shown in Figure 7, the correctly synchronized system has the load instruction displayed at the top of the screen (identified by its address), and the load data is interpreted correctly.

State/Timing E | Listing 1 | Invasm | Print | Run

Markers Off

Label> Base>	ADDR Hex	R3000 Mnemonic hex	STAT Hex	Time Absolute
-3	1FC00224	NOP	0010	2.24 us
-2	**1FC00228**	**SRL t4,zero,t8**	**0010**	**3.00 us**
-1	1FC0022C	NOP	0010	3.76 us
0	1FC00230	J 0X1FC084F0	0010	4.52 us
1	1FC00234	NOP	0010	5.24 us
2	1FC00238	LW v0,0x0000(s0)	0010	6.00 us
3	1FC0023C	NOP	0010	6.76 us
4	00000000	STORE DATA 0xAAAA5555	0000	7.40 us
5	1FC00240	LW t1,0x0000(v0)	0010	7.88 us
6	00000004	STORE DATA 0x00000000	0000	8.52 us
7	1FC00244	NOP	0010	9.00 us
8	00000000	LOAD DATA 0xAAAA5555	0010	9.64 us
9	1FC00248	B 0x1FC00258	0010	10.32 us

Figure 6. Incorrectly Synchronized Capture (Note line -2)

State/Timing E | Listing 1 | Invasm | Print | Run

Markers Off

Label>	ADDR	R3000 Mnemonic	STAT	Time
Base>	Hex	hex	Hex	Absolute
-4	1FC00220	LW v0,0x0008(s0)	0010	2.24 us
-3	1FC00224	NOP	0010	2.24 us
-2	**1FC00228**	**LOAD DATA 0x12620003**	**0010**	**3.00 us**
-1	1FC0022C	NOP	0010	3.76 us
0	1FC00230	J 0X1FC084F0	0010	4.52 us
1	1FC00234	NOP	0010	5.24 us
2	1FC00238	LW v0,0x0000(s0)	0010	6.00 us
3	1FC0023C	NOP	0010	6.76 us
4	00000000	STORE DATA 0xAAAA5555	0000	7.40 us
5	1FC00240	LW t1,0x0000(v0)	0010	7.88 us
6	00000004	STORE DATA 0x00000000	0000	8.52 us
7	1FC00244	NOP	0010	9.00 us
8	00000000	LOAD DATA 0xAAAA5555	0010	9.64 us
9	1FC00248	B 0x1FC00258	0010	10.32 us

Figure 7. Correctly Synchronized Capture (Note line -2)

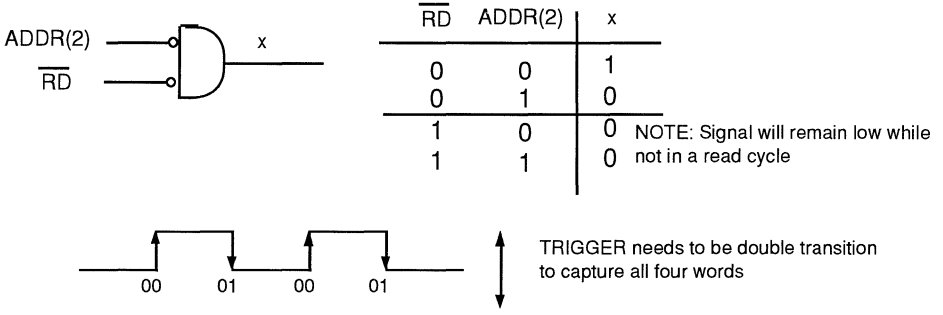


Figure 8. Simulated $\overline{\text{RDCEN}}$ signal

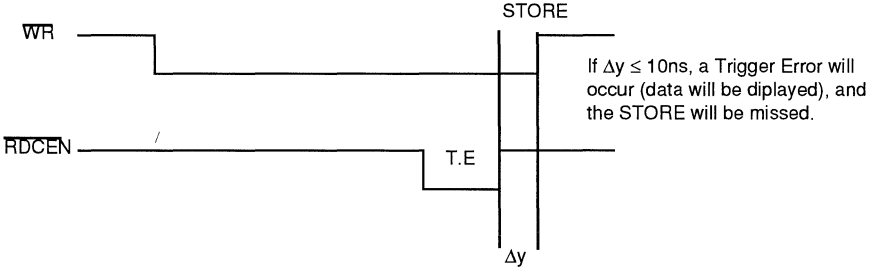


Figure 9. $\overline{\text{RDCEN}}$ Asserted during STORE

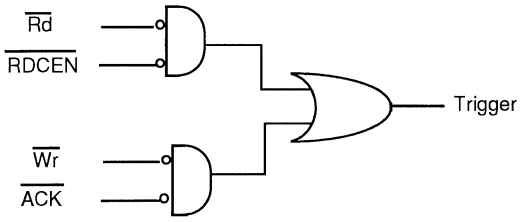


Figure 10. Simple Trigger Logic

To synchronize the system and to insure valid results, the following steps must be taken:

1. Identify the first instruction fetch by its address, not its displayed mnemonic, of the captured data and scroll this line to the top of the screen display.
2. At the top of the HP16500 screen is the field "Ivasm". Select this, and the currently displayed capture will be synchronized.
3. Always make sure that each new capture, or a jump ahead in the analyzer's buffer memory is re-synchronized properly or erroneous data might be displayed. The same applies for any move backwards for any displayed capture.

HAZARDS

For interleaved memory systems that do not toggle the $\overline{\text{RDCEN}}$ four times, but rather keep it asserted, the only data to be captured during quad-word reads will be the last word of the transfer. In order to fix this, the user might wish to simulate a $\overline{\text{RDCEN}}$ strobe during the quad-word read by utilizing the lower order address pins Addr(3:2). This can be accomplished by gating the Addr(2) pin of this 2-bit bus with the $\overline{\text{RD}}$ signal from the CPU. Whenever the next word in the sequence comes across the bus during a read cycle, the transition from LOW-to-HIGH, or HIGH-to-LOW will begin an acquisition, and thus simulate the strobbing of $\overline{\text{RDCEN}}$. Note however, the trigger transition on the HP must be set to both rising and falling transitions as seen in Figure 8.

Another hazard to be cautious about is if the $\overline{\text{RDCEN}}$ comes at precisely, or within a 10ns window (Δy) of the rising edge of the $\overline{\text{WR}}$ signal. If so, then this would be regarded as an invalid write with a trigger error (T.E) ocurring and the data on the bus at the time of the invalid capture will be displayed. In this case, the capture on the rising edge of write will be missed and the data displayed with the T.E. is the valid capture as shown in Figure 9. During any case that a $\overline{\text{RDCEN}}$ comes in on a write cycle, a T.E. will occur.

Finally, a feature in HW that would be extremely useful for triggering is a specified trigger signal for the HP logic analyzer that would distinguish between the status of reads and writes triggered by $\overline{\text{ACK}}$. The trigger would simply be established by gating the read and write signals and ORing the results as shown in Figure 10. This should eliminate any trigger edge problems associated with simple data acquisitions for inverse assembly.

SUMMARY

The use of the HP16500 and the IDT7RS364 Disassembler helps to ease the task of software development and debugging on the R305x and the R3081. The disassembler formats logic analyzer state traces into assembly level mnemonics to allow easier user interpretation. It is one of the many useful development tools already available for IDT's MIPS R3000 compatatible CPUs. Similarly, other R3000 software, compilers, as well as other development tools such as the IDT7RS901 IDT/sim ROMable Kernel/Boot Monitor can also be used on R3051 and R3081 systems with little or no modification.

Integrated Device Technology, Inc.

IDT79R3081™ PERFORMANCE ANALYSIS

APPLICATION NOTE AN-112

By Samuel Y. Shen

INTRODUCTION

IDT79R3081™ is a powerful, high-integration MIPS®-compatible processor that combines the R3000A RISC CPU, R3010A FPA, 16kB instruction cache and 4kB data cache (dynamically configurable to 8kB I/8kB D), and 4-word deep read and write buffers. It is packaged in an 84-pin PGA or MQUAD, and is available at clock rates of 20, 25, 33, and 40MHz. The R3081 is designed to bring the high performance inherent in the MIPS RISC architecture into low-cost, simplified, power-sensitive applications. The R3081 extends the capabilities of the R3051™, by integrating additional resources into the same pin-out. This new chip is aimed at two separate markets: low-cost reprogrammable systems and high-performance embedded applications. The block diagram of R3081 is shown in Figure 1.

Like the R3051 and R3052, the R3081 is available in versions with or without the on-chip MMU (i.e. E or non-E version). In addition, the R3081 incorporates a number of design improvements which include: an optional half-frequency bus interface with support for low-cost, low-speed memory systems with high computational throughput, user configurable data-cache refill size, hardware-cache coherency support, etc. This applications note provides some guidelines for quantifying the performance available from the

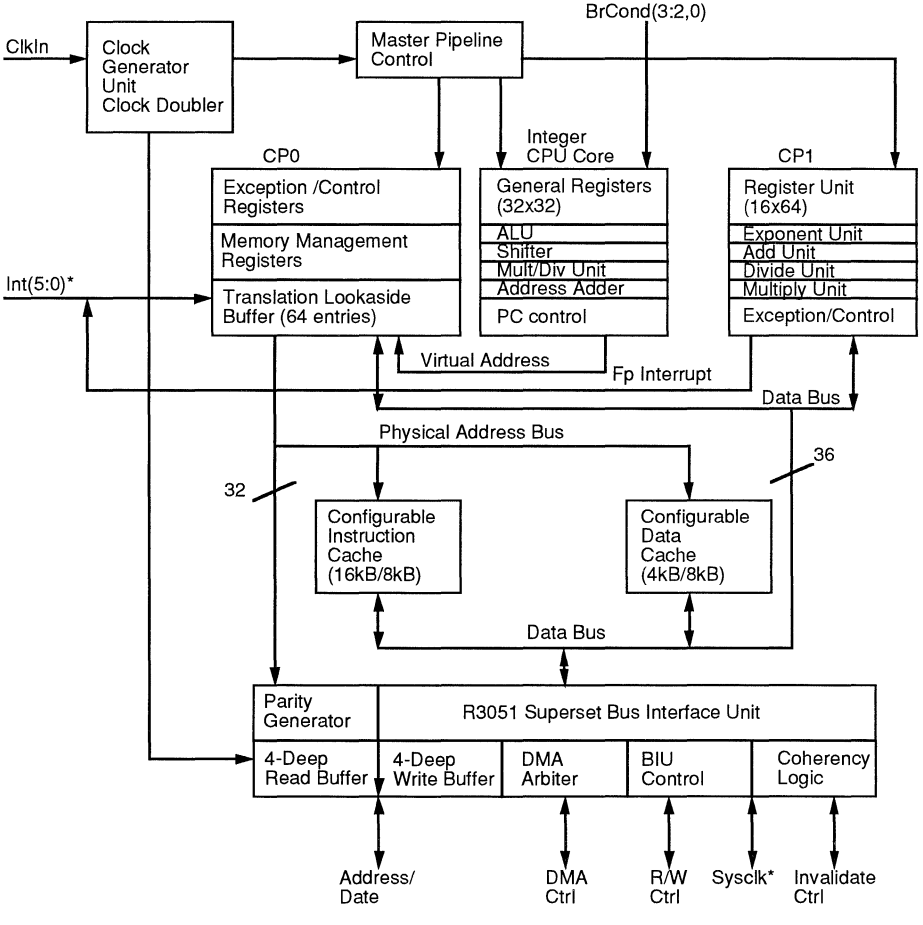


Figure 1. The IDTR3081 Block Diagram

R3081. The performance evaluation of the R3081 will be presented based on the standard embedded, integer, floating-point, and the SPEC benchmarkings. Finally, a performance comparison between an R3000/R3010-based system and an R3081-based system will be used to demonstrate the high performance of the high-end version of the R3051 family.

FAMILY OVERVIEW

This section is intended to provide a brief overview of the R3051 family, with emphasis on the R3081. For a detailed description of these devices, the reader is referred to the appropriate hardware user's manual. In addition, IDT has prepared additional applications notes describing differences between the various members of the family, and strategies to insure system upgradeability. This technical literature is available from your local IDT sales representative.

On-Chip Caches

The R3051 family achieves its high standard of performance by combining a fast, efficient execution engine (R3000A) with high-memory bandwidth, supplied from its large internal instruction and data caches. These caches insure that the majority of processor execution occurs at the rate of one instruction per clock cycle, and serves to decouple the high-speed execution engine from slower, external memory resources. The R3051 family caches are direct-mapped. This mapping coupled with the large cache sizes resident on the R3051 family, achieve extremely high hit ratios (both instruction and data) while maximizing speed and minimizing complexity and power comsumption. The R3051 family on-chip caches are indexed with physical addresses. Therefore it does not need to worry about cache flush on context switch.

As to the write policy, the R3051 family utilizes a write through strategy. This means, whenever the processor issues a write operation to memory, then both the cache and main memory are written. If it is an uncacheable reference, then only memory is written. Write through has the advantage that main memory has the most current copy of the data. Also, write through is easier to implement than write back. In the R3051 family, the on-chip 4-word deep write buffer is used to allow the processor to continue execution while the memory is updated. This optimized write buffer implementation efficiently reduces write stalls, a common disadvantage of the write through policy.

The line size of a cache refers to the number of cache elements mapped by a single TAG element. In the R3051 family, the instruction cache line size is 16 bytes, or 4 words, and the data cache line size is 4 bytes, or 1 word. The reason to have 4 words for instruction and 1 word for data is because the instructions typically execute sequentially. Thus, there is a high probability that the instruction address right after the current one will be the next instruction according to the principle of locality. Using a larger line size allows more instructions to be stored on-chip with equivalently fewer memory bits.

The current family offers a variety of different cache sizes. The R3051(E) contains 4kB of instruction cache and 2kB of data cache, the R3052(E) contains 8kB of instruction cache and 2kB of data cache while the R3081(E) doubles the R3052's caches, providing a 16 kB instruction cache and a 4kB data cache. The R3081's caches can also be dynamically configured as 8k each for instructions and data, so software can select the most effective organization. Later in this applications note, benchmarks will be shown which illustrates the impact of cache organization on performance.

R3081 FPA

Unlike the R3051/52, the R3081 contains an on-chip Floating-Point Accelerator (FPA), which operates as a coprocessor for the R3000A integer processor and extends the instruction set to perform arithmetic operations on values in floating-point representations. The FPA, with associated system software, fully conforms to the requirements of ANSI/IEEE Standard 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic." In addition, the MIPS architecture fully supports the standard's recommendation.

The R3081 allows the on-chip FPU interrupt to be internally connected to any of the CPU's six interrupt inputs. This can be done by software.

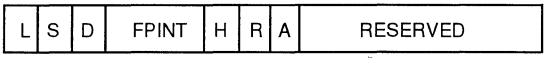
R3081 Additional Enhancements

Although with the bigger cache and on-chip FPA, the R3081 also incorporates some other design improvements. For example, the data cache block refill size can now be dynamically set to either 1 or 4 words. (Again, this can be done by programming the unique configuration register, shown in Figure 2.) In the R3051/52, the refill size is selected at reset and can not be changed dynamically.

The R3081 bus interface was modified to allow an external bus master to invalidate selected cache lines. This cache invalidation function is intended for DMA operations, and can result in a net system throughput improvement.

A 1x clock input mode is provided in the R3081. The R3081 can use either a 2x clock or a 1x clock as an input clock. Thus, a 40MHz R3081 can be plugged into an existing 20MHz R3051 design. Half-frequency mode has been included in the R3081. Again, when enabled, the bus will operate as for an R3051 operating at half the frequency of the R3081 CPU.

Power consumption reduction is a big issue for today's microprocessors especially in the embedded system and notebook market. Therefore, a halt mode and a reduce frequency mode is incorporated in the R3081 to reduce power consumption when the processor is idled. To enable these modes, appropriate bits in the configuration register needs to be set by software.



L: lock bit
S: slow bus b
D: data block refill size bit
FPINT: floating-point interrupt bit
H: stall processor bit
R: reduce frequency bit
A: cache configuration bit
RESERVED: must be zero

Figure 2. R3081 Configuration Register

Additionally, the R3081 provides a slow bus turnaround mode. This mode will allow extra cycles added between changes in A/D bus direction. This helps to eliminate high-speed buffers from the system design, by allowing more time for memory to tri-state.

UNIX® BENCHMARKS

The UNIX benchmarks consist of eight benchmarks which are briefly described below. These benchmarks are significant in that they will stress even the relatively large caches present on the R3081, thus providing better insight into the performance gain achievable in large (real-world) applications. Typical "standard" benchmarks are typically too small to stress the R3081 caches, and thus do not provide such representative results.

The various benchmarks shown are:

idtc-cc1 (v3.5)—The executable GNU 'C' compiler program modified to generate code for the IDT/C cross assembler. This benchmark uses input file "cca16355.cpp" with the "-quiet -dumpbase st.c -O -version" enabled.

idt-cpp (v3.5)—The executable GNU 'C' pre-processor program compliant with the ANSI 'C' standard. This bench-

Table 1. Execution Times and Cache Misses for UNIX Benchmarks Running on the R3051 Family

Benchmarks \ Cache		3051-33 i=4/d=2		3051e -33 i=4/d=2		3052 -33 i=8/d=2		3052e -33 i=8/d=2		3081 -33 i=8/d=8		3081 -33 i=16/d=4		3081e -33 i=8/d=8		3081e -33 i=16/d=4	
		I	D	I	D	I	D	I	D	I	D	I	D	I	D	I	D
CC 1	Sec	4.69		4.78		4.04		4.12		3.72		3.35		3.81		3.42	
	Miss Ratio	9.52	13.40	9.62	14.00	6.33	13.40	6.42	13.40	6.33	6.97	3.80	10.06	6.42	7.50	3.85	10.60
COMPRESS	Sec	159		166		159		165		148		153		153		159	
	Miss Ratio	0.02	14.36	0.02	15.52	0.01	14.36	0.01	15.52	0.01	10.53	0.01	12.18	0.01	11.22	0.01	13.08
CPP	Sec	36.8		37.1		34.5		34.8		30.0		29.6		30.2		29.9	
	Miss Ratio	1.97	5.76	1.98	5.79	1.15	5.76	1.16	5.81	1.15	1.83	0.36	3.57	1.16	1.86	0.36	3.60
DIFF	Sec	9.92		9.93		9.92		9.93		8.79		9.07		8.80		9.08	
	Miss Ratio	0.00	17.27	0.00	17.29	0.00	17.27	0.00	17.29	0.00	9.96	0.00	11.90	0.00	9.98	0.00	11.93
DIS	Sec	1.43		1.44		1.00		1.07		0.90		0.86		0.92		0.87	
	Miss Ratio	9.93	9.18	9.93	9.19	3.69	9.18	3.69	9.19	3.69	3.61	2.19	6.80	3.69	3.61	2.19	6.88
GREP	Sec	560		560		560		560		557		558		557		558	
	Miss Ratio	0.00	0.27	0.00	0.27	0.00	0.27	0.00	0.27	0.00	0.01	0.00	0.06	0.00	0.01	0.00	0.06
NM	Sec	130		131		92		92		85		79		86		80	
	Miss Ratio	9.97	5.22	9.97	5.24	3.90	5.22	3.90	5.24	3.90	1.97	2.20	3.17	3.90	1.99	2.20	3.19
YACC	Sec	5.38		5.38		5.24		5.25		4.61		4.99		4.61		4.99	
	Miss Ratio	0.42	19.79	0.42	19.79	0.12	19.79	0.12	19.79	0.12	10.45	0.05	16.42	0.12	10.45	0.05	16.42

NOTE:
Miss ratio — instruction miss ratio = total instruction miss / total instruction number *100%
— data miss ratio = total data miss / total load instruction *100%
— the marked area means the best configuration based on the execution time

mark has an input file "st.c" with the "-v -undef -D__GNUC__ -D__CHAR_ UNSIGNED__ -D__OPTIMIZE__ -DR3000 -DLANGUAGE_C" options enabled.

compress—A BSD4.3 data compression file. "Esp_pixie" is an input file for this benchmark.

diff—A BSD4.3 differential file and directory comparator. This benchmark compares two files f1 and f2 to see their differences.

dis—Dis disassemble object files into machine instructions. Standard is the object file in our example. -h and -S are specified to print general register names and source listings.

grep—A BSD4.3 UNIX function. It searches a file for a pattern. ATAN was used as a pattern to search file grepinput.

nm—A name list dump of MIPS object file. Again, standard file was used as an input.

yacc—A standard compiler-compiler type language. Yacc converts a context-free grammar into a set of table for a simple automaton which executes a parsing algorithm. Yaccinput is the input file for this benchmark.

During the tests, pseudo-FPA was added to the R3051(E)/52(E) to be able to eliminate the FPA factor; that is, if the program requested an R3010 FPA operation, the performance shown assumes an FPA was available (that is these results are as for an R3081 with its on-chip cache size reduced). This is done to force a cache-effect only shown in this result. Table 1 is the summary of the test result.

These results basically illustrate the cache issues playing a critical role on the microprocessor performance. In real-world applications, larger caches can have a substantial impact on system performance. In this example, bigger caches and dynamical configuration are pushing the performance of the R3051 family up 15%.

In addition, Table 1 illustrates the unique and important dynamic configuration effect. It can be noticed that the 16KI/4KD cache configuration provides the best hit ratios for IDTC-cc1, IDTC-cpp, dis, and nm, while 8KI/8KD is more suitable for the other four programs. Note that the same profiling tool used to obtain these results is also available to system designers attempting to tune the performance of their application.

Table 2. Stanford Benchmark Test Result

Benchmark	R3081 System	RC3240
Perm	0.063	0.090
Towers	0.066	0.068
Queen	0.047	0.045
Intmm	0.052	0.054
Puzzle	0.047	0.050
Quick	0.346	0.309
Bubble	0.047	0.047
Tree	0.054	0.055
FFT	0.089	0.094
Mm	0.083	0.086

STANDARD BENCHMARKS

There are a number of popular benchmarks that are commonly used to compare processor performance. Five of the most popular are the Stanford, Dhrystone, Linpack, Whetstone, and SPEC suite programs. Embedded system performance is addressed by the Stanford benchmark. Integer performance is addressed by the Dhrystone benchmark. Floating-point performance is addressed by the Linpack and Whetstone programs. As to the SPEC suite, SPECmark is used as a standard performance index nowadays for most UNIX processor comparisons. The main memory used on the R3081-based simulation system is 80ns DRAM, in a two way interleaved configuration. This is viewed to be a fairly realistic design around this processor (no zero wait state SRAMs).

Stanford Benchmarks

This is a suite of benchmarks that are relatively short, both in program size and execution time. It requires no input, and prints out the execution time for each program, using the system-dependent routine Getclock to find out the current CPU time. It does a rudimentary check to make sure each program gets the right output. This suite consists of ten different benchmarks which covers both integer and floating-point operation, as described below:

perm—Computes permutations of seven elements five times. Heavy use of arrays and procedure calls.

towers—Solves Towers of Hanoi for fourteen disks. Heavy use of recursive procedures.

queen—Solves the eight queens problem fifty times. Extensive use of both loops and recursion with backtracking.

intmm—Multiplies two 40x40 integer matrices. Entirely limited by integer multiply time.

puzzle—Forest Baskett's program solves a Soma Cube type problem. Heavy use of small, tight loops.

quick—Performs a quick sort of 5000 elements. Tests recursion and array indexing.

bubble—Reads a file and does a bubble sort of 500 elements. Heavy use of array manipulation.

tree—Performs binary tree sort of 50000 items. Heavy use of pointers, dynamic data structures.

fft—computes a 256-point Fast Fourier Transform twenty times. (This is an FP benchmark.)

mn—multiplies two 40 x 40 single-precision matrices. (This is an FP benchmark.)

Table 2 illustrates the separate execution time for the Stanford benchmark suite. Numbers for the 25MHz MIPS RC3240 system are also given for comparison issues. This system is a server based on the R3000A CPU, R3010A FPA, and external caches. More description of this system will be given later in this AP note.

Dhrystone Integer Benchmark

Dhrystone is a CPU-intensive synthetic benchmark consisting of a mix of higher level language instructions. Dhrystone has become a de facto standard measure of integer performance. In the synthetic benchmark program, 100 statements are dynamically executed between the comment lines "start timer" and "stop timer". The statements are balanced with

Table 3. Dhrystone 1.1 Benchmark Test Results

Benchmark	Icache Miss	Dcache Miss	Result
Dhrystone	0.00%	1.52%	44,052 Dhry.

Table 4. Linpack/Whetstone Benchmark Test Results

Benchmark	Icache Miss	Dcache Miss	Result
Linpack	0.01%	12.67%	2.6 Mflops
Whetstone	0.01%	0.00%	11,764 Whet.

regard to statement types, data types, and data locality. Dhrystone does not contain any floating-point data or operations.

The benchmark also does not make any system calls. However, its performance is highly dependent on two C library functions, strcpy() and strcmp(), which represent about 25% of computation.

Owing to the above attributes, Dhrystone performance can be significantly impacted by compiler techniques and C library implementation. There is a separate applications note which mentions the common pitfalls when benchmarking with the 7RS385 (an R3051/52 evaluation board) especially with Dhrystone program.

Another problem in using Dhrystone to measure performance has to do with the size of its executable. Dhrystone achieves a virtually 0% miss rate even in the R3051; thus, the benefit of the larger caches in the R3081 will not be adequately displayed by this benchmark.

Table 3 reflects Dhrystone v1.1 running on an R3081-25-based system. Again, however, IDT does not consider the Dhrystone benchmark to be an adequate yardstick for modern microprocessor performance, due to its unusual reliance on two library functions and due to its small size. Further, the rules for Dhrystone benchmarking (e.g. no procedure inlining) do not allow the true capabilities of the MIPS compiler suite to be demonstrated.

Linpack/Whetstone Floating-Point Benchmarks

Written in FORTRAN, Linpack is a general-purpose mathematical library of functions that solves systems of linear equations. The Linpack benchmark is a program that solves a dense system of linear equations using a small subset of the standard Linpack library functions. As a linear-equations package, Linpack emphasizes floating-point addition and multiplication. The results, measured in millions of floating-point operations per second (Mflops), are typically derived from a calculation of a 100 x 100 submatrix of linear equations.

Whetstone is a synthetic mix of integer and floating-point calculations, transcendental functions, conditional jump, function calls and array indexing. This benchmark was originally developed in 1970 and was written in ALGOL 60s. Since that time, it has been rewritten in FORTRAN and, like Linpack, has evolved into a standard benchmark of floating-point performance. Results display in thousands or millions of Whetstone interpreter instructions per second (Kwhips or Mwhips, sometimes referred to as MegaWhetstones).

Table 4 shows the Linpack and the Whetstone number (double precision only) coming out of the R3081 simulated system. The system parameters are as described earlier.

SPECmarks

SPECmark is the geometric mean of the SPEC® benchmark suite. Compared with the arithmetic mean (average), the geometric mean is a fairer way of reporting suite results because it compensates for varying run lengths while giving each program equal importance.

The SPEC benchmark suite includes ten different programs drawn from real-world applications and other scientific and engineering areas. These programs are described briefly below:

gcc—the GNU C compiler distributed by the Free Software Foundation. This benchmark measures the time it takes for the GNU C to convert 19 preprocessed source files into optimized SUN-3 assembly language (.s file) output.

espresso—one of a collection of tools for the generation and optimization of Programmable Logic Arrays (PLAs). This benchmark was developed by UC Berkeley. It takes a set of seven input models which are represented as truth tables and produces the same format outputs.

spice2g6—an analog circuit simulation and analysis application. This benchmark was developed by UC, Berkeley also. It takes an input model from HP that simulates a bipolar circuit.

doduc—a Monte Carlo simulation of the time evolution of a thermohydraulical modelization for a nuclear reactor's component.

nasa7—a collection of seven floating-point intensive kernels. The input data is double-precision.

li—a lisp interpreter written in C. This benchmark measures the time to solve the 8-queens problem.

eqntott—an integer intensive benchmark developed by UC, Berkeley. This benchmark translates a logical representation of a boolean equation to a truth table.

matrix300—a vectorizable FORTRAN scientific benchmark using double-precision floating-point arithmetic.

fppp—a quantum chemistry benchmark. It measures performance on one style of computation which occurs in the Gaussian XX series of programs.

tomcatv—a highly vectorizable double precision floating-point FORTRAN benchmark.

Our purpose for running the SPEC benchmarks is merely to compare the performance of a typical desktop R3081-based system with the performance of a discrete R3000A system. Note that compiler technology continually advances; this has been demonstrated by the recent breakthroughs in performance of the matrix300 benchmark by MIPS and others, using compiler techniques. The results of these techniques are obviously not included in this table, and thus this table should not be construed as an absolute system performance indicator. Rather, it should be used to relate the performance of the R3081 to an existing, available, 25MHz UNIX workstation/server.

Table 5. SPECmark of R3081/R4000

Benchmarks		R3081-25	RC3240
Integer	gcc	17.2*	16.6
	espresso	16.1	18.4
	li	15.7	20.3
	eqntott	19.0	17.5
Floating Point	doduc	13.8	16.4
	nasa7	19.2	17.1
	spice2g6	13.1	12.4
	tomcatv	16.7	14.1
	fpppp	13.4	20.5
	matrix300	10.5	8.8
Geometric Mean	SPECmark	15.2	15.8
	Integer only	17.0	18.1

* This is an approximate value and for reference use only.

Table 5 illustrates the detailed SPEC results, comparing simulated SPEC mark estimates for the R3081 with the RC3240. For simulation, an estimate of the overhead for the operating system (especially important to gcc) has been included but not actually measured.

R3081-BASED SYSTEM VS. RC3240 (AN R3000- AND R3010-BASED SYSTEM)

The MIPS RC3240 RISC computer is a 25MHz R3000/R3010-based system. These processing units are complemented by large, high-speed caches of 64kB each for instructions and data, and by sophisticated read/write buffers for minimizing memory access overhead. The resulting processing power is measured at over 18mips, 15.0SPEC marks, 40,000 Dhrystones, 13,800 Whestones for double precision, 17,100 Whetstones for single precision, 3.1Mflops Linpack for double precision, and 5.9Mflops Linpack for single precision.

The RC3240 requires a CPU, FPA, 30-device SRAM cache, and a number of logic devices to implement the CPU subsystem. The R3081 merely requires a single, 84-pin monolithic device. While the cache sizes are smaller than the discrete external caches used in the RC3240, the cache are large enough to handle a wide variety of real-world applications with high-performance. Further, memory design techniques such as interleaving on the mother board serve to mitigate the disadvantage of smaller caches. The result is that the R3081 system described performs within 10% of the RC3240, while dramatically reducing device cost, count, and power consumption in the CPU.

Figure 3 shows the bar chart of the comparison between the R3081-based system and RC3240.

SUMMARY

The highly integrated R3081 was designed to take advantage of the computing power inherent in the MIPS architecture, with a priority on reducing overall system cost and design complexity. This chip, which provides an excellent cache hit ratio and a floating-point HW solution attaining 15.2 SPECmark at 25MHz, fills a vital performance niche between the standard R3000A and the R4000. Because it is software and pin compatible with the R3051, the CPU offers a upward compatibility allows the R3051/81 user to implement a single HW/SW base system that can be easily upgraded by choosing the appropriate processor to fit the target price/performance range.

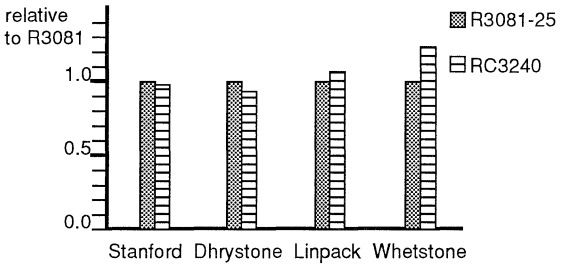


Figure 3. R3081-25 vs. RC3240

Integrated Device Technology, Inc.

UPGRADE STRATEGIES FOR IDT79R3051™-BASED DESIGNS

APPLICATION NOTE AN-113

By Phil Bourekas

INTRODUCTION

The IDT RISController™ family includes various highly-integrated microprocessors providing high levels of performance with low system cost. Currently, the R3051™ family includes three different devices, each providing differing levels of price performance, yet each pin-compatible with each other. This allows the system designer to implement a single base system, yet offer various end products at different capability levels. The end result to the customer is reduced time to market for a product family, and the amortization of a single development effort over a wider variety of end products. This wide range of pin-compatible performance is not currently achieved by any other RISC processor family.

This application note describes system design techniques that insure a high degree of interchangeability with no real design impact.

THE R3051 FAMILY

Common characteristics of the R3051 family include high integration at low cost. All current family members are pin-compatible. All family members include:

- Substantial amounts of separate instruction and data caches integrated on-chip. Although the amount of caches varies across different family members, all devices contain enough cache on-chip to achieve extremely high performance with low-cost memory systems. The caches on the R3052 and on the R3081™ are actually larger than the cache on the Intel 80486 high-end processor, enabling these devices to offer higher performance at lower cost.
- MIPS R3000A compatible integer CPU. The R3051 family was designed by integrating cache and a low-cost bus interface around the standard MIPS R3000A CPU. This RISC core is widely recognized as an extremely high-performance execution engine, with powerful compiler and development tools. Some of the features of the core include a large register file, single cycle ALU, rich set of branch instructions (including compare operations as part of the branch), and separate, autonomous integer multiply and divide. Since the R3051 was designed using the standard core, 100% software compatibility is guaranteed. Thus, compiler tools, real-time operating systems, and other software tools developed around the standard R3000A work without modification on the R3051 family.
- Optional Translation Look-aside Buffer (TLB). The "E" (Extended Architecture) versions of the RISController family feature a 64-entry, fully associative TLB. The TLB allows virtual addresses to be translated into physical addresses on a 4kB page basis. The TLB is useful in providing memory protection and debug utilities in any application; in other applications, such as those using a real-time operating system, or in an X-windows server, the TLB allows increased system functionality to be provided.
- Simple, low-pin count bus interface. The R3051 family uses a time-multiplexed 32-bit address and data bus to communicate with memory. Internal to the processor are 4-deep read buffer and write buffer FIFO's to decouple the speed of the internal execution core from the slower speed memory system. The multiplexed bus arrangement has many advantages, such as lower-cost interface chips and ASICs, without impacting system performance.

Currently, there are three family members. These are:

- The R3051/51E. This device features 4kB of Instruction cache and 2kB of Data Cache. There is no hardware floating-point unit available on this device.
- The R3052/52E. This device features 8kB of Instruction cache and 2kB of Data Cache. As with the R3051, there is no hardware floating-point unit available on this device.
- The R3081/81E. This device introduces a number of new features to the family. The primary features of interest are changes to the caches, and inclusion of a hardware floating-point unit; other features will be described throughout this application note. The R3081 implements 16kB of Instruction Cache and 4kB of Data Cache; kernel software can dynamically reconfigure the on-chip caches as 8kB of Instruction and 8kB of Data Cache.

POTENTIAL UPGRADE OPPORTUNITIES

A number of possible system upgrades from a single, base design are possible. Elsewhere in this application note, design considerations to assure interchangeability are described.

Possible upgrade strategies include the following techniques:

Upgrading Cache Size

As all devices are pin compatible; it is possible to increase performance of an application by upgrading the amount of cache available on-chip. Thus, holding all other components the same, an R3051 may be removed and replaced by an R3052 to double the instruction cache. An R3052 can be removed and replaced with an R3081, doubling both the instruction and data caches.

Add Hardware Floating-Point

One upgrade to higher performance involves upgrading an R3051 or R3052 to an R3081 and taking advantage of the on-chip floating-point accelerator. Later in this applications note, software considerations for such an upgrade are described.

This upgrade will obviously substantially increase the performance of software containing floating-point operations; while the IDT software floating-point environment is very efficient, the floating-point unit of the R3081 dramatically outperforms integer emulation, and may result in a significant speed-up of some applications.

Increasing Frequency

Obviously, one way to increase performance is to increase the system frequency. This may or may not be easy to do, depending on the exact system design. Obviously, such an upgrade will typically require the replacement of multiple devices on the PCB.

Note, however, that R3051 family packaging insures that the same footprint and pinout is available across the full frequency range of the family, and for all of the family members. Thus, the same 84-pin PLCC footprint used for a 20MHz R3051 accommodates the package for a 40MHz R3081, even though that device consumes more power. This obviously simplifies upgrading a design to a higher frequency processor. Design techniques for increasing frequency may include:

- Using faster memory devices to achieve the same relative access time.
- Using faster control logic, such as faster PALs or transceivers, to increase set-up time and reduce propagation delays. For example, a 15ns PAL may be replaced with a 10ns PAL, effectively allowing the clock period to be reduced 5ns.
- Re-programming PALs and control logic to increase the number of wait cycles. While this will reduce the frequency normalized performance, the absolute performance will be increased substantially, since the processor will execute (typically out of its internal cache) at a higher rate.

"Clock Doubler" Operation

The R3081 presents a particularly unique opportunity to upgrade systems using an R3051 or R3052. This is particularly due to the "half-frequency bus" mode of operation of the R3081.

A dramatic system upgrade can be achieved by:

1. Removing a 20MHz R3051 or R3052 and replacing it with a 40MHz R3081.
2. Selecting the "half-frequency bus" and "1x clock" modes via the reset vectors.

The resulting system bus will continue to operate at 20MHz, but the CPU will execute out of its internal cache at 40MHz. The resulting system will typically see its performance more than double (recall that the upgrade to the R3081 will also increase the on-chip caches and add hardware floating-point, relative to the R3051 or R3052).

It is also interesting to note that the performance impact of running a 40MHz processor with a 20MHz bus is not as severe as one would intuitively guess. This is due to the fact that memory access time is really in units of time, rather than in wait states. That is, 200ns access memory is 4 clock cycles at 20MHz and is 8 cycles at 40MHz; the absolute time is not improved by running the bus faster.

Intel has estimated that for the i486 with clock doubling, running the bus at one-half the CPU execution rate is approximately 11% less efficient than running the bus at the full CPU rate on benchmarks such as the SPEC benchmark suite. The R3081 contains more than twice the amount of on-chip cache as does the i486, and thus will be even less dependent on bus performance; thus, the performance degradation should be even less.

DESIGN CONSIDERATIONS FOR UPGRADING

The remainder of this applications note details specific techniques which facilitates the interchange of various members of the R3051 family. In general, all devices are pin and footprint compatible, so there are no PCB issues to be concerned about. In general, the only things needed to upgrade a design are:

- Design it around an R3051. The R3081 does include some superset features relative to the R3051 which simplifies high-speed systems; however, if a system works for the R3051, it will work for an R3081.
- Make the software independent of cache size. The various devices include varying amounts of cache on-chip. An algorithm to determine the amount of cache available is presented in this applications note.
- Have a strategy for software floating-point versus hardware floating-point. The R3081 adds a high-performance hardware floating-point accelerator, as well as increasing the cache size. This applications note describes various software techniques for dealing with software emulation versus hardware acceleration of floating-point.

Thus, this application note details specific hardware choices and software choices which facilitate interchanging CPUs. In addition, the application note illustrates techniques for determining the presence or absence of the R3081 config register, the R3081 FPA, and the amount of cache on-chip.

SOFTWARE CONSIDERATIONS FOR UPGRADING SYSTEMS

Some of the system upgrade considerations should be accommodated in the application software (especially the kernel). It is possible to develop a single binary set of code which performs across all of the family members.

Sensitivity to Cache Size

Obviously, one characteristic difference among the various family members is the amount of Instruction and Data cache available. Thus, to insure interchangeability among these devices, the software should be written to be insensitive to the cache sizes.

Typically, very little of the actual application will be functionally sensitive to the amount of on-chip cache; the primary difference will be in the performance achieved. This is the primary advantage of caches with respect to memory mapped zero-wait state RAM; caches are transparent to the software, and do not affect the memory map.

Typically, the only part of the software that may be sensitive to the cache size will be the boot/initialization software, which may perform certain memory (including on-chip cache) diagnostics, and which must initialize the on-chip cache by performing a cache flush.

Figure 1 shows a listing of a routine to perform cache sizing. This routine uses bits of the on-chip status register to isolate the cache (to prevent writes or cache misses from propagating to memory), and to swap the cache (to perform the algorithm on the Instruction cache). In order to determine hit or miss, the algorithm places a marker in the first word of the cache, and then looking for the cache size such that a read of the cache forces a wrap-around to reading location zero. Once this occurs, the maximum cache size has been exceeded, and thus the cache size is known. Other algorithms could use the cache miss bit of the status register, rather than a marker value. This capability is provided in the IDT/kit™ and IDT/sim™ software packages from IDT.

Once the cache size has been determined, it is used in the cache flush routines (for example) to completely flush the caches. Note that if the only time the cache is flushed is at system start-up, it is acceptable to assume a worst case (large) cache size and flush that amount of cache; caches smaller than the size assumed will merely be flushed multiple times, resulting in wasted execution time but correct functionality. On the other hand, applications which perform cache flushing as part of ongoing operation (e.g. to assure cache coherency when DMA operations are used) would be sensitive to performance, and thus would desire to flush only the proper amount of cache.

Floating-Point Presence

Another difference between various family members has to do with the presence or absence of the floating-point. This distinction may have two impacts on the software environment:

- The initial setting of the coprocessor 1 usable bit should reflect whether or not a hardware floating-point is available. It is possible to create a software environment which can dynamically determine the presence or absence of the FPA.
- The actual binary executable of the application may be best optimized according to the presence or absence of a hardware floating-point. This is discussed below.

How to Determine Floating-Point Presence

There are at least two different methods for determining whether a floating-point is present. One way is to perform floating-point operations and determine whether the results are reasonable; these operations could be as simple as moving data into and out of the FPA registers to see if they are present, through performing floating-point calculations and examining the results (or even possibly seeing if an exception is reported). If the floating-point is detected as present, coprocessor 1 should be marked as usable by the kernel.

Another method would be to use the CpCond(1) (coprocessor 1 condition) flag. The hardware could tie the CpCond(1) to a known state (e.g. HIGH); software could then perform a compare operation (or move to the fp cscr register) to cause CpCond(1) to report the opposite polarity. A simple branch on coprocessor (1) condition will then determine whether the CpCond(1) signal is driven by an on-chip FPA, or by the off-chip pull-up resistor.

FPA Impact on the Binary Code

There are two methods for dealing with the software which may or may not have a hardware floating-point unit. The optimal method depends on trade-offs between a single binary set operating either with or without a hardware FPA, versus a single source set compiled twice resulting in two binaries (one targeted to a hardware FPA and one targeted to an integer only environment).

Using a Single Binary with and Without an FPA

If the system designer chooses to implement a single binary capable of taking advantage of a hardware FPA when one is available, all that needs to be done is to tap into the inherent capabilities of the MIPS coprocessor architecture. Specifically, if the kernel marks the coprocessor 1 FPA as unavailable, FPA instructions will cause a trap to occur. The kernel can then perform an integer interpretation of the FPA instruction. The application software is then compiled to assume the availability of a hardware FPA: if one is available in the system fine; if not, traps will occur when FPA operations are encountered, and the kernel can perform an emulation of the function.

Using this technique requires two things in the software:

- Boot software must perform the diagnostics described above to determine the appropriate setting for the coprocessor 1 usable bit.
- The kernel must include the capability to emulate the entire FPA unit, including the FPA operations, the register file, and the FPA exception mechanisms used by the application.

While this technique has the advantage of resulting in a single binary which works in either environment, the result is added complexity and a loss of performance in the environment in which no FPA is available. Specifically, the kernel must provide an emulation library of the entire FPA; and, software FPA operations will include additional overhead from the CPU exception model and from emulating all aspects of the FPA, even though a given operation only requires a subset of the FPA functionality.

Developing Two Binaries from a Single Source

Another technique exists whereby two distinct binaries are developed from a single source tree. Each of the resulting binaries is fully optimized for either an integer only environment, or for an environment in which a hardware floating-point is available.

This is accomplished by taking advantage of the software floating-point library capabilities of the IDT/c™ environment. IDT/c includes a compile time flag which can be used to control whether hardware FPA instructions (coprocessor 1 instructions) are generated, or whether direct calls to a software floating-point library are generated. Thus, software floating-point is not forced to emulate the register set and data type conversions of the hardware FPA, and execution is not forced to go through the CPU exception model. The resulting binary operates much more efficiently than one which goes through the trap and emulation model described above.

A separate applications note describes how to determine the optimal compilation environment for a given application.

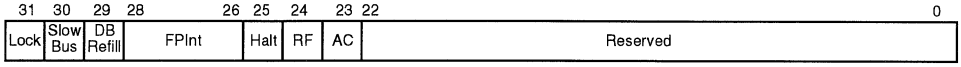
```
/*****************************************************************************
**
** _size_cache()
** returns cache size in v0
**
*****************************************************************************/

FRAME(_size_cache,sp,0,ra)
      .set        noreorder
      mfc0        t0,C0_SR                    /* save current sr */
      and         t0,~SR_PE                   /* do not inadvertently clear PE */
      or          v0,t0,SR_ISC                /* isolate cache */
      mtc0        v0,C0_SR
      /*
       * First check if there is a cache there at all
       */
      move        v0,zero
      li          v1,0xa5a5a5a5               /* distinctive pattern */
      sw          v1,K0BASE                   /* try to write into cache */
      lw          t1,K0BASE                   /* try to read from cache */
      nop
      mfc0        t2,C0_SR
      nop
      .set        reorder
      and         t2,SR_CM
      bne         t2,zero,3f                  /* cache miss, must be no cache */
      bne         v1,t1,3f                    /* data not equal -> no cache */
      /*
       * Clear cache size boundries to known state.
       */
      li          v0,MINCACHE
1:
      sw          zero,K0BASE(v0)
      sll         v0,1
      ble         v0,MAXCACHE,1b

      li          v0,-1
      sw          v0,K0BASE(zero)             /* store marker in cache */
      li          v0,MINCACHE                 /* MIN cache size */

2:    lw          v1,K0BASE(v0)               /* Look for marker */
      bne         v1,zero,3f                  /* found marker */
      sll         v0,1                        /* cache size * 2 */
      ble         v0,MAXCACHE,2b              /* keep looking */
      move        v0,zero                     /* must be no cache */
      .set        noreorder
3:    mtc0        t0,C0_SR                    /* restore sr */
      j           ra
      nop
ENDFRAME(_size_cache)
      .set        reorder
```

Figure 1. Cache Sizing Software



Lock: 1 -> Ignore subsequent writes to this register
Slow Bus: 1 -> Extra time for bus turnaround
DB Refill: 1-> 4 word refill
FPInt: Power of two encoding of FPInt <-> CPU Interrupt
Halt: 1 -> Stall CPU until reset or interrupt
RF: 1 -> Divide frequency by 16
AC: 1 -> 8kB per cache configuration
Reserved: Must be written as 0; returns 0 when read

Figure 2. R3081 Config Register

The method of dealing with floating-point operations in an integer CPU only environment is particularly important in the evaluation of a compiler platform; techniques such as the "mix and match" approach supported by IDT/c allows the best capabilities of the MIPS compiler toolchain to be integrated with efficient software floating-point emulation.

The obvious advantage of this approach is the optimum performance achieved for both the integer only system and the R3081-based (hardware FPA) system. Using distinct EPROM sets at manufacturing time, or upgrading both the EPROMs and processor as a field upgrade, are obvious consequences, but in general are not particularly onerous (EPROM upgrade can be a replacement of EPROMs, or, for FLASH EPROM, a re-programming of the EPROMs resident on the board).

The R3081 Config Register

The R3081 includes, as part of coprocessor 0, an additional control register called "Config". The R3081 Config Register is shown in Figure 2.

The Config register controls various aspects of system functionality. If these features are used in an R3081 system, software must first determine whether they are available.

To determine whether the current device is an R3081 (and thus whether the config register is available), software can use various techniques. One straightforward technique is to determine whether or not there is an FPA; if so, the device is an R3081. Similarly, software could determine the cache sizes available, and see if these correspond to the organization the R3081.

Other techniques are also possible; for example, size the cache, then reconfigure the cache by writing to the config register; re-size the cache to determine that the change occurred. Obviously, if the change occurs, the config register is available.

Note that writes to this register location in the R3051 or R3052 will have no effect; no side effects occur, and no traps are signalled. Reads of the config register produce an undefined data result for the R3051 and R3052.

If the config register is used when an R3051 is in place, various other considerations exist. These are:

- *Floating Point Interrupt.* In general, if an R3051 application intends to also work with an R3081, one of the CPU interrupt inputs needs to be reserved for the hardware FPA of the R3081. The default interrupt is Int(3), but the config register allows a different interrupt assignment to be used. The corresponding interrupt input pin of the R3081 is then ignored. Thus, the PCB should contain a pull-up resistor at the interrupt pin; when an R3051 is used in the application, no interrupt will be signalled.
- *Reduced Frequency.* This mode dramatically reduces the power consumption of the R3081, by reducing its operation frequency. This mode is unavailable in the R3051. In general, the only real functional system change that occurs is that the SysClk output clock frequency is also reduced; thus, if DRAM refresh, for example, was derived from this clock, the counter value should be reprogrammed. If an R3051 is told to "reduce frequency", nothing will happen.
- *Halt.* This control bit forces the R3081 to stall until an interrupt input is asserted, or a reset is encountered. This mode is unavailable in the R3051, and no simple software equivalent exists.
- *Data Block Refill.* The R3081 allows the block size read on a data cache miss to be dynamically reconfigured by software. The initial value is set by the reset value. In general, this bit may affect the performance of software, but is unlikely to impact its functionality.
- *Alternate cache.* This bit allows the caches to be dynamically reconfigured for the R3081. A cache flush should be performed after the cache is reconfigured. An earlier section of this applications note discussed how to make software independent of the cache organization.
- *Lock.* This bit allows software to inhibit subsequent writes to the Config register. Thus, boot software can set up the operation mode, and then protect it from other software.
- *Slow Bus Turnaround.* This bit allows systems to enjoy longer time between A/D bus mastership transitions. However, this software control is not available on the R3051. If the system designer desires extra time, and also desires to be able to interchange R3051s and R3081s, the hardware technique described in applications note AN-97 is appropriate. This technique uses the DMA arbiter interface of the CPU to insure that new transactions are not begun until ample time for bus turn-off has passed. This hardware technique works equally well with both the R3051 and R3081.

HARDWARE DESIGN ISSUES

There are various hardware design considerations that may impact the ability to interchange various members of the CPU family. With proper design, these considerations can be dealt with no real system impact.

Slow Bus Turn

Bus turn is the amount of time allowed to change mastership on the A/D bus of the processor. In general, a read followed by a write can cause a change in bus direction in one-half bus cycle. At 33MHz, this is 15ns.

The system designer may implement an architecture which, by using appropriate transceivers and control signals, can tolerate a rapid bus turn. Alternatively, the designer may desire to increase the minimum amount of time.

Although the R3081 includes a bit in the Config register to slow the bus, this technique does not work with the R3051. Instead, the hardware technique of using BusReq to insure a longer tri-state time is recommended. This technique is described in applications note AN-97.

Coherent DMA

The R3081 includes a hardware interface to insure cache-coherency in systems using DMA. This interface is unavailable in the R3051.

Many MIPS applications perform multi-master cache coherency via software techniques, and thus do not require hardware-based coherency. While hardware-coherency will improve the performance of some applications, relying on software (which may, for example, flush the entire data cache once a DMA operation is completed to insure coherency. This technique will function equally well with either the R3051 or R3081.

Floating-Point Interrupt

The R3081 uses one of the interrupt input pins to report exceptions to the CPU. The hardware should reserve one of the input pins for this function, and provide logic or pull-up resistors to insure that this input is held HIGH for an R3051 or R3052.

CpCond(1)

The R3081 uses this input to report the results of comparisons back to the CPU; thus, the external input pin is ignored. R3051 systems should provide a pull-up resistor for this pin. Earlier in this applications note, a method to use this pin to determine the presence or absence of an FPA was described.

Reset Mode Vectors

Both the R3051 and R3081 use the same basic technique to perform reset mode selection of various options. Figure 3 illustrates the mode vector logic for the R3081. Note that for the R3051, Int(5:3) mode vectors are reserved, and must be held HIGH during reset.

Options include:

- *Tri-state.* This option is used to perform board testing, and is available in all devices.
- *BigEndian.* This option selects the data byte ordering convention, and is available in all devices.
- *Data Block Refill.* This option selects single versus four-word refill on data cache misses. Although this option is available in all devices, software (via the config register) can dynamically change the value for the R3081.
- *Coherent DMA Enable.* This option enables the coherent DMA interface of the R3081. For the R3051, this input must be HIGH at reset.

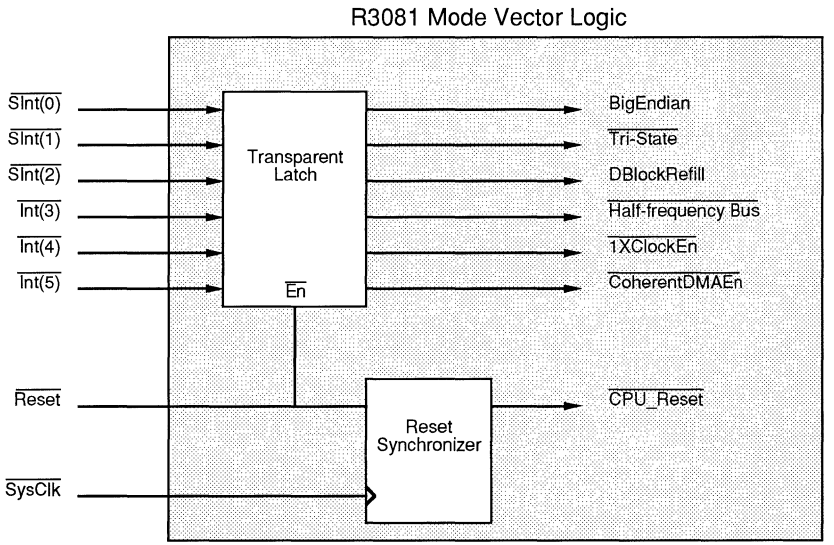


Figure 3. R3081 Mode Vector Assignment

- *1x Clock Mode.* This option instructs the R3081 that the input clock provided is at the CPU operation frequency, rather than at twice the frequency. In the R3051, only the "2x" clock is available, and this vector must be held HIGH.
- *Half-frequency Bus.* This option instructs the R3081 to operate its bus interface at one-half the execution rate. This option is unavailable in the R3051, and must be held HIGH at reset.

In order to design a system to accommodate either an R3051 or R3081, it may be desirable to include jumpers for the R3081-only options. Thus, when an R3081 is included in the design, various of the hardware options may be changed. This may open up other upgrade strategies, such as the clock doubling capability described earlier.

SUMMARY

By following a few simple rules, the system designer can implement a base R3051 system which can easily upgraded to higher performance. Upgrade options include more amounts of cache on-chip, the addition of hardware floating-point, and increases of frequency. With the R3081 half-frequency bus mode, the operation frequency of the execution engine can be substantially increased while maintaining the same (or even slower) bus interface frequency.

Thus, the IDT RISController family effectively reduces the time to market of new product families, and maximizes engineering return on investment by enabling one design effort to result in multiple end products.

Integrated Device Technology, Inc.

THE IDT RISController™ FAMILY: AN ARCHITECTURE WELL-SUITED TO X-WINDOWS

CONFERENCE PAPER CP-04

As presented at Silicon Valley Networking Conference
By V.R. Ranganath and Phil Bourekas

ABSTRACT

Although an X-Terminal is an extension of the traditional graphics terminal, the X-Windows system places additional system level constraints. These constraints require a well-balanced system architecture, not just good graphics throughput. Evidence of this is the fact that many of today's commercial X-Terminals use a dual-server architecture: a good, general purpose CPU such as an MC68020 to handle networking and I/O functions, and a dedicated graphics chip such as the TI 34010 to perform graphics functions.

Many of today's RISC microprocessors offer enough performance to integrate both functions into a single CPU, resulting in higher performance, simpler designs, and lower cost. However, the system requirements of an X-Windows application forces the system designer to evaluate more than just the raw performance of the CPU. Factors such as memory interface, memory management, and interrupt response are at least as important as computational throughput.

This paper will discuss some of the other architectural requirements of an X-Terminal system, and size up these requirements against the IDT RISController™ family. Areas of investigation will include:

- The role of memory management in X-Windows. There is a certain similarity between X-Windows systems and the traditional virtual memory system associated with general purpose computers. The paper will discuss methods of window management, using a memory management unit to provide both the client/server and server/hardware interfaces within the X-Terminal. The use of the MMU incorporated within various members of the IDT RISController family will be discussed.
- The importance of interrupt response in X-Windows. X-Windows systems are designed to interface between the network and the graphics device. Interrupt response and network handling become key determinants in X-Windows performance. This paper will discuss the importance of the network interface, as well as the interrupt handling capabilities of the IDT RISController family.
- The particular nature of the memory requirements of X-Terminals, and the difficulty in establishing a single, generic memory controller to handle the wide range of X-Terminal requirements. Differences between X-Terminals and other embedded systems, such as Laser Printers, will be discussed.

Finally, this paper will discuss an example of the use of a MIPS RISC processor in an X-Terminal application, drawing on the MIPS Magnum™ workstation as an example of the application of a generic, high-performance microprocessor engine in an X-Window environment. Many of the design goals of a workstation overlap with the design goals of an X-Window system, as exemplified by this workstation. Specific performance numbers for this implementation, as measured by X-Stones, will also be presented. We will finally discuss some X-Window specific support that could be added to this basic architecture to further increase X-Window performance.

BACKGROUND

X-Windows has emerged as the networking standard to facilitate applications sharing and interaction on a heterogeneous network. X-Terminals, which are basically graphics terminals which implement the X-Windows protocol, have been developed as a way to lower the "per-seat" cost of the network, and to optimize the cost/performance RISC has brought to the general computing world. Further, X-Terminals have facilitated the interaction and sharing of data and applications amongst multiple users which the PC Revolution made difficult, by allowing centralization of computing and storage resources while distributing the accessibility of the system through low cost terminals.

The market for X-Terminals has grown dramatically recently, for a variety of reasons:

- X-Terminals lower the "per-seat" (per-user) cost of high-performance networked computer systems.
- The availability of excess "MIPS" in high-performance compute servers, which allow X-Terminals networked to a RISC-based server to outperform PC's at lower cost per user.
- The development and use of Graphical User Interfaces (GUI's), which make high-performance, centralized computers as flexible and easy to use as PC's and MacIntosh'es.
- The X-Windows protocol has been widely adopted as the "back-end" of the GUI available from most system vendors, making it attractive to port applications software to the X-Windows environment.
- Finally, the technology required for X-Terminals to achieve high-graphics and network performance with high-resolution monitors has become increasingly available and cost effective, allowing the "break-even" point of a network of X-Terminals with a high-performance compute server to be lowered. This means that even relatively small networks can lower the cost-per-seat of this environment relative to the cost of a network of PCs.

BASIC TERMINOLOGY

The hierarchy of software and interfaces of an X-Windows environment is illustrated in Figure 1. The architects of X-Windows reversed the traditional (intuitive) nomenclature used in client/server relationship definitions: in the X-Windows environment, the client (the applications software running on the computer server) requests X-Windows activity from the X-server (the terminal) running on the network. Following initiation of an application program on the host machine, the client enters a loop and waits to be notified of an event by the server. The loop continually waits for the input from the server and, depending on the type of request made, will execute a given section of code in the application program. Following execution of the specific section of code, control returns to the main loop of the program to await a subsequent event.

The architecture of the software for the client portion of the application is beyond the scope of this paper. Instead, this paper will discuss some of the hardware considerations involved in designing a high-performance, low cost X-Terminal server.

As is obvious from the above discussion, the X-server sits between the software of the X-protocol and the hardware specific interfaces of the graphics output device, the local keyboard and mouse resources of the terminal, and the network interface.

An obvious architecture for this type of application is to use heterogenous multiple processors, each specialized to a particular aspect of the terminal, and coordinated under the control of a centralized CPU responsible for interpreting the "X" protocol and coordinating the various processors. This architecture predominated the earliest implementations of X-terminals, where graphics was managed by a graphics processor such as a 34010 and a general purpose processor such as a 68020 managed the network, keyboard and mouse, and the interaction of these various subsystems.

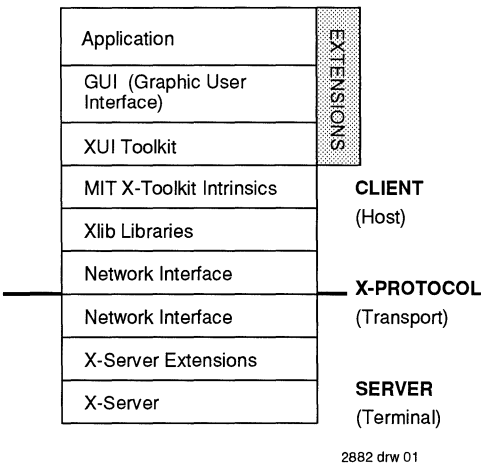


Figure 1. X-Windows Display System

The advent of low cost, high-performance RISC processors such as the IDT R3001 and R3051™ family, however, allows the integration of these subsystems into a single, high-performance CPU, allowing both lower cost and higher performance to be realized in X-terminals. RISC-based solutions absolutely are a factor in improving the viability and price/performance of the X-terminal marketplace.

TASKS OF THE TERMINAL PROCESSOR

The processor in the terminal obviously must respond to variety of different types of requests: requests from the network, requests from the mouse or keyboard, and requests to perform graphics. It is not obvious that these responsibilities all stress the same attributes of a processor: for example, network services will stress the interrupt response and real-time aspects of the system, while graphics will tend to stress the memory bandwidth and computational capabilities of the system.

Although in some ways, the X-Windows architecture sounds very similar to a laser printer architecture (which takes PostScript® from a variety of applications, often from a network, and performs interpretation and graphics operations to render an image on a page), the relative drawing and copy speed requirements of an X-Terminal are substantially higher. Recall that typical laser printers feature performance in the 8–20 pages per minute range, rather than a 70 times per second refresh rate of a terminal (this comparison is slightly apples to oranges, but it does reflect the fact that a terminal must feel more like "real-time" response to graphics requests than a printer does, at lower resolution than the 300–400 dots per inch of a printer), and that the connection to a printer is rarely a high-bandwidth network like Ethernet, but rather a slower channel such as AppleTalk®, or a dedicated channel such as Centronics.

THE MIPS® ARCHITECTURE IN X-TERMINALS

The MIPS architecture is well balanced, allowing its high-performance capabilities to benefit all of the performance critical areas of an X-Terminal design. The architectural highlights pertinent to X-Terminal design include:

- Fast, efficient interrupt handling and context switching. The MIPS architecture use a simple machine model, which uses a single set of 32 orthogonal registers (speeding context switch) and which hides the details of the pipeline from software (speeding both interrupt handling and context switch).
- On-chip Memory Management. There are actually a number of uses for memory management in an X-Terminal, some of which will be discussed later. The MIPS architecture includes memory management on-chip, and in fact also maintains a large memory space that is unmapped. This memory structure turns out to be a very good fit with the software operations of an X-Terminal.
- High-bandwidth memory. The R3000 architecture is able to fully utilize as much memory bandwidth as the system designer can supply. Note that this is dramatically different from traditional CISC architectures, which often cannot fully utilize the bandwidth of memory because of their multi-cycle

operations. Note that the R3000 family can take advantage of both high-bandwidth caches and high-bandwidth main memory (e.g. the frame buffer).
- Single-cycle ALU operations, including rotation, multi-bit shifts, and basic integer arithmetic.
- Atomic, high-speed integer multiply and divide on the integer chip, as a basic part of the architecture.
- The ability to include or exclude floating-point at will. Applications such as 3-D require the support of the high-performance floating-point co-processor, while 2-D applications do not. The mix-and-match attributes of the R3000 architecture allows a single base set of software to address multiple price-performance points by using various system-level or chip-level implementations.
- Efficient, effective compiler technology, which allows programs coded in high-level languages to obtain most of the inherent performance of the chip. Note that MIPS technology is unique in considering the compilers an integral part of the chip architecture.

THE IDT RISController FAMILY

IDT has performed modifications to the initial R3000 design which maintain these high-performance attributes, but reduce the cost of achieving that performance level by recognizing the distinction between UNIX® computers and embedded systems.

The RISController family achieve these system cost reductions by integrating those functions which affect both the performance and cost of R3000 based systems. For example, the R3001 performs integration at the cache subsystem level by reducing the overhead resulting from the TAG bits required to implement a discrete cache subsystem. The R3001 still allows the use of the high-performance R3010 hardware floating-point unit, and thus achieves lower system cost while preserving the full flexibility of the original R3000.

The R3051 family integrates the entire cache subsystem onto the CPU function, eliminating the need for external fast static rams. Today's technology allows enough cache to be integrated onto the CPU function so that performance is substantially maintained (other processors are so complex that smaller, less effective caches are implemented, with substantial performance loss; the R3000A core incorporated into the R3051 family is compact enough that substantial caches can be added into a low cost device family)[1]. However, the current members of the R3051 family do not allow access to a hardware FPA, leaving the R3001 as the appropriate choice for high end, 3-D applications.

Thus, the IDT RISController family is an architecture ideally suited to the requirements of X-Terminals. Specific vendor product announcements in the upcoming months will further serve to substantiate these claims, although certain existing computing products will serve to show the X-performance inherent in the MIPS architecture.

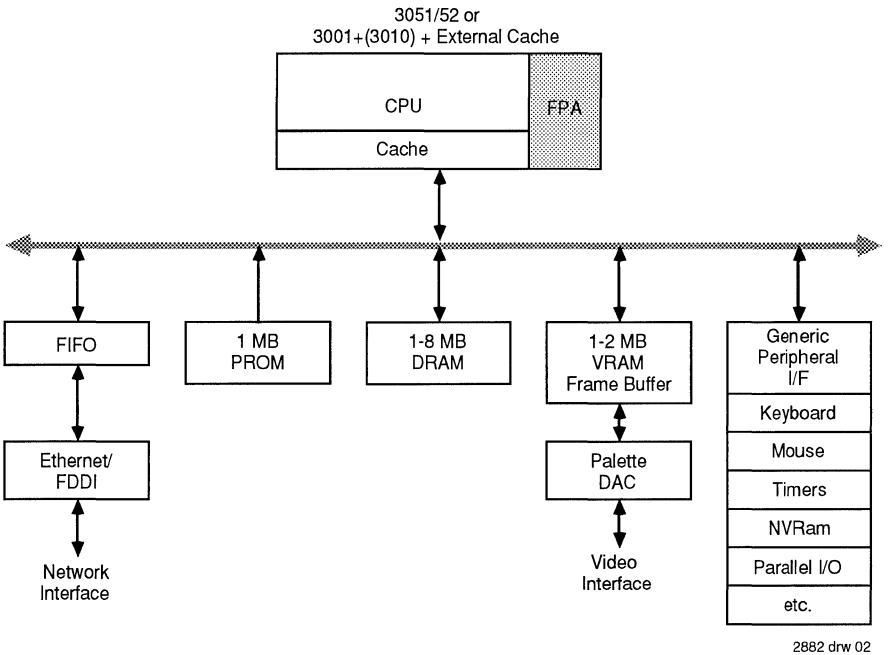


Figure 2. X-Terminal System Architecture

SYSTEM ARCHITECTURE

A typical X-Terminal system features the following areas (Figure 2) which much be managed by CPU hardware and software:

- A Frame buffer for the active video area. The actual size of the frame buffer is dependent on the resolution of the screen and the number of bits per pixel.
- A DRAM area, used to manage the various distinct windows merged together in the frame buffer, and also used to manage the display list.
- A high speed network connection, such as Ethernet, to support the TCP/IP communications protocol which has become the de facto standard.
- As much as 1MB of PROM space to hold the X-server code. Note, however, that some terminals have small boot proms, and accept the server downloaded across the network when the terminal is turned on.

In order to provide the highest levels of performance, the system strives to support extremely fast memory to memory copying, such as when copying a window into the frame buffer or moving a window within the frame buffer. Given this, a generic DRAM controller interface or PROM interface does not solve the memory interface problems of an X-terminal. Interface support for the network, the frame buffer, and for tightly couple interactions between these subsystems must also be provided. Thus, the CPU must support fast interrupt response, to minimize the performance lost in servicing network requests.

Additionally, it must perform addressing and general drawing of lines, rectangles, etc., very quickly. Typically, this involves the capability to read and write memory quickly, and to perform integer arithmetic and logic functions quickly.

Much of the software can easily be derived directly from the "C" source provided in the X11 X-windows reference source. However, substantial tuning typically occurs, both in the graphics drawing functions, and in the memory movement operations. Some of these optimizations are performed in reaction only to the architecture of the CPU chip (e.g. register file size, load or branch delays, etc.), and others are performed to tune both the chip and software to the bandwidth capabilities of the memory system (for example, is it better to perform block copies by performing burst reads followed by burst writes, and should the register file be used, etc.; is there hardware assist, such as BitBlt functions, provided in the hardware system, etc.). However, to minimize the development time required, these optimizations are kept to as small a portion of the software as possible.

MEMORY MANAGEMENT SCHEMES

One of the more controversial areas is the area of memory management schemes in an X-Terminal.

To a certain extent, the requirements of the X-Server serving client applications from hosts on the network is very analogous to the requirements of a UNIX operating system handling multiple user tasks: the single operating system (server) is responsible for managing the multiple tasks (clients and their windows) interaction with the various resources (e.g. the screen) under control of the kernel. The user tasks/client programs each assume that they "own" the entire screen and unlimited resources of the host/terminal, while in fact the resources of the terminal are limited and shared between multiple clients.

This is an obvious situation that is solved by memory management in general purpose computers. A similar approach can be taken by X-terminals.

The X-server can allocate memory to various clients from its fixed store. If it runs out of memory, a number of actions are possible:

- No action, in which case the actions of the terminal are unpredictable.
- It can send a "fault" message to either the user or client program to advise that it is out of memory.
- It can use the network to find a backing store for least recently used windows.

Obviously, many of these alternatives can be simplified by taking advantage of the memory management facilities provided in many processor architectures. Although an MMU is not strictly necessary to implement an X-terminal, a well conceived MMU (such as that provided in the MIPS R3000 architecture) can facilitate the X-terminal software.

The R3000 memory management structure (Figure 3) provides all of the features desired in an X-terminal: the large, unmapped kernel segments can be used to contain both the frame buffer and X-server software, while the mapped segments can be used to "translate" client references into memory addresses on a page by page basis. Finally, to allow the software to better interact directly with the memory system, the software can reference any memory region using either cacheable or uncacheable references, thus optimizing the use of the cache resources of the processor.

THE MAGNUM AS AN X-TERMINAL

An existence proof of the capabilities of the MIPS architecture in an X-terminal can be found in the Magnum workstation, designed and sold by MIPS computer systems.

- The Magnum includes a 25MHz R3000 and R3010, and 32kB of Instruction and 32kB of Data Cache, and is a high-performance UNIX workstation/server. The Magnum is capable of operating as an X-Terminal, although it would have added responsibilities and costs relative to a simpler, terminal-only design:
- The Magnum must manage disks, and the entire UNIX operating system, as well as the X-Server responsibilities.
- The memory bus structure of the Magnum is designed to support a standard I/O bus, and DMA transfers between unspecified I/O and the memory. Thus, the memory is less tightly coupled to the processor than for a terminal, and memory and compute bandwidth may be lost to serving "extraneous" I/O requests such as Disks.

The Magnum, on the other hand, may exceed the design of a lower cost X-terminal, in some areas:

- The Magnum includes a hardware floating-point unit. To reduce cost, this would not be provided in most X-terminals. Trace analysis has shown that the floating-point content of

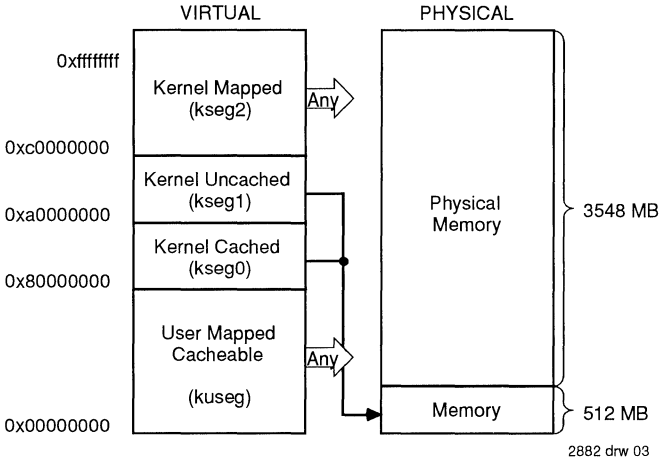


Figure 3. Virtual to Physical Memory Mapping of R3000 Architecture

an X-server is minimal, and the absence of hardware floating-point should not measurably affect performance.

- The cache size implemented in the Magnum is larger than the cache size in an R3051 family RISController. Although the smaller cache sizes of the R3051 family should translate into somewhat lower performance, it is expected that this effect will be relatively small. Many of the performance critical operations of an X-terminal are independent of cache size: for example, a block copy, which moves data between various memory regions, would not benefit from a data cache, and the algorithm to perform the block copy would easily fit within the smaller caches of the R3051 family.
- The net effect of these differences would imply that a dedicated X-terminal design, based on the RISController family, and absent the system concerns of disk drive control and long latency memory, should improve on the already substantial performance achieved by the Magnum workstation. Also note that higher-frequency versions of the RISController family will increase performance, while having only a small effect on system cost.

The Magnum workstation achieves 42,000 X-stones color drawing performance, and 91,000 X-stones monochrome (X-stones is the reference benchmark for X-terminal display performance; although it is slightly controversial, it is the most widely used indicator of a terminals graphics performance).

SUMMARY

This paper analyzed some of the basic considerations in the design of an X-terminal, and how the MIPS architecture, embodied in the IDT RISController family, serves those requirements while eliminating the dedicated graphics CPU's of first generation X-terminals.

There are obviously significantly more than these considerations in the evaluation of a processor for an X-terminal: considerations such as complexity of design, power consumption/dissipation, development environment, etc., are also considered in the choice of an X-terminal CPU. Finally, there is the intangible but often considered point that particular CPUs are code compatible with CPU hosts in some networks, opening the possibility that future terminals may actually be able to "off-load" some of the tasks or computations of the host.

These considerations, when weighed together, point to the MIPS architecture as an obvious solution to X-terminal design. Forthcoming products, to be announced by various vendors, will further substantiate this analysis.

[1] Additional information on the RISController family is available from IDT.

Integrated Device Technology, Inc.

DESIGNING MEMORY SUBSYSTEMS FOR THE R3051™ FAMILY

CONFERENCE PAPER CP-05

By Bob Napaa

INTRODUCTION

The IDT79R3051™ RISController™ family utilizes a high-performance computing core to achieve high performance across a variety of applications. Further, the amount of cache incorporated in the R3051 family allow these CPUs to achieve very high performance even with simple, low-speed low-cost memory subsystems.

The R3051 and the R3081™ RISController CPU families include a full R3000A core RISC processor, and thus are fully compatible with the standard MIPS processors. In order to provide high band-width to the CPU core, the families also incorporate relatively large instruction and data caches. The external memory interface from the R3051 family is very flexible and allows a wide variety of implementations depending on the price/performance goal of the application. The R3081 is upward compatible to the R3051 family with the same footprint and bus interface and the benefit of larger caches and a hardware floating-point coprocessor.

This paper will discuss the cost and performance impact of various trade-offs, and provide a concrete design of a DRAM memory subsystem around the R3051 and the R3081. This paper will specifically address the trade-offs between high-performance and low-cost memory systems, the impact of a high-frequency system on the memory interface and the impact of systems which are intended to be field upgradeable.

DIFFERENT TYPES OF MEMORY

SRAM, DRAM and EPROM are today's industry standard for memory subsystems. EPROMs usually provide boot code in most systems and are much slower and more expensive than SRAMs or DRAMs. SRAMs are typically less dense and more expensive than DRAMs; however, they provide faster memory access time with a simpler interface and can be used in systems where performance (rather than cost) is the primary criterion. DRAMs are the most popular choice for main memory because of their position on the cost/performance curve and the densities in which they are available.

MEMORY SYSTEMS

Most of today's systems use one of two memory architectures: Non-Interleaved or Interleaved architectures. In this paper, a memory array is defined as the group of memory devices that produce a full width CPU data bus. For example a 16-bit data bus CPU requires 4 "x4" DRAMs to compose a memory array while a 32-bit data bus CPU requires 8 "x4" DRAMs to compose a memory array.

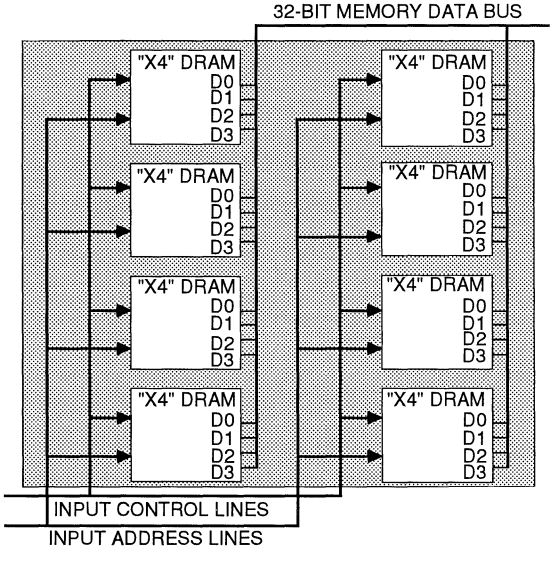


Figure 1a. Single-Bank Non-Interleaved System

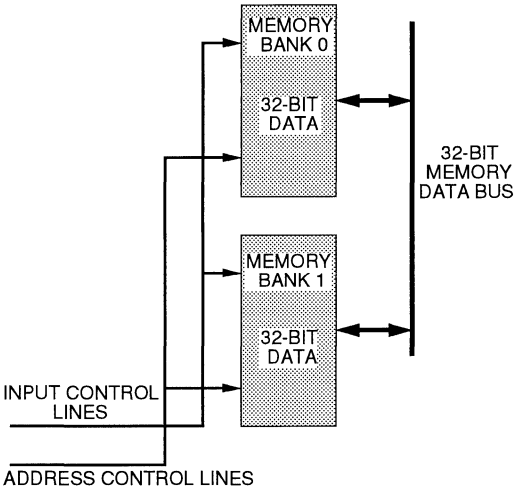


Figure 1b. Two-Bank Non-Interleaved System

In non-interleaved architectures, a memory bank consists of a single memory array with sequential addresses. Any read or write to a memory bank accesses a single location. Figure 1a illustrates the architecture of a single non-interleaved memory bank. Non-interleaved memory architectures are usually composed of multiple memory banks to satisfy the memory requirements of the system. In these topologies, the high order address lines select among the multiple memory banks and only one memory bank can be selected at a time. Figure 1b illustrates the architecture of a non-interleaved two banks memory system.

There are various types of interleaved architectures. The most popular one is the address interleaved. There are numerous variations of the address interleaved architectures. Mainly, 2-way address interleaved, 4-way address interleaved and so on. In a 2-way address interleaved architecture two memory arrays are grouped together in parallel to form a Super memory bank. This Super memory bank thus has double the data bus width and double the memory density of a single non-interleaved bank, and consists then of an even array and an odd array. A memory controller must be able to select both arrays together or independently based on the type of access. The memory controller uses the low order address bit to select between the two arrays. It must be able to direct the data path from every memory array independently to the CPU through some data buffers. Figure 2 illustrates the architecture of a 2-way interleaved single Super memory bank system. In a 4-way address interleaved architectures four memory arrays are grouped together in parallel to form a Super memory bank. This Super memory bank consists thus of four quarters. The memory controller must be able to select these four arrays together or independently using the two low

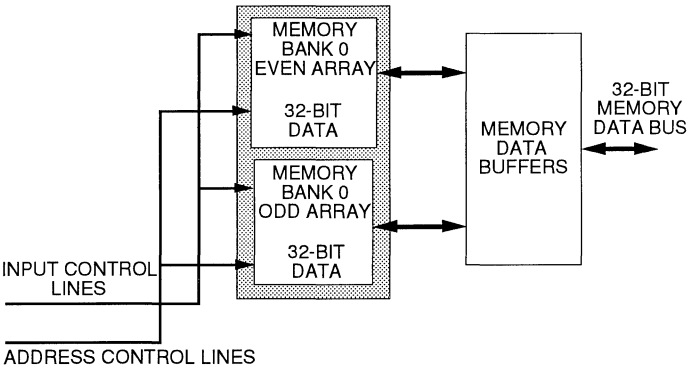


Figure 2. 2-Way Interleaved Single Super Memory Bank

order address bits. It must be able to direct the data bus of every quarter independently to the CPU through some data buffers.

Address interleaved memory systems are thus inherently more expensive than non-interleaved architecture since they require a much more complex memory controller and wider data paths. The basic amount of memory banks in address interleaved architectures is a multiple of the basic memory bank in non-interleaved architectures; however, for systems with large amount of memory, the same memory banks could be configured as interleaved or non-interleaved. The major advantage of interleaved systems lie in block of data elements accesses from/to the CPU. Interleaved systems can double or quadruple the memory band-width and thus dramatically improve the performance when the CPU reads or writes 4, 8, 16, 32... data elements at a time. Interleaved systems do not offer any advantage for single independent read or write accesses. Interleaved architectures are usually used in systems where performance (rather than cost) is of importance. For embedded cost sensitive applications, non-interleaved is usually the architecture of choice.

GENERAL DESCRIPTION OF THE DRAM SYSTEM AROUND THE R3051

The R3051 is designed around the R3000A MIPS RISC core and features a high level of integration with large on-chip instruction and data cache. It incorporates up to 8kB of instruction cache and 2kB of data cache. These relatively large caches achieve hit rates in excess of 90% and substantially contribute to the performance inherent in the R3051 family. The R3051 has also implemented on-chip a four-deep read and a four-deep write buffers that isolate the high frequency CPU core from the much slower external memory and modules. This high level of integration simplifies the interface between the R3051 and the external memory modules as is illustrated in Figure 3 and allows the use of low cost memory subsystems without penalizing the performance.

The R3051 family uses a double frequency input clock for its internal operation and provides a nominal frequency output clock for the external system. This output clock, Sysclk, synchronizes the external memory subsystems to the CPU. Memory transactions from the R3051 use a single, time multiplexed 32-bit address and data bus and a simple set of

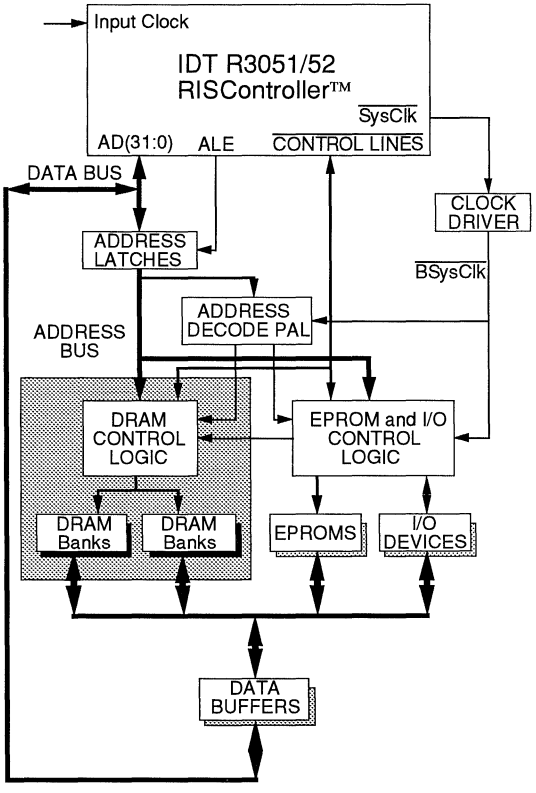


Figure 3. R3051 RISController Family-Based System

control signals. External logic then performs address demultiplexing and decoding, memory control, interface timing and data path control.

The system shown in Figure 3 is a 25MHz system with a 50MHz input clock. The R3051 interfaces to a DRAM system as the main memory, to an EPROM system and to various I/O devices and controllers. Address latches decouple the address bus from the data bus. Address decoders select among the various external modules. The output clock from the R3051 (Sysclk) is usually buffered to reduce the loading effect and to provide clock drive capability with minimum clock skew for the system.

The main DRAM memory system is based on 1 to 4 banks of non-interleaved DRAMs with 80ns of access time (t_{rac} = 80ns). The DRAMs used are 256k x 4 to provide a maximum memory space of 4MB. The DRAM memory space occupies the lower 4MB of the physical memory space. Figure 4 illustrates the architecture of the main DRAM memory system. The DRAM memory space resides between addresses 0000_0000 and 3FFF_FFFF. Address bits A(21:20) select among the four banks while the Rd and Wr outputs from the R3051 differentiate between read and write accesses.

Each memory bank (32-bit array) of DRAM, which corresponds to 1MB when using 256k x 4 DRAMs, is individually controlled by a separate RAS signal. RAS0 controls DRAM bank 0, RAS1 controls DRAM bank 1,... Each bank of DRAM is also controlled by an individual WriteEnable signal. WriteEnable0 controls DRAM bank 0, WriteEnable1 controls DRAM bank 1,... This architecture enables only a single DRAM bank for any DRAM read or a write access. The DRAM banks are arranged so that each bank represents a single, contiguous range of 1MB.

In an R3051 system, it is possible to perform a 32-bit read even when smaller data elements are requested. However on writes, it is important to enable only those bytes which are actually being written by the CPU. The R3051 bus interface provides four individual byte-enable signals to indicate which byte lanes are involved in a particular transfer. The DRAM subsystem encodes the byte-enable information from the R3051 into the CAS control signals of the DRAMs. In this encoding, CAS0 corresponds to byte lane 0, CAS1 corresponds to byte lane 1, etc. Each CAS signal is connected to the DRAM devices that correspond to the byte lane under its control in all four banks of the DRAM subsystem. That is to say that CAS0 is connected to the two DRAM devices that compose byte 0 in every DRAM bank.

Data buffers isolate the DRAM banks from the R3051 data bus to reduce the loading effect and to prevent contentions between the R3051 and the DRAMs. Note that this also alleviates concerns about the relatively slow tri-state times associated with DRAM devices. The data buffers selected are industry standard bidirectional transceivers (74FCT245). These data buffers actually isolate the data bus of the R3051 from all the external modules.

DRAM addresses are provided by multiplexing the latched R3051 address bus using the IDT FBT2827B memory drivers.

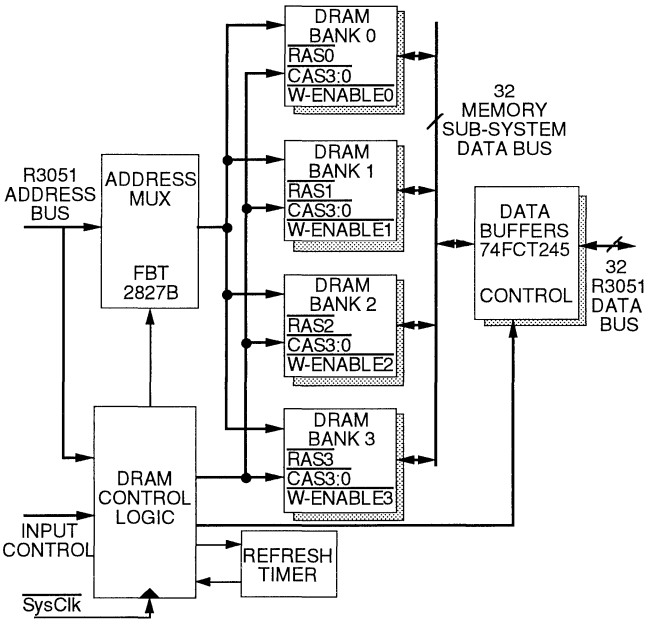


Figure 4. DRAM Memory Subsystem Architecture

This device type was selected based on its ability to drive large capacitive loading, such as found when driving 32 DRAM devices. A single FBT output has a series resistance incorporated in the output driver and is capable of driving all four banks of the DRAM subsystem. To minimize the signal skew among the DRAM devices, the address lines and the control lines to the DRAMs must use the "star" or the "fork" topology on the PCB board. In this method, all the loads on a given signal are lumped at the far end of the PCB trace. Series termination is also well suited to drive lumped (or forked) CMOS loads (like DRAMs) at the end of a PCB trace. The series termination minimizes overshoots and undershoots at the receiving end and does not add any power dissipation to the system.

Every DRAM cell consists of a MOS cell and a capacitor which encodes logic 1 and 0 in its charge. The capacitors in the DRAM cells tend to loose their charges with time through leakage. This is why DRAMs require to be refreshed at a regular time interval. The refresh mechanism is internal to the DRAMs where bits (cells) are rewritten with the same value to keep the capacitors charged. This refresh mechanism is enabled by the input control signals to the DRAM devices through the RAS and the CAS signals. In this design a refresh timer requests the refreshing of the DRAMs every 9.6µs. This refresh timer can be driven by the Sysclk from the R3051 or from an independent oscillator. The 9.6µs refresh interval chosen is more frequent than is actually required by the DRAMs. The use of this value simplified the control logic associated with page mode write. DRAMs require that RAS be maintained low no longer than 10µs; by choosing a refresh value smaller than this maximum time, the system is assured that maximum RAS low time will not be violated.

DRAM STATE MACHINE DESIGN

For the system described in this paper, a simple state machine performs the major aspects of DRAM control. The state machine uses a simple four-bit counter (C(3:0)) to dictate the timing for the DRAM control and CPU response, and is sequenced using SysClk. There are nine major states to the state machine as is illustrated in Figure 5. These states are dictated by the type of transfer requested and the state the DRAM control logic was left in by the prior transfer.

The DRAM control logic uses the Reset pulse to reset its internal states and to synchronize its operation to the R3051. During the RESET state, it also performs one refresh cycle before entering the IDLE state. In the IDLE state, the DRAM control logic arbitrates between a refresh cycle and a bus access. A DRAM bus access is started whenever the DRAM-Chip-Select and the Rd or the Wr signals are asserted. A refresh request is detected using the REF_REQ (Refresh_Request) pulse from the refresh timer. The DRAM controller supports 4 types of CPU bus accesses: "quad-word read", "Single-word read", "Single-word write" and "Page-word write". After a "Single-word write" or a "Page-word write" access, the DRAM control logic enters the IDLE RAS ASSERTED state which is an IDLE state with the RAS signals kept asserted. The RAS signals need to be precharged upon exiting this state.

Reset Cycle

A reset cycle is initiated by the assertion of the Reset signal. This is a hardware reset which initializes the control logic to the correct IDLE state. After the Reset signal is de-asserted, one DRAM refresh cycle is initiated. Most DRAMs require at least 8 refresh cycles for proper initialization. This DRAM

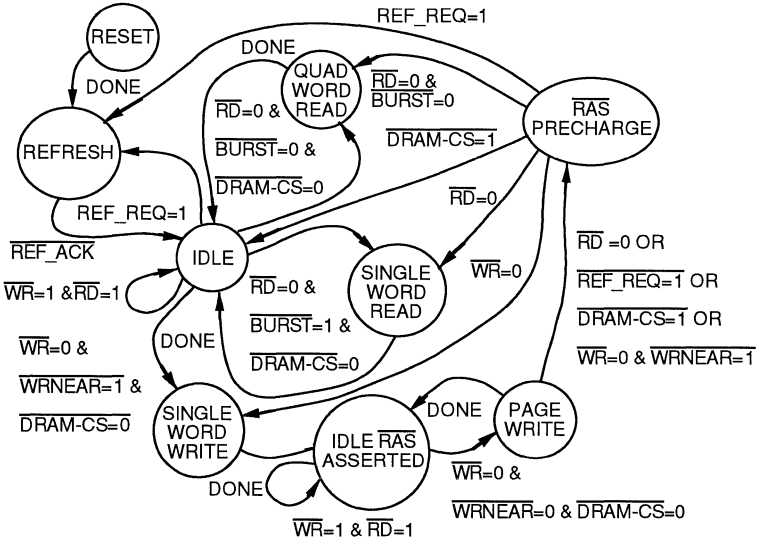


Figure 5. DRAM Control State Machine

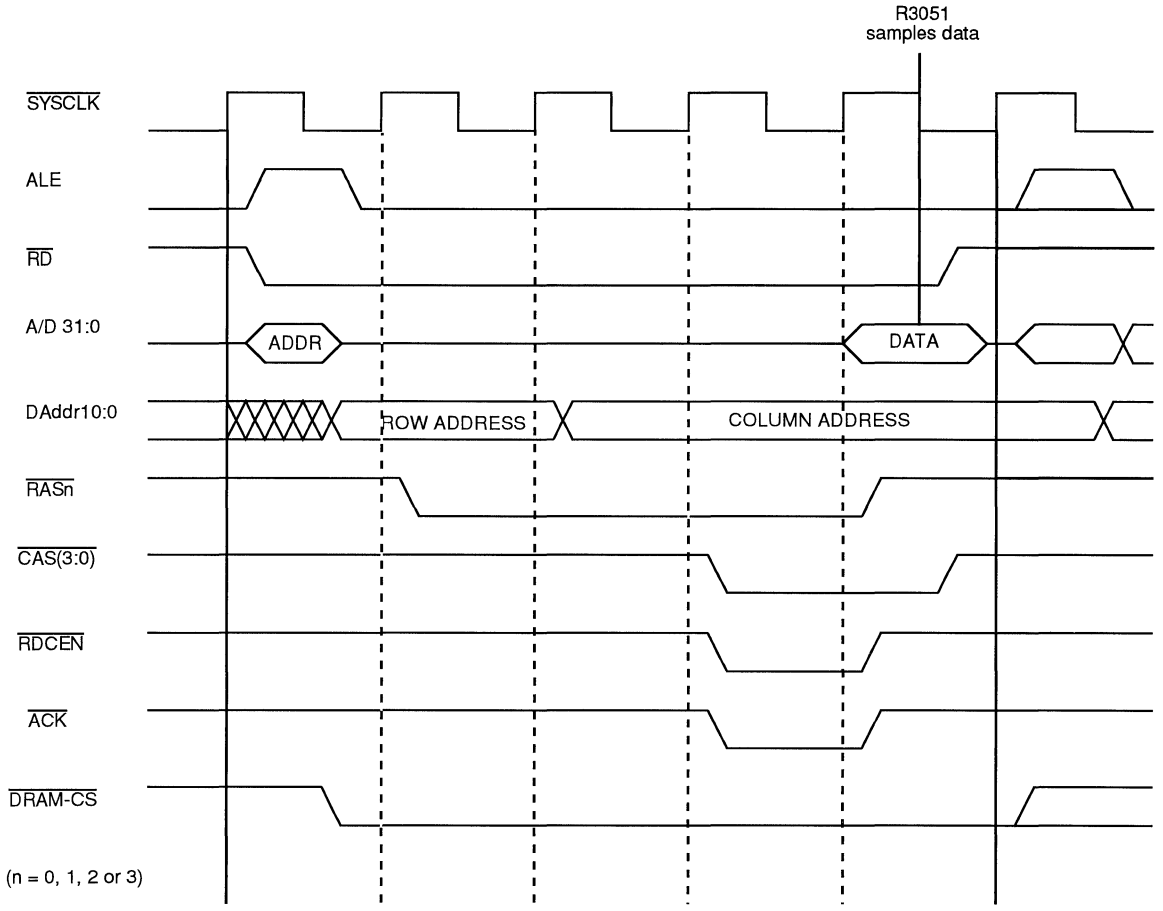


Figure 6. Single-Word Read Access Timing

control logic provides only one refresh cycle at reset time. It is the responsibility of the software to ensure that no DRAM access is made prior to the elapsing of 8 refresh periods. This can be insured by normal operation of the boot PROM; however, software could "spin-lock" for a predetermined number of loops to insure that sufficient time has elapsed.

Refresh Cycle

A refresh cycle is initiated every time a REF_REQ pulse from the refresh timer is detected. The refresh timer issues a REF_REQ pulse every 9.6µs. The DRAM control logic responds with a refresh acknowledge (REF-ACK) signal which locks the refresh timer until the refresh is serviced. The refresh interval has been set to 9.6µs which is shorter than the maximum 15.5µs refresh period that most DRAM require. The 9.6µs refresh period ensures that for an IDLE RAS ASSERTED state, where the RAS signals can be left asserted for long time periods, the maximum RAS pulse width of 10µs is not violated.

In the DRAM control logic, a refresh request has the highest priority over any other CPU requests. However, if a CPU bus requested is being serviced at the time the refresh is requested, the refresh cycle will be delayed until the end of the current bus cycle. The inverse is also true when bus requested are being delayed until the end of a refresh cycle. In this design, only the RAS-before-CAS refresh method is implemented.

Idle State

The Idle state is when the state machine is not performing any bus access or a refresh access but is constantly monitoring the bus for any access request. All the signals are deasserted and the operation of the 4-bit counter is halted.

Single-Word Read Cycle

There are two types of read transactions from the R3051: quad-word reads and single-word reads. A single-word read access is initiated by the R3051 by asserting the Rd signal.

The DRAM control logic responds by providing the R3051 with a single data element (32-bit word). Both the Ack and the RdCEn signals are used to terminate the single-word read access. In the system described in this paper, the Ack and the RdCEn signals are returned to the R3051 after 4 clock cycles, as illustrated in Figure 6.

Quad-Word Read Cycle

Quad-word reads from the R3051 occur only in response to internal cache misses. All instruction cache misses are processed as quad-word reads while data cache misses may be processed as either quad-word reads or single-word reads. The R3051 indicates quad-word read accesses by asserting both the Rd and the Burst signals. In the quad-word read access, address lines Addr(3:2) from the R3051 act as a two-bit counter to provide the address of 4 consecutive words, always starting on a word boundary.

The DRAM control logic handles quad-word read accesses using the Throttled Block Refill mode of the R3051. In a throttled read, RdCEn controls the data rate of the memory back to the CPU (latches the data into the on-chip read buffer). The Ack input is not provided back to the processor until the read transfer has sufficiently progressed such that the last word of the transfer is clocked into the on-chip read buffer (using RdCEn) one clock cycle before the processor core requires it.

In this non-interleaved system, the first word read of a quad-word read access takes the same time as a single read while the 3 subsequent words are read into the on-chip read buffer at the rate of 1 word every two clock cycles. The RdCEn is asserted for every word being read to latch the data into the R3051 read buffer. The Ack is asserted between the second and the third-word read. This ensures that for 4 subsequent falling edges of Sysclk the on-chip read buffer can provide data to the R3000A core at the rate of a word every clock cycle. Figure 7 illustrates the timing involved in quad-word read accesses.

Quad-word read accesses use the page-mode characteristics of the DRAM to obtain subsequent data word at a higher data rate. In this access, the RAS signal is kept asserted while the CAS signals are toggled 4 times to produce 4 data words.

Single-Word Write cycle

Unlike instruction fetches and data loads, which are usually satisfied by the on-chip caches, all write activity to the caches is seen at the bus interface of the R3051 as single write transactions. The R3051 indicates a single-word write access by asserting the Wr signal. The DRAM control logic enables the writing of the CPU word or partial word into the DRAMs and returns the Ack signal to terminate the write access. The Ack signal is returned to the R3051 after 3 clock cycles, as illustrated in Figure 8.

The DRAM memory system takes advantage of the WrNear signal from the R3051 by defaulting to the case that any single write to the DRAM subsystem will be followed by another write with the same upper 22 address bits. Based on this information the RAS signal must be kept asserted after every write access to enter the page mode of the DRAMs. The end of a single-word access is then different from a single read access in that the RAS signal is kept asserted.

Idle RAS Asserted State

At the end of a write access the DRAM control logic enters this idle state where a RAS signal is kept asserted while the

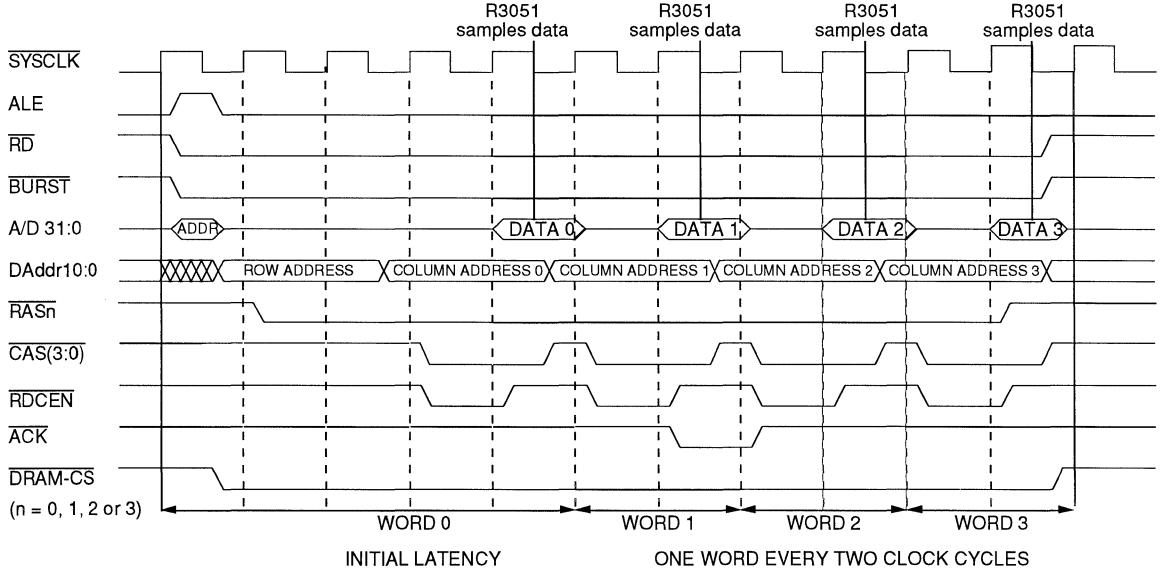


Figure 7. Quad-Word Read Access Timing

state machine awaits a subsequent transaction. If the next access is a local write (WrNear from the R3051 is asserted) the DRAM control logic enters the page write mode. If a different access type occurs, the state machine exits this state.

Page Write Cycle

A page write cycle is a single write access from the R3051 following a previous single write access with the same upper 22 address bits. The R3051 indicates a page write access by asserting the Wr and the WrNear signals.

The timing for a page write access is very similar to a single-write access but shorter since the RAS signal has been kept asserted from the previous write cycle. The Ack is returned back to the R3051 after 2 clock cycles. Figure 8 illustrates the timing for a page write access.

Precharge RAS

Any access, except a page write access, following an Idle RAS Asserted state needs to have the RAS signal precharged (driven to a level HIGH) before the access is responded to.

PERFORMANCE

The performance of the different types of R3051 bus accesses to the DRAM memory subsystem is usually measured by the number of clock cycles it takes to send the Ack back to the R3051. This time is computed from the beginning of the external access. The performance of the DRAM system can be summarized as follows:

- single read: 4 clock cycles
- block refill: 7 clock cycles
- first write: 3 clock cycles
- page write: 2 clock cycles.

This is a relatively high performance for a low-cost and easy-to-implement DRAM memory subsystem. The performance of the system can be improved by using more elaborate DRAM memory controller and/or more complex memory architectures such as address interleaving. Such systems should be able to achieve optimum performance.

FIELD UPGRADEABILITY

Many of today's systems are designed to allow for future fields upgrades of the base memory system to more memory banks and/or deeper DRAM devices. The ability to offer a base configuration (at a lower selling price) with upgrade capabilities is often a selling feature of the end product.

The system software should then run diagnostics at boot time to determine the maximum size of the available memory. Typical strategies for such diagnostics include writing distinct values into a given location within each bank, and then reading the data back to see if any of the writes did not occur properly

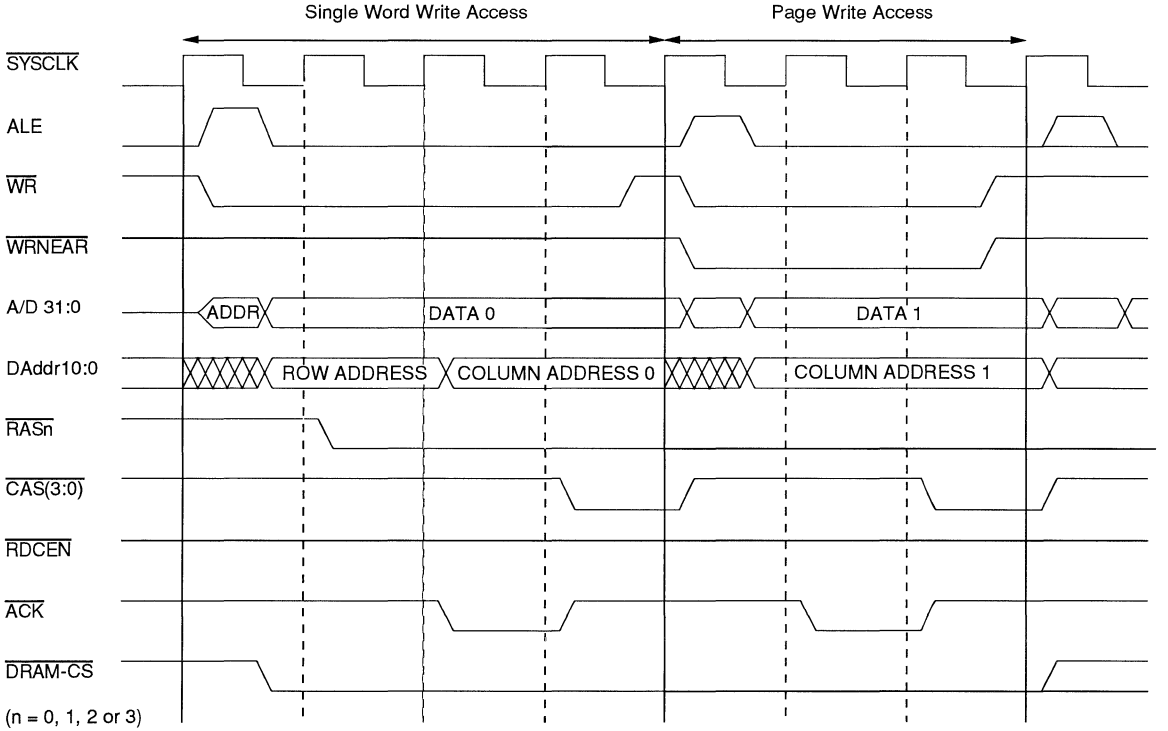


Figure 8. Single-Word Write Access Timing and Page Write Access Timing

or altered data previously written. Non-interleaved or interleaved memory architectures should be transparent to the system software.

The system hardware should make provision for extra memory banks or deeper memory devices by routing all the necessary signals to unused pins or sockets of future upgrade memory. The system hardware should try to minimize the use of jumpers to make the system much more user friendly.

In the system described in this paper, the user can upgrade to deeper memory by replacing the 256k x 4 DRAMs with deeper 1MB x 4 DRAMs to obtain a maximum memory space of 16MB. It is also possible to replace the R3051 with the R3081 to increase the performance of the system since they both have the same footprint. The R3081 with its on-chip FPA will have a great impact on the performance of floating-point intensive applications; a further benefit is the larger on-chip caches of the R3081.

CONCLUSION

The R3051 and the R3081 RISController families bus interface was designed to allow memory systems of differing complexity and performance to be implemented. Even a relatively simple DRAM system, as the one described here, offers very high performance. With simple modifications, this approach is applicable to higher frequencies (33 and 40MHz) and to interleaved memory systems yielding even higher performance. The R3081 can also be used for existing R3051 designs to improve the floating-point performance and the overall system throughput with no modifications of the external hardware.

REFERENCES

- AN-50: "Series Termination" Application Note, by Suren Kodical, 1990/91 IDT Logic Data Book.

Integrated Device Technology, Inc.

TRADE-OFFS IN LASER PRINTER APPLICATION DESIGNS AROUND THE IDT79R3051™ FAMILY

CONFERENCE PAPER CP-06

By Bob Napaa

INTRODUCTION

The IDT79R3051™ and R3081™ RISController™ families are a series of high-performance 32-bit microprocessors featuring a high level of integration. The R3051 and the R3081 are designed to bring the high performance inherent in the MIPS RISC architecture into low-cost, simplified, power-sensitive applications.

The R3051 and the R3081 families are specially targeted for high-performance, cost-sensitive embedded processing applications such as laser printers. The R3051 and the R3081 families currently offer a variety of pin-compatible and software-compatible CPUs in a common footprint. The R3051 and the R3081 families allow the system designer to implement a base design capable of accommodating a wide variety of printer market places: low cost systems through high resolution and color printers.

This paper will go through a low-cost laser printer base design around the R3051 family and will discuss the different design decisions and their impact on performance and cost. This paper will also discuss the impact of software and hardware development. Specifically, the impact on performance due to the variations among the R3051 and the R3081 family members: cache size, hardware FPA, etc. The paper will describe various models which allow a single hardware and software design effort to result in multiple customer products.

THE R3051 DEVICE OVERVIEW

The R3051 is designed around the R3000A MIPS RISC core and implements the MIPS-I ISA (instruction set architecture). The R3051 family incorporates on-chip 4kB or 8kB of instruction cache with a cache line size of 16 bytes. These relatively large caches achieve instruction hit rates in excess of 95% in most applications and substantially contribute to the performance inherent of the R3051 family. The R3051 family also incorporates 2kB of data cache with a cache line size of 4 bytes. Both caches are implemented as direct mapped physical address caches.

The R3051 family bus interface uses a 32-bit address and data bus multiplexed onto a single set of pins and provides simple handshake signals to process CPU read and write requests. The R3051 family incorporates a 4-deep write buffer to decouple the speed of the execution engine from the speed of the memory system. The write buffer captures and FIFO processes the address and data information (from the R3000A core) in store operations, and presents it to the bus interface as write transactions at a rate the memory system can accommodate. The R3051 also incorporates a 4-deep read buffer FIFO to allow the external memory system to queue up the data within the R3051 when performing a quad-word burst refill of the internal caches. Figure 1 illustrates the internal architecture of the R3051 family.

LASER PRINTER CONTROLLER DESIGN AROUND THE R3051 FAMILY

The following design example is a very basic laser printer controller around the R3051 and implements the minimum required configuration for a printer controller. This design can be extended into a more complex and more powerful one to accommodate the specific requirements of the various laser printer controllers. In any generic system, the R3051 family uses a double frequency input clock for its internal operation and provides a nominal frequency output clock for the external memory subsystems. Memory transactions from the R3051 use a single, time multiplexed 32-bit address and data bus and a simple set of control signals. External logic then performs address demultiplexing and decoding, memory control, interface timing and data path control. In this basic design of laser printer controller, the R3051 interfaces to 1MB of DRAM space expand-able to 4MB, 512kB of EPROM space expandable to 2MB, a Centronics (parallel) interface, two serial ports (RS-232) and a Canon video (print engine) interface. Figure 2 illustrates the simplified block diagram of the laser printer controller.

The following sections describe in detail the implementation of various subsystems (I/O and memory) of the laser printer controller and the impact of these implementations on the cost and the performance of the system.

External Logic and State Machine

The external logic and state machine necessary to perform address demultiplexing and decoding, memory control, interface timing and data path control is implemented using off-the-shelf, low-cost parts such as address latches, data transceivers, and programmable logic devices. This approach has the advantage of minimizing the cost of the components and the disadvantage of not obtaining the maximum performance out of the external system. It is possible to implement the above functions in a Gate Array or an ASIC chip. This second approach has the advantage of extracting the maximum performance out of the external system and minimizing the board space and the disadvantage of occurring the one time charge of the design of the ASIC chip.

DRAM Memory Subsystem

The DRAM memory subsystem is implemented as a single non-interleaved bank using "x4" DRAM devices. The basic configuration has a memory space of 1MB using 256k x 4 DRAMs. It can be extended to 4MB by using 1M x 4 DRAMs.

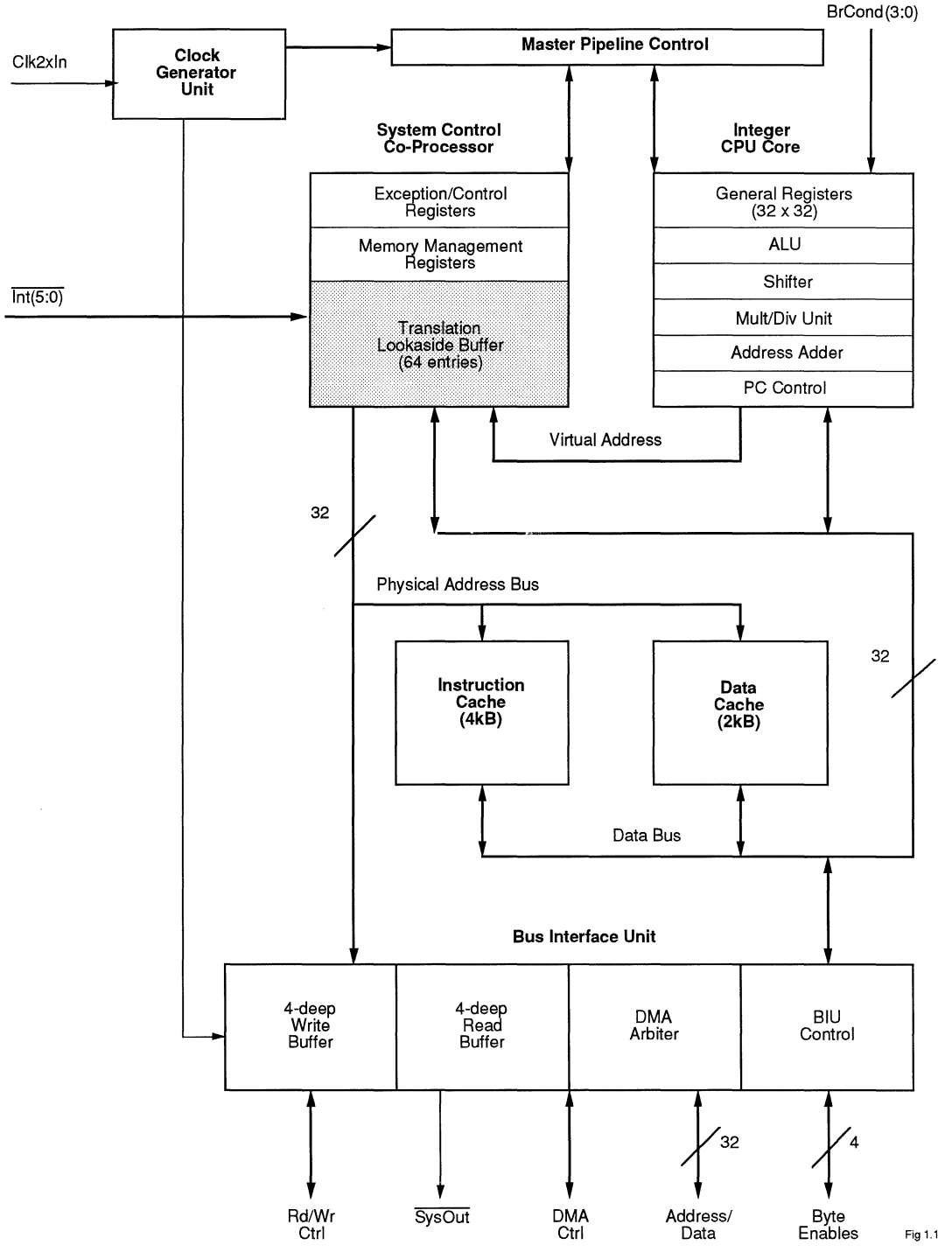


Figure 1. Internal Architecture of R3051 Family

The 1MB memory configuration (or even 512kB) is the minimum requirements for most non-PostScript® laser printer controllers. For systems which are PostScript or HP LaserJet III PCL5 compatible, the 4MB configuration is more appropriate. This DRAM memory architecture has the advantage of minimizing the cost and the disadvantage of limiting the flexibility of the DRAM subsystem (only 1 increment to 4MB possible). It is possible to offer more flexibility to the end user by implementing 4 or more non-interleaved DRAM banks to obtain the same memory depth or even a deeper memory. This approach however will increase the board space to accommodate the multiple banks of DRAMs and add more complexity to the DRAM controller in order to independently select the different banks. A third design yet is to interleave the DRAM banks to maximize the performance of the DRAM subsystem. This is a more expensive solution usually implemented on large Network printers.

EPROM Memory Subsystem

The EPROM memory subsystem is implemented as a single non-interleaved bank using "x8" EPROM devices. The basic configuration has a memory space of 512kB using 32k x 8 EPROMs. It can be extended to 2MB by using 512k x 8 EPROMs. The requirements of the EPROM memory subsystem is tightly coupled to the design and to the target application. In systems where all the code resides in the EPROM section, a deep EPROM space is usually required. This is the case for this basic design where the 2MB EPROM space is always used. In systems where only the boot code resides in EPROM and the remaining of the code is downloaded to DRAM from a diskette (or other storage media), a shallow EPROM space is sufficient (512kB). Again it is possible to maximize the performance of the EPROM memory subsystems—if code is running out of EPROM—by interleaving 2 or more banks of EPROMs which will also increase the cost of the system.

Centronics Input Port

The Centronics (parallel) input port is an 8-bit parallel port commonly used as a high-speed communications link between the computer and the printer. To enhance the performance of the Centronics interface, a standard 256 x 9 FIFO (IDT7200) is used to buffer the input parallel data. This low-cost implementation of the Centronics interface greatly enhances the performance of the I/O system. Figure 3 illustrates the block diagram of the Centronics interface. In this implementation, two methods are used to respond to incoming printer data files. The FIFO empty flag signal is connected to a branch condition input, BRCND2, on the R3051. The software should take advantage of the R3051's ability to branch on condition inputs by regularly polling this pin. The FIFO full flag disables further writes to the FIFO and is used to generate an interrupt to the R3051. There are also other possible methods to implement the parallel interface to even increase the performance of the Centronics interface.

Serial Communications Ports

The laser printer controller design uses a DUART to control two serial communications ports. The first port is usually used to connect to a CRT terminal during the debugging phase of

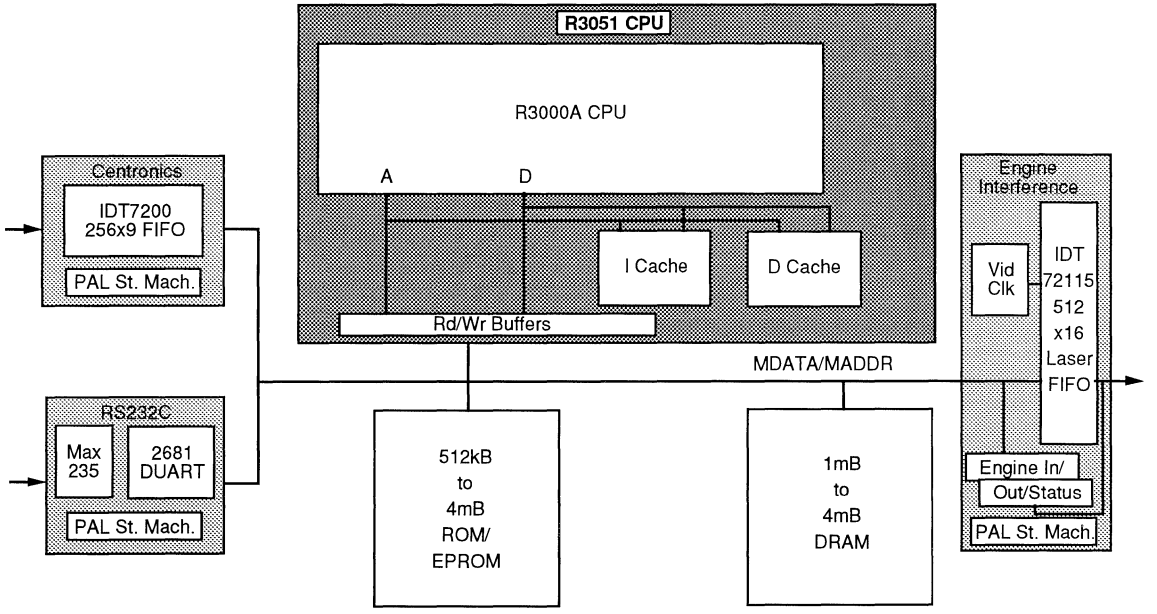


Figure 2. Block Diagram of Laser Printer Controller

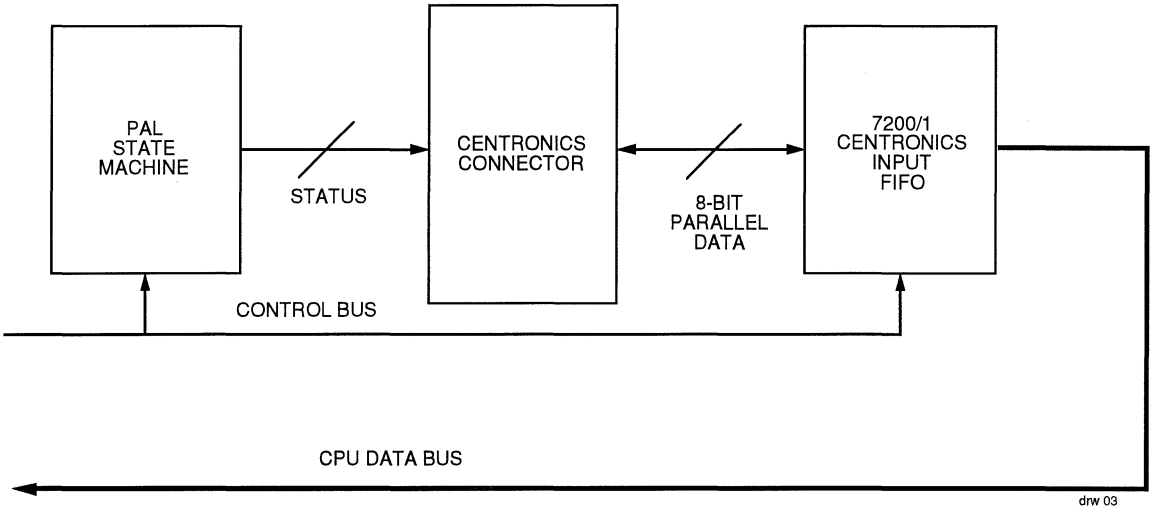


Figure 3. Centronics Interface Block Diagram

the project while the second is usually used to download data from a computer. In the final product both serial ports can be used to download data.

Canon Video Interface

The laser printer controller board is designed to interface to the industry standard Canon LBP-SX laser printer engine. To enhance the performance of the video interface and to minimize the part count—and thus the cost—the video interface is implemented using two registers and an IDT72115 512x16 LaserFIFO. The two register buffer the handshaking signals between the controller and the print engine. The LaserFIFO is large enough to buffer a horizontal scan line at 300 or 400 DPI—up to 400 bytes. For higher-resolution printers or for larger page sizes, the LaserFIFO size can be increased to 1024 x 16 with a pin-compatible IDT72125 with no other hardware modifications. Depending on the final product and the print engine of choice, the video interface can be modified to connect to the appropriate laser engine (TEC, IBM, etc.).

Unimplemented Features

There are other features that can be added to the above basic design to satisfy the need of the applications. These additional features, such as Font Cartridges interface, Banding Coprocessor, AppleTalk interface, SCSI interface, etc., have not been implemented. They are usually added to a given printer application to increase the system performance and/or to position the final printer product in a specified market place. These features are beyond the scope of this document.

Software Implementation

The main Operating system of the laser printer controller is the PeerlessPage™ Printer Operating System with a PCL5 or PostScript compatible emulations. Other printer emulations are available but have not been implemented in this design. The PeerlessPage POS provides a portable and extensible environment for printer controllers and a flexible platform for integration of other combinations of fonts, panel control or emulations. Other off-the-shelf Operating Systems are also available with similar or different set of capabilities. It is also possible to develop a proprietary OS that best suits the target application. The performance and the cost model of the various software approaches differ from one hardware/software combination to the other.

THE IDT7RS385 LASER PRINTER CONTROLLER

To evaluate the performance of its RISController family in a laser printer environment, IDT developed the REAL8™ Laser Printer Controller (IDT7RS388) board based on the R3001 CPU. The IDT7RS388 is completely self-contained and is intended for use as an evaluation system for a variety of software and memory configurations. To also evaluate the performance and system cost of IDT's R3051 RISController family in a laser printer application, IDT developed an emulation of the REAL8 board complete with ports of the PeerlessPage Imaging Environment, Microsoft TrueImage™ (PostScript®-compatible) PDL and PeerlessPrint5 (HP LaserJet III PCL5-compatible) languages using the IDT7RS385 Evaluation Board as the hardware platform. This new hardware platform, the IDT7RS385-LPC is a basic laser printer controller and is implemented according to the specifications of the design explained in this paper. It is a 25MHz design which includes a fast Centronics parallel input port, two serial ports, and a video interface for the Canon LBP-SX print engine. The memory subsystems of the IDT7RS385-LPC are designed to minimize the cost of the system rather than maximize the performance. The IDT7RS385-LPC includes a single non-interleaved bank of 4MB DRAMs and a single non-interleaved

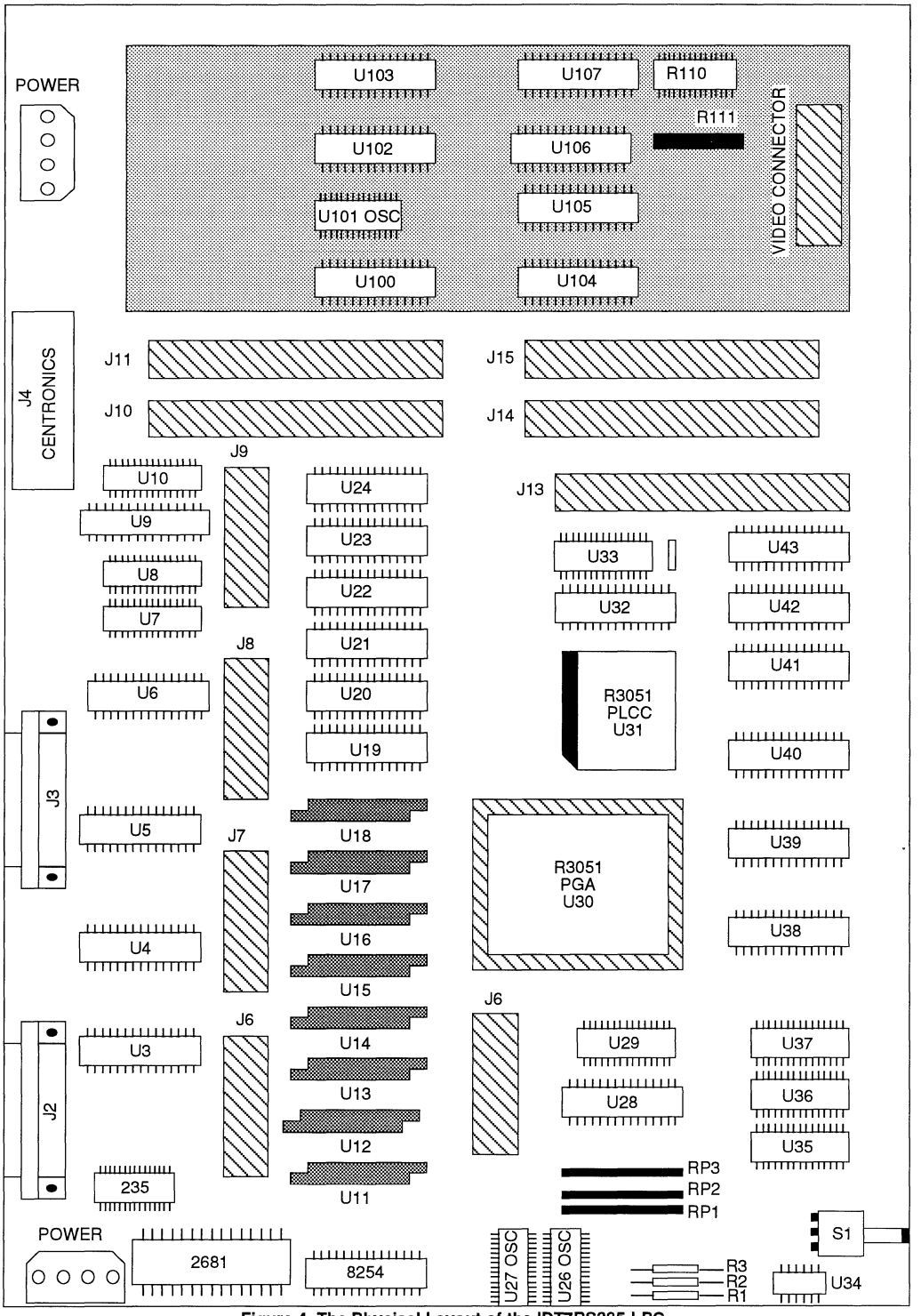


Figure 4. The Physical Layout of the IDT7RS385-LPC

bank of 2MB EPROMs. The DRAM and the EPROM memory subsystems access latencies expressed in terms of external clock cycles are the following:

DRAM:
- single read: 5 clock cycles
- quad word read: 5 clock cycles for the first word, 2 clock cycles for the remaining 3 words
- single word write: 5 clock cycles
- page write: 4 clock cycles

EPROM:
- single word read: 5 clock cycles
- quad word read: 5 clock cycles for the first word, 4 clock cycles for the remaining 3 words

The above memory subsystems latencies represent the number of external clock cycles required to process an external access (read or write from the R3051) and do not include the internal clock cycles involved in the internal arbitration for the bus and the fix-up cycle of the R3000A core. These external memory latencies can be greatly reduced by using faster memories and/or interleaving the memory banks or by using an alternate/integrated DRAM control system. These more elaborate schemes could inherently increase the cost of the system. Figure 4 illustrates the physical layout of the IDT7RS385-LPC board. The IDT7RS385-LPC can run both the PeerleesPrint5 or the Microsoft TrueImage languages. Both these languages require floating-point operations which are emulated in software using the IDT/c floating-point libraries.

PERFORMANCE OF THE IDT7RS385-LPC

The performance measure of the simple laser printer controller design represented by the IDT7RS385-LPC gives an insight of the high performance inherent to the R3051 family. It is always possible to optimize a basic, simple design, like this one, to obtain a better performance by using more elaborate memory schemes.

The R3051 family offers a set of four pin-compatible CPUs in the same footprint—the R3051, the R3051E, the R3052 and the R3052E. The "E" suffix stands for the Extended Architecture parts in which the Memory Management Unit (MMU) is present. The Software (PeerlessPrint5) does not make use of the MMU, and thus the performance for these parts will be identical to the non-E parts.

Table I illustrates the relative performance among the different configurations of the R3051 and the R3081 families normalized to the R3051. The performance was measured on an IDT7RS385-LPC board at 25MHz and running the PeerlessPrint5 language which uses extensively the floating-point operations. The floating-point operations are implemented on the R3051 family using the IDT/c™ floating-point library. To keep a common hardware platform, the performance test is implemented on an IDT7RS385-LPC with different CPU configurations. The test setup is the following: an IBM-PC is connected to the board via the Centronics interface. The board is powered by the print engine and the video interface links the board to the print engine. The relative performance

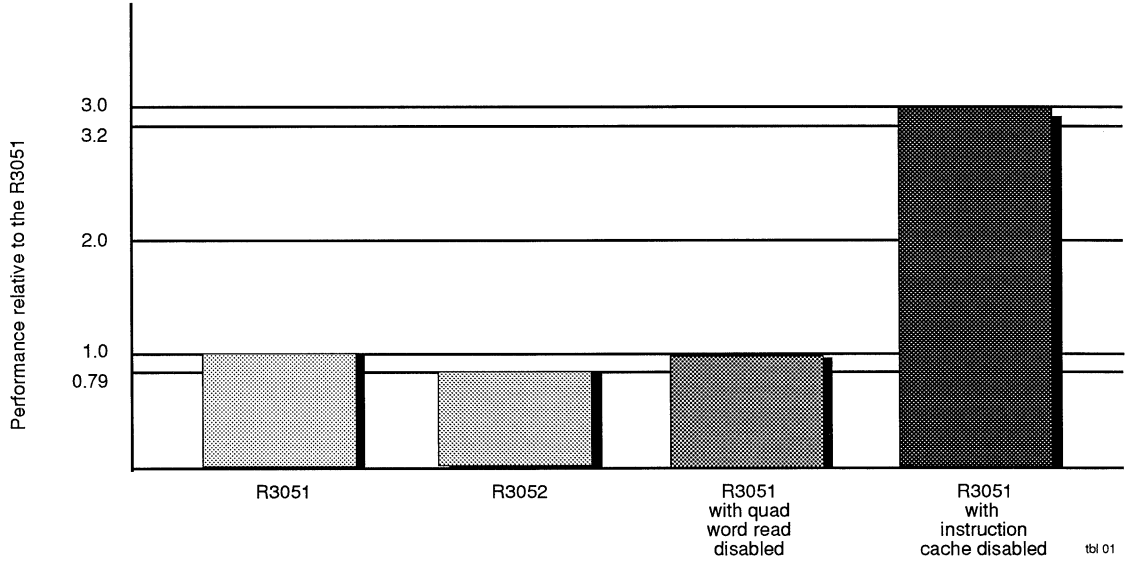


Table 01. Relative Performance of the R3051 Family and the R3081 Family in a Laser Printer Environment. (The Performance is Normalized to the R3051.)

presented here is the total of 10 different PCL5 benchmarks (these benchmarks are listed at the end of this text). Every benchmark measures the total time in seconds it takes from the moment the carriage return is hit on the PC to the moment the print engine starts running. These benchmarking comparison demonstrate the effect of the cache, the FPA, the read/write buffer on the performance of the system.

The IDT7RS385-LPC using an R3052 CPU

The best performance for the R3051 family is obtained on the IDT7RS385-LPC with the R3052 CPU. The R3052 CPU offers 8kB of instruction cache and 2kB of data cache. Both caches are implemented as direct mapped physical address caches. There is more than 20% improvement in the performance when doubling the size of the instruction cache for a given size of the data cache.

The IDT7RS385-LPC using an R3051 CPU

The reference point of the benchmark performance is based on the IDT7RS385-LPC with the R3051 CPU. The R3051 CPU offers 4kB of instruction cache and 2kB of data cache. Both caches are implemented as direct mapped physical address caches.

The IDT7RS385-LPC using an R3051 CPU and a Single-Word Data Read

The third benchmark performance is based on the IDT7RS385-LPC with the R3051 CPU. The R3051 CPU offers 4kB of instruction cache and 2kB of data cache. The R3051 family processes the data cache miss as a single-word read or a quad-word read and the instruction cache miss as quad-word read. It is possible then, at reset time, to disable the data cache quad-word refill capability and, thus, to treat every data cache miss as a single-word read. There is a minimal difference in the overall performance (for these types of embedded applications) between refilling the data cache with a single-word or a quad-word stream. For some benchmarks, it has also been noticed that the performance with only a single-word read enabled is slightly better than with quad-word cache refills.

The IDT7RS385-LPC using an R3051 CPU and No Instruction Cache

The fourth benchmark performance is based on the IDT7RS385-LPC with the R3051 CPU. The R3051 family processes the data cache miss as a single-word read or a quad-word read and the instruction cache miss as quad-word read. It is possible then, through the debug mode of the R3051, to enable data caches and the data cache quad-word refill while forcing instruction cache miss on every instruction cycle. This is equivalent to reading and executing only one instruction at a time from main memory, or in another word, an R3051 without an instruction cache. This test is then very memory intensive which is reflected in the relative performance of such systems. The effect of the presence (or the absence) of the instruction cache on the performance of a system is very noticeable in this example. The average performance of a system without an instruction cache is more than three times slower than a system with the instruction cache enabled.

The IDT7RS385-LPC using an R3081 CPU

The absence of the Floating-Point Coprocessor (FPA) from the R3051 family impacts the performance of the applications that require floating-point operations. The new R3081 family from IDT offers pin-compatible CPUs, with the same footprint and bus interface, to the R3051 family with a Floating-Point Coprocessor incorporated on-chip. The presence of the FPA in the R3081 family greatly improves the performance of floating-point intensive applications such as PostScript-compatible laser printers.

The R3081 CPU has 20kB of caches which can be configured as either 16kB of instruction cache and 4kB of data cache, or 8kB of instruction cache and 8kB of data cache. Both caches are implemented as direct-mapped physical address caches. The R3081 family is pin- and footprint-compatible with the R3051 and incorporates a hardware floating-point coprocessor on-chip. The presence of the FPA greatly increases the system performance for floating-point intensive applications. Similarly, the deeper instruction and data caches enhance the overall performance compared to the R3051 family (this effect is already noticeable in the R3051 family when upgrading from the R3051 to the R3052). Comparison data for the relative performance of the R3081 was not available when this text was prepared.

The relative performance listed in this text addresses the variations among different internal CPUs implementations, such as the effects of the caches sizes, the presence of an FPA, etc. The presence of an instruction cache (even a very shallow one) has a major impact on the overall performance of the system. Better performance can also be obtained by increasing the instruction cache size. In most embedded applications the presence of the data cache improves the performance; however, the size of the data cache is not of major importance. The FPA improves the performance of applications that use the floating-point operations intensively. For a given CPU architecture, R3051 or R3081, it is always possible to optimize the absolute performance by implementing more elaborate memory and I/O design techniques such as: interleaved memory systems, burst EPROMs and/or integrated memory controllers. The system designer is always left in making the trade-offs between the cost and the performance among the various choices to best suit the end product and the application at hand.

CONCLUSION

IDT offers a wide spectrum of embedded RISController CPUs in the form of the R3051 family and the R3081 family targeted for price-sensitive or performance-sensitive applications. It is then possible to design a base system around any processor within these families and then upgrade the design as the performance requirements increase. For systems that require more horsepower, the R3081 family can seamlessly replace the R3051 family while keeping the same hardware and software investment.

BENCHMARKS

The following are the names of the 10 PCL5 files (as well as their sizes in bytes) used in compiling the performance results of the R3051 and the R3081 families:

- ARCS. PRN 988 bytes
- BITWF0. PRN 1935 bytes
- CIRCLES. PRN 1094 bytes
- PCL5TBIT. PRN 7219 bytes
- PCL5TEXT. PRN 3336 bytes
- PCL5X10. PRN 1247 bytes
- PCL5X11. PRN 1617 bytes
- PCL5X14. PRN 1274 bytes
- PCL5X2. PRN 1560 bytes
- TRIANGLE. PRN 1633 bytes

Integrated Device Technology, Inc.

NEXT-GENERATION MIPS® RISC ARCHITECTURE FOR EMBEDDED APPLICATIONS

CONFERENCE PAPER CP-07

By Philip Bourekas

INTRODUCTION

The IDT79R3051™ RISController™ RISC Family is a high-performance 32-bit microprocessor featuring a high-level of integration, and targeted to high-performance but cost-sensitive embedded processing applications. The R3051 is designed to bring the high-performance inherent in the MIPS RISC architecture into low-cost, simplified, power-sensitive applications. The R3051E adds a full featured Memory Management Unit to the core architecture of the R3051, to support the requirements of particular embedded applications.

Functional units were integrated onto the CPU core in order to reduce the total system cost, rather than to increase the inherent performance of the integer engine. Thus, the R3051 is able to offer 20mips of integer performance at 25MHz without requiring zero wait-state memory or caches.

Further, the R3051 achieves dramatic power reduction over the R3000/R3001, allowing the use of low-cost packaging for devices up to 25MHz. The R3051 allows customer applications to bring maximum performance at minimum cost, by reducing both component cost and eliminating the need for fast external memory.

FEATURES

- Instruction set compatible with IDT79R3000A MIPS RISC CPU
- High level of integration minimizes system cost
- 20 MIPS at 25MHz
- Low cost 84-pin PLCC packaging
- Large on-chip instruction and data caches
- Flexible bus interface allows simple, low-cost designs.
- Single double-frequency clock input
- 12.5 through 33MHz operation
- On-chip 4-deep write buffer eliminates memory write stalls
- On-chip 4-deep read buffer supports the use of slow memory devices
- On-chip DMA arbiter

DEVICE OVERVIEW

Figure 1 shows a block level representation of the functional units within the R3051 and R3051E. The R3051 could be viewed as the embodiment of a discrete solution built around the R3000 or R3001. However, by integrating this functionality on a single chip, dramatic cost and power reductions are achieved. An overview of these blocks is presented here.

CPU Core

The CPU core is a full 32-bit RISC integer execution engine, capable of sustaining close to single-cycle execution rate. The CPU core contains a five-stage pipeline, and 32 orthogonal 32-bit registers. The R3051 implements the MIPS-I ISA. In fact, the execution engine of the R3051 is the same as the execution engine of the R3000 and R3001, eliminating the risk of incompatibility issues and speeding development time. Thus, the R3051 is fully binary compatible with the R3000/R3001.

System Control Coprocessor

The R3051 also integrates on-chip the System Control Coprocessor, CP0. CP0 manages both the exception handling capability of the R3051 as well as the virtual to physical mapping of the R3051.

There are two members of the R3051 Family. The R3051E (Enhanced) version incorporates the same MMU as the R3000 and R3001. This version contains a fully associative 64-entry TLB which maps 4kB virtual pages into the physical address space. The virtual to physical mapping includes kernel segments which are directly mapped to fixed physical addresses and kernel and user segments which are mapped page by page by the TLB into anywhere in the 4GB physical address space. In this TLB, 8 pages can be "locked" by the kernel to insure deterministic response in real-time applications.

The standard R3051 removes the TLB and institutes a fixed address mapping for the various segments of the virtual address space. The R3051 supports distinct kernel and user mode operation without requiring page management software, leading to a simpler software model.

Clock Generation Unit

The R3051 is driven from a single double-frequency input clock. On-chip, the clock generator unit is responsible for managing the interaction of the CPU core, caches, and bus interface. The clock generator unit logically replaces the external delay line required in R3000 and R3001 based applications.

Instruction Cache

The R3051 incorporates an on-chip instruction cache of 4kB (1k instructions) organized as a line size of 16 bytes (four entries). This relatively large cache achieves a hit rate in excess of 95% in most applications and substantially contributes to the performance inherent in the R3051. The cache is implemented as a direct mapped cache and is capable of caching instructions from anywhere within the 4GB physical address space. The cache is implemented using physical addresses (rather than virtual addresses) and thus does not require flushing on context switch.

Data Cache

The R3051 incorporates an on-chip data cache of 2kB, organized as a line size of 4 bytes (one word). This relatively large data cache achieves hit rates in excess of 90% in most applications and contributes substantially to the performance

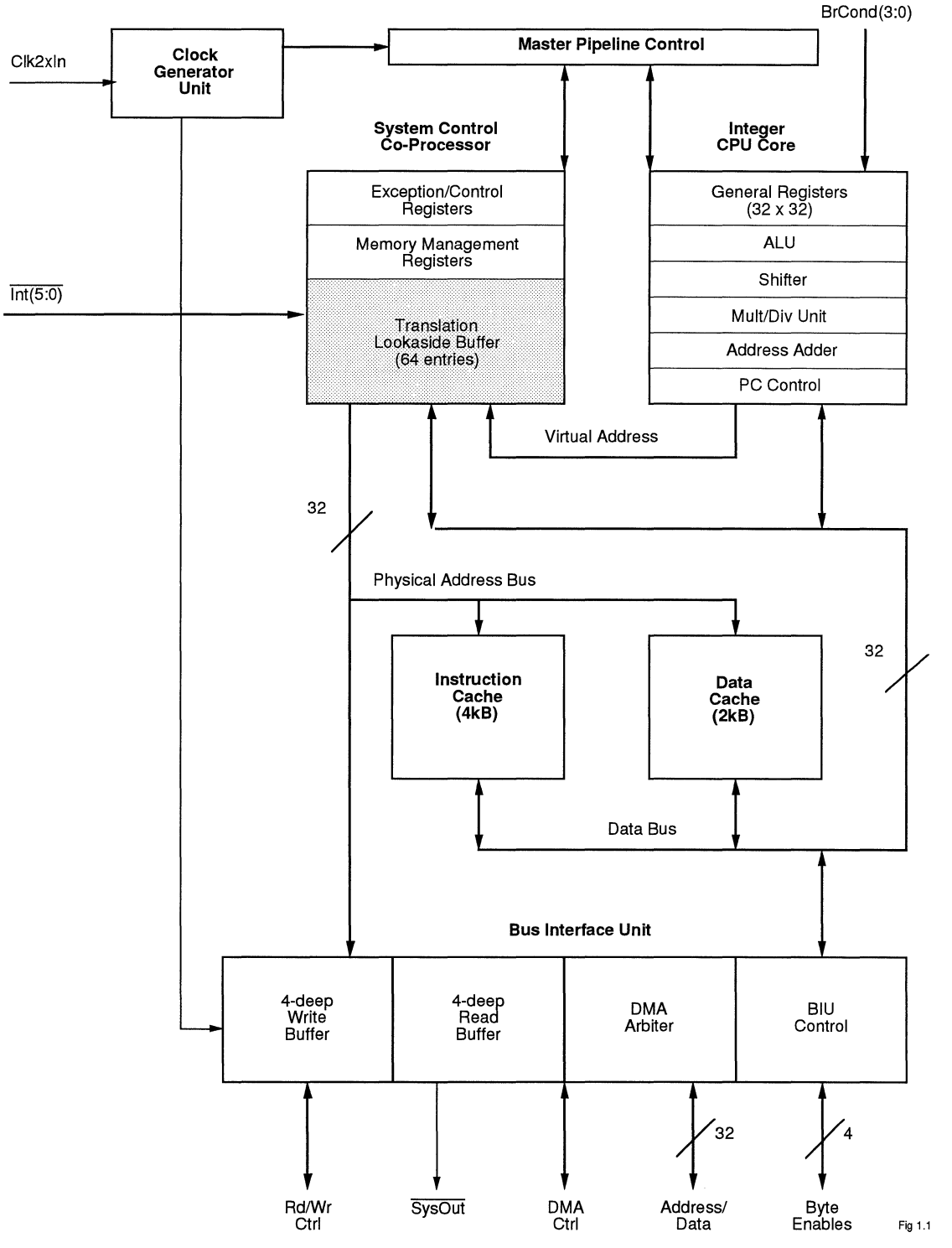


Figure 1. R3051/R3051E Block Diagram

inherent in the R3051. As with the instruction cache, the data cache is implemented as a direct mapped physical address cache. The cache is capable of mapping any word within the 4GB physical address space.

The data cache is implemented as a write through cache to insure that main memory is always consistent with the internal cache. In order to minimize processor stalls due to data write operations, the bus interface unit incorporates a 4-deep write buffer which captures address and data at the processor execution rate, allowing it to be retired to main memory at a much slower rate without impacting system performance.

Bus Interface Unit

The R3051 uses its large internal caches to provide the majority of the bandwidth requirements of the execution engine and thus can utilize a simple bus interface connected to slow memory devices.

The R3051 bus interface utilizes a 32-bit address and data bus multiplexed onto a single set of pins. The bus interface unit also provides an ALE signal to demultiplex the AD bus and simple handshake signals to process processor read and write requests. In addition to the read and write interface, the R3051 incorporates a DMA arbiter to allow an external master to control the external bus.

The R3051 incorporates a 4-deep write buffer to decouple the speed of the execution engine from the speed of the memory system. The write buffers capture and FIFO processor address and data information in store operations, and presents it to the bus interface as write transactions at the rate the memory system can accommodate.

The R3051 read interface performs both single-word reads and quad-word reads. Single-word reads work with a simple handshake. Quad-word reads can either utilize the simple

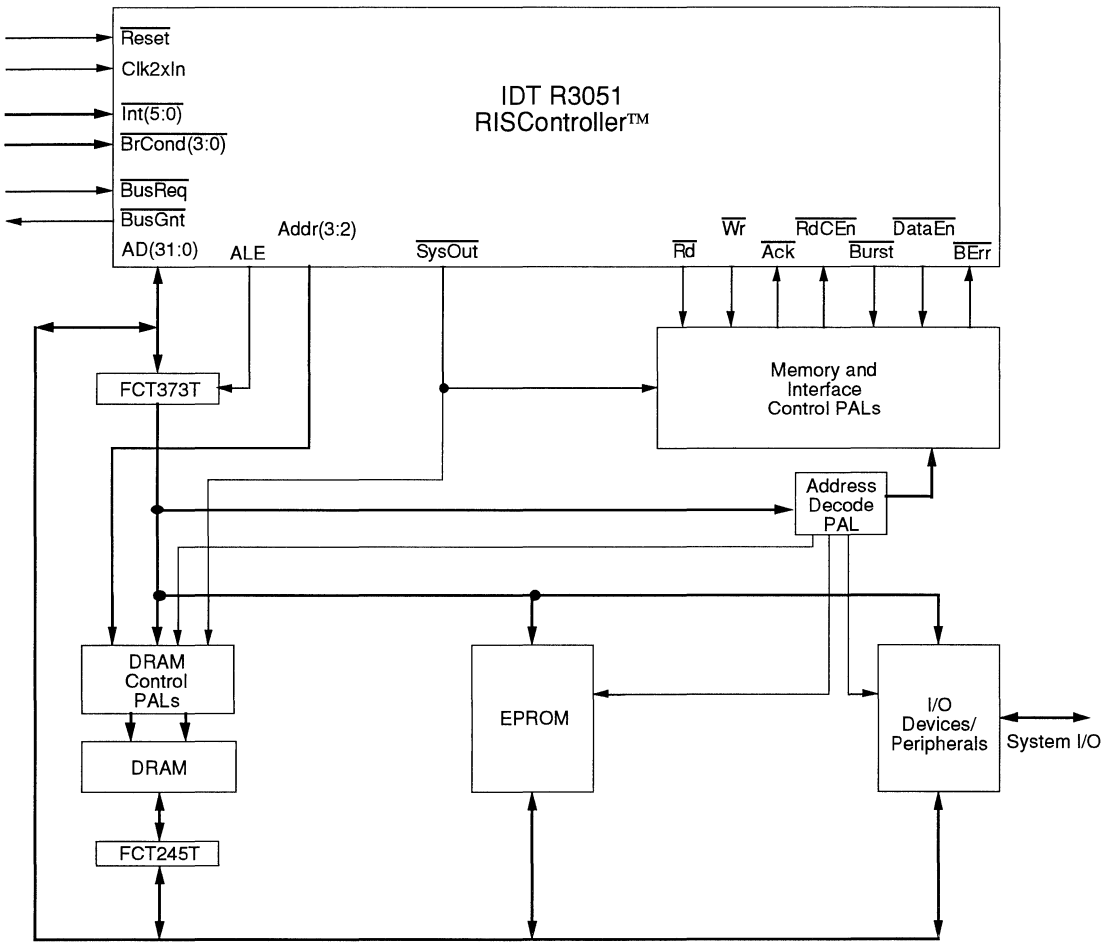


Figure 2. R3051-Based System

handshake (in lower-performance, simple systems) or utilize a tighter timing mode when the memory system can generate burst data at the processor clock rate. Thus, the system designer can choose to utilize page- or nibble-mode DRAMs (and possibly use interleaving, if desired, in high-performance systems, or use simpler techniques to reduce complexity).

In order to accommodate slower quad-word reads, the R3051 incorporates a 4-deep read buffer FIFO so that the external interface can queue up data within the processor before releasing it to perform a burst fill of the internal caches.

SYSTEM USAGE

The IDTR3051 has been specifically designed to easily connect to low-cost memory systems. Typical low-cost memory systems utilize slow EPROMs, DRAMs and application specific peripherals. These systems may also typically contain large, slow static RAMs although the IDTR3051 has been designed to not require the use of external SRAMs to achieve high performance.

Figure 2 shows a typical system block diagram. Transparent latches are used to demultiplex the R3051 address and data busses from the AD bus. The data paths between the memory system elements and the R3051 AD bus is managed by simple octal devices. A small set of simple PALs is used to control the various data path elements, and to control the handshake between the memory devices and the R3051.

Depending on the cost versus performance trade-offs appropriate to a given application, the system design engineer could include true burst support from the DRAM to provide for high-performance cache-miss processing.

DEVELOPMENT SUPPORT

The IDTR3051 is supported by a rich set of development tools, ranging from system simulation tools through Prom monitor support, logic analysis tools and subsystem modules.

Figure 3 is an overview of the system development process typically used when developing R3051 applications. The R3051 is supported in all phases of project development. These tools allow timely, parallel development of hardware and software for R3051 based applications and include tools such as:

- A program, Cache-3051™, which allows the performance of an R3051-based system to be modeled and understood without requiring actual hardware.

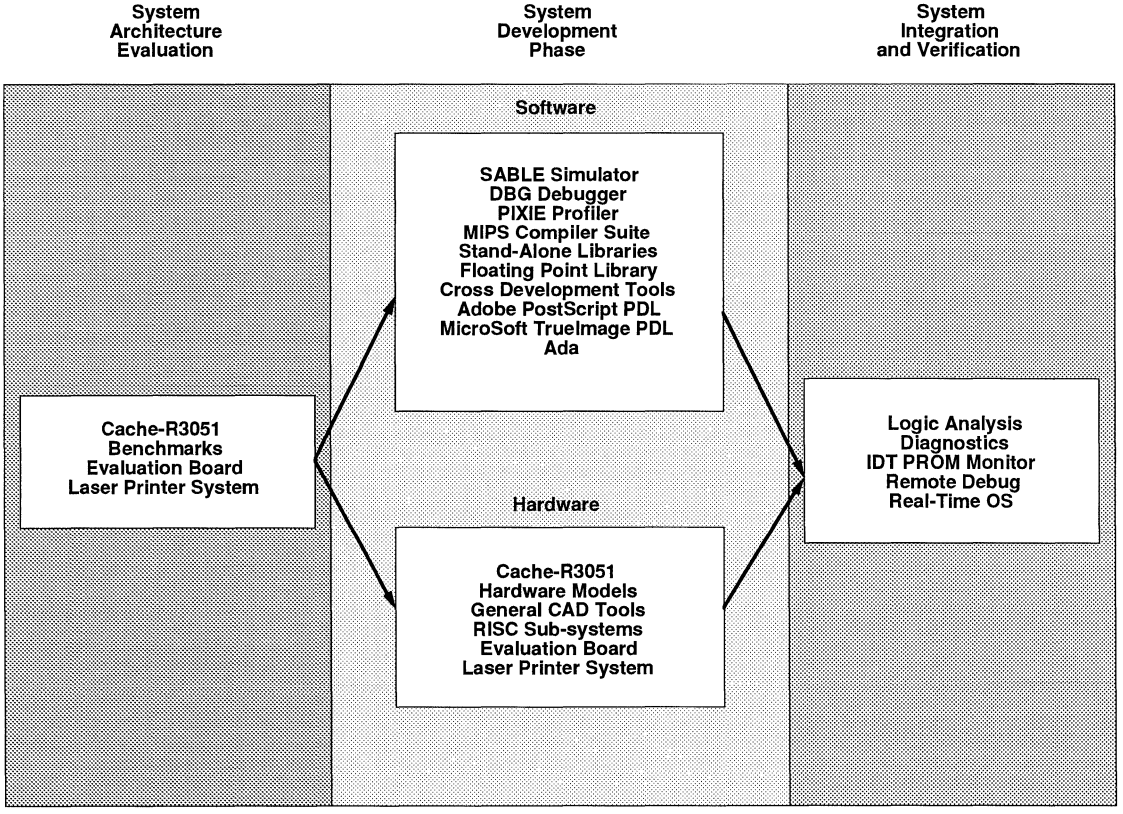


Figure 3. R3051 Development Support Tool Chain

- Sable, an instruction set simulator.
- Optimizing compilers from MIPS, the acknowledged leader in optimizing compiler technology.
- Cross development tools, available in a variety of development environments.
- The high-performance IDT floating-point library software.
- The IDT Evaluation Board, which includes RAM, EPROM, I/O, and the IDT PROM Monitor.
- The IDT Laser Printer System board, which directly drives a low-cost print engine, and runs Microsoft TrueImage™ Page Description Language on top of PeerlessPage™ Advanced Printer Controller BIOS.
- Adobe PostScript® Page Description Language, ported to the R3000 instruction set, runs on the IDT R3051.
- The IDT R3051 PROM Monitor, which implements a full PROM monitor (diagnostics, remote debug support, peek/poke, etc.).

PERFORMANCE OVERVIEW

The R3051 achieves a very high level of performance. This performance is based on:

- An efficient execution engine. The CPU performs ALU operations and store operations in single cycle, has an effective load time of 1.3 cycles, and branch effective execution time of 1.5 cycles (based on the ability of the compilers to avoid software interlocks). Thus, the execution engine achieves over 22mips performance when operating out of cache.
- Large on-chip caches. The R3051 contains caches which are substantially larger than those on the majority of today's microprocessors. These large caches minimize the number of bus transactions required, and allow the R3051 to achieve actual sustained performance very close to its peak execution rate.
- Autonomous multiply and divide operations. The R3051 features an on-chip integer multiplier/divide unit which is separate from the other ALU. This allows the R3051 to perform multiply or divide operations in parallel with other integer operations, using a single multiply or divide instruction rather than "step" operations.
- Integrated write buffer. The R3051 features a 4-deep write buffer which captures store target addresses and data at the processor execution rate and retires it to main memory at the slower main memory access rate. Use of on-chip write buffers eliminates the need for the processor to stall when performing store operations.
- Burst read support. The R3051 enables the system designer to utilize page-mode or nibble-mode RAMs when performing read operations to minimize the main memory read penalty and increase the effective cache hit rates.

These techniques combine to allow the processor to achieve over 20mips integer performance without the use of external caches or zero wait-state memory devices.

Integrated Device Technology, Inc.

INTRODUCTION TO THE NEW R3081™ PROCESSOR

CONFERENCE PAPER CP-08

By Philip Bourekas

INTRODUCTION

The IDTR3051™ family is a series of high-performance 32-bit microprocessors featuring a high level of integration, and targeted to high-performance but cost-sensitive processing applications. The R3051 family is designed to bring the high performance inherent in the MIPS RISC architecture into low-cost, simplified, power-sensitive applications.

Thus, functional units have been integrated onto the CPU core to reduce the total system cost, rather than to increase the inherent performance of the integer engine. Nevertheless, the R3051 family is able to offer 35VUPS performance at 40MHz without requiring external SRAM or caches.

The R3081™ extends the capabilities of the R3051 by integrating additional resources into the same pinout. The R3081 family thus extends the range of applications addressed by the R3051 family, and allows designers to implement a single, base system and software set capable of accepting a wide variety of CPUs, according to the price/performance goals of the end system.

In addition to the embedded applications served by the R3051 family, the R3081 allows low-cost, entry-level computer systems to be constructed. These systems will offer many times the performance of traditional PC systems, yet cost approximately the same. The R3081 is able to run any of the various other operating systems ported to the MIPS R3000 architecture. Thus, the R3081 can be used to build a low-cost system, further widening the range of performance solutions of the ACE Initiative.

This paper provides a brief overview of the R3081 processor; consult the "R3081 Family Hardware User's Guide" for a complete description of this processor.

DEVICE OVERVIEW

The R3051 family offers a wide range of functionality in a compatible interface. The R3051 family allows the system designer to implement a single base system, and utilize interface-compatible processors of various complexity to achieve the price/performance goals of the particular end system.

Differences among the various R3051 family members pertain to the on-chip resources of the processor. Current family members include:

- The R3052E, which incorporates an 8kB instruction cache, a 2kB data cache, and full function memory management unit (MMU) including 64-entry fully associative Translation Look-aside Buffer (TLB).
- The R3052, which also incorporates an 8kB instruction cache and 2kB data cache, but does not include the TLB, and instead uses a simpler virtual to physical address mapping.
- The R3051E, which incorporates 4kB of instruction cache and 2kB of data cache, along with the full function MMU/TLB of the R3000A.
- The R3051, which incorporates 4kB of instruction cache and 2kB of data cache, but omits the TLB, and instead uses a simpler virtual to physical address mapping.
- The R3081E, which incorporates a 16kB instruction cache, a 4kB data cache, and full function memory management unit (MMU) including 64-entry fully associative Translation Look-aside Buffer (TLB). The cache on the R3081E is user-configurable to an 8kB Instruction Cache and 8kB Data Cache.
- The R3081, which incorporates a 16kB instruction cache, a 4kB data cache, but uses the simpler memory mapping of the R3051/52, and thus omits the TLB. The cache on the R3081 is user configurable to an 8kB Instruction Cache and 8kB Data Cache.

Figure 1 shows a block level representation of the functional units within the R3081E. The R3081E could be viewed as the embodiment of a discrete solution built around the R3000A and R3010A. However, by integrating this functionality on a single chip, dramatic cost and power reductions are achieved.

CPU Core

The CPU core is a full 32-bit RISC integer execution engine, capable of sustaining close to single cycle execution. The CPU core contains a five-stage pipeline, and 32 orthogonal 32-bit registers. The R3081 uses the same basic integer execution core as the entire R3051 family, which is the R3000A implementation of the MIPS instruction set. Thus, the R3081 family is binary-compatible with the R3051, R3052, R3000A, R3001 and R3500 CPUs. In addition, the R4000 represents an upwardly software compatible migration path to still higher levels of performance.

The execution engine in the R3081 uses a five-stage pipeline to achieve near single-cycle instruction execution rates. A new instruction can be initiated in each clock cycle; the execution engine actually processes five instructions concurrently (in various pipeline stages). Figure 2 shows the concurrency achieved in the R3081 execution pipeline.

System Control Coprocessor

The R3081 family also integrates on-chip the System Control Coprocessor, CP0. CP0 manages both the exception handling capability of the R3081, as well as the virtual to physical address mapping.

As with the R3051 and R3052, the R3081 family offers two versions of memory management and virtual to physical

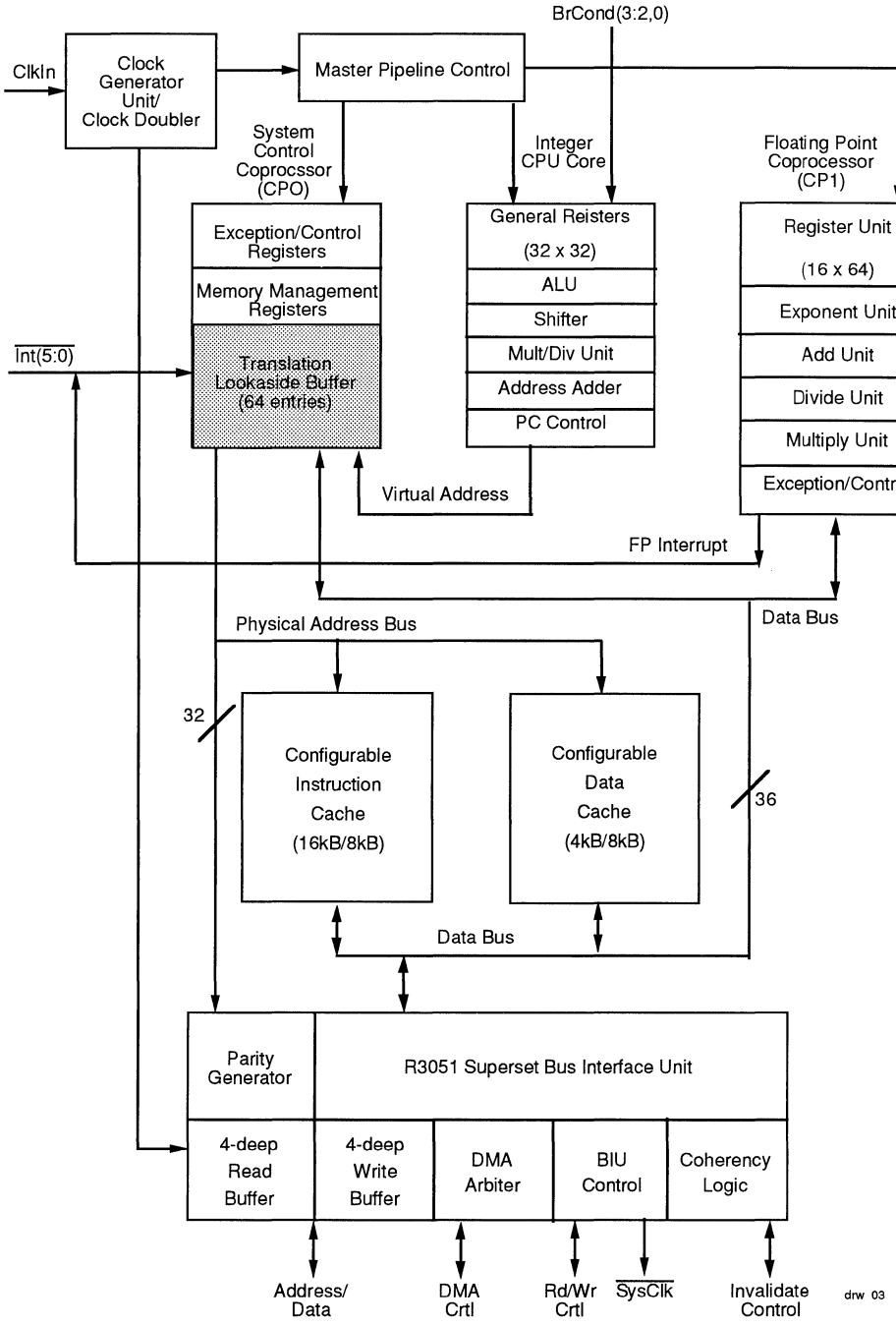


Figure 1. R3081 Block Diagram

address mapping: the extended architecture versions (the R3051E, R3052E and R3081E) incorporate the same full-function MMU as the R3000A. These versions contain a fully associative 64-entry TLB which maps 4kB virtual pages into the physical address space. The virtual to physical mapping thus includes kernel segments which are hard-mapped to physical addresses and kernel and user segments which are mapped page by page by the TLB into anywhere in the 4GB physical address space. In this TLB, 8-page translations can be "locked" by the kernel to insure deterministic response in real-time applications. Figure 3 illustrates the virtual to physical mapping found in the R3081E.

The Extended architecture versions of the R3051 family (the R3051E, R3052E, and R3081E) allow the system designer to implement kernel software which dynamically manages User task utilization of system resources, and also allows the Kernel to protect certain resources from User tasks. These capabilities are important in general computing applications such as ARC computers and are also important in a variety of embedded applications, from process control (where protection may be important) to X-Window display systems (where virtual memory management can be used). The MMU can also be used to simplify system debug.

R3051 family base versions (the R3051, R3052, and R3081) remove the TLB and institute a fixed address mapping for the various segments of the virtual address space. These devices still support distinct kernel and user mode operation, but do not require page management software, leading to a simpler software model. The memory mapping used by these devices is shown in Figure 4. Note that the reserved spaces are for compatiblity with future family members which may map on-chip resources to these addresses. References to these addresses in the R3081 will be translated in the same fashion as the rest of their respective segments with no traps or exceptions signalled.

When using the base versions of the architecture, the system designer can implement a distinction between the user tasks and the kernel tasks without having to implement page management software. This distinction can be implemented by decoding the output physical address. In systems which do not need memory protection, and wish to have the kernel and user tasks operate out of the same memory space, high-order address lines can be ignored by the address decoder and thus all references will be seen in the lower gigabyte of the physical address space.

Floating-Point Coprocessor

The R3081 also integrates an R3010A compatible floating-point accelerator on-chip. The FPA is a high-performance coprocessor (coprocessor 1 to the CPU) providing separate add, multiply, and divide functional units for single- and double-precision floating-point arithmetic. The floating-point accelerator features low latency operations and autonomous functional units which allow differing types of floating-point

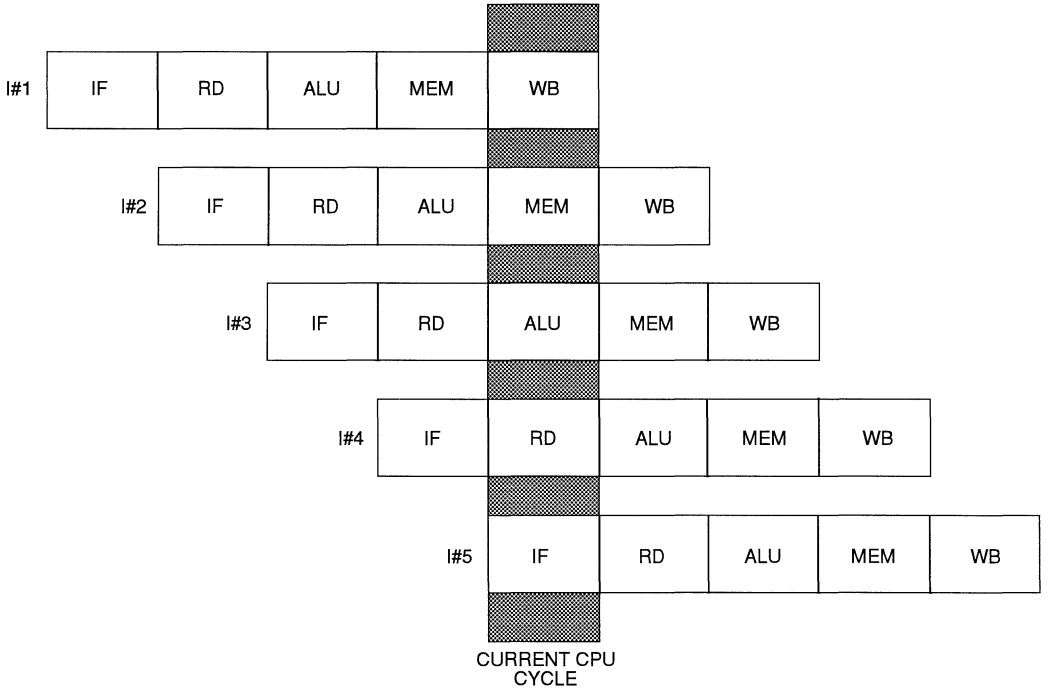


Figure 2. R3081 5-Stage Pipeline

operations to function concurrently with integer operations. The FPA appears to the software programmer as a simple extension of the integer execution unit with 16 dedicated 64-bit floating-point registers (software references these as 32 32-bit registers when performing loads or stores). Figure 5 illustrates the functional block diagram of the on-chip FPA.

Clock Generator Unit

The R3081 is driven from a single input clock which can be either at the processor rated speed, or at twice that speed. On-chip, the clock generator unit is responsible for managing the interaction of the CPU core, caches, and bus interface. The R3081 includes an on-chip clock doubler to provide higher-frequency signals to the internal execution core; if 1x clock mode is selected, the clock doubler will internally convert it to a double-frequency clock. The 2x clock mode is provided for compatiblity with the R3051. The clock generator unit replaces the external delay line required in R3000A-based applications.

Instruction Cache

The R3081 implements a 16kB Instruction Cache. The system may choose to repartition the on-chip caches, so that the instruction cache is reduced to 8kB but the data cache is increased to 8kB. The instruction cache is organized with a line size of 16 bytes (four entries). This large cache achieves hit rates in excess of 98% in most applications and substantially contributes to the performance inherent in the R3081. The cache is implemented as a direct mapped cache and is capable of caching instructions from anywhere within the 4GB physical address space. The cache is implemented using physical addresses (rather than virtual addresses) and thus does not require flushing on context switch.

The instruction cache is parity protected over the instruction word and tag fields. Parity is generated by the read buffer during cache refill; during cache references, the parity is checked, and in the case of a parity error, a cache miss is processed.

Data Cache

The R3081 incorporates an on-chip data cache of 4kB, organized as a line size of 4 bytes (one word). The R3081 allows the system to reconfigure the on-chip cache from the default 16kB I-Cache/4kB D-Cache to 8kB of Instruction and 8kB of Data caches.

The relatively large data cache achieves hit rates in excess of 95% in most applications, and contributes substantially to the performance inherent in the R3081. As with the instruction cache, the data cache is implemented as a direct mapped physical address cache. The cache is capable of mapping any word within the 4GB physical address space.

The data cache is implemented as a write-through cache, to insure that main memory is always consistent with the internal cache. In order to minimize processor stalls due to data write operations, the bus interface unit incorporates a 4-deep write buffer which captures address and data at the processor execution rate, allowing it to be retired to main

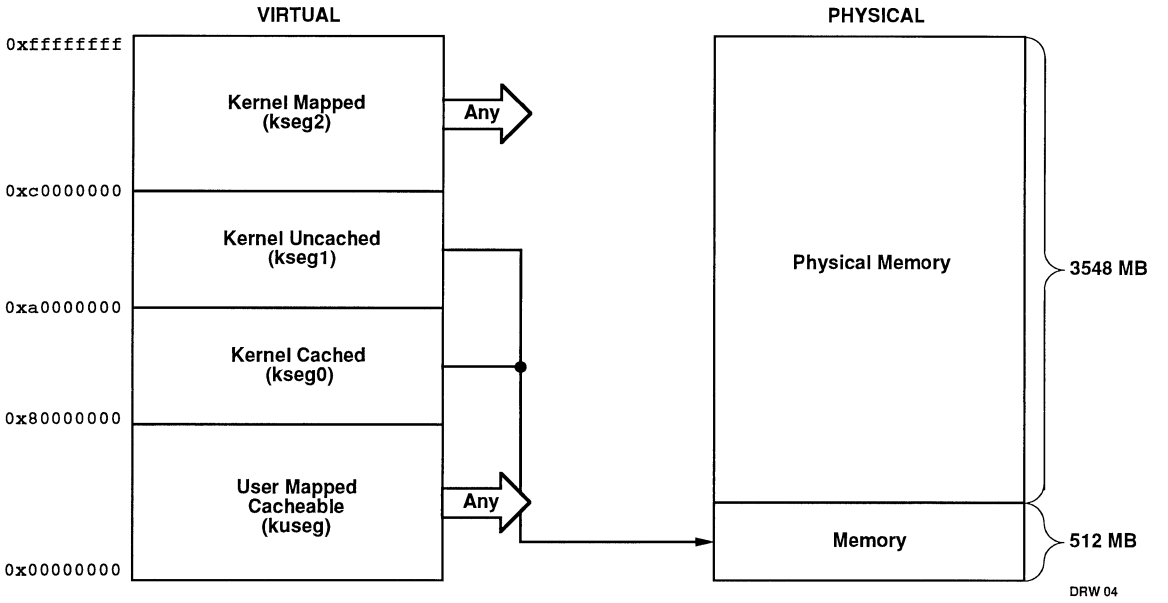


Figure 3. Virtual to Physical Mapping of Extended Architecture Versions

memory at a much slower rate without impacting system performance. Further, support has been provided to allow hardware-based data cache coherency in a multi-master environment, such as one utilizing DMA from I/O to memory.

The data cache is parity protected over the data and tag fields. Parity is generated by the read buffer during cache refill; during cache references, the parity is checked, and in the case of a parity error, a cache miss is processed.

Bus Interface Unit

The R3081 uses its large internal caches to provide the majority of the bandwidth requirements of the execution engine, and thus can utilize a simple bus interface connected to slower memory devices. Alternately, a high-performance, low-cost secondary cache can be implemented, allowing the processor to increase performance in systems where bus bandwidth is a performance limitation.

The R3051 family bus interface utilizes a 32-bit address and data bus multiplexed onto a single set of pins. The bus interface unit also provides an ALE (Address Latch Enable) output signal to demultiplex the A/D bus, and simple handshake signals to process CPU read and write requests. In addition to the read and write interface, the R3051 family incorporates a DMA arbiter to allow an external master to control the external bus.

The R3081 also supports hardware-based cache coherency during DMA writes. The R3081 can invalidate a specified line of data cache, or in fact can perform burst invalidations during burst DMA writes.

The R3081 incorporates a 4-deep write buffer to decouple the speed of the execution engine from the speed of the memory system. The write buffers capture and FIFO processor address and data information in store operations and present it to the bus interface as write transactions at the rate the memory system can accommodate.

The R3081 read interface performs both single-datum reads and quad-word reads. Single reads work with a simple handshake and quad-word reads can either utilize the simple handshake (in lower-performance, simple systems), or utilize a tighter timing mode when the memory system can burst data at the processor clock rate. Thus, the system designer can choose to utilize page- or nibble-mode DRAMs (and possibly use interleaving, if desired, in high-performance systems) or use simpler techniques to reduce complexity.

In order to accommodate slower quad-word reads, the R3081 incorporates a 4-deep read buffer FIFO, so that the external interface can queue-up data within the processor before releasing it to perform a burst fill of the internal caches.

The R3081 is R3051 superset-compatible in its bus interface. Specifically, the R3081 has additional support to simplify the design of very high-frequency systems. This support includes the ability to run the bus interface at one-half the processor execution rate, as well as the ability to slow the

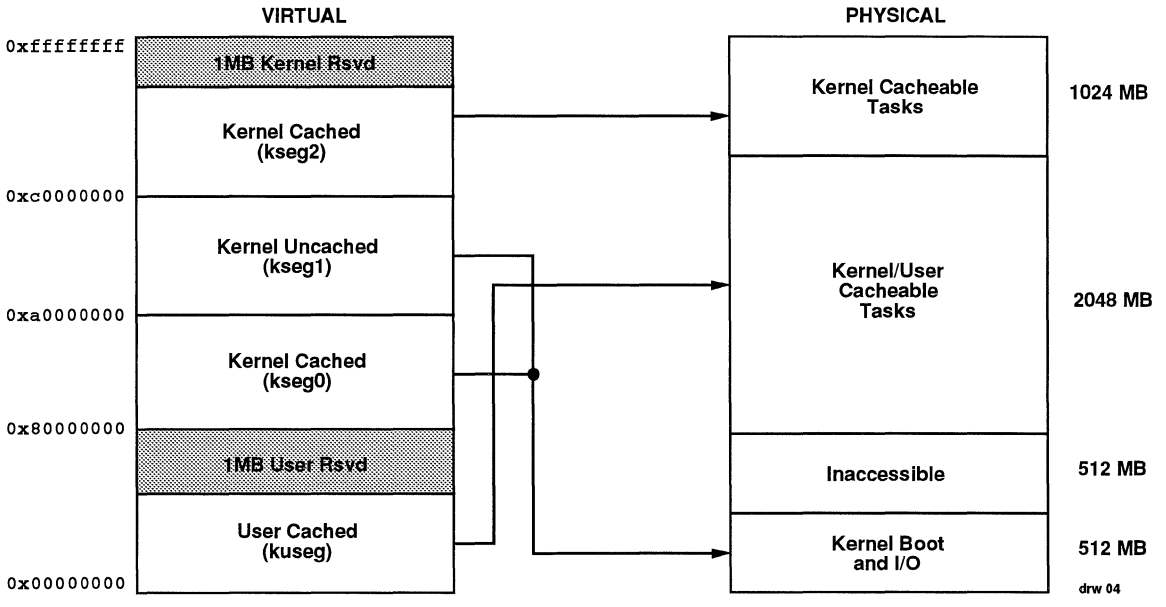


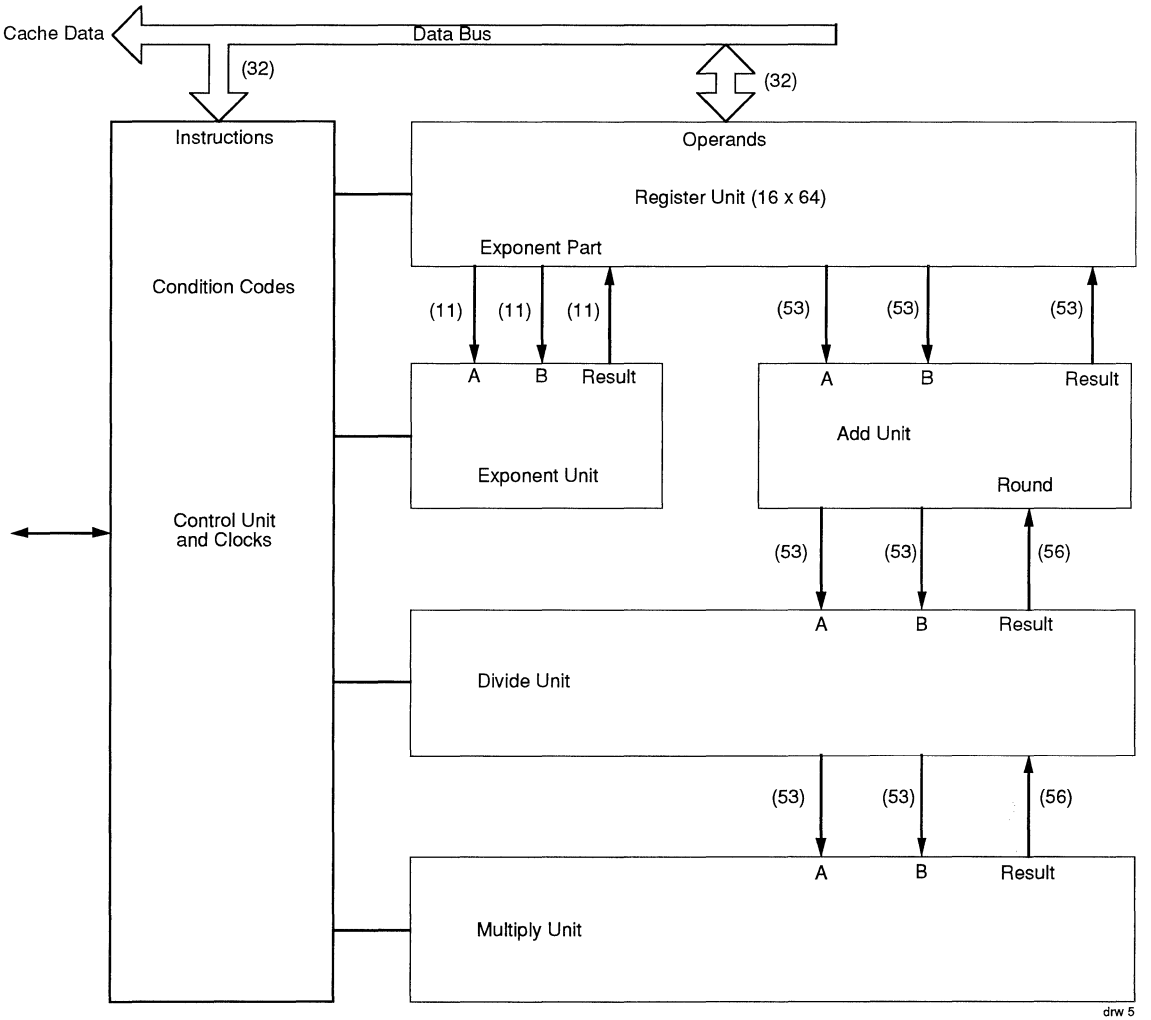
Figure 4. Virtual to Physical Mapping of Base Architecture Versions

transitions between reads and writes to provide extra buffer disable time for the memory interface. However, it is still possible to design a system which, with no modification to the PC Board or software, can accept either an R3051, R3052, or R3081.

SYSTEM USAGE

The IDT R3051 family has been specifically designed to allow a wide variety of memory systems. Low-cost systems can use slow-speed memories and simple controllers, while other designers may choose to incorporate higher frequencies, faster memories, and techniques such as DMA to achieve maximum performance. The R3081 includes specific support for high-performance systems, including signals necessary to implement external secondary caches and the ability to perform hardware-based cache coherency in multi-master systems.

Figure 6 shows a typical system implementation. Transparent latches are used to demultiplex the R3081 address and data busses from the A/D bus. The data paths between the memory system elements and the A/D bus is managed by simple octal devices. A small set of simple PALs is used to control the various data-path elements and to control the handshake between the memory devices and the CPU. IDT has implemented the R3720/21 support chip set specifically



Figre 5. FPA Functional Block Diagram

tailored to R3051 family systems. This chip set directly interfaces the processor to DRAM, simplifying design and eliminating discrete logic chips and PAL devices.

Depending on the cost versus performance trade-offs appropriate to a given application, the system design engineer could include true burst support from the DRAM to provide for high-performance cache-miss processing, or utilize a simpler, lower-performance memory system to reduce cost and simplify the design. Similarly, the system designer could choose to implement techniques such as external secondary cache, or DMA, to further improve system performance.

DEVELOPMENT SUPPORT

The IDT R3051 family is supported by a rich set of development tools, ranging from system simulation tools through PROM monitor and debug support, applications software and utility libraries, logic analysis tools, subsystem modules and shrink-wrap operating systems. The R3081, which is pin and software compatible with the R3051, can directly utilize these existing tools to reduce time to market.

Figure 7 is an overview of the system development process typically used when developing R3051 family applications. The R3051 family is supported in all phases of project devel-

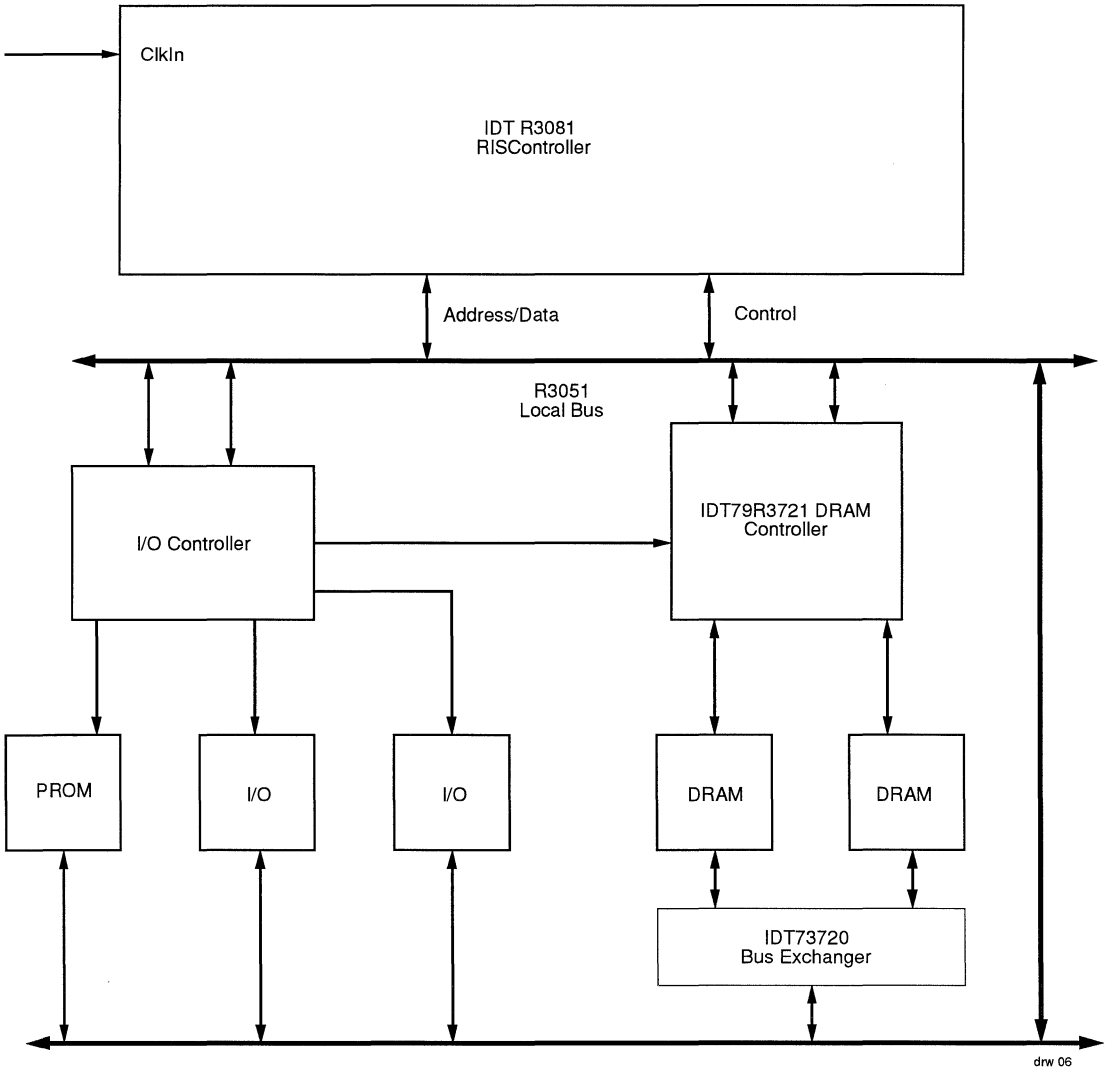


Figure 6. R3081 RISChipset Based System

opment. These tools allow timely, parallel development of hardware and software for R3051 family applications, and include tools such as:

- A program, Cache-3051™, which allows the performance of an R3051 family system to be modeled and understood without requiring actual hardware.
- Sable, an instruction set simulator.
- Optimizing compilers from MIPS, the acknowledged leader in optimizing compiler technology.
- Cross development tools, available in a variety of development environments.
- The high-performance IDT floating-point library software, including transcendental functions and IEEE-compliant exception handlers.
- The IDT Evaluation Board, which includes RAM, EPROM, I/O, and the IDT PROM Monitor.
- The IDT Laser Printer System board, which directly drives a low-cost print engine, and runs Microsoft TrueImage™ Page Description Language on top of PeerlessPage™ Advanced Printer Controller BIOS.
- Adobe PostScript® Page Description Language, ported to the R3000 instruction set, runs on the IDT R3051 family.
- IDT/sim, which implements a full PROM monitor (diagnostics, remote debug support, peek/poke, etc.).
- IDT/sae, which implements a run-time support package for R3051 family systems.
- Various Operating Systems ported to the R3000, including ACE UNIX®.

PERFORMANCE OVERVIEW

The R3081 achieves a very high level of performance. This performance is based on:

- **An efficient execution engine.** The CPU performs ALU operations and store operations in a single cycle, and has an effective load-time of 1.3 cycles, and branch execution rate of 1.5 cycles (based on the ability of the compilers to avoid software interlocks). Thus, the execution engine achieves over 35VUPS performance when operating out of cache, and equivalently high SPECmark performance.
- **A full-featured floating-point accelerator/coprocessor.** The R3081 incorporates an R3010A compatible floating-point accelerator on-chip, with independent ALUs for floating-point add, multiply, and divide. The floating-point unit is fully hardware-interlocked, and features overlapped operation and precise exceptions. The FPA allows floating-point adds, multiplies and divides to occur concurrently with each other, as well as concurrently with integer operations.
- **Large on-chip caches.** The R3051 family contains caches which are substantially larger than those on the majority of

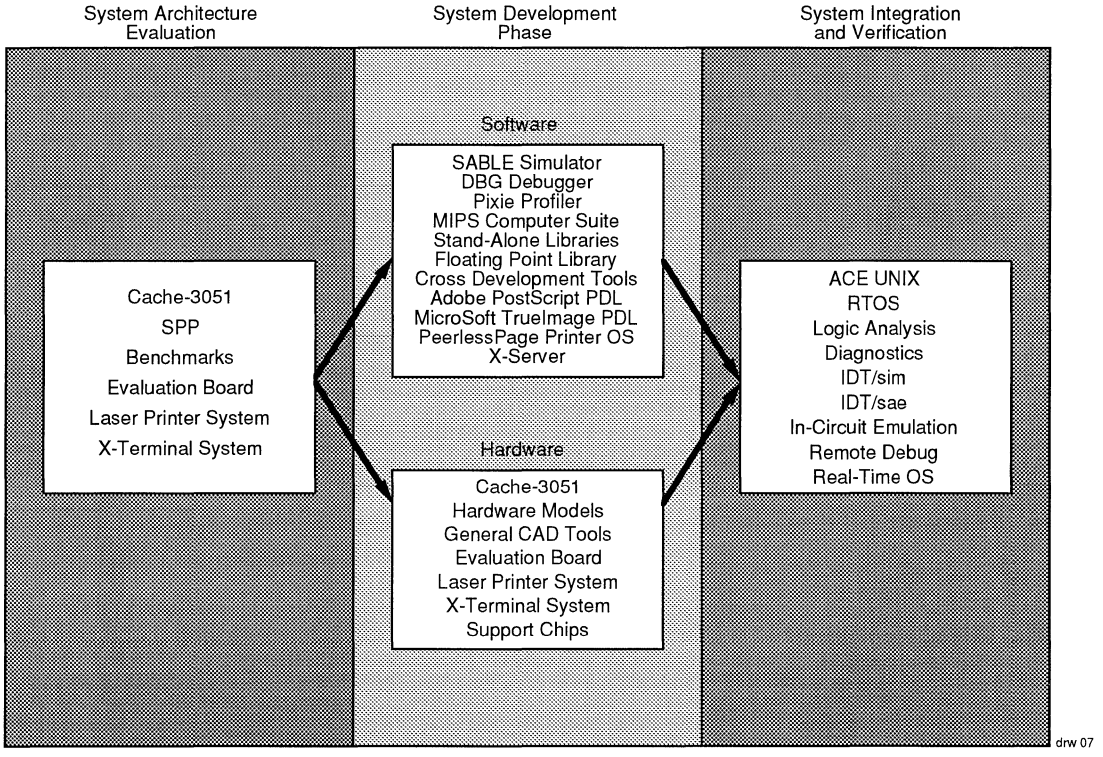


Figure 7. R3051 Family Development Toolchain

today's microprocessors. These large caches minimize the number of bus transactions required, and allow the R3051 family to achieve actual sustained performance very close to its peak execution rate. The R3081 doubles the cache available on the R3052, making it a suitable engine for many general-purpose computing applications, such as ACE systems.

- **Autonomous multiply and divide operations.** The R3051 family features an on-chip integer multiplier/divide unit which is separate from the other ALU. This allows the CPU to perform multiply or divide operations in parallel with other integer operations, using a single multiply or divide instruction rather than "step" operations.
- **Integrated write buffer.** The R3081 features a four deep write buffer, which captures store target addresses and data at the processor execution rate and retires it to main memory at the slower main memory access rate. Use of on-chip write buffers eliminates the need for the processor to stall when performing store operations.
- **Burst read support.** The R3051 family enables the system designer to utilize page-mode or nibble-mode RAMs when performing read operations to minimize the main memory read penalty and increase the effective cache hit rates.

The performance differences between the various R3051 family members depends on the application software and the design of the memory system. The impact of the various cache sizes, and the hardware floating-point, can be accurately modeled using Cache-3051. Since the R3051, R3052, and R3081 are all pin and software compatible, the system designer has maximum freedom in trading between performance and cost. A system can be designed, and later the appropriate CPU inserted into the board, depending on the desired system performance.

SELECTABLE FEATURES

The R3081 allows the system designer to configure certain aspects of operation. Some of these options are established when the device is reset, while others are enabled via the Config registers:

- **Big Endian vs. Little Endian byte ordering.** The part can be configured to operate with either byte ordering. ACE systems typically use Little Endian byte ordering. However, various embedded applications, written originally for a Big Endian processor such as the MC680x0, are easier to port to a Big Endian system.
- **Data Cache Refill of one or four words.** The memory system must be capable of performing four word refills of instruction cache misses. The R3081 allows the system designer to enable D-Cache refill of one or four words dynamically. Thus, specialized algorithms can choose one refill size, while the rest of the system can operate with the other.
- **Half-frequency bus mode.** The processor can be configured such that the external bus interface is at one-half the frequency of the processor core. This simplifies system design; however, the large on-chip caches mitigate the performance impact of using a slower system bus clock.
- **Slow bus turn-around.** The R3081 allows the system designer to space processor operations, so that more time is allowed for transitions between memory and the processor on the multiplexed address/data bus.
- **Configurable cache.** The R3081 allows the system designer to use software to select either a 16kB Instruction Cache/4kB Data Cache organization, or an 8kB Instruction/8kB Data Cache organization.
- **Cache Coherent interface.** The R3081 has an optional hardware based cache coherency interface intended to support multi-master systems such as those utilizing DMA between memory and I/O.
- **Optional 1x or 2x clock input.** The R3081 can be driven with an R3051-compatible 2x clock input, or a lower frequency 1x clock input.

INTERCHANGEABILITY WITH OTHER R3051 DEVICES

The R3081 family has been designed to allow interchangeability with other members of the R3051 family, with no changes to the PCB. The last chapter of the R3081 User's Manual describes the various design considerations involved. Upgrade options within the R3051 family now include:

- Upgrading an R3051 to an R3052. This doubles the amount of instruction cache, without modifying the frequency of the system, and thus could be offered as a field upgrade.
- Upgrading an R3051 or R3052 to an R3081 at the same frequency. This would have the effect of increasing both the instruction and data cache sizes. In addition, the hardware floating-point unit would be available to upgrade system floating-point performance. This upgrade results in a substantial performance gain, with no board redesign.
- Upgrading an R3051 or R3052 to an R3081 running at twice the frequency but the same bus interface speed. For example, it is possible to upgrade a 20MHz R3051 to a 40MHz R3081. The R3081 would run in "1x clock mode", so no changes to the input clock would be required; the R3081 bus could be run at one-half the processor speed (20MHz), so no other system changes are required. This upgrade results in twice the amount of cache, at twice the execution rate, with additional hardware floating-point support, yet requires no real modifications to the PCB or other components.

SUMMARY

The R3081 further extends the range of price-performance served by the R3051 family. By offering different devices with the same footprint, a single hardware design effort is leveraged into multiple end products, each addressing different price/performance points.

Integrated Device Technology, Inc.

IDT RISC TECHNOLOGY AND DESIGNING WITH CACHE COHERENCY IN MIND

CONFERENCE PAPER CP-09

By Barry Seidner

INTRODUCTION

The objective of this paper is to discuss new performance standards by which to judge the attributes of new RISC architectures that were designed for very different applications. As any microprocessor design engineer with experience will testify, microprocessors seem architected to ease design—not validate performance. Digital design engineers can get their arms around the interface requirements, which are finite enough to make trade-off studies, and they can look at support chips to make sure the peripherals meet the needs of the design. If the needs are not met they can look at what other peripherals can meet those needs and how many PAL devices it will take to use them on the selected processor.

The less tangible part of microprocessor selection is determining if a processor will meet the challenging demands of today's software requirements. Most cursory investigations will involve the comparison of standard benchmarks. One commonly used set is called the "Intel Benchmark Suite", because Intel showed the original six that include Quicksort, Bubblesort, Pi-500, Anneal, Matmult and Dhrystone 1.1. The first problem not readily apparent is that Intel does not state their latencies into main memory. In fact, they boast performance on an all-SRAM system and then a DRAM system. What is of note here is that all the details of main memory latency are important and missing. This leaves the designer to depend on his benchmarks and define the latency he will use.

The fact is that the majority of RISC chip designers are just like the user. Because it takes years to get a reliable CPU into production, they design processors on hardware features, not proven performance. But MIPS RISC architectures are different. They were designed by *first* architecting the instruction set, writing compilers, selecting 20 large benchmarks (the code sets were UNIX®-oriented applications in the multiple-megabyte size range) and performing two years of optimization. The *last* thing MIPS did was to define the CPU hardware architecture to execute the instruction set and the compilers.

Trade-offs continued into the CPU development as well. Illustrated later in the feature set of MIPS RISC processors is a background autonomous multiply/divide unit with 64-bit result versus the implementation of a multiply step instruction. This feature allows multiply/divides to run while other code continues, and because the compilers are designed to facilitate this, a vectorization of code occurs. Also, for the first time, a Write Buffer was included in the architecture to prevent CPU write stalls. This feature helps lessen the problem of CPU Write bandwidth to main memory Write bandwidth.

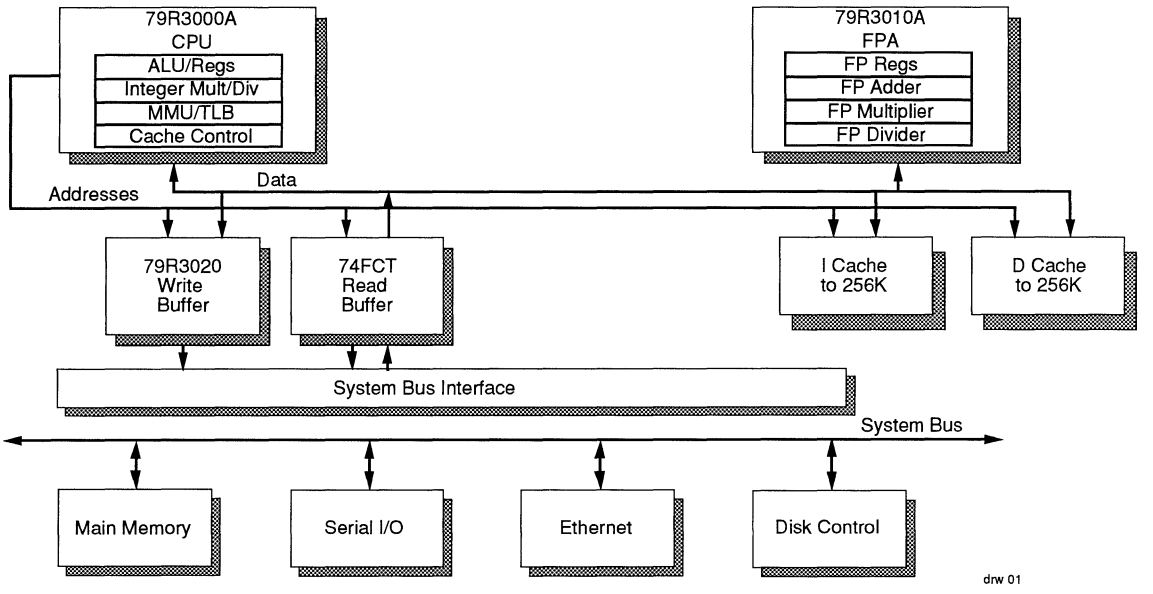


Figure 1. Typical R3000A System Block Diagram

Briefly discussed, one can see a clear difference between MIPS RISC architecture development versus others. In the other camp, processor architectures are presented to compiler writers after the fact and they are forced to deal with a non-conforming architecture. Even in situations where many compiler technology companies are pitted against each other, (e.g., Metaware and Green Hills funded to do compilers for a given architecture where the best gets the business) the outcome falls short.

SYSTEMS SHIPPING TODAY

Todays' high-performance systems have been shipping the 79R3000A for three years. Most notably, Silicon Graphics, DEC, CDC, Tandem, Pyramid, NEC and Sony have found the performance of the R3000A to be profound. These systems still ship the MIPS RISC architecture known as MIPS 1. It includes the R3000A CPU, the optional R3010A floating-point hardware accelerator, write buffer logic and as many as thirty 64K SRAMs to make up the cache. MIPS 2, which has been in production for over a year, executes on the R6000 processor. Implemented in ECL logic, the processor is now shipping in excess of 66MHz. MIPS 3, recently announced, defines the R4000. The R4000 is a super-pipelined RISC that is a superset of MIPS 1 and MIPS 2, where all application code is upwardly binary compatible.

MIPS 1 systems define yesterday's two-chip set—the 79R3000A RISC CPU and the optional 79R3010A FPA, as described in Figure 1. From a software standpoint, they appear seamless operationally because both execution units examine every opcode simultaneously. If an integer operation is seen on the bus, the 79R3000A executes, or if it is a floating-point operation, the FPA executes it.

The CPU contains thirty-two 32-bit registers. All registers can be an ALU, can be shifted, and can be used as pointers, offsets, etc. On-chip is a 32-bit multiply/divide unit that gives 64-bit results that can execute concurrently with integer operations. The compiler technology explained previously examines the source code and picks the most effective method to schedule the multiply/divide unit. If the result is needed immediately, the compiler will pick the most effective way to perform the operation; if not, the autonomous unit will be scheduled, if inactive. In cases where a register is modified by a constant, the compilers will use clever code sequences to perform the operation. Where two registers are used, and the data is needed right away, the autonomous unit will be scheduled and the processor will interlock on references to the unit before data completion.

The 79R3000A RISC processor employs a five-stage pipeline to effectively schedule all phases of instruction execution. The stages are: instruction fetch, register read, ALU, memory read and write back. The FPA has a six-stage pipeline. Both pipelines move at the clock rate, synchronized by a phase-lock loop on the 79R3010A.This facilitates the seamless operation of the two independent devices. Since they have their own set of registers, integer operations don't interfere with floating-point operations and, as they are autonomous units, the compiler can overlap execution units for maximum performance. Also on the CPU is the MMU, which is coprocessor 0. This coprocessor is responsible for virtual-to-physical translations and exception processing. The MMU is fully associative, resulting in high MMU hit rates. The exception model is precise, which allows easy decoding of which instruction caused the exception, integer or floating-point. The 79R3000A has six dedicated interrupt inputs that are level-sensitive and can be sensed individually for six unique interrupts or can produce an offset into a jump table realizing 64 vectored interrupts.

The 79R3010A FPU has a separate set of sixteen 64-bit registers. Three separate operational units exist on the chip that include the add/subtract, multiply and divide functions. Their functions are overlappable and scheduled by the compiler. An example of overlap would show that multiplies can be overlapped with divides and add/subtracts overlapped with multiplies—resulting in all execution units operating in the FPU concurrently with integer processing. The CPU and FPA communicate over a set of individual lines indicating status. As an example, if the FPA cannot accept another operation, it will signal the CPU to stall the additional operation with the FpBusy signal. Additionally, since the instruction set includes floating-point comparisons, the architecture does not require the data to be moved into the R3000A. Compare is done by the FPA and the results are communicated via the FP-condition signal. This signal connects to the CPU CpCond input and the processor can jump on a true or false condition. Lastly, if a floating-point exception occurs as a result of an operation, the FpInt output of the FPA will signal the CPU to examine the situation.

CACHE MEMORIES

The data bus on the 79R3000A is a multiplexed data bus, time-shared by the data cache and the instruction cache, with 1/2 clock cycle coming from each. The cache is organized in a classical Harvard architecture, maximizing standard SRAM technology by providing a cache controller on-chip. The cache is configured as a 60-bit-wide path, consisting of 32 bits of data, 4 bits of data parity, 20 bits of tag, 3 parity bits on the tag and a valid bit. All cache accesses require all 20 bits of tag to be presented for comparison—allowing all 4GB to be cacheable.

Several clever techniques can be implemented to reduce cache density. One easy method takes advantage of the fact that on instruction cache misses a block refill occurs. Since the block always has the same tag, and the instruction cache is not written to otherwise, the tag cache depth can be shallower. Other methods include wrapping the tag bus back with latches for the low-order tag to eliminate some tag SRAMS there, as well. Semiconductor manufacturers continue to integrate functions on-chip, like incorporating an FCT373 function, dual cells that include I and D sides, to reduce parts count.

MAIN MEMORY INTERFACE

Whenever a cache miss occurs, main memory is referenced to obtain the required data. Data is cacheable if the address range falls within the user mode range, in special kernel mode

segments and is automatically loaded into the cache when recovered from memory. On writes, the 79R3000A maintains a cache write-through policy, so data is written to main memory with the cache. The Write Buffer captures the 36 bits of data, including parity and 32 bits of address, which can be retired at the bandwidth acceptable to main memory. The reason for Write Buffers become clear when an examination of cache bandwidth, main memory bandwidth and the rate of writes are examined. It is easy to see that the CPU can easily overwhelm memory and have to stall frequently. The Write Buffer separates the bandwidth mismatch by storing the data until main memory can accept the Write.

SOLUTIONS TO REDUCE COST

Integration has always been a key method in reducing system cost. Not only does this reduce the individual parts counts and costs, it also reduces PC board costs, PC layers, power, manufacturing costs and increases performance and reliability, etc. Semiconductor manufacturers continue to integrate for better solutions, to improve customer relationships through improved business and to separate themselves from the simple "we sell chips" suppliers.

IDT announced the release of RISCore™, the 79R3500, in September of 1991. The 79R3500 integrates the 79R3000A CPU and the 79R3010A FPA into a package that fits into the 175 PGA socket (actually 161 pins)—the most popular footprint shipped. Because the 79R3500 includes hardware floating-point, FP interrupts and the FP condition code are internally routed via the reset vectors to pick the decisions that are wired externally in discrete designs. Other benefits of the 79R3500 include reduced power, better board layout permitted by a smaller cache bus, an additional memory mapping option for non-TLB versions, a new set of features to reduce cache size and lower cost.

Three modes programmed via the reset vectors allow the elimination of several cache SRAMs. These options allow the elimination of tag bit comparisons, which are not possible on the 79R3000A as it requires all 20 bits to be compared. The first option limits cacheable main memory to 128MB and saves four bits of tag in both caches. Very few systems today have the bus width or allow the capacity to address the full 4GB of cacheable memory permitted by the 79R3000A. The second option eliminates the lower four bits to the tag bus. This mandates that the cache use at least 16k depth (i.e., 16k x 4) cache SRAMs (64kB). The lower four tag bits are redundant in caches of this depth. The last mode includes both features, eliminating 16 bits of cache SRAMs, and is clearly popular. Additionally, there is an option to eliminate parity

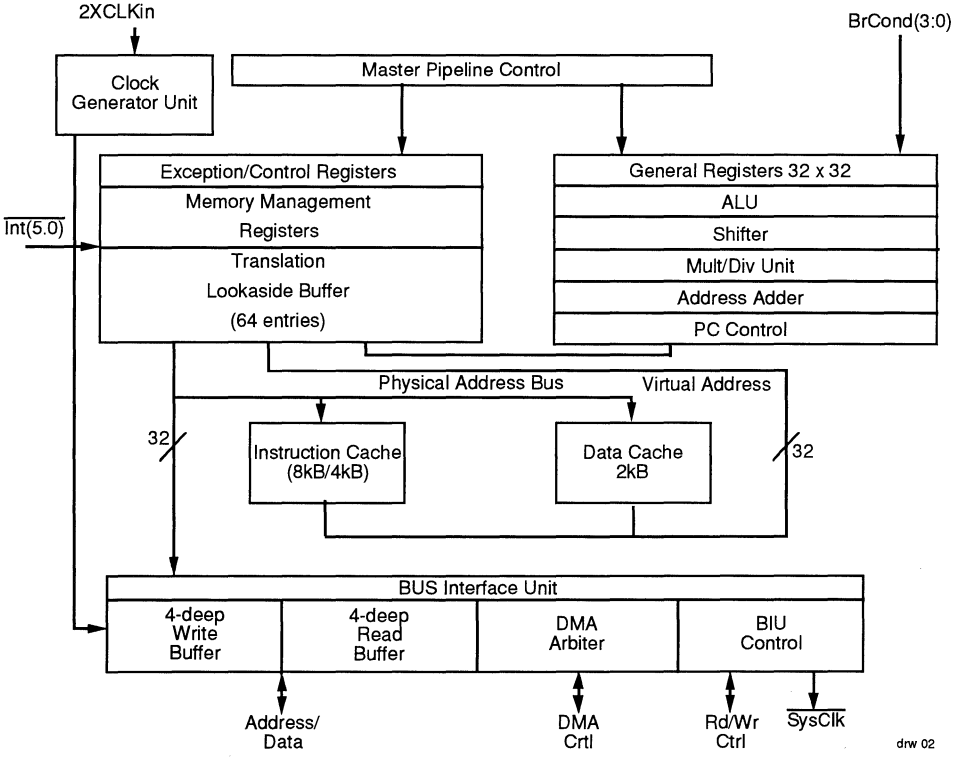


Figure 2. The R3051 RISController

checking in the cache. This option, combined with the last mode, reduces the cache bus length to 32 data + 1 valid + 12 tag = 45 bits—a clear cost savings!

Another cost savings solution is to address reducing the number of cache SRAMs. Currently available from IDT is the 71B229, a cache SRAM for all R3000A/R3500 systems. The IDT71B229 includes the latching function on-chip, eliminating the need for the external FCT373 address latches. It is x9 in width and is bicameral (16K x 9 x 2). The bicameral feature cuts the number of SRAMs needed in half because the bicameral SRAMs actually have two differentiated sections, where one is part of the instruction cache and the other is part of the data cache. This solution is also footprint-effective as it is available in a 300mil SOJ package. Using RISCore (which integrates the CPU and FPA) and the third cache option (which reduces the cache bus width to 45 bits), in conjunction with five 71B229 cache SRAMs, reduces yesterday's 32-chip solution down to 6 devices. This represents significant board and cost savings.

Clearly, these cost reduction solutions are significant for the ultimate-performance systems, but what about systems that don't require high-performance and are more cost sensitive? IDT recognized the need for cost-sensitive solutions where customers had experience with the MIPS ISA and compiler suite and wanted to stick with that architecture in their lower-performance applications. Also, designers in the x86 and 68020 performance market needed a cost-sensitive solution to improve performance and still have complete development support. IDT's solution was to maintain 100% code compatibility, *no* modification of the architecture and integrate to reduce cost. This solution is called the R3051™ RISController™. Second-sourced by Siemens and costing $30 when purchased in volume, it has clearly become a leading solution for the embedded marketplace.

THE R3051 RISController

The R3051 integrates the R3000A CPU, cache, a 4-deep read and 4-deep write buffer, signals to deal with page-mode DRAMs and other features to reduce cost at the system level, as shown in Figure 2. The cache includes 4kB of Instruction cache and 2kB of Data cache. Simulations of many benchmark suites, including SPEC, show the cache size to be above the knee of the curve to effectively address typical program loops. The upgrade to the R3051 is the R3052, which doubles the instruction cache to 8kB, is pin-for-pin compatible for instant performance upgrades without *any* hardware or software modifications.

The R305x products all come in cost-effective 84-pin PLCC packages that reduce board space and manufacturing, facilitated by a multiplexed 32-bit address/data bus. This bus is demultiplexed on a cache miss with ALE, which connects to standard FCT373 latches, in one clock. No performance is lost because the address flows through the FCT373, maximizing address propagation. When an examination of non-multiplexed systems are performed, the first clock is used the same way—to propagate the address. The DataEN output signal from the R3051 is used by the memory interface to detect when the processor has stopped driving the bus and external memory can now supply data for read misses. As for writes, the WrNear signal goes true if this write falls within the same page as the last write, improving write bandwidth into main memory. Just like the standard R3000A, the processor supports burst reads and includes a 4-deep read buffer which allows instruction streaming. The refill can be as fast as a single clock or multiple clocks, gated by RdClkEn.

The processor has the same precise exception model as the R3000A, allowing six independent interrupts directly into the processor. Upon sensing an interrupt, the general exception vector goes to the address bus and the exception routine executes. Under software control, the processor can implement any priority-based interrupt scheme imaginable—and dynamically change it, which is especially useful when interrupt loading varies in a system. Additionally, four independent inputs, called branch conditions, can be used to sense external conditions without any additional hardware or logic.

Integration of all the basic CPU components eliminated the need for glue logic, but did not integrate peripherals—why not? The answer lies in the types of peripherals needed by the majority of the designs and the expertise of those customers. A study of major customers in the embedded market was conducted to discover what devices, and how many, were most often needed. The conclusion of that study showed that there were few functions that the majority of designs needed and that most customers had significant design expertise in ASIC technology to implement those differentiating functions. Another result was that, though most needed some type of timing function, the requirements were all different. If one function is integrated and the user needed two, he gained nothing because an additional chip is still needed. Since these basic functions cost less than $2.00, IDT decided to concentrate on improving performance and lowering the CPU cost. This has proven to be a significant factor in market acceptability of the R305x products.

MULTIMASTER MAIN MEMORY SYSTEMS

Systems where main memory can accept data from various sources (i.e., contains arbitration logic to decide who controls the data into and out of main memory) are classified as multimaster systems. This is very common in standard systems and is true whenever the system incorporates a DMA function, which is a peripheral that can master the bus and the more elaborate systems that accept an additional CPU card into a backplane. Cache coherency, a problem in these types of systems, occurs when processors have private memory that is a small copy of main memory and is not updated at the same time as main memory. Cache is a typical form of this configuration, as main memory typically does not have the ability to update cache.

When implementing multimaster systems with R3000A cache structures, designers have several choices to keep the data cache coherent (identical with memory). The R3000A has two control signals frequently called MP Stall and MP Invalidate, which are used to stop the processor and invalidate the cache entry. The major issue is when to do this. Trade-offs are organized by cost and performance. The first method

involves using an external tag memory that stores the addresses of the items in the data cache. It is written to with the address of the data item on every data cache miss and cleared on data cache flush to maintain its correctness. When a memory Write is requested from another device, the tag SRAM is checked to see if an address appears that matches what is contained in the data cache. On a hit, a simple state machine stalls the transfer, stalls the processor, invalidates the cache item and continues. This is the most expensive and highest-performance process. One drawback is that the CPU stalls whenever a cache invalidate cycle occurs. A second, and simpler, method does not include the tag SRAM, but stalls and invalidates any shared address. This, of course, affects performance because the CPU is stalled more often.

One key in reducing the amount of CPU stalls due to data cache hits is to reduce the amount of main memory that is shared. When this is done, the state machine can decode the address to first see if the address range is appropriate for shared memory. This greatly simplifies the overall architecture but has some impact on the software. It is important to minimize shareable memory in all multimaster systems.

Systems that do not include a second processor but have a multimaster structure allow simpler methods to keep data coherent. The first method is simple because the processor probably programmed the address transfer and can flush those cacheable data addresses; the trade-off being the overhead of the code it takes to flush the data cache. A second method would include treating all DMA memory references as uncacheable references; the trade-off again being lowered performance, but a simpler software model. Both methods are effective, cost sensitive and achieve the desired result.

The R305x products allow any of these methods to be used, except the most sophisticated one where the external state machine invalidates the data cache. This arrangement is akin to the highest performance engines and the R305x CPUs are targeted at lower-performance and lower-cost systems. The next-generation product, the R3081™, has the necessary handshake signals to invalidate data cache locations inside a highly-integrated CPU.

THE R3081 RISController

The R3081 RISController is the next logical step in integration, addressing performance applications beyond the reach of the R305x products. The R3081, as shown in Figure 3, has many improved features, including twice the cache size (up to 20kB), on-chip R3010A identical functionality, an optional 1x clock input, up to 50MHz operation and data cache coherency hooks for invalidation. The cache features dynamic reconfiguration, allowing the opportunity to change the cachefrom 16kB of instruction cache and 4kB of data cache to 8kB of each. This feature is extremely important in certain applications. For the embedded world, there are subroutines that will improve in performance when more data cache is available: the 8k/8k configuration. In standard applications, the SPEC benchmarks have shown a performance improvement preference to 16k instruction caches. The configurable feature allows the R3081 to perform well in a variety of situations.

During the definition of the R3081, IDT wanted to provide an upgrade path from the R305x by providing socket upgradeability. That ambition is met by the this device because a user can simply unplug a top-end R3052-20MHz and plug in the R3081-40MHz for immediate performance improvement. This is facilitated by two major factors: twice the cache and a 1x clock. The cache size greatly improves the hit rate and since the R3081 can be programmed at reset to run at the 1x rate while the bus to main memory is at half speed, the system will not see the change and the result is double the pipeline rate.

For more performance, the application can be recompiled to take advantage of the hardware floating-point accelerator. In a few applications for the R305x, some floating-point calculations were needed. To use the integer-only nature of the R305x, floating-point emulation software is linked into the object code to emulate the hardware acceleration available in the R3010A. The R3081 can still execute those instructions, but more performance can be achieved by direct execution of the FPA instruction set. The application can be compiled without the FP emulation library and the final binary will contain the FP opcodes. This will improve the floating-point code execution by approximately 40%.

The R3081 has the added capability of modifying the data cache refill size. On the R3000A and R305x products, data cache refill size is set at reset and, as a data cache reference is missed, the entire block is fetched, minimizing the miss rate. For non-cacheable misses, a single data word is fetched and is an example of reading an I/O register. The R3081 processor can dynamically change refill from 4 words to 1 word on cacheable references and, as always, non-cacheable references are one word. This feature is selectable via a CP0 configuration register. One type of application that can benefit from this feature is data manipulation, e.g., updating a graphic display. When a bit line is drawn, it is not necessary to get the non-adjacent four words, as these words may be on different color planes. This feature allows individual subroutines to be optimized, without any changes to the compiler suite.

Data cache invalidation is a significant feature of the R3081 RISController. This feature, enabled at reset, involves several signals: ACK, WR, Address, CohReq (coherent request) and InvReq (invalidate request). If hardware-based cache coherency is enabled at reset, the processor will stall on bus requests when CohReq is true to allow the maximum invalidate bandwidth. When that occurs, the processor will latch in the address with ALE and invalidate the appropriate cache line if InvReq is true. It is also possible to perform a burst invalidate. As discussed, the R3081 captures address on the ALE signal supplied by the external master device. At the end of the cycle, whether an invalidate occurs or not, the internal address counter increments with ACK. This allows the external logic to continue the transfer, without another address phase and invalidate the next address. Of course, the external logic can invalidate all addresses as they come by or use an external tag SRAM as described earlier to pick appropriate addresses.

All the hooks are supplied to support the addition of a secondary cache, as well. Common in today's systems,

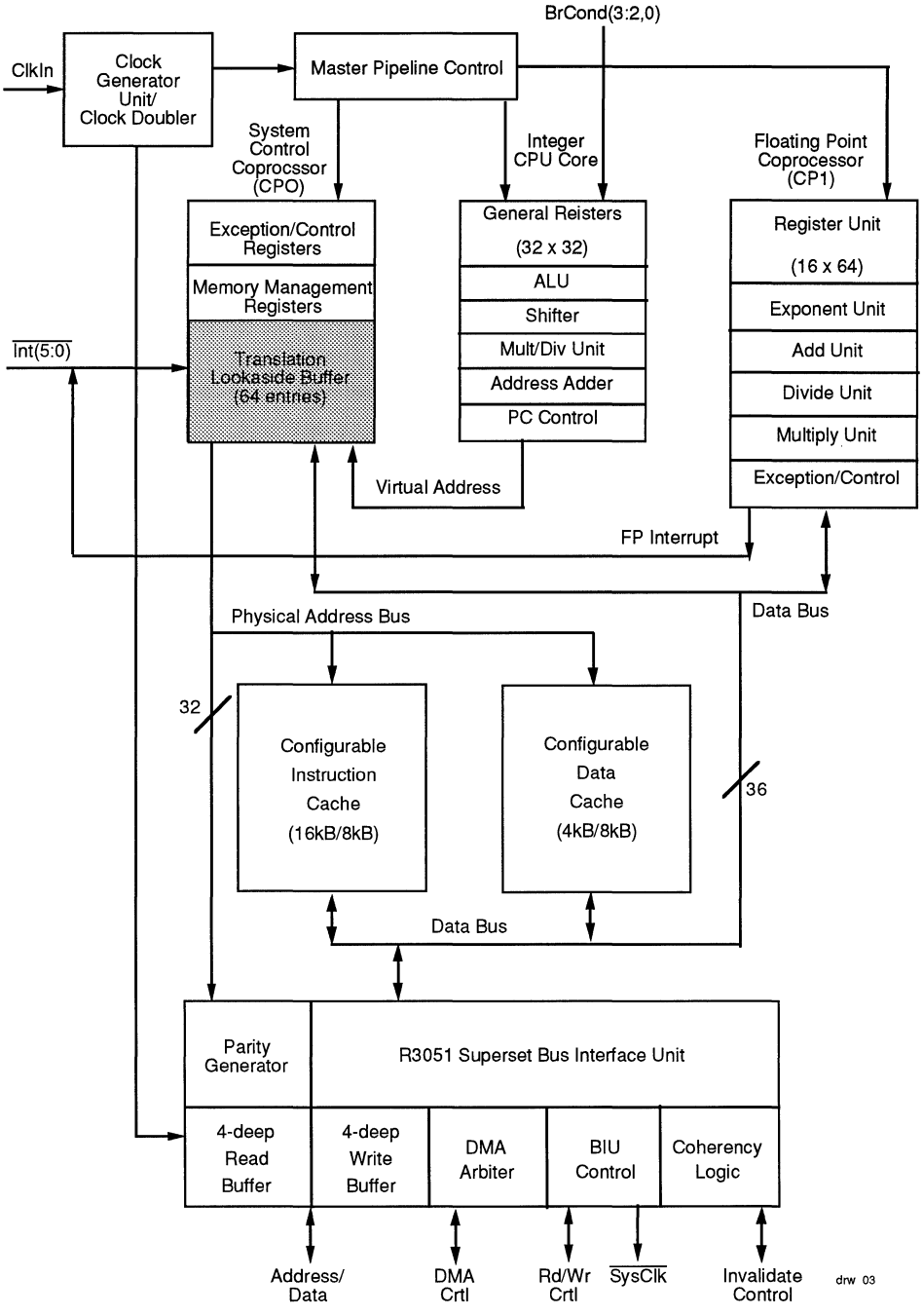


Figure 3. R3081 RISController

secondary caches improve CPU performance by providing a faster intermediate memory between the primary cache and main memory. Facts are facts. When dealing with cache coherency, the secondary cache would first be checked to see if a hit occurs within itself. If that happened, the location could be invalidated or updated and another bit could define if that word was in the primary cache. If the primary cache is incoherent, the R3081 is stalled for an invalidate cycle. This additional layer of insulation improves performance not only by reducing CPU latency to memory, but also results in fewer stalls in coherent systems.

CONCLUSION

The microprocessor selection process has changed dramatically over the past two years. Motorola, who dominated the embedded market, is significantly challenged where it once was a leader. This is because Intel has done well with the 960 line of embedded products and because they are Intel. In fact, this author believes that a major component in the microprocessor selection process is the vendor, not the product ("if it has bat wings or an "i" on it, I will design it in"). Intel, the most predominant processor supplier because of the IBM PC era, still out in front by the definition of units shipped, is being threatened by the ACE initiative and the Apple/IBM "Power PC" product in the future. AMD has a significant lead in the embedded RISC marketplace due to timeframe—the 29000 was on the market long before other RISC solutions were available.

However, the dynamics of the microprocessor industry are evolving. No longer are designers buying the fact that the 'manufacturer' is the key in the decision-making process. The deciding factor is now performance, cost, integration and how the architecture meets the requirements of the design. Growth paths are an increasing concern. With six suppliers now in the market, the designer is assured that many more products will be available soon using the MIPS architecture. IDT has already demonstrated three industry specific RISC processors that the majority of designers are admiring and designing-in. And all incorporate ISA-identical, feature-common compilers, development equipment, operating systems and design mindset. This benefit has provided designers and engineering managers a guiding light to what the next generation of microprocessor goals are: in the 1980s, we centralized on one processor ISA, the 68000, and for 10 years the evolution of derivatives served well. In the 1990s, MIPS RISC solutions have all the essential attributes to establish the next common platform for designers to use. IDT, dedicated to that product arena, will continue to provide the necessary products to keep designers' goals met, by providing products to keep them successful in their marketplace.

Integrated
Device Technology, Inc.

2975 Stender Way
P.O. Box 58015
Santa Clara, CA 95052-8015
(408) 727-6116
FAX: 408-492-8674

MISC-RISC-00062