Simple Embedded Control NS32CG16 System Graphics Application Note 2

National Semiconductor
Application Note 564
June 1989

1.0 INTRODUCTION

This design is for a stand alone NS32CG16 execution vehicle. The design includes the NS32081 Floating Point Unit, DP8511 BitBlt Processing Unit, NS32202 Interrupt Control Unit and SCN2681 Dual channel serial interface. MONCG, a modified version of MON16, is supported in this design for interface to DBG32 debug utilities. The NS32202 timers may be used to time program execution.

1.1 Specification

1. 256 Kbytes of Static RAM (32K x 8).
2. 4 sockets of EPROM, capable of 27C256 or 27C512 EPROM.
3. Serial I/O—2 ports RS-232, configured for MONCG/DBG16 debug.
4. Circuits required to interface the DP8511 with the NS32CG16. The interface utilizes an 8-bit counter to control the DP8511 allowing a maximum of a 256 word wide pattern to be BitBlted.
5. Memory and I/O map controlled with PALs to allow changes.
6. Simple LED indicators to show board status.
7. NS32081D-15 installed for floating point tests.
8. NS32202-10 interface to verify interrupts.
9. 3 push buttons, INT into NS32202, NMI and RESET to NS32CG16.
10. NS32CG16V-15 installed.
11. MONCG (a new mon16) installed in EPROMs.
12. $\overline{\text{HOLD}}$/input available for testing.
13. ''SPLICE'' Control Signal interface.
14. PROM Shadow feature.
15. Operates at 15 MHz.

1.2 BPU Control Circuit Programming Information

On the board, the addresses of the control registers are PAL programmable, but default to the following:

Address	Description
0xFF0000	Duart (SCN2861)
0xFF0020	BPU Control Register on DP8511
0xFF0022	BPU Function select register on DP8511
0xFF0040	BPU Mask Register
0xFF0060	BPU Counter Register

The BPU Control register and Function select register are as described in the *DP8511 BitBlt Processing Unit* Data sheet, and are 13 bits and 4 bits wide respectively.

When programming the control logic, the following sequence should be used.

1. Write BPU Counter
2. Write BPU Mask Register
3. Write BPU Control Register
4. Write BPU Function Select Register

Note that the BPU Enable in the Mask register must be turned on prior to writing the BPU Control register.

When the BILBLT operation is complete, it is recommended that the BPU be turned off by writing a zero to the BPU Mask register. This is not required, however.

2.0 NS32CG16/DP8511 INTERFACE

The following sections describe the logic required to interface the NS32CG16 to the DP8511 BitBlt Processing Unit (BPU). The NS32CG16 facilitates the interface requirements by supporting a special signal, $\overline{\text{BPU}}$, and an BitBlt instruction, EXTBLT.

The schematic and PAL equations in this document describe an implementation that supports BitBlt operations in 4 directions, left to right or right to left while moving top to bottom or bottom to top. Most typical printer applications require only left to right BitBlting while moving top to bottom.

2.1 Features

Following are the features of described interface:

- 32-bit CPU.
- 16 megabyte address range for BitBlt operations.
- BitBlt operations in all 4 directions.
- 16 logical BitBlt functions.

```
  0                 (0 to d)
 -s  AND  -d
 -s  AND   d
 -s
  s  AND  -d
 -d
  s  XOR   d
 -s  OR   -d
  s  AND   d
  s  XNOR  d
  d                 (d to d)
 -s  OR    d
  s                 (s to d)
  s  OR   -d
  s  OR    d
  1                 (1 to d)
```


RRD-B30M105/Printed in U. S. A.

- Operates with conventional DRAMs.
- High-speed barrel shift of data.
- Hardware masking of data.
- Bus bandwidth limits on external BitBlting.

2.2 Image Memory Configuration

To obtain optimum performance in graphics applications the Image Memory must be organized to support the *Series 32000* byte and bit manipulating instructions. *Figure 2.1*, below, illustrates the memory organization at the byte (8 bits), word (16 bits) and double-word (32 bits) level.

In the rest of this document hexadecimal numbers will be represented with a leading 0x. e.g., hexadecimal 1a5a will appear in this document as 0x1a5a.

Figure 2.1 represents one scan line of a standard 8½ inch by 11 inch page, on a 300 Dot Per Inch (DPI) laser printer in the portrait orientation (8½ inches wide, 11 inches high). There are 3300 such scan lines on each page. The start of the second scan line on the page would be at byte offset 320 decimal, 140 hex. Since the first scan line is at the top of the page, successive scan lines proceed down the page.

All *Series 32000* microprocessors have 32-bit internal data paths, with a ''natural'' size of 32 bits, or 4 bytes. Memory accesses are always Least Significant Byte (LSB) to Most Significant Byte (MSB). Referring to *Figure 2.1*, writing a byte of 0xA5 to address zero would result in address zero containing 0xA5. Writing a word of 0xA55A to address zero would result in address zero containing 0x5A, and address 1 containing 0xA5. Writing a doubleword of 0xFFA55A00 to address zero would result in address zero containing 0x00, address 1 containing 0x5A, address 2 containing 0xA5, and 3 containing 0xFF.

The *Series 32000* microprocessors do **not** have an alignment restriction in that data of byte, word or doubleword size need not reside on an even memory address. The Bus Interface Unit, internal to all *Series 32000* microprocessors, request multiple bus transfers as required, aligning the data automatically.

The bit offset is equally consistent. Bit ordering is always least significant to most significant bit. In *Figure 2.1*, bit zero of byte zero would be the first pixel imaged on the page. Bit one would be the next pixel, bit two the next, and so on. Bit 2549 would be the last pixel imaged on the page in the horizontal direction, since 8½ inches * 300 DPI yields a width of 2,550 dots, or pixels. Bit 2549 is contained within byte 318, at bit position 5. Both Bit Addressing (e.g., *SBITD 2549,page*) and Byte Addressing with a byte address and a bit offset (e.g., *SBITD 5,page + 318*) are available in *Series 32000*. *Figure 2.2* is an expansion of the first three bytes of the scan line, showing the bit addressing, as it would appear on the page printer or graphics screen.

To clarify these conventions further, the following example illustrates how a line 1 dot high and 10 dots wide is drawn. This line appears on scan line one, starting at the ninth pixel, or bit position 8. This will result in 0xFF in address one, and 0x03 in address two. This is referred to as the *horizontal* direction.

The width of the memory for an image is referred to as the image *warp*. The *warp* of the page printer image in the previous example is 320 decimal (140 hex) bytes, or 2560 bits. Note that the image width is actually 2550 on this sample page printer at 300 DPI, since 8½ inches * 300 DPI yields a width of 2,550 dots. The width is rounded up to 2560 bits (320 bytes) to make memory addressing simpler in a typical hardware design.

When the *warp* is known, perpendicular (or vertical, in this case) lines can be drawn. A vertical line 10 dots high and 1 dot wide starting at the first line, ninth pixel, with a *warp* of 320 (140 hex) and a *base address* of 0 would result in addresses 1 (1 hex), 321 (141 hex), 641 (281 hex) . . . 2881 (B41 hex) each containing 01 hex.

To summarize, for portrait applications, the ''top left'' pixel is bit 0. The ''top right'' pixel is bit 2,549. The ''bottom left'' pixel is bit 8,445,440. The ''bottom right'' pixel is bit 8,447,989. To calculate *x,y* bit positions on the page, the formula:

$$\text{Bit offset} = (y * 2560) + x$$

may be used, where *y* is the scan line number ranging from 0 to 3299, for the sample 8½ by 11 inch page, and *x* is the pixel displacement across the page from the left hand edge.

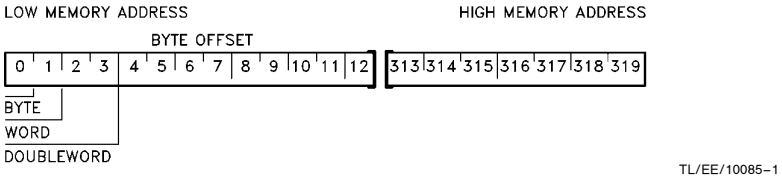


FIGURE 2.1. Memory Organization

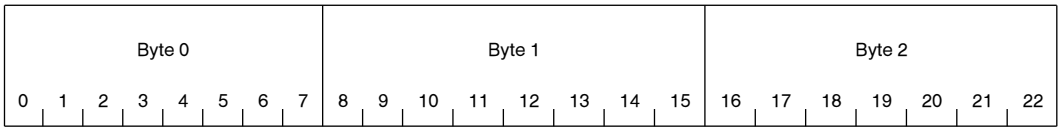


FIGURE 2.2. Bit Order within a Byte

2.3 BitBlt Operation

It is assumed that the reader is familiar with the fundamentals of BitBlt operations. More information on the BitBlt algorithm may be found in the DP8511 BitBlt Processing Unit (BPU) data sheet.

In this circuit the BPU is connected to an NS32CG16 CPU. During the external BitBlt instruction execution, the CPU generates an optional source word pre-read, source word read, destination word read and destination word write, while asserting the $\overline{\text{BPU}}$ signal. The BPU interface circuit monitors the $\overline{\text{BPU}}$ signal and controls the BPU and asserts left and right masks when appropriate. The interface has an 8-bit counter, enabling up to 256 word wide blocks to be transferred.

Prior to executing the EXTBLT instruction the software must first load the *mask register* with left and right beginning and ending mask bits, pre-read enable, BPU enable, and then load the character width *count register* with the width (in words) of the BitBlt operation. Then the software loads the BPU *control register* with the *Barrel Swap bit*, the *shift amount*, the *left mask*, and the *right mask* values and the BPU *Function select register*. The software must then load the CPU registers with all the information required by the EXTBLT instruction, for more details, refer to the NS32CG16 Programmer's Reference Manual. The EXTBLT instruction will then cause the BPU to perform the required BitBlt instruction for the width of the character by the height of the character.

The *mask register*, the character width *counter register*, the BPU *control register* and the BPU *Function select register* are mapped as output devices in the CPU's address space. The address of these registers are implementation dependent since the EXTBLT instruction does not reference them directly. The BPU is selected by the $\overline{\text{BPU}}$ signal which is asserted during the execution of the EXTBLT instruction.

The BPU control register is shown in *Figure 2.3a*. This register is on the BPU device. Note: the left mask is asserted at the beginning of a line on the left side of the page, the right mask is asserted at the end of a line on the right side of the page. This satisfies the bit-ordering of a *Series 32000* bit zero of byte zero is the first imaged pixel.

The programming of the BPU control register changes depending on the direction of the BitBlt, either left to right or right to left. The mask enable bits for the left and right masks and the BIS bit remain the same for top to bottom or bottom to top.

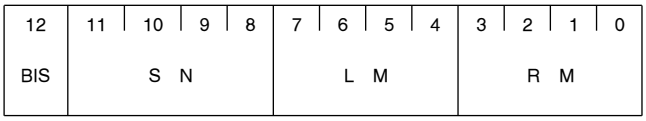


FIGURE 2.3a. BPU Control Register

Description of *control register* bits.

RM	MSB			LSB
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1
0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 1
0 0 1 1	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1
0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 1	1 1 1 1
0 1 0 1	0 0 0 0	0 0 0 0	0 0 1 1	1 1 1 1
0 1 1 0	0 0 0 0	0 0 0 0	0 1 1 1	1 1 1 1
0 1 1 1	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1
1 0 0 0	0 0 0 0	0 0 0 1	1 1 1 1	1 1 1 1
1 0 0 1	0 0 0 0	0 0 1 1	1 1 1 1	1 1 1 1
1 0 1 0	0 0 0 0	0 1 1 1	1 1 1 1	1 1 1 1
1 0 1 1	0 0 0 0	1 1 1 1	1 1 1 1	1 1 1 1
1 1 0 0	0 0 0 1	1 1 1 1	1 1 1 1	1 1 1 1
1 1 0 1	0 0 1 1	1 1 1 1	1 1 1 1	1 1 1 1
1 1 1 0	0 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1

LM	MSB			LSB
0 0 0 0	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
0 0 0 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
0 0 1 0	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 0
0 0 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 0 0 0
0 1 0 0	1 1 1 1	1 1 1 1	1 1 1 1	0 0 0 0
0 1 0 1	1 1 1 1	1 1 1 1	1 1 1 0	0 0 0 0
0 1 1 0	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 0
0 1 1 1	1 1 1 1	1 1 1 1	1 0 0 0	0 0 0 0
1 0 0 0	1 1 1 1	1 1 1 1	1 0 0 0	0 0 0 0
1 0 0 1	1 1 1 1	1 1 1 0	0 0 0 0	0 0 0 0
1 0 1 0	1 1 1 1	1 1 0 0	0 0 0 0	0 0 0 0
1 0 1 1	1 1 1 1	1 0 0 0	0 0 0 0	0 0 0 0
1 1 0 0	1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0
1 1 0 1	1 1 1 0	0 0 0 0	0 0 0 0	0 0 0 0
1 1 1 0	1 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1 1 1 1	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

SN	
0 0 0 0	Barrel shift quantity. Causes the 32-bit barrel shifter to rotate 0–15 bits MSB to LSB.
|	
1 1 1 1	

BIS	
0	Left to right, shift > zero.
1	Left to right, shift = zero.
1	Right to left, any shift.

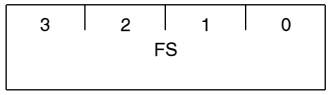


FIGURE 2.3b. BPU Function Select Register

Description of *Function Select register* bits.

FS	
0 0 0 0	0
0 0 0 1	–s AND –d
0 0 1 0	–s AND d
0 0 1 1	–s
0 1 0 0	s AND –d
0 1 0 1	–d
0 1 1 0	s XOR d
0 1 1 1	–s OR –d
1 0 0 0	s AND d
1 0 0 1	s XNOR d
1 0 1 0	d
1 0 1 1	–s OR d
1 1 0 0	s
1 1 0 1	s OR –d
1 1 1 0	s OR d
1 1 1 1	1

The structure of the *mask register* is shown in *Figure 2.4*. It is an 8-bit register, located on the least significant byte of the data bus. Following is a description of valid mask enable sequences that can be programmed into the *mask register*.

A left mask in the following documentation means a mask that preserves the least significant bits, a right mask preserves the most significant bits. The BPU applies the left and right masks on the write to the destination.

```
BRME  BLME  ERME  ELME
 0     1     1     0          Left to right, both masks, width > 1
 1     0     0     1          Right to left, both masks, width > 1
 0     0     0     0          Shift of zero, no masks, width = n * 16 bits
 1     1     0     0          Any direction, both masks, width = 1
 0     1     0     0          Any direction, left mask, width = 1
 1     0     0     0          Any direction, right mask, width = 1
```

ENBPU Active High. When low, resets the BPU and the interface circuitry state machine.

PRERD Active High. When asserted, forces the interface circuitry to perform an additional read at the beginning of each BitBlt source read. This is termed *source pre-read*, and is required at the beginning of each line if the width of the first destination data write is greater than the amount of valid data contained in the first source read.

7	6	5	4	3	2	1	0
BRME	BLME	ERME	ELME		ENBPU	PRERD	

FIGURE 2.4. Mask Register

The structure of the *counter register* is shown in *Figure 2.5*. It is an 8-bit register, located on the least significant byte of the data bus. The value written to this register is the two's complement of the width in words of the destination data to be BitBlted. Each word is 16 bits in width, thus for a font character 32 pixels wide, the *counter register* would be loaded with the value 0xFE. The two's complement of the width can easily be obtained via the NS32CG16's **NEGi gen,gen** instruction.

7	6	5	4	3	2	1	0
Width in Words (Two's Complement)							

FIGURE 2.5. Counter Register

2.4 Interface Circuit Description

The EXTBLT instruction causes the CPU to output source and destination addresses, the $\overline{\text{BPU}}$ signal and read/write bus cycles. The BPU interface circuit monitors these signals and issues control signals to the BPU. The BPU must be interfaced to the data bus on the system side of the transceivers. During BPU data transfer cycles, a signal is generated by the BPU interface circuit to disable the data bus transceivers. The CPU must be disabled from driving the system data bus during an EXTBLT write cycle because the BPU will be driving the BitBlt result onto the bus.

A detailed description of the BPU interface circuit and all the external signals follows.

Description of interface signals follows.

Signal	Description
$\overline{\text{BPU}}$	From NS32CG16, $\overline{\text{BPU}}$ cycle active.
ST1–ST3	From NS32CG16, status signals describing the type of bus cycle.
BD00–BD15	Buffered system data bus.
$\overline{\text{RESET}}$	From NS32CG16, reset signal.
$\overline{\text{TSO}}$	From NS32CG16, signifies data portion of bus cycle.
CTTL	From NS32CG16, TTL clock output.
$\overline{\text{DDIN}}$	From NS32CG16, signifies direction of data transfer.
CSYNC	Similar timing to $\overline{\text{TSO}}$, but is deasserted one T-state early, at the end of T3.
$\overline{\text{BPUCYC}}$	Signal to turn off the data bus transceivers. The CPU data bus must be isolated from the BPU-memory data bus during the execution of the EXBLT instruction.
The following signals are decoded address lines, gated with $\overline{\text{TSO}}$	
$\overline{\text{BPUMSKWR}}$	Write the 8-bit mask register.
$\overline{\text{BPUCNTWR}}$	Write the 8-bit counter register.
$\overline{\text{BPUCTLWR}}$	Write the BPU control register.

Refer to the BPU interface schematic diagram for the rest of this section. Functional description of the interface circuit follows.

2C and 6C decode status signals from the CPU along with the $\overline{\text{BPU}}$ signal to produce the $\overline{\text{BPUCYC}}$ signal. $\overline{\text{BPUCYC}}$ indicates that the current bus cycle is an EXTBLT data transfer cycle. The status signals must be used to qualify the $\overline{\text{BPU}}$ signal because the CPU can assert $\overline{\text{BPU}}$ and then perform instruction pre-reads to fill it's internal pre-fetch queue. The status signals are decoded to uniquely detect a data transfer cycle, as against any other type of bus cycle.

3F is the *mask register*. It is a 6-bit write only latch with reset. On power-on the NS32CG16 $\overline{\text{RESET}}$ output signal will cause all the outputs of the latch to be cleared to zeroes. Writing to the latch simply involves a move of a byte to the $\overline{\text{BPUMSKWR}}$ address. Once programmed the *mask register* will remain unchanged until it is written to again or $\overline{\text{RESET}}$ is asserted.

3D is the *counter register*. It is an 8-bit write only latch and counter. Writing to the latch simply involves a move of a byte to the BPUCNTWR address. Once programmed the *counter register* will remain unchanged until it is written to again. The synchronous binary up counter portion of 3D is loaded when 4D asserts the CTR__LOAD output. CTR__LOAD is asserted when the BPUMSKWR signal is active, which implies that the programmer must program the *counter register* prior to the *mask register*. The counter in 3D enables characters up to 256 words wide to be BitBlted with the BPU. Note again that the value programmed into the *counter register* must be the two's complement of the width in words of the character.

BPU bus cycles consist of an optional source pre-read (only on the first word of a new line), source read, destination read and destination write. 4D detects destination writes and only increments or re-loads 3D on completion of the write. The PRERD bit in the *mask register* must be set by the programmer if pre-reads are required. This bit causes the state machine in 4D to perform an extra source read (pre-read), at the beginning of each BitBlt line.

CTR__LOAD is also asserted on the last BPU write cycle at the end of each BitBlt line, causing the value in the *counter register*, 3D, to be re-loaded into 8-bit counter in preparation for the next BitBlt line. The TSO signal from the NS32CG16 is the clock, rising (positive-going) edge sensitive, for 4D and 3D. During a BitBlt of more than one word in width, 4D also asserts the CTR__ENBP signal to enable 3D to count on the next rising edge of TSO. 3D counts up until it reaches a count of 255, at which point the RCO of 3D is asserted. 4D treats the assertion of the RCO signal as an indication that the next BPU bus cycle (source read, destination read and destination write) is at the end of BitBlt line. 4D then asserts the MASK__SEL and MASK__ENB signals, and causes 3D to re-load from the *counter register* as explained above. The MASK__SEL and MASK__ENB signals control the multiplexer in 4F selecting the appropriate masks control signals that connect to the BPU, refer to the description of the *mask register* in Section 2.3.

Refer to the attached timing diagrams for detailed BPU control signal timing. *Figure 2.6* depicts a complete BPU cycle with pre-read. Only the data path control signals are shown. *Figure 2.7* depicts a write bus cycle to the *mask register*. *Figure 2.8* depicts the assertion of the appropriate masks during the first, a middle word and then the last word of a BPU BitBlt line.

4D generates a signal, DESTCYC, that indicates the type of bus access that the EXTBLT instruction is performing. When high, it indicates a source pre-read or read, when low, it indicates either a destination read or destination write. DESTCYC connects to 4F which controls most functions of the BPU.

4F controls the BPU data paths, FIFO operation, and control registers. The programmer must load the BPU *control register* prior to executing the EXTBLT instruction, refer to Section 2.3 for a detailed description of the bits. The programmer accesses the 13-bit, write only *control register* by writing to the BPUCTLWR address. 2C generates the CRE signal which writes the 13-bit data into the BPU *control register*.

2C also generates the FSE signal which writes 4-bit data into the BPU *Function select register*. All references to registers within the BPU use the same terminology as in the DP8511 data sheet. 2D is clocked from inverted CTTL, its function is to delay the assertion of the FRD, FWR, RME, LME, BSE and DLE signals to satisfy the setup and hold time requirements of the DP8511. The B__DLE signal causes the BPU to latch the data on the data bus into the DIL-MASTER register. Both the source and destination read data is temporarily stored in this register during the EXTBLT instruction execution. The B__BSE signal causes the BPU to latch the data from the DIL-MASTER register into the DIL-SOURCE register during a source pre-read or read bus cycle.

When B__BSE is not asserted, the data contained in the DIL-MASTER register will be latched into the DIL-DEST register. The DIL-DEST register contains the read destination data.

The barrel rotator performs the rotation, 0 to 15 bits. The result is transferred to a multiplexer at the input of the 16 word FIFO. 4F generates the FWR signal to write this value into FIFO location 0. 4F delays the assertion of FRD two clocks, as required by the BPU. The FRD signal transfers the data stored in FIFO location 0 to a holding latch, then through another multiplexer (always in BitBlt mode, B/L low) to the source input of the BitBlt Logic Unit, BLU.

The B__BSE is deasserted during the destination source read, causing the DIL-MASTER data to be loaded into the DIL-DEST register, through to the destination input of the BLU.

The BLU performs the required logical operation, based in the 4-bit Function Select code programmed into the BPU *control register*. The left and right masks are then applied to the result and finally the destination read data is ored with this result. This method of masking is called *destination masking* and is different to the NS32CG16's software BitBlt instructions which perform *source masking*. The final result is the same regardless of the method used.

The result from the BLU is now available for writing back to memory. The NS32CG16 performs a write bus cycle, 6C generates DOE which enables the BPU output buffers, the result appears on DQ00−DQ15 and is written to memory.

The entire BitBlt operation takes 12 clock cycles to perform a source read, destination read and destination write. The whole BitBlt cycle can then repeat for the next word of the BitBlt line. Note that interrupts if enabled and pending will be serviced at the end of each BPU write cycle. The preread (optional), read source, read destination and write destination cycle is indivisible at the interrupt level. The NS32CG16 will deassert the BPU signal prior to fetching the vector from the interrupt source. The BPU signal remains deasserted during the entire interrupt service routine and only on return from interrupt and resumption of the EXTBLT instruction will the NS32CG16 again assert the BPU signal.

The DP8511 BPU has many functions that are not used by the NS32CG16 during the EXTBLT instruction execution. *Figure 2.9* depicts the functional blocks inside the BPU that are used during an EXTBLT instruction. Refer to the DP8511 Data Sheet for the complete BPU model.

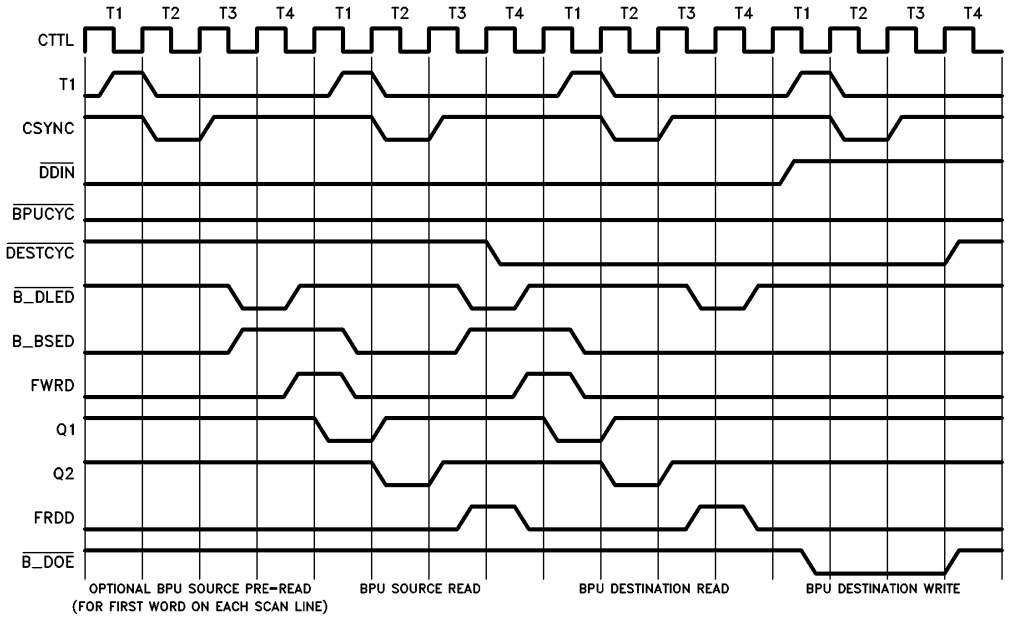
Following are two example programs that perform tests of the EXTBLT instruction and interface. The first program performs a left to right, top to bottom test, the second performs a right to left, bottom to top test. The programs check the result of the EXTBLT instruction by comparing the output with that from the BBFOR instruction. If the results are the same, the shift amount is incremented and the test is performed again. The programs test the EXTBLT for shifts of zero through to 15.

```
#Program extblt.s
#Program to test the extblt instruction, left to right, top to bottom
        .globl  __test,dest
#BitBlt  test program
__test:  movqd    0,shift                  # start with shift of zero
loopl:   movd     $108,height              # height in lines
         movqd    1,width                  # start with width of 1 word
loop:    addr     dest,r0                  #point to destination block
         movqd    4,r1                     #increment value
         addr     1024,r2                  #number of patterns to write
         movqd    0,r3                     #pattern to write
         movmpd                            #fill area
         addr     0xff0000,r0              #point to the control base
         movd     width,r2                 # get current width
         movd     shift,r1                 # get current shift value
         movb     $0x0e,r3                 # assume shift is zero.
         movqd    4,r7                     # set destination warp
         movqd    0,r6                     # set source warp
         cmpqd    0,r1                     # is shift zero?
         beq      noinc                    # yes, all is ok, else
         movb     $0x6e,r3                 # set up left and right masks
         addqd    1,r2                     # one extra word of destination
         movqd    2,r7                     # set destination warp
         movqd    -2,r6                    # set source warp
noinc:   negb     r2.0x60(r0)              # set up counter
         movb     r3,0x40(r0)              #set up mask register
         movw     bputab[r1:w],0x20(r0)    #set up BPU register
         movw     $0x7,0x22(r0)            #set up BPU register, OR function
         addr     chara-2,r0               #point to source character
         addr     dest,r1                  #point to destination
         movd     height,r3                # get current height
         movqd    2,r4                     #increment value
         addd     r2,r2                    # width = r4 * r2
         movd     r2,r5

         cmpqb    1,$2                     #do pre read
         extblt

         movqb    $0,0x40 + 0xff0000       #clear mask register, disable
                                           #bpu, reset logic
         addr     chara,r0                 #point to source char
         addr     destl,r1                 #point to destination
         movd     shift,r2                 #shift value wanted
         movd     height,r3                #height in lines
         movd     $0xffff,r4               #first mask
         movd     $0xffff,r5               #second mask
         movqd    2,r6                     #source warp
         movqd    4,r7                     #dest warp
         movd     width,tos                #width in words
         cmpb     r2,$0
         bbfor
         cmpqd    0,tos                    #unstack
         addr     512 ,r0                  #number of doubles to compare
         addr     destl,r1                 #destl
```

```
        addr    dest,r2                 #dest
        cmpsd                           #compare those strings
        bne     bad

        addqd   1,width                 # next width
        movd    $54*4,r0                        # get max lines
        divd    width,r0                # divide to get current lines
        movd    r0,height               # and store it
        cmpqd   1,r0                    # is it OK?
        blt     loop
        addqd   1,shift                 # next shift
        cmpd    $16,shift               # done yet?
        bne     loopl                   # no, back for more
#
        ret     $0

bad:    bpt
        .data
        .data
bputab: .word   0x100f                  # shift of zero, set masks & BIS
        .word   0x0f10                  # shift of one
        .word   0x0e21                  # shift of two
        .word   0x0d32                  # shift of three
        .word   0x0c43                  # 4
        .word   0x0b54                  # 5
        .word   0x0a65                  # 6
        .word   0x0976                  # 7
        .word   0x0887                  # 8
        .word   0x0798                  # 9
        .word   0x06a9                  # 10
        .word   0x05ba                  # 11
        .word   0x04cb                  # 12
        .word   0x03dc                  # 13
        .word   0x02ed                  # 14
        .word   0x01fe                  # 15
width:  .double 0
shift:  .double 0
count:  .double 0
height: .double 0
        .comm   dest,2048
        .comm   destl,2048
```

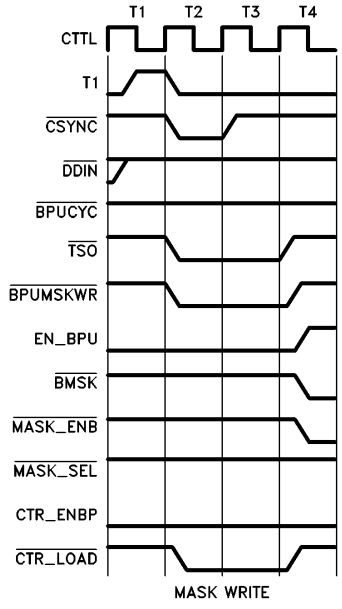
```
#Program extblt.s
#Program to test the extblt instruction right to left, bottom to top
        .globl  __test,dest
#BitBlt test program
__test:  movqd    0,shift                  # start with shift of zero
loopl:   movd     $108,height              # height in lines
         movqd    1,width                  # start with width of 1 word
loop:    addr     dest,r0                  #point to destination block
         movqd    4,r1                     #increment value
         addr     1024,r2                  #number of patterns to write
         movqd    0,r3                     #pattern to write
         movmpd                            #fill area
         addr     0xff0000,r0              #point to the control base
         movd     width,r2                 # get current width
         movd     shift,r1                 # get current shift value
         movb     $0x0e,r3                 # assume shift is zero
         movqd    -4,r7                    # set destination warp
         movqd    0,r6                     # set source warp
         cmpqd    0,r1                     # is shift zero?
         beq      noinc                    # yes, all is ok, else
         movb     $0x6e,r3                 # set up left and right masks
         addqd    1,r2                     # one extra word of destination
         movqd    -2,r7                    # set destination warp
         movqd    2,r6                     # set source warp
noinc:   negb     r2,0x60(r0)              # set up counter
         movb     r3,0x40(r0)              #set up mask register
         movw     bputab[r1:w],0x20(r0)    #set up BPU register
         movw     $0x7,0x22(r0)            #set up BPU register, OR function
         addr     chara+220,r0             #point to source character
         addr     dest+1024,r1             #point to destination
         addd     r6,r1                    #pre-increment destination for sh>0
         movd     height,r3                # get current height
         movqd    -2,r4                    #increment value
         muld     r4,r2                    # width = r4 * r2
         movd     r2,r5

         cmpqb    1,$2                     #do pre read
         extblt

         movqb    $0,0x40 + 0xff0000       #clear mask register, disable
                                           #bpu, reset logic
         addr     chara+218,r0             #point to source char
         addr     dest1+1024,r1            #point to destination
         movd     shift,r2                 #shift value wanted
         movd     height,r3                #height in lines
         movd     $0xffff,r4               #first mask
         movd     $0xffff,r5               #second mask
         movqd    -2,r6                    #source warp
         movqd    -4,r7                    #dest warp
         movd     width,tos                #width in words
         cmpb     r2,$0
         bbor     -da
         cmpqd    0,tos                    #unstack
         addr     512 ,r0                  #number of doubles to compare
```

```
          addr      destl,rl                #destl

          addr      dest,r2                 #dest
          cmpsd                             #compare those strings
          bne       bad

          addqd     1,width                 # next width
          movd      $54*4,r0                # get max lines
          divd      width,r0                # divide to get current lines
          movd      r0,height               # and store it
          cmpqd     1,r0                    # is it OK?
          blt       loop
          addqd     1,shift                 # next shift
          cmpd      $16,shift               # done yet?
          bne       loopl                   # no, back for more
#
          ret       $0

bad:      bpt
          .data
          .data
bputab:   .word     0x100f                  # shift of zero, set masks & BIS
          .word     0x1f10                  # shift of one
          .word     0x1e21                  # shift of two
          .word     0x1d32                  # shift of three
          .word     0x1c43                  # 4
          .word     0x1b54                  # 5
          .word     0x1a65                  # 6
          .word     0x1976                  # 7
          .word     0x1887                  # 8
          .word     0x1798                  # 9
          .word     0x16a9                  # 10
          .word     0x15ba                  # 11
          .word     0x14cb                  # 12
          .word     0x13dc                  # 13
          .word     0x12ed                  # 14
          .word     0x11fe                  # 15
width:    .double   0
shift:    .double   0
count:    .double   0
height:   .double   0
          .comm     dest,2048
          .comm     destl,2048
```



TL/EE/10085–2

FIGURE 2.5. BPU Data Cycle



TL/EE/10085–3

FIGURE 2.6. Mask Register Write

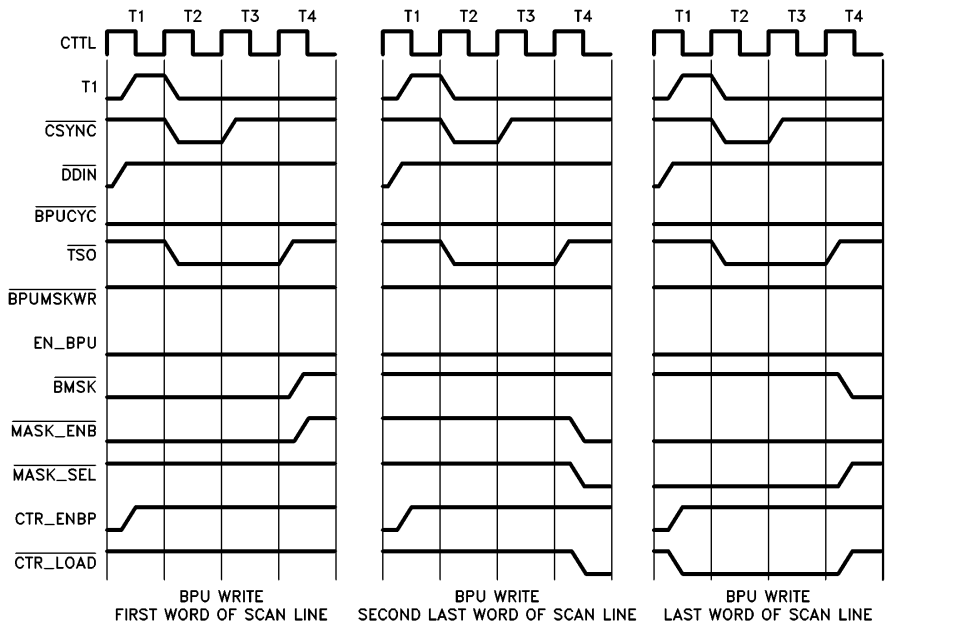
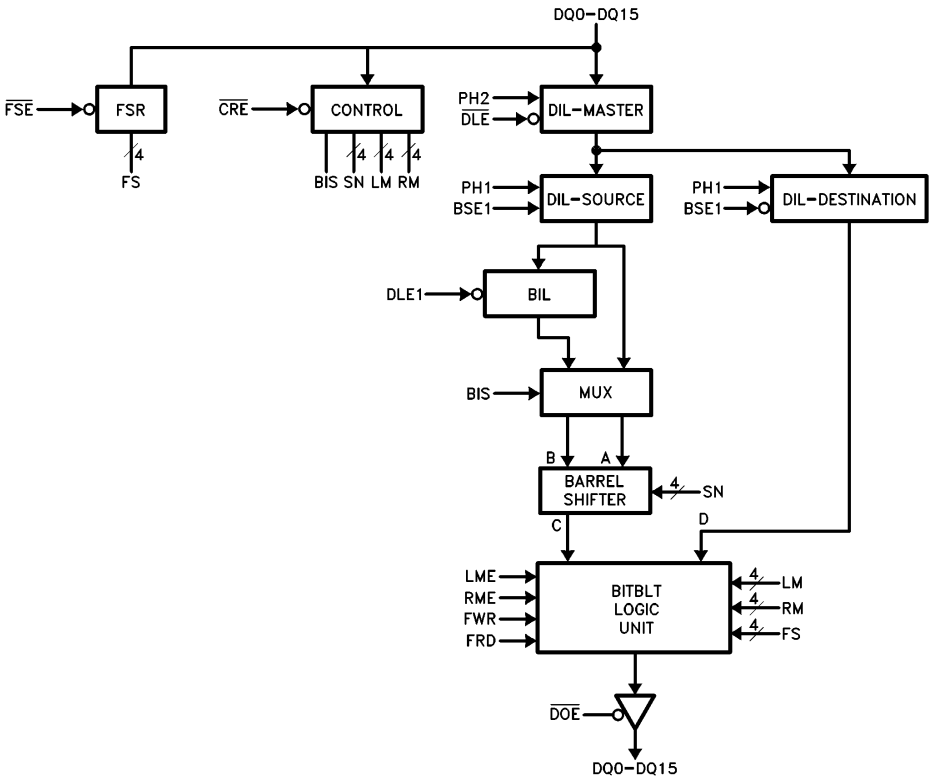


FIGURE 2.7. BPU Mask Register Timing



TL/EE/10085–5

FIGURE 2.9. BPU Model

Figure 2.9 is a block diagram of the functional model of the BPU as used with the NS32CG16. All the data paths in the figure are 16 bits wide. The barrel shifter is actually a rotator that rotates from right to left, i.e., least significant bits, DQ0 to DQ15, are shifted towards the most significant bits. Referring to *Figure 2.9* the data paths to and from the barrel shifter are A, B and C. Path A is the current source read data and is loaded into the 16 LSB's of the barrel shifter, path B is the source read data from the previous word (if in the middle of a BitBlt block) and is loaded into the 16 MSB's. The data is rotated left by the appropriated number of bits specified by the SN inputs and the resulting 16 MSB's are output via data path C to the BitBlt Logic Unit (BLU). Path D contains the 16 bits from the destination read data, and connects to the BLU destination data input. The BLU performs the required function and asserts the appropriate masks and the result is then made available at the output of the BLU for writing back to the destination BitBlt address.

```
Name            MASK.PLD;
Date            03/06/89;
Revision        1B;
Designer        Bill Fox;
Company         NSC;
Assembly        APP Note;
Location        U5;
Device          p16r4;
Partno          0d81;

/*****************************************************************/
/*                                                               */
/* BPUMASK: DP8511 MASK AND MASK COUNTER CONTROL                 */
/*                                                               */
/*****************************************************************/
/*  Allowable Target Device Types: PAL16R4A                      */
/*****************************************************************/

/**  Inputs  **/

Pin     1       = clk           ; /* tso from cg16 */
Pin     2       = nc0           ; /* */
Pin     3       = !mrco         ; /* counter ripple carry out */
Pin     4       = nc1           ; /* */
Pin     5       = !ddin         ; /* data direction in */
Pin     6       = !bpumskwr     ; /* bpu mask write strobe */
Pin     7       = b_prerd       ; /* bpu preread */
Pin     8       = b_enbpu       ; /* enable BPU */
Pin     9       = !bpucyc       ; /* bpu cycle */
Pin     11      = !oe           ; /* output enable, always gnd */

/**  Outputs  **/

Pin     12      = !ctr_enbp     ; /* counter enable p */
Pin     13      = !ctr_load     ; /* counter load */
Pin     14      = !destcyc      ; /* destination cycle indicator */
Pin     15      = !bmsk         ; /* beginning mask */
Pin     16      = !read_cnt     ; /* count of readsq */
Pin     17      = nc2           ; /* */
Pin     18      = !mask_enb     ; /* mask enable */
Pin     19      = !emask_sel    ; /* ending mask select */

/** Declarations and Intermediate Variable Definitions **/

bpuread = bpucyc & ddin;
bpuwrite= bpucyc & !ddin;

field   bpuseq = [destcyc, read_cnt];

$define source0 'b'00
$define source1 'b'01
$define dst     'b'10
```

TL/EE/10085–13

```
/**  Logic Equations  **/

sequence bpuseq {

present source0 if !b_enbpu
                        next source0;

                if  b_enbpu & bpuread & b_prerd
                        next source1;

                if  b_enbpu & bpuread & !b_prerd
                        next dst;

                default
                        next source0;

present source1 if !b_enbpu
                        next source0;

                if  b_enbpu & bpuread
                        next dst;

                default
                        next source1;

present dst     if !b_enbpu
                        next source0;

                if  b_enbpu & bpuwrite &  end_mask
                        next source0;

                if  b_enbpu & bpuwrite & !end_mask
                        next source1;

                default
                        next dst;
}

bmsk.d  = b_enbpu & (
                  !bpumskwr &  bpuwrite & mrco  /* bpu write cycle */
                # !bpumskwr & !bpuwrite & bmsk  /* hold data */
                )
                # bpumskwr;                     /* load initial mask */

ctr_enbp= bpuwrite;

ctr_load = bpuwrite & mrco
         # bpumskwr;

mask_enb = bmsk
         # mcro;

emask_sel= mrco;
```

TL/EE/10085–14

```
/** Logic Equations **/

b_fwr.d = bpusrd & b_dle;

q0.d    = b_fwr;

q1.d    = q0;

b_frd.d = q1;

b_dle.d = bpudrd & csync
        # bpusrd & csync;

b_bse.d = bpusrd;

b_lme   = elme & mask_enb &  mask_sel
        # blme & mask_enb & !mask_sel;

b_rme   = erme & mask_enb &  mask_sel
        # brme & mask_enb & !mask_sel;
```

TL/EE/10085–15

```
Name            BCTL;
Partno          ;
Date            03/06/89;
Revision        1B;
Designer        George Scolaro;
Company         NSC;
Assembly        BPU interface PAL;
Location        ;
Device          p20r6;

/*************************************************************************/
/* Control pal for DP8511 interface                                      */
/*************************************************************************/
/*  Allowable Target Device Types:      PAL20R6A                         */
/*************************************************************************/

/**  Inputs  **/

Pin     1       = clk           ; /* clock */
Pin     2       = !bpucyc       ; /* bpu cycle in progress */
Pin     3       = !ddin         ; /* data direction */
Pin     4       = !csync        ; /* 1 t state prior to T3 */
Pin     5       = !destcyc      ; /* destination bpu cycle  */
Pin     6       = brme          ; /* beginning right mask */
Pin     7       = blme          ; /* beginning left mask */
Pin     8       = erme          ; /* ending right mask */
Pin     9       = elme          ; /* ending left mask */
Pin     10      = !mask_sel     ; /* select left/right mask */
Pin     11      = !mask_enb     ; /* enable masks */
Pin     13      = !oe           ;
Pin     14      = nc0           ;
Pin     23      = nc1           ;

/**  Outputs  **/

Pin     15      = b_lme         ; /* left mask enable */
Pin     16      = !b_dle        ; /* data input latch */
Pin     17      = b_bse         ; /* bpu source enable */
Pin     18      = !q0           ; /* internal delay */
Pin     19      = !q1           ; /* internal delay */
Pin     20      = b_fwr         ; /* fifo write */
Pin     21      = b_frd         ; /* fifo read */
Pin     22      = b_rme         ; /* right mask enable */

/** Declarations and Intermediate Variable Definitions **/

bpusrd  = bpucyc & !destcyc &  ddin;
bpudrd  = bpucyc &  destcyc &  ddin;
bpuwrite= bpucyc &            !ddin;
```

TL/EE/10085–16

```
Name       DECODE.PLD;
Date       07/08/87;
Revision   1A;
Designer   FOX;
Company    NSC;
Assembly   APP Note;
Location   5F;
Device     p16l8;

/*****************************************************************/
/*                                                               */
/* DECODE:      Memory & I/O decode                              */
/*                                                               */
/*****************************************************************/
/*  Allowable Target Device Types: PAL16L8B                      */
/*****************************************************************/

/**  Inputs  **/

Pin     [1..9]  = [a23..16,ba15] ;/* address bus                      */
Pin     10      = gnd            ;/* ground                           */
Pin     11      = shdwn          ;/* shadow enable                    */

/**  Outputs  **/

Pin     12      = !ram0          ;/* ram0 enable                      */
Pin     13      = !ram1          ;/* ram1 enable                      */
Pin     14      = !ram2          ;/* ram2 enable                      */
Pin     15      = !ram3          ;/* ram3 enable                      */
Pin     16      = !ramsel        ;/* common ram select for wait state ctl*/
Pin     17      = !promsel       ;/* prom select                      */
Pin     18      = !iosel         ;/* io device select                 */

/** Declarations and Intermediate Variable Definitions **/

$define |       #

/**  Logic Equations  **/

field adr       = [a23..16,ba15];

romn    = adr:[0100000..013ffff];

ramdcd  = adr:[0200000..0effffff] | (adr:[0..03ffff] & !shdwn);

ram0    = !a17 & !a16 & ramdcd;
ram1    = !a17 &  a16 & ramdcd;
ram2    =  a17 & !a16 & ramdcd;
ram3    =  a17 &  a16 & ramdcd;
ramsel  = ramdcd;

promsel = romn | (shdwn & adr:[0..03ffff]);

iosel   = adr:[0ff0000..0ffffff];
```

TL/EE/10085–17

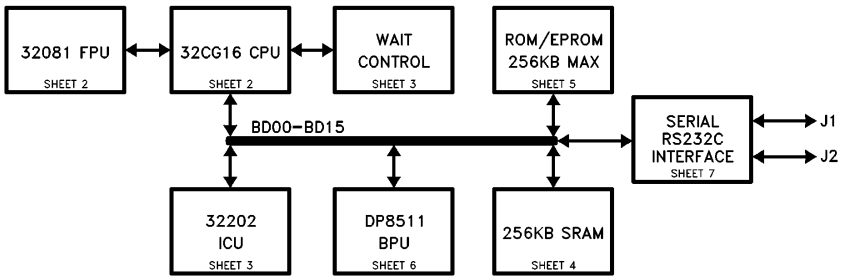
```
Name       WAIT.PLD;
Date       07/08/87;
Revision   1A;
Designer   FOX;
Company    NSC;
Assembly   APP note;
Location   5D;
Device     p16r4;

/*****************************************************************/
/*                                                               */
/* WAIT.PLD:    Wait recreation logic                            */
/*                                                               */
/*****************************************************************/
/*  Allowable Target Device Types: PAL16R4A                      */
/*****************************************************************/

/**  Inputs  **/

Pin     1       = cttl          ;/* CTTL                        */
Pin     2       = t1            ;/* T1 indication from CPU      */
Pin     3       = t2            ;/* T2                          */
Pin     4       = !ddin         ;/* data direction              */
Pin     5       = !dbe          ;/* data bus enable             */
Pin     7       = !cwait        ;/* cwait into CG               */
Pin     8       = !wait2        ;/* wait 2                      */
Pin     9       = !wait1        ;/* wait 1                      */
Pin     11      = gnd           ;/* ground                      */

/**  Outputs  **/

Pin     12      = !iord         ;/* i/o read strobe             */
Pin     [16..14] = ![ctr2..0]   ;/* wait state counter          */
Pin     17      = !dctr2        ;/* peripheral strobe for write */
Pin     18      = csync         ;/* sync strobe to bpu          */
Pin     19      = !iowr         ;/* i/o write strobe            */

/** Declarations and Intermediate Variable Definitions **/

$define |       #

/**  Logic Equations  **/

field cnt = [ctr2..0];

load    = (cwait & ctr2) | t1;

count   = ctr2;

ctr0.d  =  load & !wait1
        | !load & (count $ ctr0);
```

TL/EE/10085–18

```
ctr1.d  =  load & !wait2
        | !load & ((count & ctr0) $ ctr1);

ctr2.d  =  load
        | !load & ((count & ctr1 & ctr0) $ctr2);

dctr2.d =  ctr2;

cycend  =  !cwait & cnt:[7];

!csync  = cycend;

iord    = dctr2 & ddin;
iowr    = (ctr2 & dctr2) & !ddin;
```

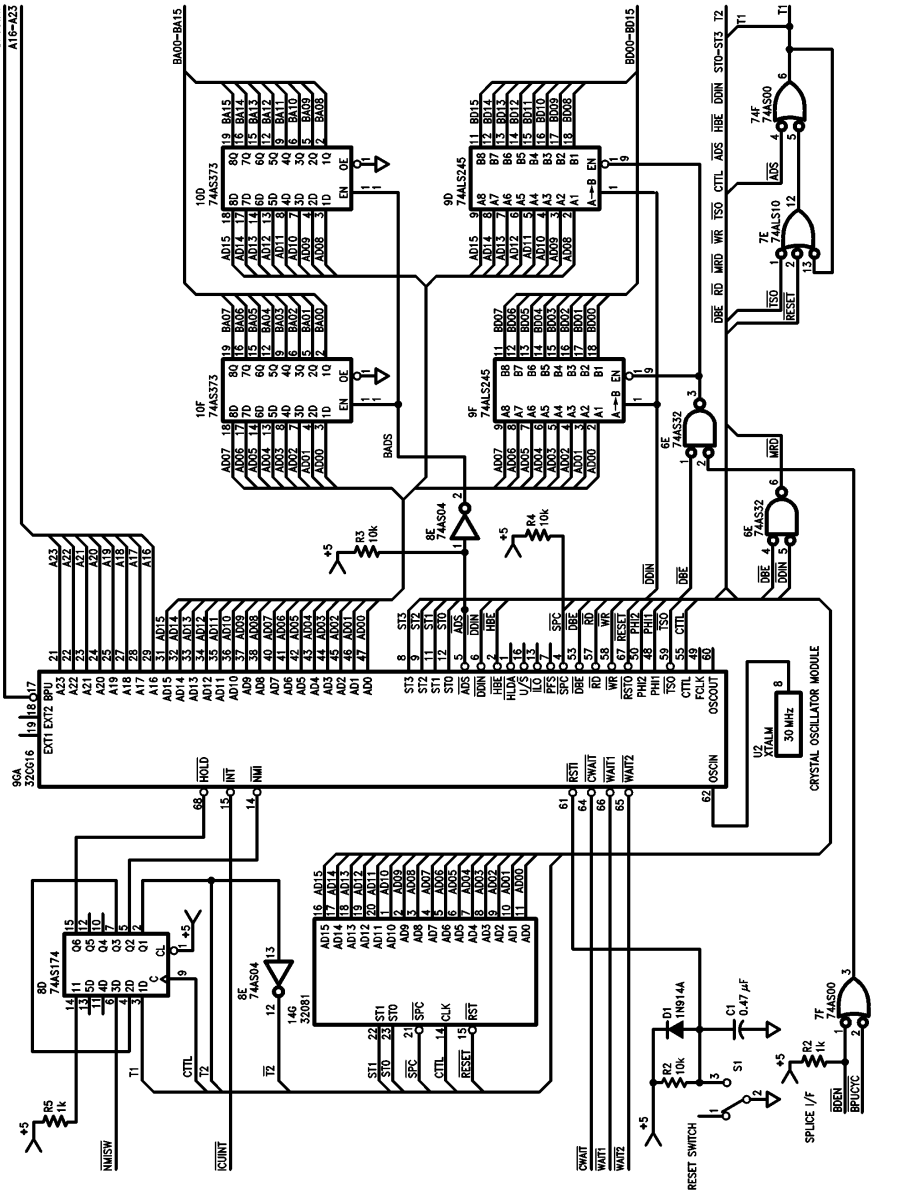
TL/EE/10085–19

32CG16 Functional Block Diagram

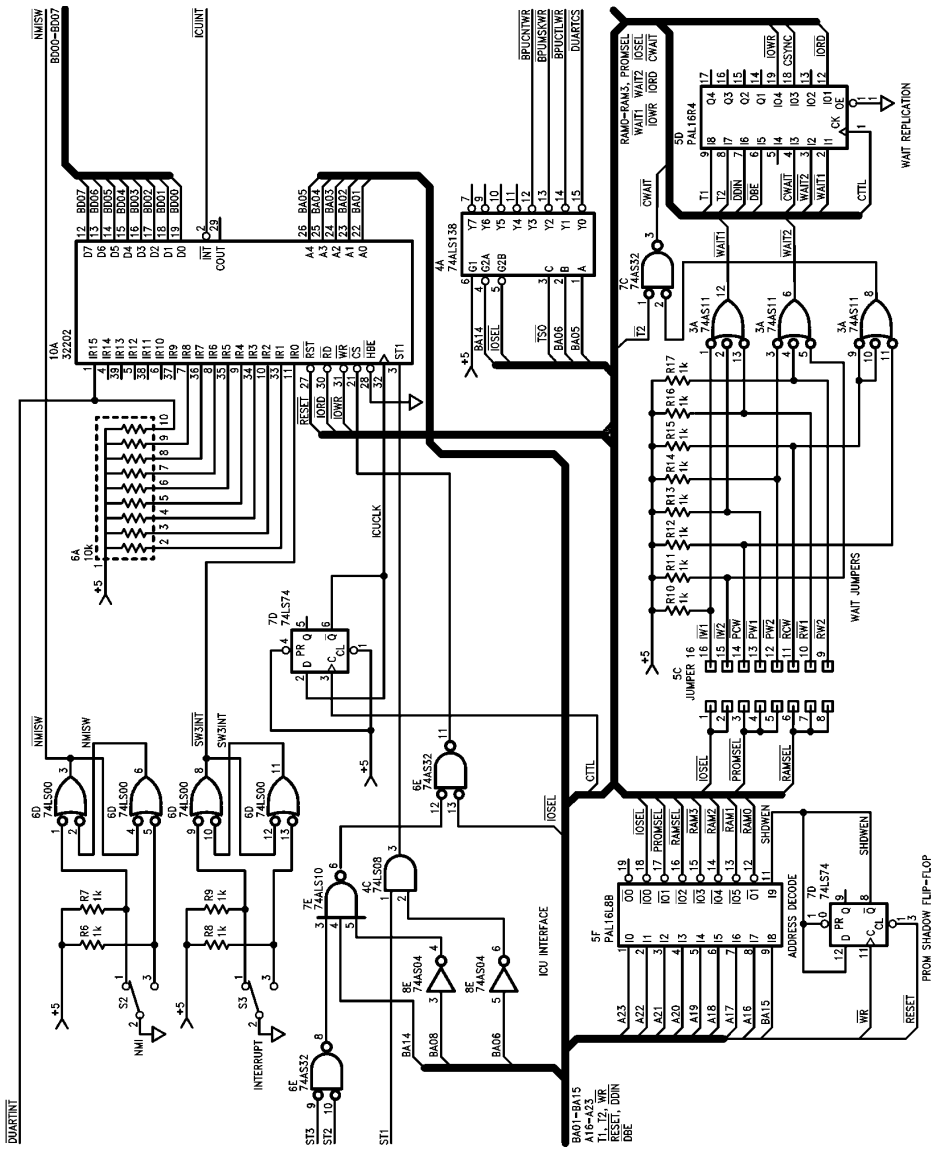


TL/EE/10085–6

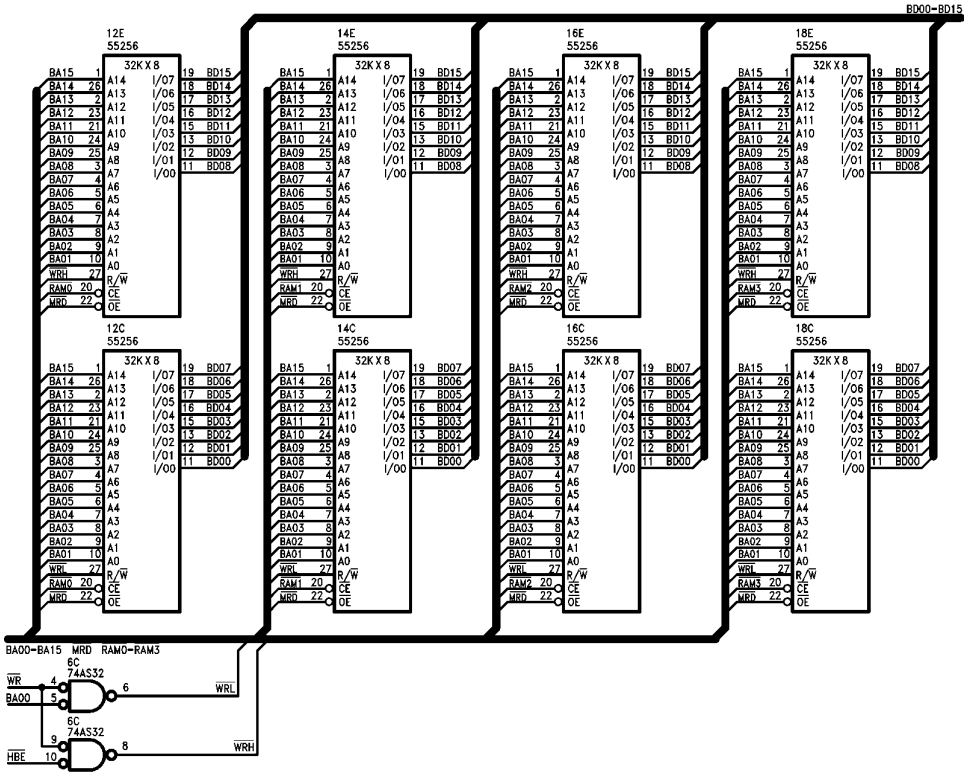
CPU and Buffering



TL/EE/10085–7

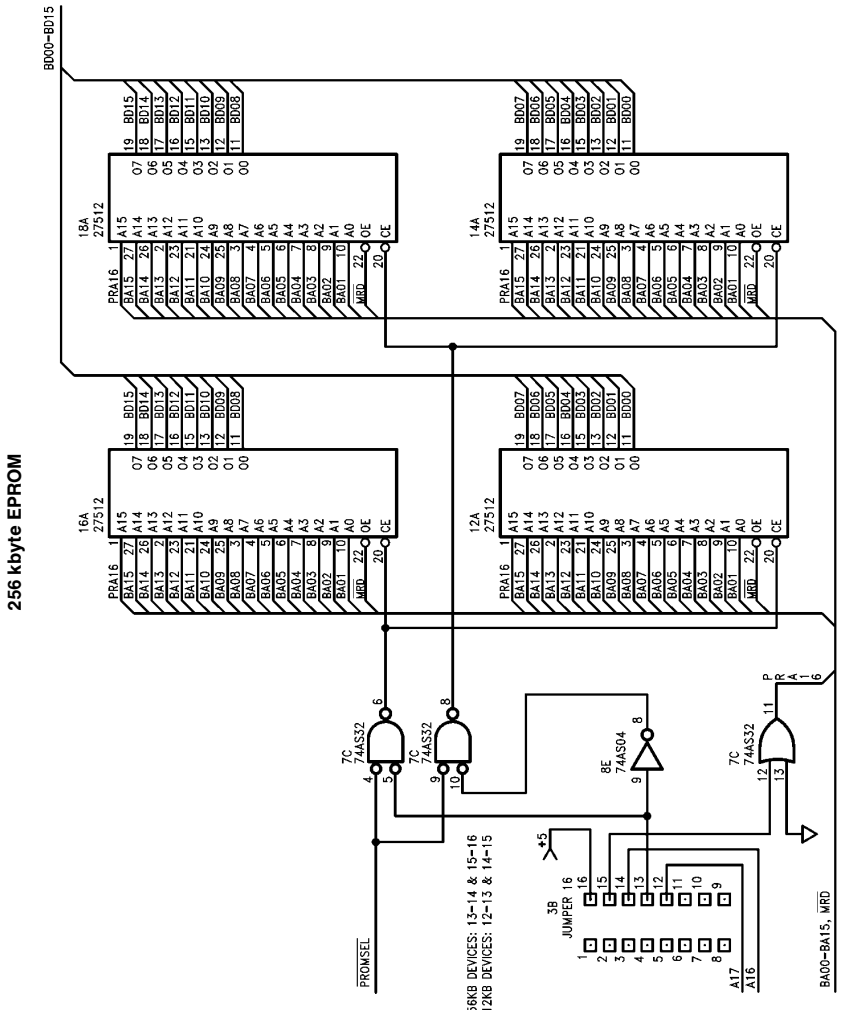

TL/EE/10085–8
WAIT REPLICATION
WAIT JUMPERS
PROM SHADOW FLIP-FLOP
ADDRESS DECODE
ICU INTERFACE
NMI
INTERRUPT
DUARTINT
ICUCLK
CTTL
IOSEL
PROMSEL
RAMSEL
SHDWEN
NMISW
SW3INT
CWAIT
WAIT1
WAIT2
IOWR
IORD
DBE
RESET
BA01–BA15
A16–A23
T1, T2, WR
RESET, DDIN
DBE
BD00–BD07
ICUINT
BPUCNTWR
BPUMSKWR
BPUCTLWR
DUARTCS
RAM0–RAM3, PROMSEL
WAIT1 WAIT2 IOSEL
IOWR IORD CWAIT
10A 32202
5D PAL16R4
5F PAL16L8B
4A 74ALS138
7C 74AS32
3A 74AS11
7D 74LS74
6D 74LS00
6E 74AS32
4C 74LS08
7E 74ALS10
8E 74AS04
5C JUMPER 16
6A 10k
S2
S3
R6 1k
R7 1k
R8 1k
R9 1k
R10 1k
R11 1k
R12 1k
R13 1k
R14 1k
R15 1k
R16 1k
R17 1k

256 kbyte Static RAM



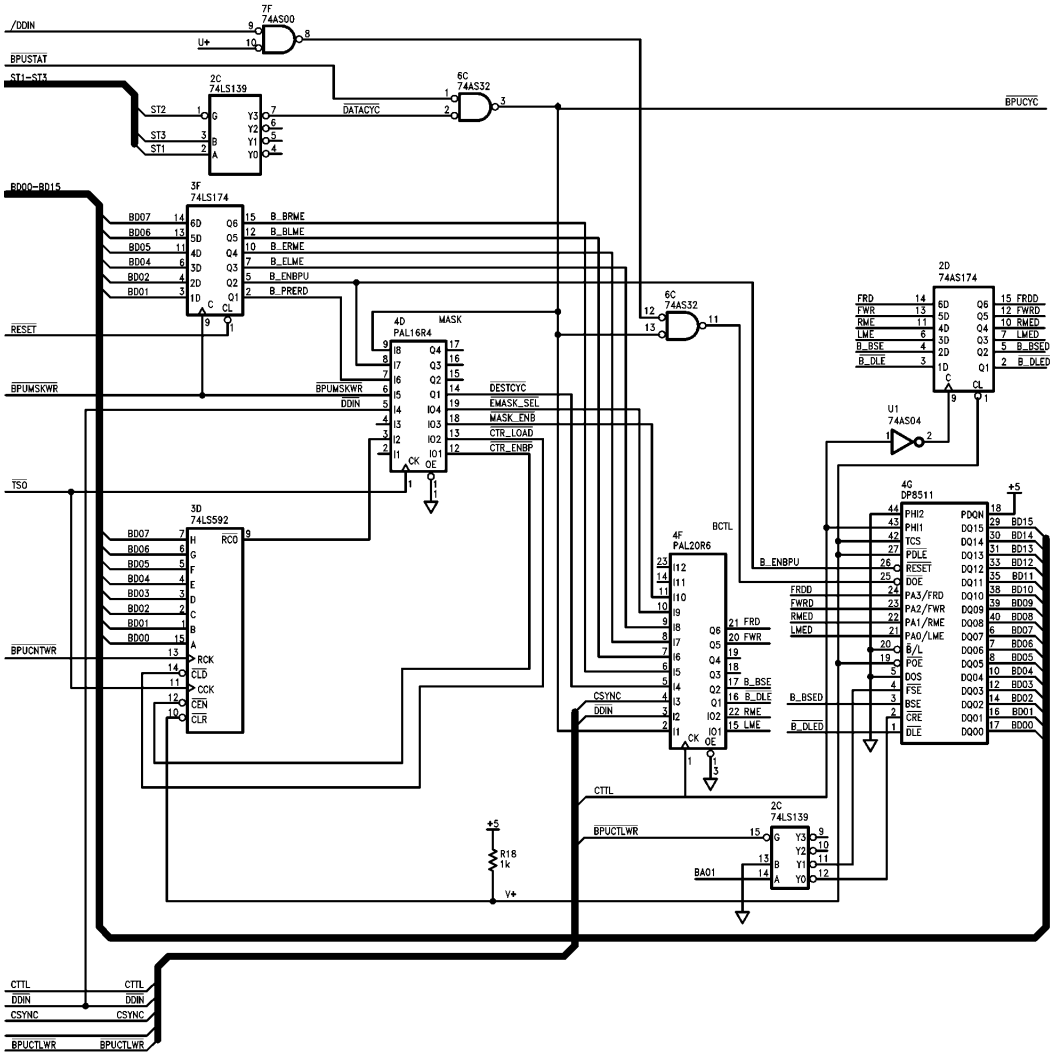
TL/EE/10085–9

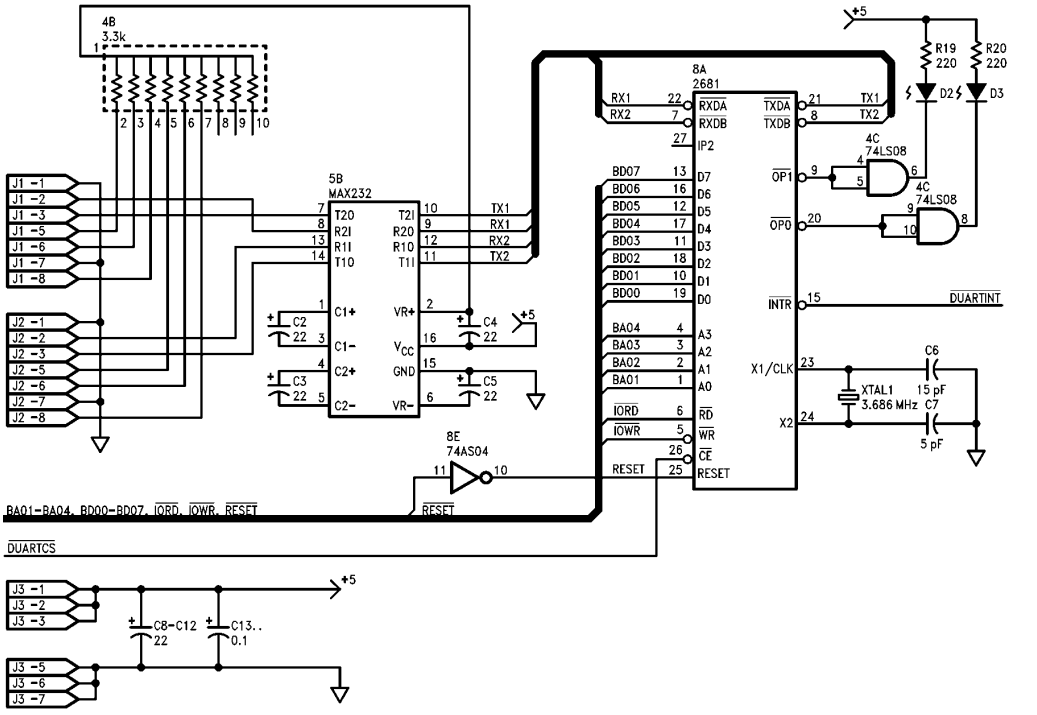
256 kbyte EPROM



TL/EE/10086–10

Bit Processing Unit (DP8511)





TL/EE/10085–12

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

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.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408