

**SBC1190  
PC/104 Embeddable  
Computer  
Board**

**Micro/sys, Inc.**  
3730 Park Place  
Montrose, CA 91020  
Phone: (818) 244-4600  
FAX: (818) 244-4246  
[www.embeddedsys.com](http://www.embeddedsys.com)

DOC1234  
Applicability: rev C and above  
3/7/00

---

## **Micro/sys Technical Support**

---

Micro/sys offers the best technical support in the computer board business - and it's free!

Our application engineers are ready to assist you in getting your SBC1190 project up and running as quickly as possible. You can contact us as follows:

### **Micro/sys Technical Support**

**Phone: (818) 244-4600**

**FAX: (818) 244-4246**

**E-mail: techsupport@embeddedsys.com**

We can also upload and download programs by modem whenever that will assist you.

Thanks for specifying Micro/sys computer boards. We'll be glad to be a part of your team as you use our products.

RUN.EXE trademark Micro/sys

DOC1234  
©2000, Micro/sys, Inc.  
All rights reserved.

---

**□ Table of Contents**

---

<b>Introduction</b>	<b>1</b>
<b>How to Work with the SBC1190</b>	<b>3</b>
<b>Factory Configuration</b>	<b>3</b>
<b>LOAD and RUN Serial Connections</b>	<b>4</b>
<b>Reset Switch Connection</b>	<b>5</b>
<b>Power Connections</b>	<b>5</b>
<b>Running Turbo Debugger</b>	<b>6</b>
<b>BIOS Boss</b>	<b>9</b>
<b>Theory of Operation</b>	<b>13</b>
<b>80C188EB Processor</b>	<b>13</b>
<b>On-Board Memory</b>	<b>13</b>
<b>Off-Board Memory</b>	<b>13</b>
<b>Serial Ports</b>	<b>14</b>
<b>Parallel I/O Lines</b>	<b>14</b>
<b>Interrupt Controller</b>	<b>15</b>
<b>Watchdog Timer</b>	<b>15</b>
<b>Timer/Counters</b>	<b>15</b>
<b>Real-time Clock</b>	<b>16</b>
<b>Analog Input</b>	<b>16</b>
<b>Analog Output</b>	<b>16</b>
<b>Ethernet</b>	<b>16</b>
<b>On-Board Flash Programming</b>	<b>16</b>
<b>Add-on PC/104 I/O Resources</b>	<b>17</b>
<b>RUN.EXE Firmware</b>	<b>17</b>
<b>SBC1190 Card Configuration</b>	<b>19</b>
<b>Memory Mapping</b>	<b>19</b>
<b>Memory Socket U13 Configuration</b>	<b>19</b>
<b>Interrupt Sources</b>	<b>20</b>
<b>Watchdog Timer</b>	<b>20</b>
<b>COM A - RS232</b>	<b>20</b>
<b>Summary of SBC1190 Jumper Settings</b>	<b>21</b>

<b>Installation and External Connections</b>	<b>23</b>
<b>Installation</b>	<b>23</b>
Connecting Power and Reset	23
Serial Ports COMA and COMB	24
Parallel I/O Lines	25
RS485 and Miscellaneous Lines	26
PC/104 Bus	27

<b>Programming the SBC1190</b>	<b>29</b>
<b>RUN.EXE</b>	<b>29</b>
System Initialization without RUN.EXE	29
Serial I/O Ports	<b>30</b>
Parallel I/O Lines	<b>31</b>
Watchdog Timer	32
Red LED	<b>32</b>
Interrupts	<b>33</b>
Timers	<b>33</b>
Real-time Clock	<b>34</b>
Accessing Memory in Socket U13	<b>34</b>

<b>SBC1190 Options</b>	<b>35</b>
A/D Converter-8 Channel	37
D/A Converter-4 Channel	41
Ethernet	45

<b>Troubleshooting</b>	<b>47</b>
------------------------	-----------

## **Appendices**

<b>A</b>	<b>Mapping</b>
<b>B</b>	<b>References</b>
<b>C</b>	<b>Major Device Data</b>
	<b>82C55 Parallel I/O</b>
	<b>DS1302 Clock/Calendar</b>
	<b>MAX197</b>
	<b>MAX525</b>
	<b>Intel 28F400 Flash Memory</b>
	<b>80C188EB Processor</b>
<b>D</b>	<b>SBC1190 Schematic Diagrams</b>



---

## □ Introduction

---

The Micro/sys SBC1190 offers a high integration CPU board at low cost. On the PC/104 form factor (3.55" x 3.775") card, a variety of features are available. Two serial ports, a watchdog timer, a real-time clock, and 24 lines of digital I/O are some of the standard features. The powerful 80C188EB CPU is compatible with the Intel 80x86 family of processors and contains an interrupt controller, three timer/counters, and both serial ports.

Additionally, the SBC1190 has the available options of an eight-channel 12-bit A/D converter, a four-channel 12-bit D/A converter, and a 10Base-T ethernet adapter.

If, however, your application has I/O demands that exceed the on-board resources, simply plug PC/104 I/O modules onto the SBC1190.

The RUN.EXE™ firmware offers reliability and ease of development. This innovative firmware allows the SBC1190 to execute application programs written in any popular PC language. No DOS licenses, special startup code, locators, or special libraries are needed.

The SBC1190 includes flash EPROM program memory, allowing applications to be downloaded to the computer through a serial port. An on-board flash programmer then places the application program in non-volatile flash memory. Upon powerup or reset, the computer henceforth executes this program. With over 100,000 download cycles possible, the SBC1190 will virtually never need any memory device changing due to program changes.





### ***Factory Configuration***

The SBC1190 is a very capable embeddable computer, and there are many possible operating modes. To optimize the SBC1190 for a particular application may take some time familiarizing yourself with the various CPU and on-board peripherals. The other sections of this Reference Manual provide in-depth documentation on all hardware and software aspects of the SBC1190.

However, if you are like most people, you probably want to "kick the tires" of the SBC1190 immediately upon receipt. For this reason, the SBC1190 is shipped complete with installed memory, firmware, and a configuration that allows it to be run directly out of the box.

#### **Here is how the SBC1190 is configured:**

Static RAM memory is installed, and all configuration jumpers are set to default settings for the devices installed.

The flash memory is loaded with the Micro/sys RUN.EXE firmware. This firmware includes an industrial BIOS, a DOS emulator, and an XMODEM program download system. This firmware is in an area of the flash memory that never needs to be modified. This leaves 128K for user programs on SBC1190s with 256K flash memory, 384K for SBC1190s with 512K flash.

As shipped, the user program area is loaded with a program that manages serial communications with a host computer running Borland's Turbo Debugger. Therefore, upon first receiving the SBC1190, you can download programs to RAM for debugging.

Until the first time you download an application program into the flash memory, this debug capability will remain. When you use the XMODEM download, the program you download overwrites the resident debug manager. To use Turbo Debugger again, you will have to download a supplied debug manager .EXE file into the flash program area, overwriting the last application program you downloaded. You can then download to RAM once again.

## **LOAD and RUN Mode Serial Port Connections**

The SBC1190 communicates through serial ports COM A (J4) and COM B (J3). COM A is used as if it were the video monitor on a PC. For example, print statements are sent out COM A. COM B can be used for download and debug. Alternatively, the COM 4 port of a Micro/sys MPC302 Dual Serial card can be used for debug.

The innovative SBC1190 development method uses a special LOAD/RUN cable set that is supplied as part of the DK1190 Kit. **The first cable, CA4020, is plugged into the SBC1190 serial port. The second cable, CA4035, connects between the CA4020 cable and a COM port on your host PC.**

The CA4035 cable is delivered in two different physical configurations which are electrically equivalent. One version has a single DB9 connector on one end, and two DB9 connectors, labeled "RUN" and "LOAD" on the other end. The other version has a single DB9 connector on both ends, with one DB9 having a switch with "RUN" and "LOAD" positions.

Depending upon the connections you make with these cables, the system enters either the BIOS Boss or RUN mode at power-up. The BIOS Boss is for downloading a program and changing the system settings. RUN mode is used for installed systems executing an application program, and also for debug using Turbo Debugger.

**The BIOS Boss is entered only when the CA4035 LOAD connector or switch position is used on the SBC1190 COMB port. The RUN mode is any other serial cabling configuration.**

The following sections indicate when to use LOAD and RUN serial connections, depending upon the desired activity.

### **Reset Switch Connection**

If you want a reset switch, connect any contact closure between the two pins of J6.

### ***Power Connections***

Power is supplied to the SBC1190 through a 5 position removable terminal block, TB1. Legends describing the voltages on each pin are printed on the board.

For most applications, all you need is 5V on TB1-1 and GROUND on TB1-5. TB1-3 can accept +12V and TB1-4 can accept -12V to supply PC/104 add-on modules with those voltages, if needed.

## ***Running Turbo Debugger***

When you power up with the LOAD/RUN cable attached to COM B, SBC1190 firmware determines whether you want to run the BIOS Boss, or the currently downloaded .EXE file. This is determined by the configuration of the CA4035 LOAD/RUN cable plugged into the open 9-pin connector of the CA4020 cable. If there are two 9-pin connectors on one end, use the one labeled 'RUN'. If there is a switch, place it in the 'RUN' position.

Note that if you previously used the BIOS Boss to download an application program, the remote debug manager has been overwritten, and must first be reloaded before Turbo Debugger can be used again. A compatible remote debug manager may be found on the support diskette.

The debug manager sends sign-on and program load status messages to its "console", COM A. While it is not necessary to connect a terminal to display these messages, some designers like to see what's going on. To display these messages, connect a dumb terminal, or a PC running a terminal program, through a null modem cable, to COM A (J3). Set the terminal to 19,200 baud, n, 8, 1.

Apply power to the SBC1190. The green LED (D2) should light. If you attached a terminal to COM A, you will see a sign-on message.

Now start Turbo Debugger on your development PC. To start Turbo Debugger in remote mode at the 115K baud rate that the SBC1190 supports, type:

**C:>td -rp1** if the CA4035 is connected to COM 1 on your PC, or

**C:>td -rp2** if the CA4035 is connected to COM 2 on your PC

The standard Turbo Debugger screen will be displayed on your development PC, allowing you to examine and modify registers, memory, and I/O ports. But more importantly, you can download a program into RAM for test.

A simple demonstration program, BLINK.EXE is supplied, and can

be used as a first program to download to RAM. Alternatively, if you have a terminal attached to COM A, you can run a version of the classic HELLO.C, or any other operator interface program. Make sure that BLINK.EXE and BLINK.C are in your current directory, or you may have to search through directory trees to find them. To download BLINK, use Turbo Debugger's menu system: File / Open / BLINK.EXE.

Now you can single step, or run full speed.  
Alt-X will exit Turbo Debugger.

The set of Turbo Debugger features available depends on the compiler used. Borland C, Pascal, or TASM (assembly language) programs can be debugged using the full feature set. Microsoft C and QuickBASIC programs, and executable files from other compilers, can be downloaded to RAM through Turbo Debugger and run with limited debug features. To set breakpoints or examine variables, you will need to have a listing of the .MAP file generated by your linker, and use the addresses listed instead of the names for routines and variables. (This is because Microsoft does not use standard debug records in their .OBJ files.)

Placing diagnostic print statements at critical points in your non-Borland code can be quite useful for tracing program execution with a terminal attached to COM A. Many designers create a small debug menu and prompt at COM A to allow program internal tables and variables to be dumped to the COM A terminal for debug.

The Turbo Debugger remote debug manager can be used indefinitely until the first time you use the BIOS Boss to download a program, which overwrites the remote debug manager. If you want to return to using Turbo Debugger after the remote debug manager has been overwritten, or to download a different remote debugger version, first locate the desired remote debugger file supplied with the SBC1190. The SBC1190 COM B port, or the COM 4 port of a serial card installed onto the SBC1190 can be used for debug. There are also several different Turbo Debugger version ranges. See the README file on the diskette for information on choosing the proper remote debug manager. Then use the BIOS Boss to download the proper file into flash.







## BIOS Boss™

When you power up the SBC1190, firmware checks for the 'LOAD' configuration of the CA4035 cable connected to COM2 (through the CA4020 cable plugged into J2). When the 'LOAD' end of the cable is used on COM2, powering up the system should result in a sign-on message similar to the following:

```
BIOS Boss, SBC1190 v1.00 (c)Micro/sys, Inc. 1993-98
```

```
Do you wish to start the BIOS Boss?
```

```
<Y>es
```

```
<N>o
```

```
<S>afe mode, no ANSI color
```

Answering 'y' to this will invoke the BIOS Boss, a utility for altering system information and programming. Answering 'S' to this will invoke the BIOS Boss as well but without ANSI display features. Even in safe mode, a minimum VT-100 emulation is still required.

When using the SBC1190 under its standard configuration, the BIOS Boss menu will look something like this:

```
----- Main menu: ----- Micro/sys, Inc. ----- System mode:RUN.EXE -----
<E>rase flash and download EXE file
E<X>it BIOS Boss
Flash <S>etup                                     <C>omm speed
----- Last error = No error -----
```

```
No EXE file found:
```

```
Max size of EXE file that can be downloaded = 393216
bytes
```

## **<E>rase flash and load stand-alone EXE file**

This menu selection allows you to transfer a single stand-alone EXE file from your desktop PC to the SBC1190 using the XMODEM protocol.

Type 'E' to begin this file download procedure. Because this function will erase any application previously loaded on the SBC1190, the BIOS Boss will require that you confirm this choice by typing 'c' to continue.

Erasing of the flash may take a few seconds, once this is complete, you will be directed to start an XMODEM transfer of your file. Since you are sending a file to the SBC1190, most terminal programs actually call this an upload.

At this point, use whatever keystrokes are necessary to enter the file upload function of your terminal program. If presented with a series of protocols from which to choose, select XMODEM. Satisfy any other needs your terminal program has, among them entering the name of the file you want to download.

Once the download has started, most terminal programs will give a live display of the current download status, for instance the total file size, the number of bytes sent so far, and estimated download time (which is, in some cases, quite inaccurate). The program is programmed into the flash memory as it is received by the SBC1190. Once the file has been downloaded and programmed into the flash memory, the BIOS Boss displays a message that the download is complete.

## **E<X>it BIOS Boss**

This menu selection will exit the BIOS Boss and attempt to start the system and (already loaded) EXE file as defined in *Flash <S>etup*.

## Flash <S>etup™

This menu selection allows you to view and alter system startup parameters and information. The Flash Setup menu will look something like this:

```
Flash setup menu -----Micro/sys, Inc.
<ESC> Return to main menu
<T>ime/date settings
<S>afe/ansi mode:   ANSI mode (with color)
<C>ommand Tail:

N<e>twork Setup
```

### <T>ime/date settings

Use this option to view and make changes to the time and date.

### <S>afe/ansi mode

Most terminals and terminal programs will support ANSI and extended ASCII which are used by the BIOS Boss to display its user interface. If the BIOS Boss menus do not appear to be displayed properly then your terminal may not support ANSI and/or extended ASCII. To get around this problem you can set the BIOS Boss to operate in Safe mode which does not use these features.

Note: The BIOS Boss does require a VT100 or better terminal, even in safe mode.

### <C>ommand Tail

The BIOS Boss allows the use of EXE files that can accept command tails that would normally be typed after the application name on the command line of a DOS prompt. If your application can accept command line information, you may enter that command tail here.

#### Example:

You may have one program which you may want to run differently on different SBC1190s or you may just want to be able to periodically alter startup parameters without recompiling

and reuploading your code each time. For example, let's say that your program (USERPROG.EXE) has the ability to communicate out either COM port at various baud rates. In a DOS environment you might have the following line in your AUTOEXEC.BAT file:

**USERPROG 2 9600**

After loading USERPROG.EXE into the SBC1190 flash, you can then enter the following information in the Command Tail setting for a similar effect:

**<C>ommand Tail: 2 9600**

### **<E>thernet setup**

If an Micro/sys MPC356 PC/104 Ethernet Adapter is installed, then you will also have an ethernet setup option available. This selection will allow you to view the system's ethernet settings and system resources. It also allows you to change IP address values.

```
— Network setup menu: — Micro/sys, Inc. —
IP address <S>ource: Specify an IP address
<I>P address:          0. 0. 0. 0
S<u>bnet mask:         0. 0. 0. 0
Default <G>ateway:    0. 0. 0. 0

<L>ocal name:

Ethernet Address:      00-60-92-00-10-00
Ethernet base address: 300h
Ethernet IR<Q>:        IRQ4

<A>dvanced network menu
```

### **<C>omm Speed**

This menu selection allows you to change the speed of the COM ports to baud rates faster than 19200bps. This is particularly useful when you are downloading large files to the flash. Changes made with this option are for this BIOS Boss session only and are not permanently saved.

### ***80C188EB Processor***

At the heart of the SBC1190 is the Intel 80C188EB processor. This chip is a 16-bit, CMOS CPU that can execute the 8086 instruction set. It has an 8-bit external data bus in order to reduce the number of external chips necessary. It also integrates a number of common PC peripherals in order to further reduce the chip count. The 80C188EB includes an interrupt controller, three 16-bit timer/counters, and a wait-state generator. It also includes two UARTs.

### ***On-board Memory***

The on-board memory consists of a 32-pin JEDEC memory socket, one surface mounted flash memory, and a surface mounted RAM. The surface mounted RAM, U10, is configured for static or pseudo-static RAM. A 128k x 8 or 512k x 8 static or pseudo-static RAM can be installed.

A 5V flash EPROM is surface mounted on the board. RUN.EXE firmware resides in this flash EPROM. However, due to the efficiency of RUN.EXE, there is still a large memory space available for the application program. 256k x 8 is the standard flash size, with 512k available as an option.

Socket U13 can accept 32k or 128k static RAM, and the RAM can be battery-backed. In addition, U13 can accept an EPROM: 27C128 (16k), 27C256 (32k), 27C512 (64k), or 27C010 (128k).

### ***Off-board Memory***

Any memory that is not allocated to on-board memory defaults to off-board memory. The memory mapping PAL equations in U7, and software initialization of the 80C188EB chip select perform this function.

## ***Serial Ports***

The SBC1190 has two asynchronous serial ports on-board. The UARTs for COM A and COM B are integrated into the 80C188EB. Note that these ports are not PC-compatible, but offer additional capabilities such as 9-bit modes of operation.

The COM ports support the RXD and TXD lines, and modem controls RTS and CTS. The COM ports have internal baud rate generators, and support baud rates over 200K bits per second.

The 80C188EB serial port 0 is routed to the COM B connector, and serial port 1 is routed to the COM A connector. Serial port 0 has integral interrupt capabilities that make it preferable for COM B connections during debug.

COM A signals are available on the J5 connector as full-duplex RS485 signals.

## ***Parallel I/O Lines***

The SBC1190 includes an 82C55 device for general-purpose digital I/O. The 24 I/O lines are CMOS/TTL compatible, and are programmable as input or output in two groups of 8 bits and two groups of 4 bits. Eight of the lines have bit set/reset capabilities. All 24 lines are presented, unbuffered, at connector J2.

## ***Interrupt Controller***

An interrupt controller is integrated into the 80C188EB, and it provides 7 maskable and 1 non-maskable interrupts. Two internal interrupt sources are the timers and the COM A serial port.

External interrupt inputs 0, 1, 2, and 3 are routed to the PC/104 interface for use by add-on I/O modules as IRQ4, IRQ5, IRQ6, and IRQ7, respectively. Configuration jumper W9 allows CPU external interrupt input 4 to be selected from the COM A serial port, or from PC/104 IRQ2 or IRQ3.

Configuration jumper W7 selects either the PC/104 IOCHCK signal or the on-board power fail signal for the NMI non-maskable interrupt input. W8 can be jumpered to cause an interrupt when an A/D conversion is complete.

## ***Watchdog Timer***

In many applications it is important that if the program gets "lost," a hardware reset will occur in order to restart the system. The SBC1190 has an on-board watchdog timer to effect such a reset.

The on-board watchdog timer is designed so that if it doesn't get strobed within 1.6 seconds, it will reset the system. It may be strobed by writing to port FE4CH. If the program continues to strobe the watchdog timer, it is assumed that the program is operating correctly and no reset will result.

## ***Timer/Counters***

The 80C188EB contains three timer/counters (T0, T1, & T2). Two of these are connected to external pins available at connector J5. The third timer (T2) is not connected to any external pins, and can only be clocked by the internal CPU clock. It can be used as a prescaler to the other two timers, or it can generate interrupts. Under RUN.EXE, the third timer is used as a prescaler for T0 which serves as the DOS timer tick for the system.

## ***Real-time Clock***

The real-time clock allows the SBC1190 to keep track of the date and time. The Dallas DS1302 clock chip is attached to 3 internal I/O port pins of the 80C188EB CPU. The DS1302 internal registers contain year, month, date, day-of-week, hours, minutes, seconds, and an AM/PM indicator.

## ***Analog Input***

This option provides 8 channels of analog input to the SBC1190. The A/D chip used is the Maxim MAX197. This chip allows for independent software configuration of the input ranges for each channel. The ranges allowed are:  $\pm 10V$ ,  $\pm 5V$ ,  $+10V$ ,  $+5V$  and provide 12-bits of accuracy.

## ***Analog Output***

The analog output option uses a Maxim MAX525 chip to provide 4 channels of analog output. This 12-bit DAC has an output range of 0 to 5V.

## ***Ethernet***

The optional ethernet interface is designed to connect to a 10BASE-T, twisted-pair network. The ethernet controller chip used is the SMC LAN91C94. When installed, this option includes the Embedded Netsock system which provides a built-in network stack that allows easy connection to a network with minimal coding.

## ***On-board Flash Programming***

The SBC1190 has the capability of programming flash memory on-board. This option uses one bit (D5) of the 80C188EB port 1 to enable writes to the flash memory.



## ***Add-on PC/104 I/O Resources***

The J1 connector is a standard PC/104 connector for adding more I/O to a system based on the SBC1190. A wide range of PC/104 modules are available, including digital I/O, analog I/O, communication, network, video, GPIB, and motor control.

The PC/104 implemented on the SBC1190 is for 8-bit transfers to add-on cards.

### ***RUN.EXE firmware***

RUN.EXE firmware installed in flash memory at U22 occupies the top of the memory map, including the power-up vector. When applying power or resetting the system, the firmware initializes the registers in the 80C188EB and then sets up the environment so that, to an application program, it looks like an IBM PC running DOS. It can then: run an .EXE file stored in the lower part of the same flash EPROM device, or load a file from the COM B port into flash memory using the XMODEM protocol.



---

## □ SBC1190 Card Configuration

---

The SBC1190 is a highly integrated CPU, with on-board memory and firmware. This means that there is very little to configure according to any application. Most system configuration will be on add-on PC/104 cards. However, there are a few configuration jumpers on the SBC1190 that may need to be set. This section details all SBC1190 configuration jumpers.

### ***Memory Mapping***

Upon power-up or reset, the SBC1190 firmware automatically sizes flash EPROM and RAM. Memory mapping is then programmed through the use of the 80C188EB chip select unit. Chip selects will be initialized properly for the RAM device in U10, and the flash code storage device in U22.

Flash EPROM at U22 is mapped at the top of the 1MB address space, and RAM in U10 is mapped at the bottom of the 1MB address space. Refer to the Memory Map and I/O Map heading in the *Programming the SBC1190* section.

### ***Memory Socket U13 Configuration***

The U13 socket can be configured for RAM or EPROM devices, depending on the jumpering of W10.

To configure U13 for 32k or 128k RAM:  
jumper W10 pins 1-3, 2-4, and 5-6.

To configure U13 for 16k, 32k, 64k, or 128k EPROM  
or 128k 5V flash EPROM:  
jumper W10 pins 1-2, 3-5, and 4-6.

The onboard 3V battery can be configured to back up low-power CMOS RAM installed in U13. Jumper W6 selects the power source for the device in U13. Jumper 1-2 for 3V battery backup power to be applied to U13 whenever system power is removed; jumper 2-3 for normal Vcc powering of U13. Do not select battery backup for an EPROM device installed in socket U13.

## ***Interrupt Sources***

In systems with many interrupts, some choices may have to be made in routing interrupt inputs to various 80C188EB interrupt pins. Jumpers W7, W8, & W9 accomplish this.

W9 selects the input for 80C188EB pin INT4. Jumper 1-3 to connect PC/104 IRQ3 to INT4; jumper 2-3 to connect PC/104 IRQ2 to INT4 this requires a wire rather than a plug-on jumper 3-4 to connect COM A interrupt to INT4.

W7 selects the input for 80C188EB non-maskable interrupt. Jumper 1-2 to connect the on-board power fail signal to NMI; leave unjumped to connect PC/104 IOCHCK signal to NMI.

## ***Watchdog Timer***

Jumper block W4 enables the watchdog timer strobe. If the watchdog timer is enabled by W4 1-2 shorted, make sure that your application program reads or writes to the appropriate port before the timeout period of about 1.6 seconds elapses.

## ***COM A - RS485***

The COM A serial port can be configured for either RS232 or RS485. Jumper W2 is used to select RS232 or RS485 for COM A. Jumper W2 1-2 for RS485 operation.

## ***Ethernet***

W5 controls the powerup settings of the ethernet controller. When W5 is jumpered, the ethernet will powerup at its default settings of 300h. W5 also disables the EEPROM where the ethernet configuration data is stored. Therefore, to program the configuration settings for the ethernet controller or to load preprogrammed settings at powerup, remove any jumper from W5.

## **Summary of SBC1190 Jumper Settings**

W1	---	Factory use only
W2	1-2 2-3	COM A - RS485 COM A - RS232
W3	---	Factory use only
W4	1-2 N/C	Enable watchdog timer Disable watchdog timer
W5	1-2 2-3	EEPROM disable - Ethernet uses default settings EEPROM enable - Ethernet loads settings from EEPROM
W6	1-2 2-3	U13 battery-backed U13 on system Vcc
W7	1-2 N/C	PFAIL to NMI PC/104 IOCHCK to NMI
W8	1-2 N/C	A/D conversion interrupt on IRQ7 (INT3) A/D polled for end of conversion
W9	1-3 2-3 3-4	IRQ3 to INT4 IRQ2 to INT4 COMA to INT4
W10	1-3, 2-4, 4-6 1-2, 3-5, 4-6	U14 configured for RAM U14 configured for EPROM





## **Installation and External Connections**

---

### ***Installation***

The SBC1190 can be installed using the four mounting holes supplied in the corners. These holes are placed according to the PC/104 standard.

### ***Connecting Power and Reset***

Power is applied to the SBC1190 through terminal strip TB1. +5V, +12V, -12V, external battery, and ground terminals are provided. A reset switch can be attached to two-pin connector J6.

## **Serial Ports COM A and COM B**

Connections to serial ports COM A and COM B are made at connectors J4 and J3, respectively. Both are 10-pin headers. A simple ribbon cable can be used to adapt J3 and J4 signals to a 9-pin male connector, which is standard on IBM AT computers. The following table shows the pinout for J3 and J4, as well as the wiring of the ribbon cable assembly that adapts to a DB9 connector.

<b>J4 COM-A and J3 COM-B Serial Port Connector/Cable</b>		
<b>J4/J3 Pin</b>	<b>RS232</b>	<b>DB9 Pin</b>
1		1
2		6
3	RX Data In	2
4	RTS Out	7
5	TX Data Out	3
6	CTS In	8
7		4
8		9
9	GND	5
10		

The Micro/sys cable CA4020 adapts the J3 or J4 10-pin header to a standard DB9 connector.



## I/O Lines

Connections to the parallel I/O lines from the 82C55 are made at connector J2, at the first 26 pins. Additionally, if the A/D or D/A options are installed, the analog signals will be available on the same connector. The following table shows the pinout for J2.

J2 I/O Connector			
J2 Pin	Signal	Signal	J2 Pin
1	PA0	PA1	2
3	PA2	PA3	4
5	PA4	PA5	6
7	PA6	PA7	8
9	PB0	PB1	10
11	PB2	PB3	12
13	PB4	PB5	14
15	PB6	PB7	16
17	PC0	PC1	18
19	PC2	PB3	20
21	PC4	PC5	22
23	PC6	PC7	24
25	GND	GND	26
27	AOUT0	GND	28
29	AOUT1	GND	30
31	AOUT2	GND	32
33	AOUT3	GND	34
35	AIN7	GND	36
37	AIN6	GND	38
39	AIN5	GND	40
41	AIN4	GND	42
43	AIN3	GND	44
45	AIN2	GND	46
47	AIN1	GND	48
49	AIN0	GND	50

## ***RS485 and Miscellaneous Lines***

A 14-pin header, J5, provides inputs and outputs for timers 0 and 1. COM A RS485 signals are also available at this connector. Additionally, there is a single digital input, signal from the 188EB processor that has been made available at J5. The following table shows the pinout for J5.

<b>J5 Timer/Misc Connector</b>	
Pin	Signal
1	GND
2	485TX+
3	485 TX-
4	485RX+
5	485RX-
6	P2.5 (from CPU)
7	GND
8	VCC
9	T0 In
10	T0 Out
11	T1 In
12	T1 Out
13	GND
14	-Bootswap

## PC/104 Bus

J1 of the SBC1190 provides a standard PC/104 connection for easy integration of 8-bit PC/104 plug-on modules. The following table shows the pinout for J1.

1A	1B
IOCHK*	GND
SD7	RESET
SD6	+5V
SD5	IRQ9
SD4	--
SD3	--
SD2	-12V
SD1	--
SD0	+12V
IOCHRDY	(KEY)
AEN	SMEMW*
SA19	SMEMR*
SA18	IOW*
SA17	IOR*
SA16	--
SA15	--
SA14	--
SA13	--
SA12	REFRESH*
SA11	SYSCLK
SA10	IRQ7
SA9	IRQ6
SA8	IRQ5
SA7	IRQ4
SA6	IRQ3
SA5	--
SA4	GND
SA3	BALE
SA2	+5V
SA1	OSC (14MHz)
SA0	GND
GND	GND



## ***RUN.EXE***

When developing applications for the SBC1190, the RUN.EXE firmware offers quick, easy development. RUN.EXE executes .EXE files created by popular PC languages without the royalty cost or memory cost of having DOS in the target system. Also, debugging is facilitated by using the powerful Borland Turbo Debugger to download and step through your program.

Using RUN.EXE, you can ignore many of the low-level, detailed programming issues involved with the SBC1190. RUN.EXE performs all device and memory initialization, starts the system timer ticks, and presents a warm, running environment to the program immediately upon power-up. For the most part, you can write a program as though you were writing it for an IBM PC.

RUN.EXE also redirects all console I/O through the COM A port so that if programming in C, printf() will print characters out COM A and scanf() will receive characters from COM A. Likewise with WriteLn() and ReadLn() in Pascal.

## ***System Initialization Without RUN.EXE***

When RUN.EXE is not installed, you will need to handle all of the initialization of the devices on-board. The 80C188EB is highly configurable for many modes of operation and it will not operate properly until the configuration registers have been initialized.

The details of SBC1190 initialization must be selected according to application needs. The Appendices include full programming details for the major SBC1190 devices.

## Serial I/O Ports

The SBC1190 BIOS supports INT 14h, RS232 I/O. An application program can use INT 14h BIOS calls to manage one or both serial ports. However, the debug manager uses INT 14h, so **INT 14h is not available when running under Turbo Debugger.**

To configure COM A and COM B, you can use BIOS INT 14h. Most C compilers offer library support (i.e. `_biosserialcom()`) for setting baud rate, stop bits, data bits, and parity. Use COM A indicators for SBC1190 COM A, and COM B indicators for SBC1190 COM B.

**Note that the SBC1190 COM A and COM B ports are not PC-compatible at the register and bit level.** This means that existing code that directly accesses COM 1, COM 2, COM 3, or COM 4 of a PC will not operate properly on the SBC1190. Micro/sys offers the CommBLOK software library with interrupt-driven, buffered serial drivers through the SBC1190 COM A and COM B ports.

RUN.EXE firmware initializes both COM ports for 19,200 baud, 8 data bits, 1 stop bit, no parity. Since it is not operating the COM ports in interrupt-driven mode, it does not enable COM interrupts. Also, the console I/O (BIOS INT 10h and INT 16h) is redirected through the COM A port. Hence, COM A is best for serial operator I/O.

When programming the SBC1190 serial ports, use the following cross reference table:

<u>SBC1190 Port</u>	<u>BIOS Port</u>	<u>80C188EB Port</u>
COM A	0	1
COM B	1	0

The RTS control outputs are implemented as two pins of the CPU parallel port P1. RTS for COM A is bit P1.4; RTS for COM B is bit P1.3. These output bits can be set and cleared through reading and writing to Port 1 at I/O ports FF52h and FF56h. The CTS inputs are readable in the Serial Status registers. Note that bit 6 in each Serial Control register affects operation of the CTS input.

COM A has a full-duplex RS485 connection available. You can use

it to link over 30 units on a single twisted-pair cable. You must manage the transmit enable of the RS485 driver with your software such that only one transmitter on one unit is enabled at any point in time. To enable the RS485 transmitter on the SBC1190, set the RTS for COM A. Clear RTS to receive. (See previous page for controlling RTS)

### ***Parallel I/O Lines***

Programming the parallel I/O lines is accomplished by merely accessing the 82C55 device at I/O ports FE40h - FE43h. Port A is at FE40, Port B is at FE41, Port C is at FE42, and the Configuration Register is at FE43. (See the data sheet on the 82C55 device in the Appendix.)

A single write to the digital I/O configuration register will set the operation of all the digital I/O ports. For simple *mode 0* operation, the following table may be used to select a suitable configuration byte.

Digital I/O Control Byte				
Port A	Port B	Port C 7-4	Port C 3-0	Control Byte
Out	Out	Out	Out	80h
Out	Out	Out	In	81h
Out	Out	In	Out	88h
Out	Out	In	In	89h
Out	In	Out	Out	82h
Out	In	Out	In	83h
Out	In	In	Out	8Ah
Out	In	In	In	8Bh
In	Out	Out	Out	90h
In	Out	Out	In	91h
In	Out	In	Out	98h
In	Out	In	In	99h
In	In	Out	Out	92h
In	In	Out	In	93h
In	In	In	Out	9Ah
In	In	In	In	9Bh

### **Watchdog Timer**

The watchdog timer is designed to restart the system in the event of a program crash. The watchdog timer is strobed by writing to port FE4Ch. If the watchdog timer is enabled, your program will need



to strobe the watchdog at least once every 1.6 seconds or a system reset will occur.

Under RUN.EXE firmware, the interrupt service routine for interrupt 1Ch strobes the watchdog timer by doing a write to port FE4Ch. Interrupt 1Ch is the DOS user timer tick routine which is called 18.2 times per second. This ensures that the system comes up properly and continues to run until the application program takes over. Your program can then revector interrupt 1Ch to a null interrupt handler or to a custom interrupt handler.

Once interrupt 1Ch is revector, it becomes the responsibility of the application program to strobe the watchdog. It is usually desirable to strobe the watchdog from mainline code rather than from an interrupt service routine because it is possible that an ISR may continue to run after a program has crashed.

### *Red LED*

Data bit D6 of 80C188EB port 1 is connected to LED2. Setting P1.6 low turns on the LED; setting P1.6 high turns off the LED. This can be used by a program for various signalling purposes.

### ***Interrupts***

The 80C188EB interrupt controller is programmed through ports FF02h to FF1Fh.

Note that there are two phases to interrupt programming: initialization and service routines.

Initialization of the interrupt controller must take place before any interrupts occur. Additionally, interrupt vectors must be set up in the first page of RAM to point to the appropriate interrupt handler for each type of hardware interrupt.

Interrupt service routines must perform End Of Interrupt (EOI) acknowledgements to the hardware. First the interrupting device must be serviced (e.g. a received character read from the COM port that caused a receive interrupt). Next, the interrupt controller must be given an EOI command to unlock it for the next interrupt.

Failure to properly sequence EOI commands can result in missing all interrupts after the first.

Refer to the Appendix for details on the 80C188EB interrupt controller.

### ***Timers***

The three timer/counters in the 80C188EB are programmed through ports FF30h to FF47h. Note, however, that on the IBM PC, the clock source for all three timers is a 1.19318 MHz oscillator. On the SBC1190, the clock source for two of the timers can be set to be either internal or external. The internal clock is the CPU clock (12 MHz, 20 MHz, or 25 MHz) divided by 2, 4, 8, or 16. The external clock is from connector J5. Thus, when programming the timers directly, the count should be changed to reflect the differences in input frequency from a standard PC.

The RUN.EXE firmware initializes the 80C188EB timer registers so that the output of timer 0 is 18.2Hz, implementing the standard PC BIOS timer tick.

### ***Real-time Clock***

The real-time clock allows the SBC1190 to constantly keep track of the time and date when the system is powered down. It also allows data to be stored in RAM and retrieved after power is restored. The clock is based on the Dallas DS1302 clock chip. It is accessed through 3 I/O port lines of the 80C188EB.

With the RUN.EXE firmware, you can call Interrupt 1Ah Function 02h through Function 05h to set or read the clock. Details on using this interrupt are available in a number of books such as the DOS Programmer's Reference (1992, Dettmann and Johnson, Que Corporation).

## ***Accessing Memory in Socket U13***

If RAM or EPROM devices are installed in socket U13, the size and starting address must be programmed into the 80C188EB chip select unit. Socket U13 is accessed through the GCS0 chip select line. You must configure the GCS0 signal through the chip select configurations registers, according to programming information in Appendix A.

**Note that you must not let U13 overlap memory areas reserved for RAM in U10, or flash addresses F0000h - FFFFFh.**

During initialization, RUN.EXE firmware opens chip select UCS to "see" the entire flash device. After initialization and loading, UCS is reduced to the 64K address range F0000h - FFFFFh for BIOS code. Therefore, your program can configure GCS0 upon startup for a desired address range for the memory in U13. By enabling and disabling GCS0, you can "unlock" and "lock" the memory contents in U13 (i.e. for battery-backed RAM integrity)



---

**SBC1190 Options**

---

8 Channel A/D

4 Channel D/A

Ethernet



---

## ◆ A/D Converter - 8 Channel

---

When ordered with 1190OPT11, the SBC1190 comes installed with the Maxim MAX197 multi-range, 12-bit data acquisition system (DAS). This system provides 8 analog input channels that are independently software programmable for a variety of ranges. Connections to the A/D can be made at connector J2. A pinout for connector J2 follows the A/D & D/A sections. Programming of the A/D converter may be accomplished in one of two ways. The device may be accessed directly, or through a special Micro/sys system function.

### Direct Method:

To access the A/D converter directly requires three steps.

1. First, the **Control Byte** must be written to the Base Address of the A/D converter which is located at port address FE44h. The Control Byte carries your configuration choices for the analog read. See the tables on the following page for setting the individual bits of the Control Byte. The A/D conversion is initiated by the writing of the Control Byte.
2. Once the conversion is complete the A/D converter will raise the ADCRDY signal. This signal is available at digital I/O Port 2.2 of the the 188EB. The port 2 pin register is loaded at address 0xFF5A by continuously polling the ADCRDY bit you will know when the converted data is ready to read. This signal can also be connected to INT3 so that the A/D converter may be controlled through the use of interrupts.
3. To read the converted data, simply **read the A/D converter** at its Base Address to get the low byte and then read the A/D converter again at its Base Address + 1 for the high byte.

Read the converted data as a signed integer. For unipolar conversions, the value will be an integer in the range 0 to +4095. For bipolar conversions, the value will be an integer in the range -2048 to +2047.

## A/D System Function:

The A/D converter may also be accessed by use of the BIOS interrupt 1Ah function DDh. Subfunctions 20h and 21h may be called to start and check the A/D conversion.

### **Subfunction 20h:** Start ADC Conversion

Inputs:

AL	20h	
AH	DDh	
BL	Analog channel (0 - 7)	
BH	0	0 to +5V range
	1	-5 to +5V range
	2	0 to +10V range
	3	-10 to +10V range

Outputs:

None

### **Subfunction 21h:** Check A/D Conversion

Inputs:

AL	21h
AH	DDh

Outputs:

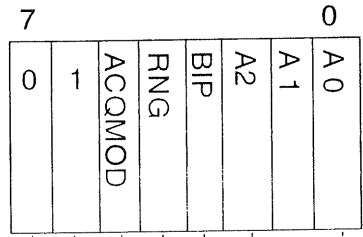
BL	0	conversion not complete
	1	conversion complete
AX	ADC reading if conversion is complete	

(Look for A/D examples on the support diskette)



### CONTROL Byte

Address= FE44h



Bit 7-6 - Set as indicated:

Bit 5 - ACQMOD:

0 = internally controlled acquisition  
1 = externally controlled acquisition

Bit 4 - RNG:

(See Table)

Bit 3 - BIP:

(See Table)

Bit 2-0 - Channel:

Range & Polarity Selection		
BIP	RNG	Input Range (V)
0	0	0 to 5
0	1	0 to 10
1	0	$\pm 5$
1	1	$\pm 10$



---

## ◆ D/A Converter - 4 Channel

---

When ordered with 1190OPT2, the SBC1190 comes installed with the Maxim, 12-bit Voltage-Output DAC with Serial Interface. This system provides four 12-bit, voltage-output digital-to-analog converters (DAC) and four precision output amplifiers. Connections to the D/A can be made at connector J2. A pinout for connector J2 follows the A/D & D/A sections. Programming of the A/D converter may be accomplished in one of two ways. The device may be accessed directly, or through a special Micro/sys system function.

### Direct Method:

1. Before starting a D/A conversion, make sure the DACCS# and the RTCDACCLK signals are set high. These signals are available at 188EB DIO port 2, bits 5 and 6 respectively. Then, to enable the converter, **drop the DACCS# signal**. This signal must remain low while the DAC is being written to.
2. Next, **write the 16-bit Input Data Word** to the Converter. The serial data is composed of two DAC address bits, two control bits, and the 12 data bits. The 4-bit address/control code determines the register or registers to be updated and the mode in which they are written to. Use the table on the following page to construct the input data word.

To clock data into the DAC, use the RTCDACCLK and RTCDACIO signals. RTCDACIO is available at 188EB DIO port 2, bit 7, and defines the data bit being sent. RTCDACCLK is available at 188EB DIO port 2, bit 6, and provides the clock pulse. Data bits are clocked in on the rising edge of RTCDACCLK.

3. Once the input data word has been clocked into the DAC's input registers, the data must be latched into the appropriate registers. To do this, **set the DACCS# signal**. The proper output voltages should then appear at J2.

## D/A System Function:

The D/A converter may also be accessed by use of the BIOS interrupt 1Ah. Subfunction 30h may be called to initiate a D/A conversion.

**Subfunction 30h:** Output to D/A converter

Inputs:

AL	30h
AH	DDh
BL	Analog channel (0 - 3)
CX	0 to FFFh (FFFh is +5V full-scale)

Outputs:

None







## Ethernet

---

The 1190OPT20 option adds IEEE 802 Ethernet capabilities to the Micro/sys SBC1190 Computer Board. As always, Micro/sys products provide ease of use, with particular attention to flexibility for use in embedded systems. For information on configuration and programming of the SBC1190 ethernet option and the Embedded Netsock system, please refer to the separate documentation that accompanies 1190OPT20.





---

## □ Troubleshooting

---

### **Red LED blinks twice then repeats**

**Problem:** The SBC1190 has attempted to start but has not found a valid EXE in flash.

**Solution:** Use the BIOS Boss to load an EXE file to the SBC1190.

### **Green LED is blinking**

**Problem:** The SBC1190 is continually resetting. Probable cause of this is poor power supply of +5V to the SBC1190.

**Solution:** Check that Vcc on the SBC1190 is a strong +5V and that ground also is properly connected.

**Problem:** If your application is loaded on the board and makes use of the watchdog timer, then your application may be causing the reset.

**Solution:** Try using a version of your application that does not make use of the watchdog timer or make sure that your application, after enabling the watchdog timer, strobes the watchdog frequently.

### **The BIOS Boss will not start**

**Problem:** The SBC1190 is not recognizing the download cable connected to COM B.

**Solution:** Verify that the 'LOAD' end of the CA4035 cable is connected to COM B of the SBC1190 via the CA4020 ribbon cable.

Also verify that the unlabeled end of the CA4035 cable is connected to a terminal running at 19200 BAUD, 'N', 8, 1.

**Problem:** The terminal is not receiving or displaying received characters properly to the screen.

**Solution:** Verify that your terminal (or terminal program) is working properly and will display characters received at 19200 BAUD, 'N', 8, 1.

**Turbo Debugger does not connect with the SBC3386EX in RUN.EXE mode.**

- Problem:** Possible version conflict. There are several versions of Turbo Debugger for which a matching version of the remote debug manager must be used.
- Solution:** See the README.DOC on the support diskette for Turbo Debugger version information then use the BIOS Boss to load the remote debug manager that is compatible with your version of Turbo Debugger.

**When attempting to load files to the SBC1190, the file transfer is aborted immediately.**

- Problem:** If using Procomm Plus the transfer is being cancelled by the terminal program.
- Solution:** Change *Setup/Protocol Options/General Options/Abort Xfer if CD lost* to "No".

**Problems loading a file or files to the SBC1190**

- Problem:** Possible Protocol conflict.
- Solution:** When loading an EXE in RUN.EXE mode, you must upload using the XMODEM protocol.

## **Appendices**

- Appendix A: Mapping*
- Appendix B: References*
- Appendix C: Major Device Data*  
*82C55 Parallel I/O*  
*DS1302 Clock/Calendar*  
*MAX197*  
*MAX525*  
*Intel 28F400 Flash Memory*  
*80C188EB Processor*
- Appendix D: SBC1190 Schematic Diagrams*



## Appendix A - Mapping

### SBC1190 Memory Map

There are a number of memory options available on the SBC1190. The following table indicates the memory maps that are automatically implemented by RUN.EXE firmware upon power-up.

Note: The location of a device installed in the U6 socket is user programmable by writing to chip select GCS0.

<b>SBC1190 Memory Maps</b>				
Address Range	128K RAM in U10	512K RAM in U10	256K Flash in U22	512K Flash in U22
00000-1FFFF				
20000-3FFFF				
40000-5FFFF				
60000-7FFFF				
80000-9FFFF				
A0000-BFFFF				
C0000-DFFFF			HIDDEN FLASH	HIDDEN FLASH
E0000-EFFFF			HIDDEN FLASH	
F0000-FFFFF				

## ***SBC1190 I/O Map***

The I/O map of the SBC1190 consists of the internal Peripheral Control Block (PCB) of the 80C188EB processor, as shown in the appendix. The PCB starts at port FF00h, so all port addresses shown have FF as a prefix (i.e. Port 1 Direction Register is at port FF50h). The 82C55 parallel I/O device is addressed by GCS1, which is programmed to include ports FE40h through FE43h. The Analog-to-Digital Converter is located at FE44h and FE45h. The watchdog timer is strobed by a write to port FE4Ch. See the following table for a complete list of I/O addresses for the SBC1190.

## SBC1190 I/O MAP

Port Address	Function
0000 - 7FFF	Offboard I/O
82C55	
FE40	DATA A
FE41	DATA B
FE42	DATA C
FE43	CONTROL
Analog	
FE44 (write)	ADC Control
FE44 (read)	ADC Data D0 - D7
FE45 (read)	ADC Data D8 - D11
Miscellaneous	
FE48	Altmap
FE4C	Watchdog
80C188EB	
FF02	EOI
FF04	POLL
FF06	POLLSTS
FF08	IMASK
FF0A	PRIMSK
FF0C	INSERV
FF0E	REQST
FF10	INSTS
FF12	TCUCON

Port Address	Function
FF14	SCUCON
FF16	I4CON
FF18	I0CON
FF1A	I1CON
FF1C	I2CON
FF1E	I3CON
FF30	T0CNT
FF32	T0CMPA
FF34	T0CMPB
FF36	T0CON
FF38	T1CNT
FF3A	T1CMPA
FF3C	T1CMPB
FF3E	T1CON
FF40	T2CNT
FF42	T2CMPA
FF46	T2CON
FF50	P1DIR
FF52	P1PIN
FF54	P1CTRL
FF56	P1LTCH
FF58	P2DIR
FF5A	P2PIN
FF5C	P2CTRL
FF5E	P2LTCH
FF60	B0CMP
FF62	B0CNT
FF64	S0CON
FF66	S0STS
FF68	S0RBUF
FF6A	S0TBUF
FF70	B1CMP

Port Address	Function
FF72	B1CNT
FF74	S1CON
FF76	S1STS
FF78	S1RBUF
FF7A	S1TBUF
FF80	GCS0ST
FF82	GCS0SP
FF84	GCS1ST
FF86	GCS1SP
FF88	GCS2ST
FF8A	GCS2SP
FF8C	GCS3ST
FF8E	GCS3SP
FF90	GCS4ST
FF92	GCS4SP
FF94	GCS5ST
FF96	GCS5SP
FF98	GCS6ST
FF9A	GCS6SP
FF9C	GCS7ST
FF9E	GCS7SP
FFA0	LCSST
FFA2	LCSSP
FFA4	UCSST
FFA6	UCSSP
FFA8	RELREG
FFB0	RFBASE
FFB2	RFTIME
FFB4	RFCON
FFB6	RFADDR
FFB8	PWRCON
FFBC	STEPID





## ***Appendix B - References***

- Intel Corp., Intel 80C186EB/80C188EB Microprocessor User's Manual, Order No. 270830-003 (or latest revision), Mt. Prospect, IL
- Intel Corp., Intel Flash Memory, Order No. 210830-013 (or latest revision), Mt. Prospect, IL
- Dettman and Kyle, DOS Programmer's Reference, Que Corp., Carmel, IN
- Edward Solari, AT Bus Design. Annalsbooks. San Diego, CA
- Frank Van Gilluwe, The Undocumented PC, Addison-Wesley



## ***Appendix C - Major Device Data***

82C55 Parallel I/O

DS1302 Clock/Calendar

MAX197 12-Bit DAS

MAX525 12-Bit D/A Converter

28F400 Flash Memory

Intel 80C188EB Microprocessor



## **82C55 Parallel I/O Interface**



# 82C55A

## Functional Description

### Data Bus Buffer

This three-state bi-directional 8 bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of Input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

### Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control buses and in turn, issues commands to both of the Control Groups.

**(CS)** Chip Select. A "low" on this input pin enables the communication between the 82C55A and the CPU.

**(RD)** Read. A "low" on this input pin enables the 82C55A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 82C55A.

**(WR)** Write. A "low" on this input pin enables the CPU to write data or control words into the 82C55A.

**(A0 and A1)** Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

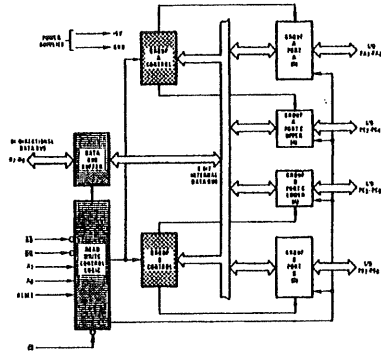


FIGURE 1. 82C55A BLOCK DIAGRAM. DATA BUS BUFFER, READ/WRITE, GROUP A & B CONTROL LOGIC FUNCTIONS

### 82C55A BASIC OPERATION

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
1	1	0	1	0	Control Word → Data Bus
OUTPUT OPERATION (WRITE)					
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
DISABLE FUNCTION					
X	X	X	X	1	Data Bus → Three-State
X	X	1	1	0	Data Bus → Three-State

**(RESET)** Reset. A "high" on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode. "Bus hold" devices internal to the 82C55A will hold the I/O port inputs to a logic "1" state with a maximum hold current of 400µA.

### Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 82C55A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the Internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7 - C4)

Control Group B - Port B and Port C lower (C3 - C0)

The control word register can be both written and read as shown in the "Basic Operation" table. Figure 4 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic "1", as this implies control word mode information.

**Ports A, B and C**

The 82C55A contains three 8 bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 82C55A.

**Port A** One 8 bit data output latch/buffer and one 8 bit data input latch. Both "pull-up" and "pull-down" bus-hold devices are present on Port A. See Figure 2a.

**Port B** One 8 bit data input/output latch/buffer and one 8 bit data input buffer. See Figure 2b.

**Port C** One 8 bit data output latch/buffer and one 8 bit data input buffer (no latch for input). This port can be divided into two 4 bit ports under the mode control. Each 4 bit port contains a 4 bit latch and it can be used for the control signal outputs with ports A and B. See Figure 2b.

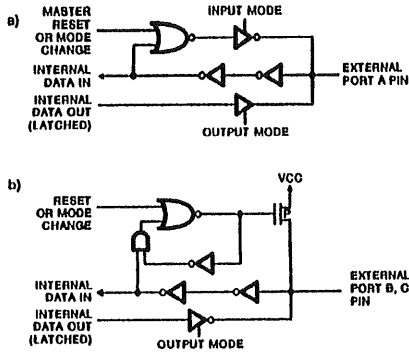


FIGURE 2. PORT A, PORT C BUS-HOLD CONFIGURATION

**Operational Description**

**Mode Selection**

There are three basic modes of operation that can be selected by the system software:

- Mode 0 - Basic Input/Output
- Mode 1 - Strobed Input/Output
- Mode 2 - Bi-directional Bus

When the reset input goes "high", all ports will be set to the input mode with all 24 port lines held at a logic "one" level by internal bus hold devices. After the reset is removed, the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need to pullup or pulldown resistors in all-CMOS designs. The control word register will contain 9Bh. During the execution of the system program, any of the other modes may be selected using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine. Any port programmed as an output port is initialized to all zeros when the control word is written.

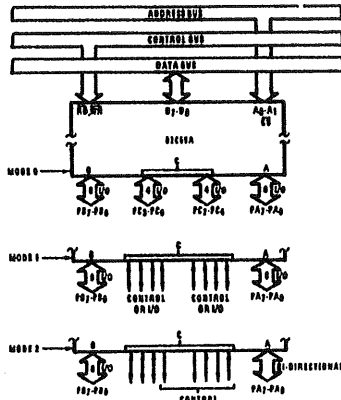


FIGURE 3. BASIC MODE DEFINITIONS AND BUS INTERFACE

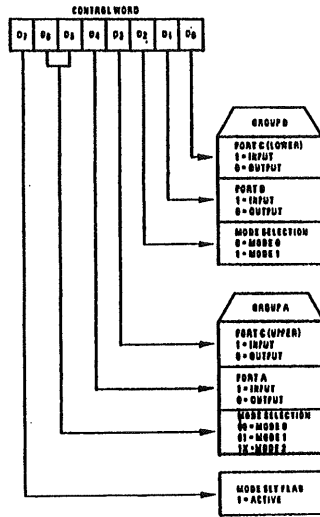


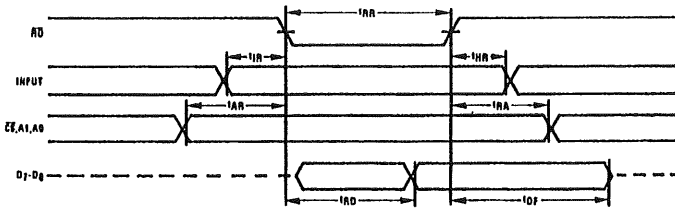
FIGURE 4. MODE DEFINITION FORMAT



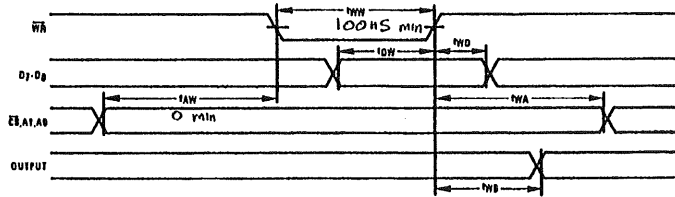


# 82C55A

## MODE 0 (Basic Input)

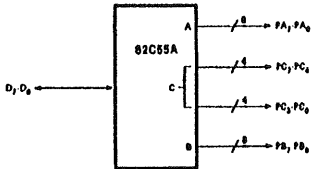
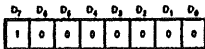


## MODE 0 (Basic Output)

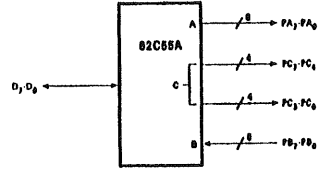
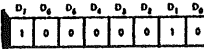


## MODE 0 CONFIGURATIONS

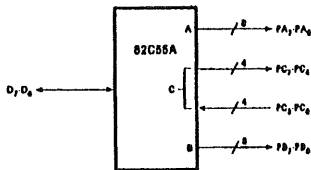
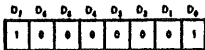
### CONTROL WORD #0



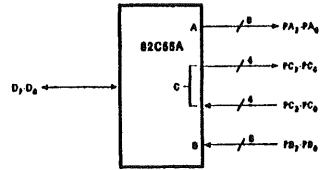
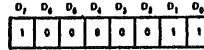
### CONTROL WORD #2



### CONTROL WORD #1



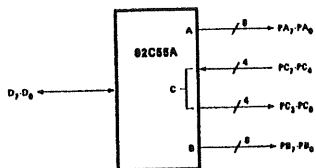
### CONTROL WORD #3



# 82C55A

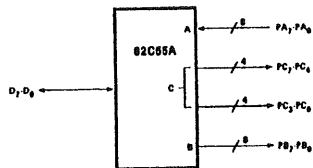
CONTROL WORD #4

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	0	1	0	0	0



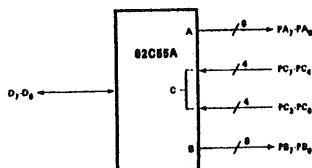
CONTROL WORD #8

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	0	0	0	0



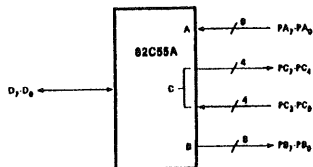
CONTROL WORD #6

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	0	1	0	0	1



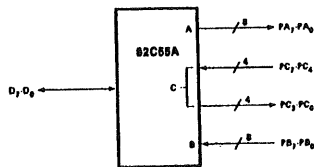
CONTROL WORD #9

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	0	0	0	1



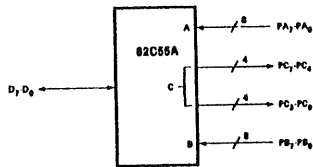
CONTROL WORD #8

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	0	1	0	1	0



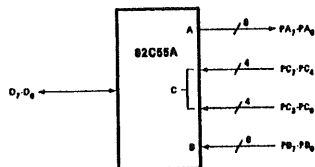
CONTROL WORD #10

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	0	0	1	0



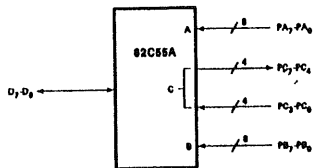
CONTROL WORD #7

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	0	1	0	1	1



CONTROL WORD #11

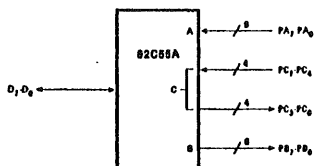
D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	0	0	1	1



## 82C55A

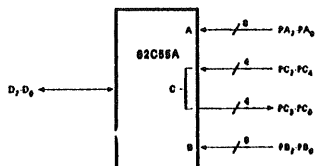
CONTROL WORD #13

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	1	0	0	0



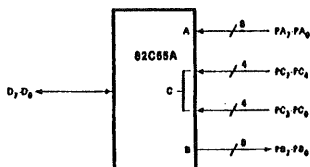
CONTROL WORD #14

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	1	0	1	0



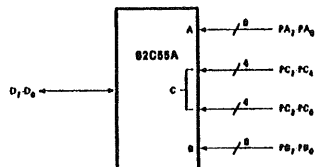
CONTROL WORD #15

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	1	0	0	1



CONTROL WORD #16

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	0	0	1	1	0	1	1



### Operating Modes

**Mode 1 (Strobed Input/Output).** This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "hand shaking" signals. In mode 1, port A and port B use the lines on port C to generate or accept these "hand shaking" signals.

#### Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8 bit port and one 4 bit control/data port.
- The 8 bit data port can be either input or output. Both inputs and outputs are latched.
- The 4 bit port is used for control and status of the 8 bit port.

**Input Control Signal Definition**  
(Figures 6 and 7)

**STB (Strobe Input)**

A "low" on this input loads data into the input latch.

**IBF (Input Buffer Full F/F)**

A "high" on this output indicates that the data has been loaded into the input latch. In essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

**INTR (Interrupt Request)**

A "high" on this output can be used to interrupt the CPU when and input device is requesting service. INTR is set by the condition: STB is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of RD. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

**INTE A**

Controlled by bit set/reset of PC4.

**INTE B**

Controlled by bit set/reset of PC2.

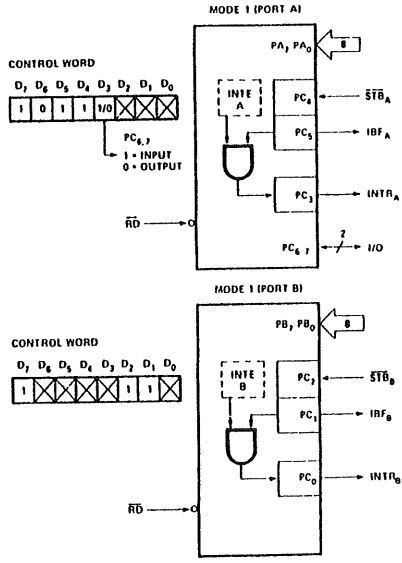


FIGURE 6. MODE 1 INPUT

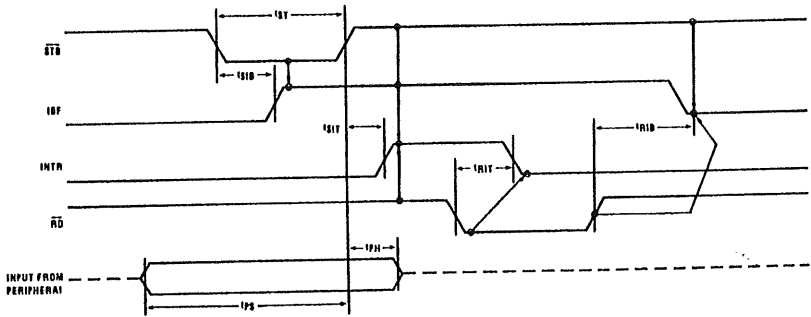


FIGURE 7. MODE 1 (STROBED INPUT)

**Output Control Signal Definition**  
(Figures 8 and 9)

**OBF** (Output Buffer Full F/F). The  $\overline{\text{OBF}}$  output will go "low" to indicate that the CPU has written data out to the specified port. This does not mean valid data is sent out of the port at this time since  $\overline{\text{OBF}}$  can go true before data is available. Data is guaranteed valid at the rising edge of  $\overline{\text{OBF}}$ . See Note 1. The  $\overline{\text{OBF}}$  F/F will be set by the rising edge of the  $\overline{\text{WR}}$  input and reset by  $\overline{\text{ACK}}$  input being low.

**ACK** (Acknowledge Input). A "low" on this input informs the 82C55A that the data from Port A or Port B is ready to be accepted. In essence, a response from the peripheral device indicating that it is ready to accept data. See Note 1.

**INTR** (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU.  $\overline{\text{INTR}}$  is set when  $\overline{\text{ACK}}$  is a "one",  $\overline{\text{OBF}}$  is a "one" and  $\overline{\text{INTE}}$  is a "one". It is reset by the falling edge of  $\overline{\text{WR}}$ .

**INTE A**

Controlled by Bit Set/Reset of  $\text{PC}_6$ .

**INTE B**

Controlled by Bit Set/Reset of  $\text{PC}_2$ .

**NOTE: 1.** To strobe data into the peripheral device, the user must operate the strobe line in a hand shaking mode. The user needs to send  $\overline{\text{OBF}}$  to the peripheral device, generate an  $\overline{\text{ACK}}$  from the peripheral device and then latch data into the peripheral device on the rising edge of  $\overline{\text{OBF}}$ .

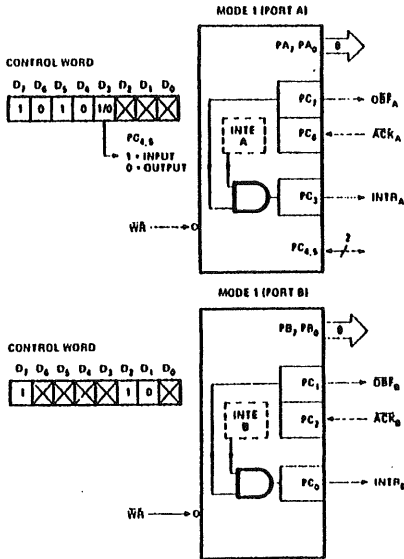


FIGURE 8. MODE 1 OUTPUT

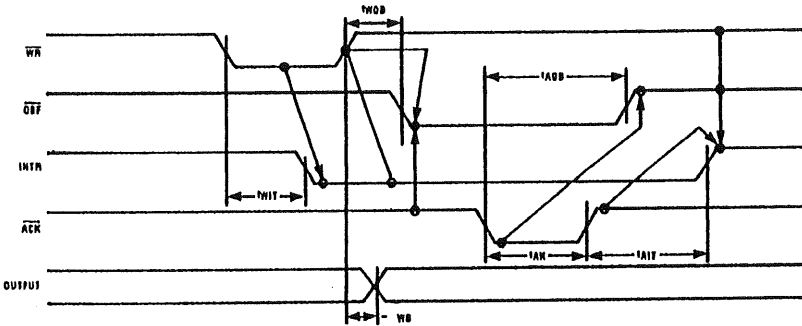


FIGURE 9. MODE 1 (STROBED OUTPUT)

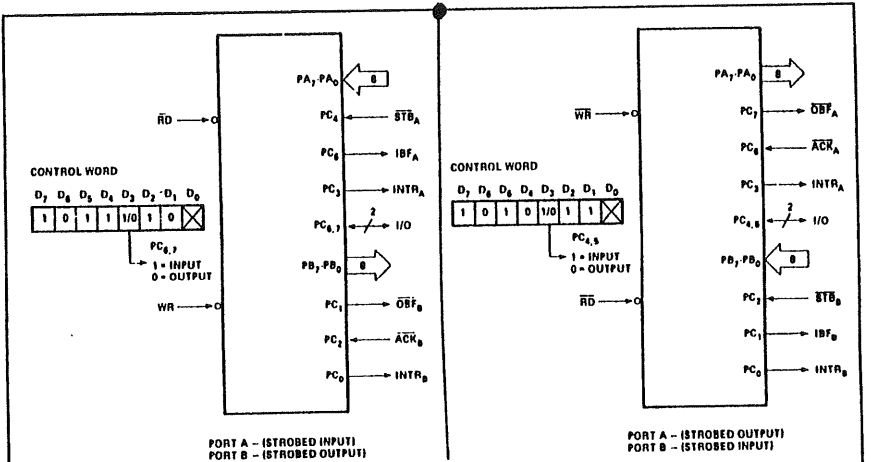


FIGURE 10. COMBINATIONS OF MODE 1

Combinations of Mode 1: Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

### Operating Modes

#### Mode 2 (Strobed Bi-directional Bus I/O)

The functional configuration provides a means for communicating with a peripheral device or structure on a single 8 bit bus for both transmitting and receiving data (bi-directional bus I/O). "Hand shaking" signals are provided to maintain proper bus flow discipline similar to Mode 1. Interrupt generation and enable/disable functions are also available.

#### Mode 2 Basic Functional Definitions:

- Used in Group A only
- One 8 bit, bi-directional bus Port (Port A) and a 5 bit control Port (Port C)
- Both inputs and outputs are latched
- The 5 bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A)

#### Bi-directional Bus I/O Control Signal Definition (Figures 11, 12, 13, 14)

**INTR** (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

#### Output Operations

**OBF** (Output Buffer Full). The  $\overline{\text{OBF}}$  output will go "low" to indicate that the CPU has written data out to port A.

**ACK** (Acknowledge). A "low" on this input enables the three-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

**INTE 1** (The INTE flip-flop associated with  $\overline{\text{OBF}}$ ). Controlled by bit set/reset of PC4.

#### Input Operations

**STB** (Strobe Input). A "low" on this input loads data into the input latch.

**IBF** (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

**INTE 2** (The INTE flip-flop associated with IBF). Controlled by bit set/reset of PC4.

82C55A

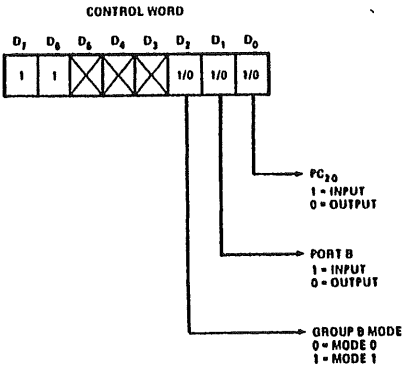


FIGURE 11. MODE CONTROL WORD

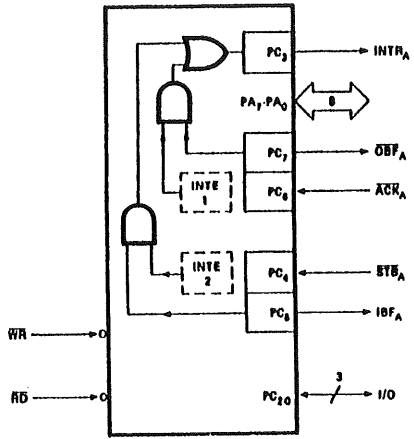


FIGURE 12. MODE 2

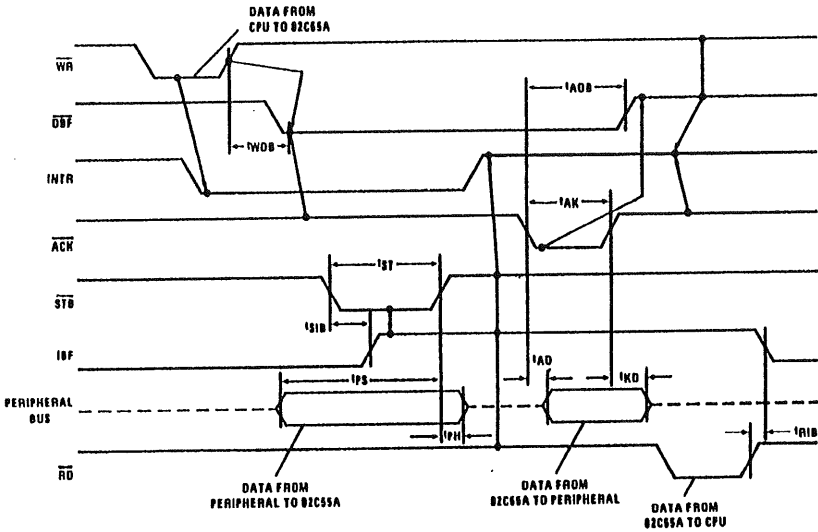
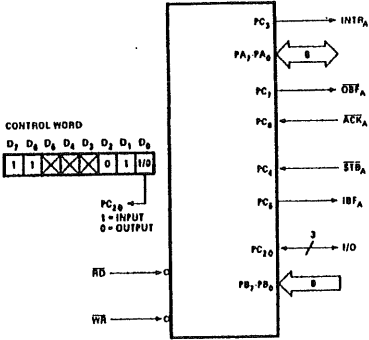


FIGURE 13. MODE 2 (BI-DIRECTIONAL)

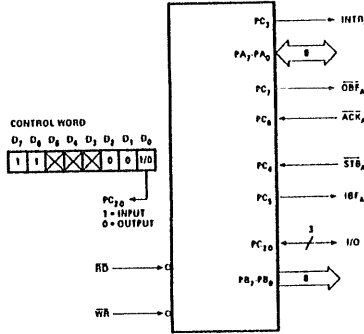
NOTE: Any sequence where  $\overline{WR}$  occurs before  $\overline{ACK}$  and  $\overline{STB}$  occurs before  $\overline{RD}$  is permissible.  
 $(\overline{INTR} = \overline{IBF} \cdot \overline{MASK} \cdot \overline{STB} \cdot \overline{RD} \cdot \overline{OBF} \cdot \overline{MASK} \cdot \overline{ACK} \cdot \overline{WR})$



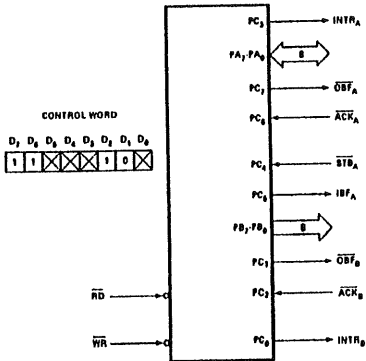
MODE 2 AND MODE 0 (INPUT)



MODE 2 AND MODE 0 (OUTPUT)



MODE 2 AND MODE 1 (OUTPUT)



MODE 2 AND MODE 1 (INPUT)

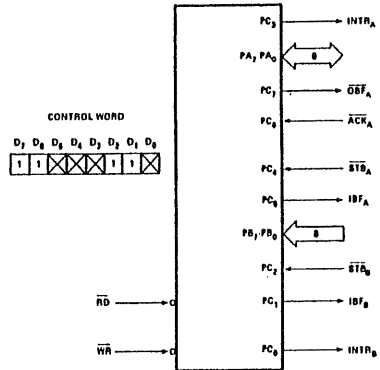


FIGURE 14. MODE 2 COMBINATIONS

MODE DEFINITION SUMMARY					
	MODE 0		MODE 1		MODE 2
	IN	OUT	IN	OUT	GROUP A ONLY
PA0	In	Out	In	Out	←→
PA1	In	Out	In	Out	
PA2	In	Out	In	Out	
PA3	In	Out	In	Out	
PA4	In	Out	In	Out	
PA5	In	Out	In	Out	
PA6	In	Out	In	Out	
PA7	In	Out	In	Out	
PB0	In	Out	In	Out	}
PB1	In	Out	In	Out	
PB2	In	Out	In	Out	
PB3	In	Out	In	Out	
PB4	In	Out	In	Out	
PB5	In	Out	In	Out	
PB6	In	Out	In	Out	
PB7	In	Out	In	Out	
PC0	In	Out	INTR <sub>B</sub>	INTR <sub>B</sub>	I/O
PC1	In	Out	IBF <sub>B</sub>	OBF <sub>B</sub>	I/O
PC2	In	Out	STB <sub>B</sub>	ACK <sub>B</sub>	I/O
PC3	In	Out	INTR <sub>A</sub>	INTR <sub>A</sub>	INTR <sub>A</sub>
PC4	In	Out	STB <sub>A</sub>	I/O	STB <sub>A</sub>
PC5	In	Out	IBF <sub>A</sub>	I/O	IBF <sub>A</sub>
PC6	In	Out	I/O	ACK <sub>A</sub>	ACK <sub>A</sub>
PC7	In	Out	I/O	OBF <sub>A</sub>	OBF <sub>A</sub>

Mode 0  
Or Mode 1  
Only

**Special Mode Combination Considerations:**

There are several combinations of modes possible. For any combination, some or all of Port C lines are used for control or status. The remaining bits are either inputs or outputs as defined by a "Set Mode" command.

During a read of Port C, the state of all the Port C lines, except the ACK and STB lines, will be placed on the data bus. In place of the ACK and STB line states, flag status will appear on the data bus in the PC2, PC4, and PC6 bit positions as illustrated by Figure 17.

Through a "Write Port C" command, only the Port C pins programmed as outputs in a Mode 0 group can be written. No other pins can be affected by a "Write Port C" command, nor can the interrupt enable flags be accessed. To write to any Port C output programmed as an output in a Mode 1 group or to change an interrupt enable flag, the "Set/Reset Port C Bit" command must be used.

With a "Set/Reset Port C Bit" command, any Port C line programmed as an output (including IBF and OBF) can be written, or an interrupt enable flag can be either set or reset. Port C lines programmed as inputs, including ACK and STB lines, associated with Port C are not affected by a "Set/Reset Port C Bit" command. Writing to the corresponding Port C bit positions of the ACK and STB lines with the "Set/Reset Port C Bit" command will affect the Group A and Group B interrupt enable flags, as illustrated in Figure 17.

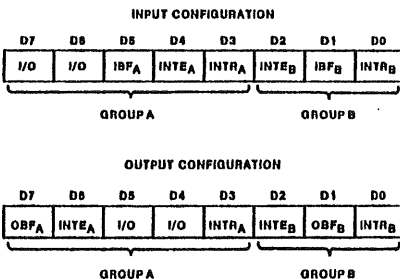


FIGURE 15. MODE 1 STATUS WORD FORMAT

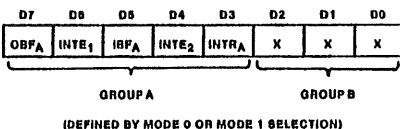


FIGURE 16. MODE 2 STATUS WORD FORMAT

## 82C55A

### Current Drive Capability:

Any output on Port A, B or C can sink or source 2.5mA. This feature allows the 82C55A to directly drive Darlington type drivers and high-voltage displays that require such sink or source current.

### Reading Port C Status (Figures 15 and 16)

In Mode 0, Port C transfers data to or from the peripheral device. When the 82C55A is programmed to function in Modes 1 or 2, Port C generates or accepts "hand shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

INTERRUPT ENABLE FLAG*	POSITION	ALTERNATE PORT C PIN SIGNAL (MODE)
INTE B	PC2	ACK <sub>B</sub> (Output Mode 1) or STB <sub>B</sub> (Input Mode 1)
INTE A2	PC4	STB <sub>A</sub> (Input Mode 1 or Mode 2)
INTE A1	PC6	ACK <sub>A</sub> (Output Mode 1 or Mode 2)

FIGURE 17. INTERRUPT ENABLE FLAGS IN MODES 1 AND 2

### Applications of the 82C55A

The 82C55A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 82C55A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 82C55A to exactly "fit" the application. Figures 18 through 24 present a few examples of typical applications of the 82C55A.

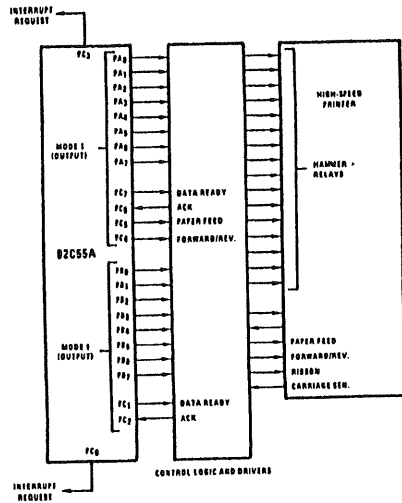


FIGURE 18. PRINTER INTERFACE

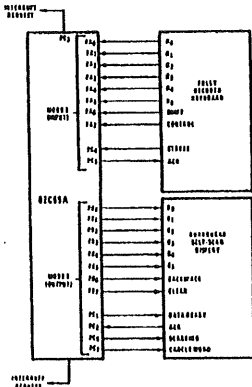


FIGURE 19. KEYBOARD AND DISPLAY INTERFACE

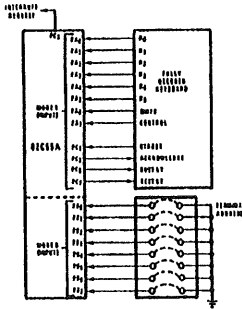


FIGURE 20. KEYBOARD AND TERMINAL ADDRESS INTERFACE

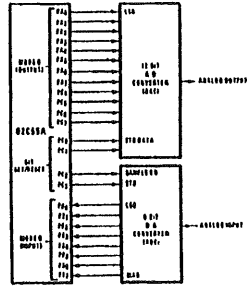


FIGURE 21. DIGITAL TO ANALOG, ANALOG TO DIGITAL

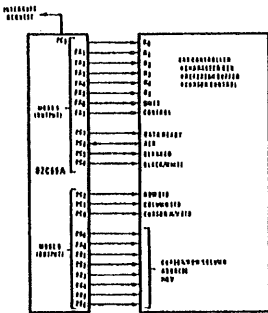


FIGURE 22. BASIC CRT CONTROLLER INTERFACE

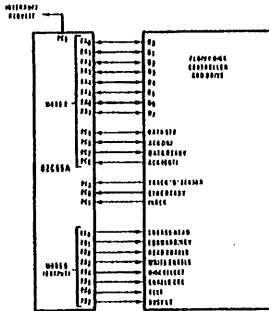


FIGURE 23. BASIC FLOPPY DISC INTERFACE

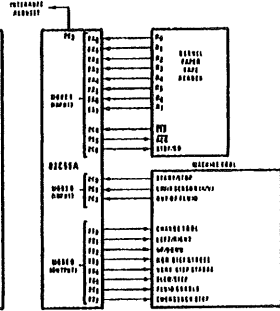


FIGURE 24. MACHINE TOOL CONTROLLER INTERFACE

**Dallas DS1302**



# DALLAS

SEMICONDUCTOR

## DS1302

### Trickle Charge Timekeeping Chip

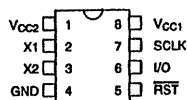
#### FEATURES

- Real time clock counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap year compensation valid up to 2100
- 31 x 8 RAM for scratchpad data storage
- Serial I/O for minimum pin count
- 2.0–5.5 volt full operation
- Uses less than 300 nA at 2.0 volts
- Single–byte or multiple–byte (burst mode) data transfer for read or write of clock or RAM data
- 8–pin DIP or optional 8–pin SOIC's for surface mount
- Simple 3–wire interface
- TTL–compatible ( $V_{CC} = 5V$ )
- Optional industrial temperature range  $-40^{\circ}C$  to  $+85^{\circ}C$
- DS1202 compatible
- Added features over DS1202
  - Optional trickle charge capability to  $V_{CC1}$
  - Dual power supply pins for primary and backup power supplies
  - Backup power supply pin can be used for battery or super cap input
  - Additional scratchpad memory (7 bytes)

#### DESCRIPTION

The DS1302 Trickle Charge Timekeeping Chip contains a real time clock/calendar and 31 bytes of static RAM. It communicates with a microprocessor via a simple serial interface. The real time clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24–hour or 12–hour format with an AM/PM indicator.

#### PIN ASSIGNMENT



DS1302  
8–PIN DIP (300 MIL)



DS1302S 8–PIN SOIC (200 MIL)  
DS1302Z 8–PIN SOIC (150 MIL)

#### PIN DESCRIPTION

X1, X2	– 32.768 kHz Crystal Pins
GND	– Ground
RST	– Reset
I/O	– Data Input/Output
SCLK	– Serial Clock
$V_{CC1}$ , $V_{CC2}$	– Power Supply Pins

#### ORDERING INFORMATION

PART #	DESCRIPTION
DS1302	Serial Timekeeping Chip; 8–pin DIP
DS1302S	Serial Timekeeping Chip; 8–pin SOIC (200 mil)
DS1302Z	Serial Timekeeping Chip; 8–pin SOIC (150 mil)

Interfacing the DS1302 with a microprocessor is simplified by using synchronous serial communication. Only three wires are required to communicate with the clock/RAM: (1) RST (Reset), (2) I/O (Data line), and (3) SCLK (Serial clock). Data can be transferred to and from the clock/RAM one byte at a time or in a burst of up to 31 bytes. The DS1302 is designed to operate on very low power and retain data and clock information on less than 1 microwatt.

The DS1302 is the successor to the DS1202. In addition to the basic timekeeping functions of the DS1202, the DS1302 has the additional features of dual power pins for primary and back-up power supplies, programmable trickle charger for  $V_{CC1}$ , and seven additional bytes of scratchpad memory.

After the first eight clock cycles have loaded the command word into the shift register, additional clocks will output data for a read or input data for a write. The number of clock pulses equals eight plus eight for byte mode or eight plus up to 248 for burst mode.

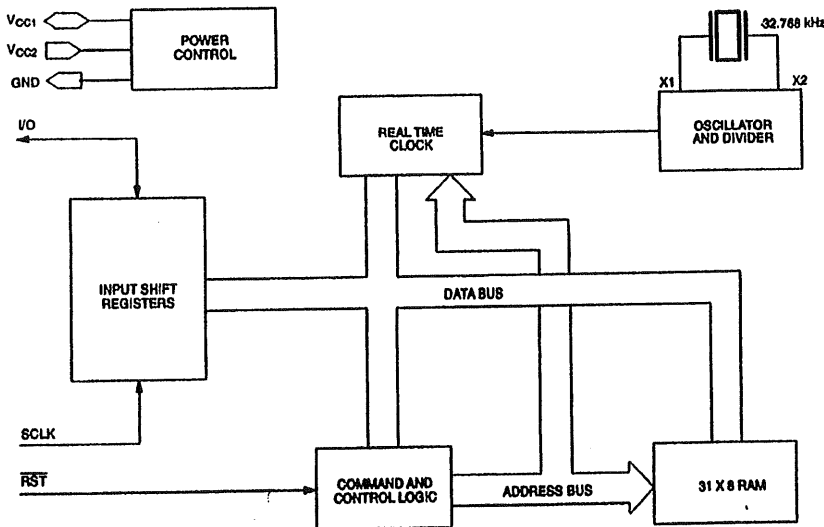
**OPERATION**

The main elements of the Serial Timekeeper are shown in Figure 1: shift register, control logic, oscillator, real time clock, and RAM. To initiate any transfer of data,  $\overline{RST}$  is taken high and eight bits are loaded into the shift register providing both address and command information. Data is serially input on the rising edge of the SCLK. The first eight bits specify which of 40 bytes will be accessed, whether a read or write cycle will take place, and whether a byte or burst mode transfer is to occur.

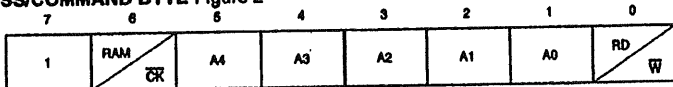
**COMMAND BYTE**

The command byte is shown in Figure 2. Each data transfer is initiated by a command byte. The MSB (Bit 7) must be a logic "1". If it is zero, writes to the DS1302 will be disabled. Bit 6 specifies clock/calendar data if logic "0" or RAM data if logic "1". Bits one through five specify the designated registers to be input or output, and the LSB (Bit 0) specifies a write operation (input) if logic "0" or read operation (output) if logic "1". The command byte is always input starting with the LSB (Bit 0).

**DS1302 BLOCK DIAGRAM Figure 1**



**ADDRESS/COMMAND BYTE Figure 2**





### RESET AND CLOCK CONTROL

All data transfers are initiated by driving the  $\overline{RST}$  input high. The  $\overline{RST}$  input serves two functions. First,  $\overline{RST}$  turns on the control logic which allows access to the shift register for the address/command sequence. Second, the  $\overline{RST}$  signal provides a method of terminating either single byte or multiple byte data transfer.

A clock cycle is a sequence of a falling edge followed by a rising edge. For data inputs, data must be valid during the rising edge of the clock and data bits are output on the falling edge of clock. If the  $\overline{RST}$  input is low all data transfer terminates and the I/O pin goes to a high impedance state. Data transfer is illustrated in Figure 3. At power-up,  $\overline{RST}$  must be a logic "0" until  $V_{CC} \cong 2.0$  volts. Also SCLK must be at a logic "0" when  $\overline{RST}$  is driven to a logic "1" state.

### DATA INPUT

Following the eight SCLK cycles that input a write command byte, a data byte is input on the rising edge of the next eight SCLK cycles. Additional SCLK cycles are ignored should they inadvertently occur. Data is input starting with bit 0.

### DATA OUTPUT

Following the eight SCLK cycles that input a read command byte, a data byte is output on the falling edge of the next eight SCLK cycles. Note that the first data bit to be transmitted occurs on the first falling edge after the last bit of the command byte is written. Additional SCLK cycles retransmit the data bytes should they inadvertently occur so long as  $\overline{RST}$  remains high. This operation permits continuous burst mode read capability. Also, the I/O pin is tri-stated upon each rising edge of SCLK. Data is output starting with bit 0.

### BURST MODE

Burst mode may be specified for either the clock/calendar or the RAM registers by addressing location 31 decimal (address/command bits one through five = logical one). As before, bit six specifies clock or RAM and bit 0 specifies read or write. There is no data storage capacity at locations 9 through 31 in the Clock/Calendar Registers or location 31 in the RAM registers. Reads or writes in burst mode start with bit 0 of address 0.

As in the case with the DS1202, when writing to the clock registers in the burst mode, the first eight registers must be written in order for the data to be transferred.

However, when writing to RAM in burst mode it is not necessary to write all 31 bytes for the data to transfer. Each byte that is written to will be transferred to RAM regardless of whether all 31 bytes are written or not.

### CLOCK/CALENDAR

The clock/calendar is contained in seven write/read registers as shown in Figure 4. Data contained in the clock/calendar registers is in binary coded decimal format (BCD).

### CLOCK HALT FLAG

Bit 7 of the seconds register is defined as the clock halt flag. When this bit is set to logic "1", the clock oscillator is stopped and the DS1302 is placed into a low-power standby mode with a current drain of less than 100 nanoamps. When this bit is written to logic "0", the clock will start. The initial power on state is not defined.

### AM-PM/12-24 MODE

Bit 7 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10 hour bit (20 - 23 hours).

### WRITE PROTECT BIT

Bit 7 of the control register is the write protect bit. The first seven bits (bits 0 - 6) are forced to zero and will always read a zero when read. Before any write operation to the clock or RAM, bit 7 must be zero. When high, the write protect bit prevents a write operation to any other register. The initial power on state is not defined. Therefore the WP bit should be cleared before attempting to write to the device.

### TRICKLE CHARGE REGISTER

This register controls the trickle charge characteristics of the DS1302. The simplified schematic of Figure 5 shows the basic components of the trickle charger. The trickle charge select (TCS) bits (bits 4 - 7) control the selection of the trickle charger. In order to prevent accidental enabling, only a pattern of 1010 will enable the trickle charger. All other patterns will disable the trickle charger. The DS1302 powers up with the trickle charger disabled. The diode select (DS) bits (bits 2 - 3) select whether one diode or two diodes are connected between  $V_{CC2}$  and  $V_{CC1}$ . If DS is 01, one diode is

selected or if DS is 10, two diodes are selected. If DS is 00 or 11, the trickle charger is disabled independent of TCS. The RS bits (bits 0 – 1) select the resistor that is connected between  $V_{CC2}$  and  $V_{CC1}$ . The resistor selected by the resistor select (RS) bits is as follows:

RS Bits	Resistor	Typical Value
00	None	None
01	R1	2K $\Omega$
10	R2	4K $\Omega$
11	R3	8K $\Omega$

If RS is 00, the trickle charger is disabled independent of TCS.

Diode and resistor selection is determined by the user according to the maximum current desired for battery or super cap charging. The maximum charging current can be calculated as illustrated in the following example. Assume that a system power supply of 5V is applied to  $V_{CC2}$  and a super cap is connected to  $V_{CC1}$ . Also assume that the trickle charger has been enabled with 1 diode and resistor R1 between  $V_{CC2}$  and  $V_{CC1}$ . The maximum current  $I_{max}$  would therefore be calculated as follows:

$$I_{max} = (5.0V - \text{diode drop}) / R1 \\ \sim (5.0V - 0.7V) / 2K\Omega \\ \sim 2.2 \text{ mA}$$

Obviously, as the super cap charges, the voltage drop between  $V_{CC2}$  and  $V_{CC1}$  will decrease and therefore the charge current will decrease.

#### CLOCK/CALENDAR BURST MODE

The clock/calendar command byte specifies burst mode operation. In this mode the first eight clock/calendar registers can be consecutively read or written (see Figure 4) starting with bit 0 of address 0.

If the write protect bit is set high when a write clock/calendar burst mode is specified, no data transfer will occur

to any of the eight clock/calendar registers (this includes the control register). The trickle charger is not accessible in burst mode.

#### RAM

The static RAM is 31 x 8 bytes addressed consecutively in the RAM address space.

#### RAM BURST MODE

The RAM command byte specifies burst mode operation. In this mode, the 31 RAM registers can be consecutively read or written (see Figure 4) starting with bit 0 of address 0.

#### REGISTER SUMMARY

A register data format summary is shown in Figure 4.

#### CRYSTAL SELECTION

A 32.768 kHz crystal can be directly connected to the DS1302 via pins 2 and 3 (X1, X2). The crystal selected for use should have a specified load capacitance (CL) of 6 pF. For more information on crystal selection and crystal layout consideration, please consult Application Note 58, "Crystal Considerations with Dallas Real Time Clocks".

#### POWER CONTROL

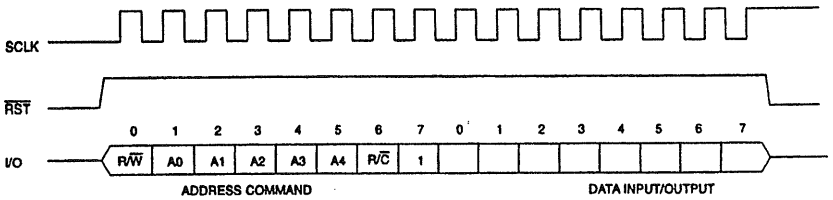
$V_{CC1}$  provides low power operation in single supply and battery operated systems as well as low power battery backup.

$V_{CC2}$  provides the primary power in dual supply systems where  $V_{CC1}$  is connected to a backup source to maintain the time and data in the absence of primary power.

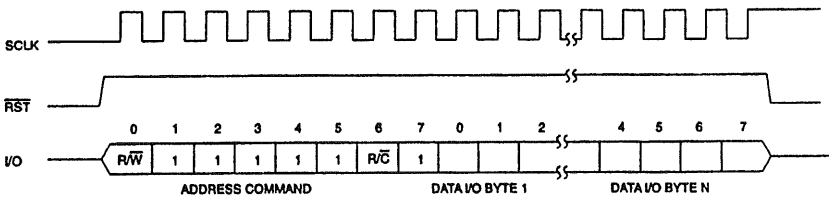
The DS1302 will operate from the larger of  $V_{CC1}$  or  $V_{CC2}$ . When  $V_{CC2}$  is greater than  $V_{CC1} + 0.2V$ ,  $V_{CC2}$  will power the DS1302. When  $V_{CC2}$  is less than  $V_{CC1}$ ,  $V_{CC1}$  will power the DS1302.

**DATA TRANSFER SUMMARY Figure 3**

**SINGLE BYTE TRANSFER**



**BURST MODE TRANSFER**

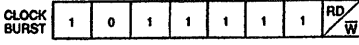
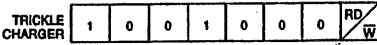
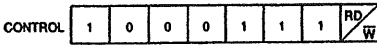
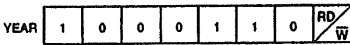
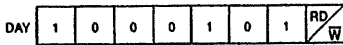
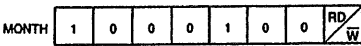
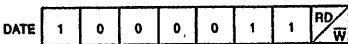
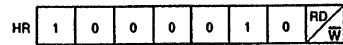
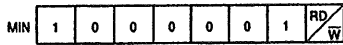
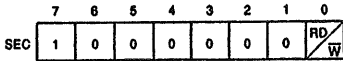


FUNCTION	BYTE N	SCLK n
CLOCK	8	72
RAM	31	258

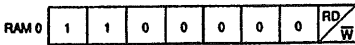
**REGISTER ADDRESS/DEFINITION Figure 4**

**REGISTER ADDRESS**

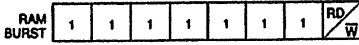
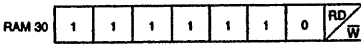
**A. CLOCK**



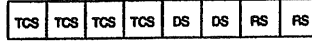
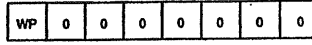
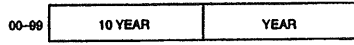
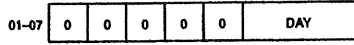
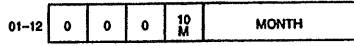
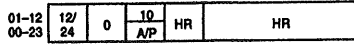
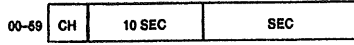
**B. RAM**



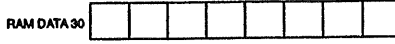
⋮



**REGISTER DEFINITION**



⋮



# Low-Power, Quad, 12-Bit Voltage-Output DAC with Serial Interface

**MAX525**

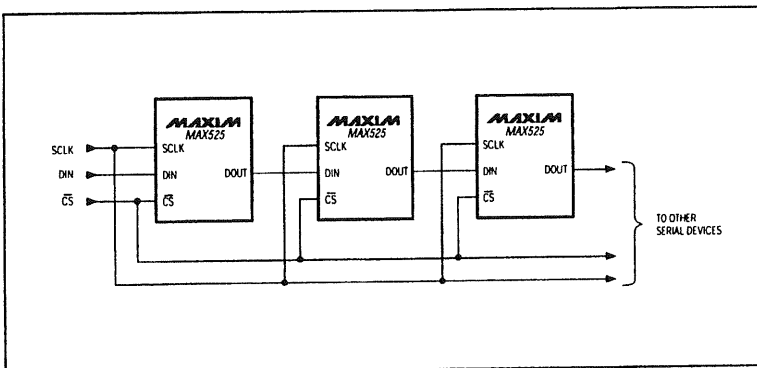


Figure 7. Daisy-Chaining MAX525s

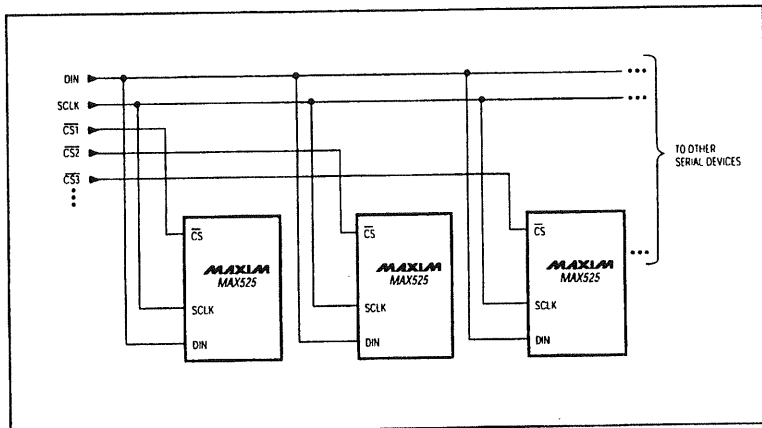


Figure 8. Multiple MAX525s Sharing a Common DIN Line

# Low-Power, Quad, 12-Bit Voltage-Output DAC with Serial Interface

**MAX525**

## Applications Information

### Unipolar Output

For a unipolar output, the output voltages and the reference inputs have the same polarity. Figure 9 shows the MAX525 unipolar output circuit, which is also the typical operating circuit. Table 2 lists the unipolar output codes.

For rail-to-rail outputs, see Figure 10. This circuit shows the MAX525 with the output amplifiers configured with a closed-loop gain of +2 to provide 0V to 5V full-scale range when a 2.5V reference is used.

**Table 2. Unipolar Code Table**

DAC CONTENTS		ANALOG OUTPUT
MSB	LSB	
1111	1111	$+V_{REF} \left( \frac{4095}{4096} \right)$
1000	0000	$+V_{REF} \left( \frac{2048}{4096} \right)$
1000	0000	$-V_{REF} \left( \frac{2048}{4096} \right) = \frac{+V_{REF}}{2}$
0111	1111	$+V_{REF} \left( \frac{2047}{4096} \right)$
0000	0000	$+V_{REF} \left( \frac{1}{4096} \right)$
0000	0000	0V

**Table 3. Bipolar Code Table**

DAC CONTENTS		ANALOG OUTPUT
MSB	LSB	
1111	1111	$+V_{REF} \left( \frac{2047}{2048} \right)$
1000	0000	$+V_{REF} \left( \frac{1}{2048} \right)$
1000	0000	0V
0111	1111	$-V_{REF} \left( \frac{1}{2048} \right)$
0000	0000	$-V_{REF} \left( \frac{2047}{2048} \right)$
0000	0000	$-V_{REF} \left( \frac{2048}{2048} \right) = -V_{REF}$

Note: 1LSB =  $(V_{REF}) \left( \frac{1}{4096} \right)$

### Bipolar Output

The MAX525 outputs can be configured for bipolar operation using Figure 11's circuit.

$$V_{OUT} = V_{REF} [(2NB / 4096) - 1]$$

where NB is the numeric value of the DAC's binary input code. Table 3 shows digital codes (offset binary) and corresponding output voltages for Figure 11's circuit.

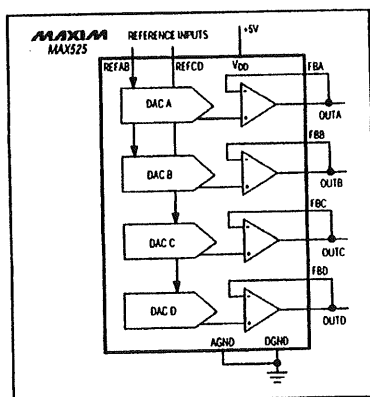
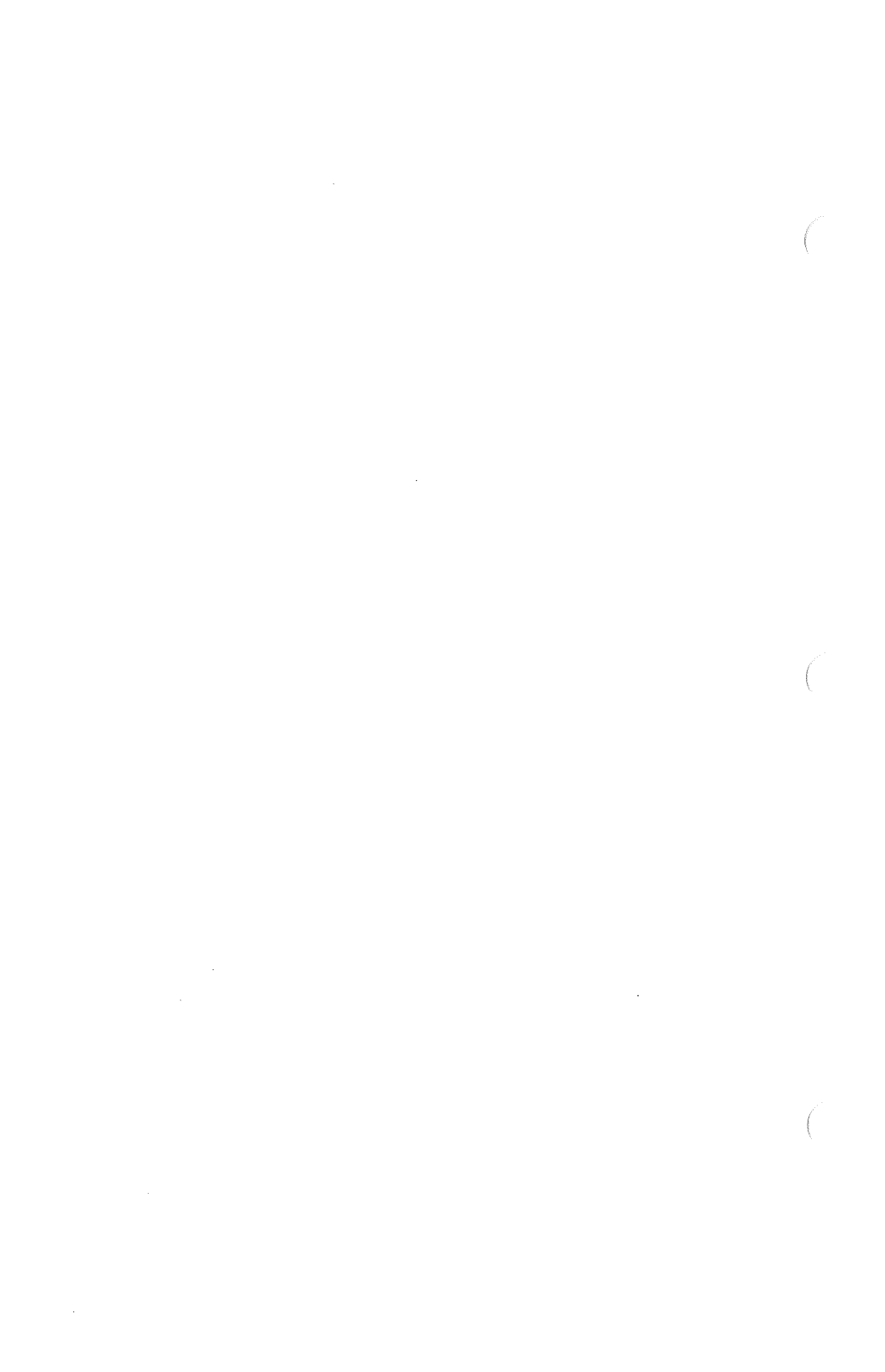


Figure 9. Unipolar Output Circuit

## **28F400 Flash Memory**





## 2.1 A28F400BX Memory Organization

### 2.1.1 BLOCKING

The A28F400BX uses a blocked array architecture to provide independent erasure of memory blocks. A block is erased independently of other blocks in the array when an address is given within the block address range and the Erase Setup and Erase Confirm commands are written to the CUI. The A28F400BX is a random read/write memory, only erasure is performed by block.

#### 2.1.1.1 Boot Block Operation and Data Protection

The 16-Kbyte boot block provides a lock feature for secure code storage. The intent of the boot block is to provide a secure storage area for the kernel code that is required to boot a system in the event of power failure or other disruption during code update. This lock feature ensures absolute data integrity by preventing the boot block from being written or erased when RP# is not at 12V. The boot block can be erased and written when RP# is held at 12V for the duration of the erase or program operation. This allows customers to change the boot code when necessary while providing security when needed. See the Block Memory Map section for address locations of the boot block for the A28F400BX-T and A28F400BX-B.

#### 2.1.1.2 Parameter Block Operation

The A28F400BX has 2 parameter blocks (8 Kbytes each). The parameter blocks are intended to provide storage for frequently updated system parameters and configuration or diagnostic information. The parameter blocks can also be used to store additional boot or main code. The parameter blocks however, do not have the hardware write protection feature that the boot block has. The parameter blocks provide for more efficient memory utilization when dealing with parameter changes versus regularly blocked devices. See the Block Memory Map section for address locations of the parameter blocks for the A28F400BX-T and A28F400BX-B.

#### 2.1.1.3 Main Block Operation

Four main blocks of memory exist on the A28F400BX (3 x 128 Kbyte blocks and 1 x 96-Kbyte blocks). See the following section on Block Memory Map for the address location of these blocks for the A28F400BX-T and A28F400BX-B products.

### 2.1.2.2 A28F400BX-T Memory Map

The A28F400BX-T device has the 16-Kbyte boot block located from 3E000H to 3FFFFH to accommodate those microprocessors that boot from the top of the address map. In the A28F400BX-T the first 8-Kbyte parameter block resides in memory space from 3D000H to 3DFFFH. The second 8-Kbyte parameter block resides in memory space from 3C000H to 3CFFFH. The 96-Kbyte main block resides in memory space from 30000H to 3BFFFH. The three 128-Kbyte main blocks reside in memory space from 20000H to 2FFFFH, 10000H to 1FFFFH and 00000H to 0FFFFH as shown below in Figure 5.

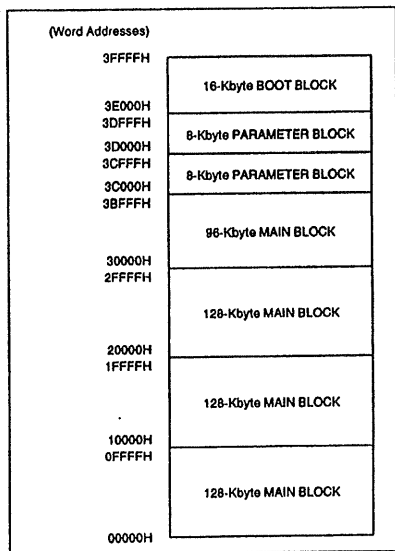


Figure 5. A28F400BX-T Memory Map

## 3.0 PRODUCT FAMILY PRINCIPLES OF OPERATION

Flash memory augments EPROM functionality with in-circuit electrical write and erase. The 4-Mbit flash family utilizes a Command User Interface (CUI) and internally generated and timed algorithms to simplify write and erase operations.

The CUI allows for 100% TTL-level control inputs, fixed power supplies during erasure and programming, and maximum EPROM compatibility.

In the absence of high voltage on the  $V_{PP}$  pin, the 4-Mbit boot block flash family will only successfully execute the following commands: Read Array, Read Status Register, Clear Status Register and Intelligent Identifier mode. The device provides standard EPROM read, standby and output disable operations. Manufacturer Identification and Device Identification data can be accessed through the CUI or through the standard EPROM  $A_9$  high voltage access ( $V_{ID}$ ) for PROM programming equipment.

The same EPROM read, standby and output disable functions are available when high voltage is applied to the  $V_{pp}$  pin. In addition, high voltage on  $V_{pp}$  allows write and erase of the device. All functions associated with altering memory contents: write and erase, Intelligent Identifier read and Read Status are accessed via the CUI.

The purpose of the Write State Machine (WSM) is to completely automate the write and erasure of the device. The WSM will begin operation upon receipt of a signal from the CUI and will report status back through a Status Register. The CUI will handle the  $WE\#$  interface to the data and address latches, as well as system software requests for status while the WSM is in operation.

### 3.1 Bus Operations

Flash memory reads, erases and writes in-system via the local CPU. All bus cycles to or from the flash memory conform to standard microprocessor bus cycles.

**Table 1. Bus Operations for WORD-WIDE Mode (BYTE# = V<sub>IH</sub>)**

Mode	Notes	RP#	CE#	OE#	WE#	A <sub>9</sub>	A <sub>0</sub>	V <sub>PP</sub>	DQ <sub>0-15</sub>
Read	1, 2, 3	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IL</sub>	V <sub>IH</sub>	X	X	X	D <sub>OUT</sub>
Output Disable		V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>IH</sub>	X	X	X	High Z
Standby		V <sub>IH</sub>	V <sub>IH</sub>	X	X <sub>I</sub>	X	X	X	High Z
Deep Power-Down	9	V <sub>IL</sub>	X	X	X	X	X	X	High Z
Intelligent Identifier (Mfr)	4	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>ID</sub>	V <sub>IL</sub>	X	0089H
Intelligent Identifier (Device)	4, 5	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>ID</sub>	V <sub>IH</sub>	X	4470H 4471H
Write	6, 7, 8	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>IL</sub>	X	X	X	D <sub>IN</sub>

**Table 2. Bus Operations for BYTE-WIDE Mode (BYTE = V<sub>IL</sub>)**

Mode	Notes	RP#	CE#	OE#	WE#	A <sub>9</sub>	A <sub>0</sub>	A <sub>-1</sub>	V <sub>PP</sub>	DQ <sub>0-7</sub>	DQ <sub>8-14</sub>
Read	1, 2, 3	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IL</sub>	V <sub>IH</sub>	X	X	X	X	D <sub>OUT</sub>	High Z
Output Disable		V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>IH</sub>	X	X	X	X	High Z	High Z
Standby		V <sub>IH</sub>	V <sub>IH</sub>	X	X	X	X	X	X	High Z	High Z
Deep Power-Down	9	V <sub>IL</sub>	X	X	X	X	X	X	X	High Z	High Z
Intelligent Identifier (Mfr)	4	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>ID</sub>	V <sub>IL</sub>	X	X	89H	High Z
Intelligent Identifier (Device)	4, 5	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>ID</sub>	V <sub>IH</sub>	X	X	70H 71H	High Z
Write	6, 7, 8	V <sub>IH</sub>	V <sub>IL</sub>	V <sub>IH</sub>	V <sub>IL</sub>	X	X	X	X	D <sub>IN</sub>	High Z

**NOTES:**

1. Refer to DC Characteristics.
2. X can be V<sub>IL</sub>, V<sub>IH</sub> for control pins and addresses, V<sub>PPL</sub> or V<sub>PPH</sub> for V<sub>PP</sub>.
3. See DC Characteristics for V<sub>PPL</sub>, V<sub>PPH</sub>, V<sub>IH</sub>, V<sub>ID</sub> voltages.
4. Manufacturer and Device codes may also be accessed via a CUI write sequence. A<sub>1</sub>-A<sub>17</sub> = X.
5. Device ID = 4470H for A28F400BX-T and 4471H for A28F400BX-B.
6. Refer to Table 3 for valid D<sub>IN</sub> during a write operation.
7. Command writes for Block Erase or Word/Byte Write are only executed when V<sub>PP</sub> = V<sub>PPH</sub>.
8. To write or erase the boot block, hold RP# at V<sub>IH</sub>.
9. RP# must be at GND ±0.2V to meet the 80 μA maximum deep power-down current.

### 3.2 Read Operations

The 4-Mbit boot block flash family has three user read modes; Array, Intelligent Identifier, and Status Register. Status Register read mode will be discussed in detail in the "Write Operations" section.

During power-up conditions (V<sub>CC</sub> supply ramping), it takes a maximum of 300 ns from when V<sub>CC</sub> is at 4.5V minimum to valid data on the outputs.

#### 3.2.1 READ ARRAY

If the memory is not in the Read Array mode, it is necessary to write the appropriate read mode command to the CUI. The 4-Mbit boot block flash family has three control functions, all of which must be logically active, to obtain data at the outputs. Chip-Enable CE# is the device selection control. Reset/Power-Down, RP# is the device power control. Output-Enable OE# is the DATA INPUT/OUTPUT (DQ[0:15] or DQ[0:7]) direction control and when active is used to drive data from the selected memory on to the I/O bus.

### 3.2.1.1 Output Control

With OE# at logic-high level ( $V_{IH}$ ), the output from the device is disabled and data input/output pins (DQ[0:15] or DQ[0:7]) are tri-stated. Data input is then controlled by WE#.

### 3.2.1.2 Input Control

With WE# at logic-high level ( $V_{IH}$ ), input to the device is disabled. Data Input/Output pins (DQ[0:15] or DQ[0:7]) are controlled by OE#.

## 3.2.2 INTELLIGENT IDENTIFIERS

The manufacturer and device codes are read via the CUI or by taking the A<sub>9</sub> pin to 12V. Writing 90H to the CUI places the device into Intelligent Identifier read mode. A read of location 00000H outputs the manufacturer's identification code, 0089H, and location 00001H outputs the device code; 4470H for A28F400BX-T, 4471H for A28F400BX-B. When BYTE# is at a logic low only the lower byte of the above signatures is read and DQ<sub>15</sub>/A<sub>-1</sub> is a "don't care" during Intelligent Identifier mode. A read array command must be written to the memory to return to the read array mode.

## 3.3 Write Operations

Commands are written to the CUI using standard microprocessor write timings. The CUI serves as the interface between the microprocessor and the internal chip operation. The CUI can decipher Read Array, Read Intelligent Identifier, Read Status Register, Clear Status Register, Erase and Program commands. In the event of a read command, the CUI simply points the read path at either the array, the Intelligent Identifier, or the status register depending on the specific read command given. For a program or erase cycle, the CUI informs the write state machine that a write or erase has been requested. During a program cycle, the Write State Machine will control the program sequences and the CUI will only respond to status reads. During an erase cycle, the CUI will respond to status reads and erase suspend. After the Write State Machine has completed its task, it will allow the CUI to respond to its full command set. The CUI will stay in the current command state until the microprocessor issues another command.

The CUI will successfully initiate an erase or write operation only when  $V_{PP}$  is within its voltage range. Depending upon the application, the system designer may choose to make the  $V_{PP}$  power supply switchable, available only when memory updates

are desired. The system designer can also choose to "hard-wire"  $V_{PP}$  to 12V. The 4-Mbit boot block flash family is designed to accommodate—either design practice. It is strongly recommended that RP# be tied to logical Reset for data protection during unstable CPU reset function as described in the "Product Family Overview" section.

## 3.3.1 BOOT BLOCK WRITE OPERATIONS

In the case of Boot Block modifications (write and erase), RP# is set to  $V_{HH}$  = 12V typically, in addition to  $V_{PP}$  at high voltage.

However, if RP# is not at  $V_{HH}$  when a program or erase operation of the boot block is attempted, the corresponding status register bit (Bit 4 for Program and Bit 5 for Erase, refer to Table 5 for Status Register Definitions) is set to indicate the failure to complete the operation.

## 3.3.2 COMMAND USER INTERFACE (CUI)

The Command User Interface (CUI) serves as the interface to the microprocessor. The CUI points the read/write path to the appropriate circuit block as described in the previous section. After the WSM has completed its task, it will set the WSM Status bit to a "1", which will also allow the CUI to respond to its full command set. Note that after the WSM has returned control to the CUI, the CUI will remain in its current state.

### 3.3.2.1 Command Set

Command Codes	Device Mode
00	Invalid/Reserved
10	Alternate Program Setup
20	Erase Setup
40	Program Setup
50	Clear Status Register
70	Read Status Register
90	Intelligent Identifier
B0	Erase Suspend
D0	Erase Resume/Erase Confirm
FF	Read Array

### 3.3.2.2 Command Function Descriptions

Device operations are selected by writing specific commands into the CUI. Table 3 defines the 4-Mbit boot block flash family commands.

**Table 3. Command Definitions**

Command	Bus Cycles Req'd	Notes	First Bus Cycle			Second Bus Cycle		
			8	Operation	Address	Data	Operation	Address
Read Array	1	1	Write	X	FFH			
Intelligent Identifier	3	2, 4	Write	X	90H	Read	IA	IID
Read Status Register	2	3	Write	X	70H	Read	X	SRD
Clear Status Register	1		Write	X	50H			
Erase Setup/Erase Confirm	2	5	Write	BA	20H	Write	BA	D0H
Word/Byte Write Setup/Write	2	6, 7	Write	WA	40H	Write	WA	WD
Erase Suspend/Erase Resume	2		Write	X	B0H	Write	X	D0H
Alternate Word/Byte Write Setup/Write	2	6, 7	Write	WA	10H	Write	WA	WD

**NOTES:**

1. Bus operations are defined in Tables 1, and 2.
  2. IA = Identifier Address: 00H for manufacturer code, 01H for device code.
  3. SRD = Data read from Status Register.
  4. IID = Intelligent Identifier Data.
- Following the Intelligent Identifier Command, two read operations access manufacturer and device codes.
5. BA = Address within the block being erased.
  6. WA = Address to be written.
  - WD = Data to be written at location WD.
  7. Either 40H or 10H commands is valid.
  8. When writing commands to the device, the upper data bus [DQ<sub>8</sub>-DQ<sub>15</sub>] = X which is either V<sub>CC</sub> or V<sub>SS</sub> to avoid burning additional current.

**Invalid/Reserved**

These are unassigned commands. It is not recommended that the customer use any command other than the valid commands specified above. Intel reserves the right to redefine these codes for future functions.

**Read Array (FFH)**

This single write command points the read path at the array. If the host CPU performs a CE# /OE# controlled read immediately following a two-write sequence that started the WSM, then the device will output status register contents. If the Read Array command is given after Erase Setup the device is reset to read the array. A two Read Array command sequence (FFH) is required to reset to Read Array after Program Setup.

**Intelligent Identifier (90H)**

After this command is executed, the CUI points the output path to the Intelligent Identifier circuits. Only Intelligent Identifier values at addresses 0 and 1 can be read (only address A<sub>0</sub> is used in this mode, all other address inputs are ignored).

**Read Status Register (70H)**

This is one of the two commands that is executable while the state machine is operating. After this command is written, a read of the device will output the contents of the status register, regardless of the address presented to the device.

The device automatically enters this mode after program or erase has completed.

**Clear Status Register (50H)**

The WSM can only set the Program Status and Erase Status bits in the status register, it can not clear them. Two reasons exist for operating the status register in this fashion. The first is a synchronization. The WSM does not know when the host CPU has read the status register, therefore it would not know when to clear the status bits. Secondly, if the CPU is programming a string of bytes, it may be more efficient to query the status register after programming the string. Thus, if any errors exist while programming the string, the status register will return the accumulated error status.

**Program Setup (40H or 10H)**

This command simply sets the CUI into a state such that the next write will load the address and data registers. Either 40H or 10H can be used for Program Setup. Both commands are included to accommodate efforts to achieve an industry standard command code set.

**Program**

The second write after the program setup command, will latch addresses and data. Also, the CUI initiates the WSM to begin execution of the program algorithm. While the WSM finishes the algorithm, the device will output Status Register contents. Note that the WSM cannot be suspended during programming.

**Erase Setup (20H)**

Prepares the CUI for the Erase Confirm command. No other action is taken. If the next command is not an Erase Confirm command then the CUI will set both the Program Status and Erase Status bits of the Status Register to a "1", place the device into the Read Status Register state, and wait for another command.

**Erase Confirm (D0H)**

If the previous command was an Erase Setup command, then the CUI will enable the WSM to erase, at the same time closing the address and data latches, and respond only to the Read Status Register and Erase Suspend commands. While the WSM is executing, the device will output Status Register data when OE# is toggled low. Status Register data can only be updated by toggling either OE# or CE# low.

**Erase Suspend (B0H)**

This command only has meaning while the WSM is executing an Erase operation, and therefore will only be responded to during an erase operation. After this command has been executed, the CUI will set an output that directs the WSM to suspend Erase operations, and then return to responding to only Read Status Register or to the Erase Resume commands. Once the WSM has reached the Suspend state, it will set an output into the CUI which allows the CUI to respond to the Read Array, Read Status Register, and Erase Resume commands. In this mode, the CUI will not respond to any other commands. The WSM will also set the WSM Status bit to a "1". The WSM will continue to run, idling in the SUSPEND state, regardless of the state of all input

control pins, with the exclusion of RP#. RP# will immediately shut down the WSM and the remainder of the chip. During a suspend operation, the data and address latches will remain closed, but the address pads are able to drive the address into the read path.

**Erase Resume (D0H)**

This command will cause the CUI to clear the Suspend state and set the WSM Status bit to a "0", but only if an Erase Suspend command was previously issued. Erase Resume will not have any effect in all other conditions.

**3.3.3 STATUS REGISTER**

The 4-Mbit boot block flash family contains a status register which may be read to determine when a program or erase operation is complete, and whether that operation completed successfully. The status register may be read at any time by writing the Read Status command to the CUI. After writing this command, all subsequent Read operations output data from the status register until another command is written to the CUI. A Read Array command must be written to the CUI to return to the Read Array mode.

The status register bits are output on DQ[0:7] whether the device is in the byte-wide (x8) or word-wide (x16) mode. In the word-wide mode the upper byte, DQ[8:15] is set to 00H during a Read Status command. In the byte-wide mode, DQ[8:14] are tristated and DQ<sub>15</sub>/A<sub>-1</sub> retains the low order address function.

It should be noted that the contents of the status register are latched on the falling edge of OE# or CE# whichever occurs last in the read cycle. This prevents possible bus errors which might occur if the contents of the status register change while reading the status register. CE# or OE# must be toggled with each subsequent status read, or the completion of a program or erase operation will not be evident.

The Status Register is the interface between the microprocessor and the Write State Machine (WSM). When the WSM is active, this register will indicate the status of the WSM, and will also hold the bits indicating whether or not the WSM was successful in performing the desired operation. The WSM sets status bits "Three" through "Seven" and clears bits "Six" and "Seven", but cannot clear status bits "Three" through "Five". These bits can only be cleared by the controlling CPU through the use of the Clear Status Register command.

**3.3.3.1 Status Register Bit Definition**
**Table 4. Status Register Definitions**

	WSMS	ESS	ES	PS	VPPS	R	R	R
	7	6	5	4	3	2	1	0
<b>SR.7 = WRITE STATE MACHINE STATUS</b> 1 = Ready 0 = Busy								
<b>SR.6 = ERASE SUSPEND STATUS</b> 1 = Erase Suspended 0 = Erase In Progress/Completed								
<b>SR.5 = ERASE STATUS</b> 1 = Error In Block Erasure 0 = Successful Block Erase								
<b>SR.4 = PROGRAM STATUS</b> 1 = Error In Byte/Word Program 0 = Successful Byte/Word Program								
<b>SR.3 = V<sub>pp</sub> STATUS</b> 1 = V <sub>pp</sub> Low Detect; Operation Abort 0 = V <sub>pp</sub> OK								
<b>SR.2–SR.0 = RESERVED FOR FUTURE ENHANCEMENTS</b>								

**NOTES:**

Write State Machine Status bit must first be checked to determine byte/word program or block erase completion, before the Program or Erase Status bits are checked for success.

When Erase Suspend is Issued, WSM halts execution and sets both WSMS and ESS bits to "1". ESS bit remains set to "1" until an Erase Resume command is issued.

When this bit is set to "1", WSM has applied the maximum number of erase pulses to the block and is still unable to successfully perform an erase verify.

When this bit is set to "1", WSM has attempted but failed to Program a byte or word.

The V<sub>pp</sub> Status bit unlike an A/D converter, does not provide continuous indication of V<sub>pp</sub> level. The WSM interrogates the V<sub>pp</sub> level only after the byte write or block erase command sequences have been entered and informs the system if V<sub>pp</sub> has not been switched on. The V<sub>pp</sub> Status bit is not guaranteed to report accurate feedback between V<sub>ppL</sub> and V<sub>ppH</sub>.

These bits are reserved for future use and should be masked out when polling the Status Register.

**3.3.3.2 Clearing the Status Register**

Certain bits in the status register are set by the write state machine, and can only be reset by the system software. These bits can indicate various failure conditions. By allowing the system software to control the resetting of these bits, several operations may be performed (such as cumulatively programming several bytes or erasing multiple blocks in sequence). The status register may then be read to determine if an error occurred during that programming or erasure series. This adds flexibility to the way the device may be programmed or erased. To clear the status register, the Clear Status Register command is written to the CUI. Then, any other command may be issued to the CUI. Note again that before a read cycle can be initiated, a Read Array command must be written to the CUI to specify whether the read data is to come from the array, status register, or Intelligent Identifier.

**3.3.4 PROGRAM MODE**

Program is executed by a two-write sequence. The Program Setup command is written to the CUI followed by a second write which specifies the address and data to be programmed. The write state machine will execute a sequence of internally timed events to:

1. Program the desired bits of the addressed memory word (byte), and
2. Verify that the desired bits are sufficiently programmed.

Programming of the memory results in specific bits within a byte or word being changed to a "0".

If the user attempts to program "1"s, there will be no change of the memory cell content and no error occurs.

Similar to erasure, the status register indicates whether programming is complete. While the program sequence is executing, bit 7 of the status register is a "0". The status register can be polled by toggling either CE# or OE# to determine when the program sequence is complete. Only the Read Status Register command is valid while programming is active.

When programming is complete, the status bits, which indicate whether the program operation was successful, should be checked. If the programming operation was unsuccessful, Bit 4 of the status register is set to a "1" to indicate a Program Failure. If Bit 3 is set then V<sub>pp</sub> was not within acceptable limits, and the WSM will not execute the programming sequence.

The status register should be cleared before attempting the next operation. Any CUI instruction can follow after programming is completed; however, it must be recognized that reads from the memory, status register, or Intelligent Identifier cannot be accomplished until the CUI is given the appropriate command. A Read Array command must first be given before memory contents can be read.

Figure 6 shows a system software flowchart for device byte programming operation. Figure 7 shows a similar flowchart for device word programming operation (A28F400BX-only).

### 3.3.5 ERASE MODE

Erasure of a single block is initiated by writing the Erase Setup and Erase Confirm commands to the CUI, along with the addresses A[12:17], identifying the block to be erased. These addresses are latched internally when the Erase Confirm command is issued. Block erasure results in all bits within the block being set to "1".

The WSM will execute a sequence of internally timed events to:

1. Program all bits within the block
2. Verify that all bits within the block are sufficiently programmed
3. Erase all bits within the block and
4. Verify that all bits within the block are sufficiently erased

While the erase sequence is executing, Bit 7 of the status register is a "0".

When the status register indicates that erasure is complete, the status bits, which indicate whether the erase operation was successful, should be checked. If the erasure operation was unsuccessful, Bit 5 of the status register is set to a "1" to indicate an Erase Failure. If V<sub>pp</sub> was not within acceptable limits after the Erase Confirm command is issued, the WSM will not execute an erase sequence; instead, Bits of the status register is set to a "1" to indicate an Erase Failure, and Bit 3 is set to a "1" to identify that V<sub>pp</sub> supply voltage was not within acceptable limits.

The status register should be cleared before attempting the next operation. Any CUI instruction can follow after erasure is completed; however, it must be recognized that reads from the memory array, status register, or Intelligent Identifier can not be accomplished until the CUI is given the appropriate command. A Read Array command must first be given before memory contents can be read.

Figure 8 shows a system software flowchart for Block Erase operation.

#### 3.3.5.1 Suspending and Resuming Erase

Since an erase operation typically requires 1.5 to 3 seconds to complete, an Erase Suspend command is provided. This allows erase-sequence interruption in order to read data from another block of the memory. Once the erase sequence is started, writing the Erase Suspend command to the CUI requests that the Write State Machine (WSM) pause the erase sequence at a predetermined point in the erase algorithm. The status register must be read to determine when the erase operation has been suspended.

At this point, a Read Array command can be written to the CUI in order to read data from blocks other than that which is being suspended. The only other valid command at this time is the Erase Resume command or Read Status Register operation.

Figure 9 shows a system software flowchart detailing the operation.

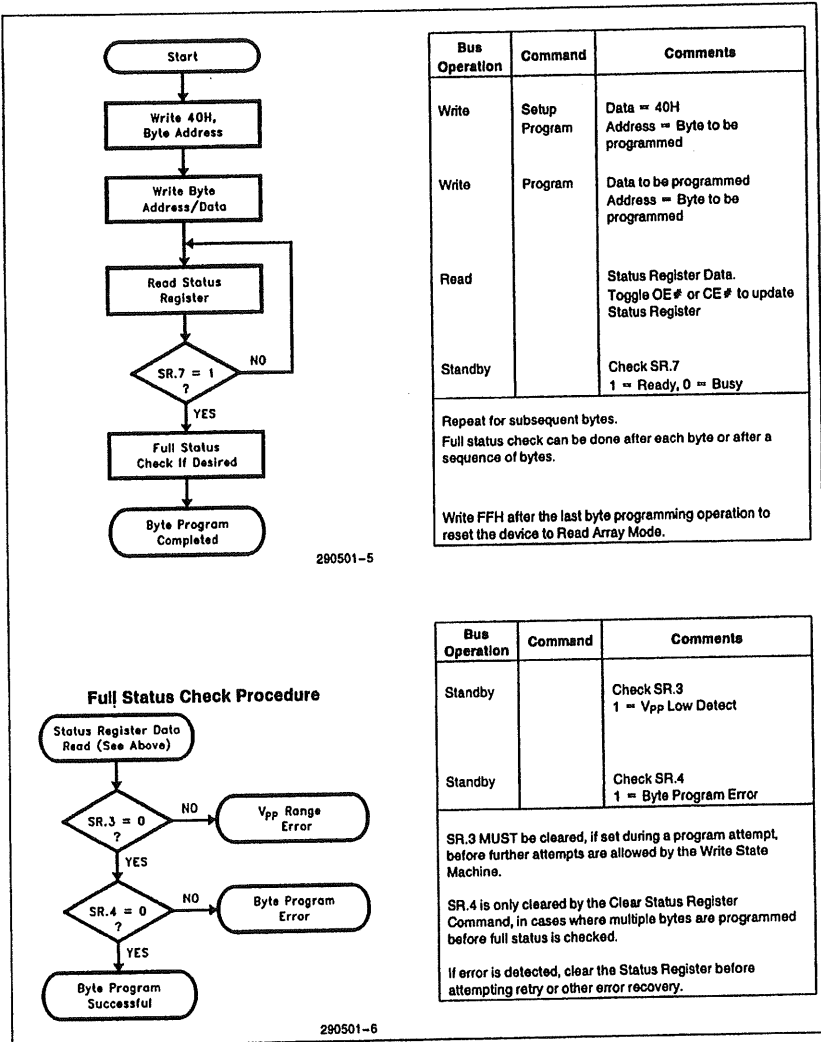


During Erase Suspend mode, the chip can go into a pseudo-standby mode by taking CE# to  $V_{IH}$  and the active current is now a maximum of 10 mA. If the chip is enabled while in this mode by taking CE# to  $V_{IL}$ , the Erase Resume command can be issued to resume the erase operation.

Upon completion of reads from any block other than the block being erased, the Erase Resume command must be issued. When the Erase Resume command is given, the WSM will continue with the erase sequence and complete erasing the block. As with the end of erase, the status register must be read, cleared, and the next instruction issued in order to continue.

### **3.3.6 EXTENDED CYCLING**

Intel has designed extended cycling capability into its ETOX III flash memory technology. The 4-Mbit boot block flash family is designed for 1,000 program/erase cycles on each of the seven blocks. The combination of low electric fields, clean oxide processing and minimized oxide area per memory cell subjected to the tunneling electric field, results in very high cycling capability.



290501-5

290501-6

Bus Operation	Command	Comments
Write	Setup Program	Data = 40H Address = Byte to be programmed
Write	Program	Data to be programmed Address = Byte to be programmed
Read		Status Register Data. Toggle OE# or CE# to update Status Register
Standby		Check SR.7 1 = Ready, 0 = Busy

Repeat for subsequent bytes.  
Full status check can be done after each byte or after a sequence of bytes.

Write FFH after the last byte programming operation to reset the device to Read Array Mode.

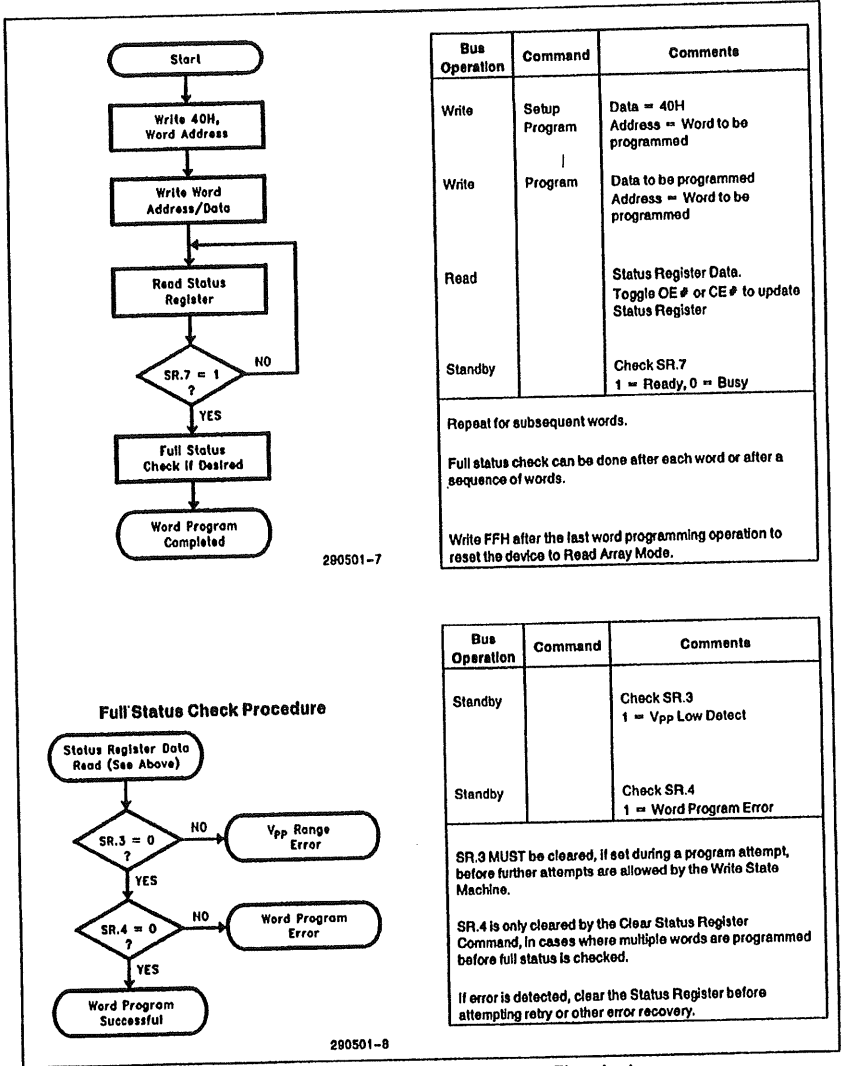
Bus Operation	Command	Comments
Standby		Check SR.3 1 = Vpp Low Detect
Standby		Check SR.4 1 = Byte Program Error

SR.3 MUST be cleared, if set during a program attempt, before further attempts are allowed by the Write State Machine.

SR.4 is only cleared by the Clear Status Register Command, in cases where multiple bytes are programmed before full status is checked.

If error is detected, clear the Status Register before attempting retry or other error recovery.

Figure 6. Automated Byte Programming Flowchart



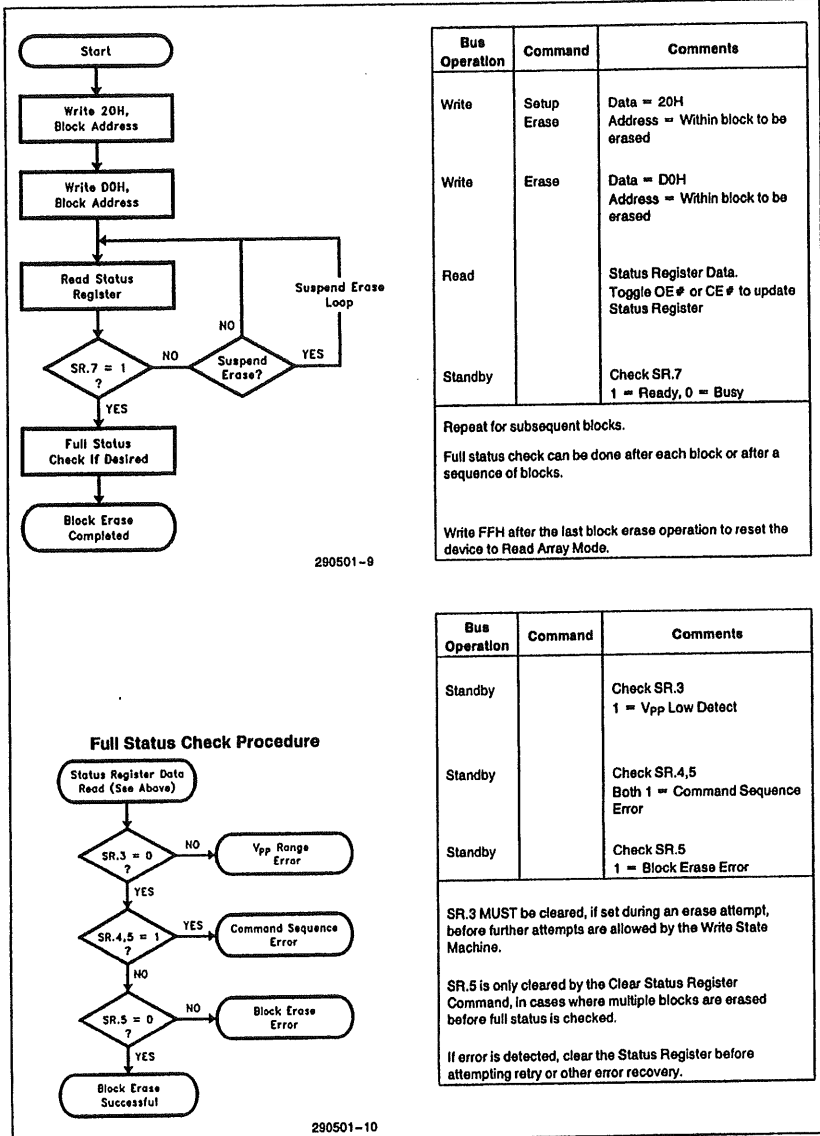
290501-7

290501-8

Bus Operation	Command	Comments
Write	Setup Program	Data = 40H Address = Word to be programmed
Write	Program	Data to be programmed Address = Word to be programmed
Read		Status Register Data. Toggle OE# or CE# to update Status Register
Standby		Check SR.7 1 = Ready, 0 = Busy
Repeat for subsequent words.		
Full status check can be done after each word or after a sequence of words.		
Write FFH after the last word programming operation to reset the device to Read Array Mode.		

Bus Operation	Command	Comments
Standby		Check SR.3 1 = Vpp Low Detect
Standby		Check SR.4 1 = Word Program Error
SR.3 MUST be cleared, if set during a program attempt, before further attempts are allowed by the Write State Machine.		
SR.4 is only cleared by the Clear Status Register Command, in cases where multiple words are programmed before full status is checked.		
If error is detected, clear the Status Register before attempting retry or other error recovery.		

Figure 7. Automated Word Programming Flowchart



Bus Operation	Command	Comments
Write	Setup Erase	Data = 20H Address = Within block to be erased
Write	Erase	Data = 00H Address = Within block to be erased
Read		Status Register Data. Toggle OE# or CE# to update Status Register
Standby		Check SR.7 1 = Ready, 0 = Busy
Repeat for subsequent blocks.		
Full status check can be done after each block or after a sequence of blocks.		
Write FFH after the last block erase operation to reset the device to Read Array Mode.		

Bus Operation	Command	Comments
Standby		Check SR.3 1 = Vpp Low Detect
Standby		Check SR.4,5 Both 1 = Command Sequence Error
Standby		Check SR.5 1 = Block Erase Error
SR.3 MUST be cleared, if set during an erase attempt, before further attempts are allowed by the Write State Machine.		
SR.5 is only cleared by the Clear Status Register Command, in cases where multiple blocks are erased before full status is checked.		
If error is detected, clear the Status Register before attempting retry or other error recovery.		

Figure 8. Automated Block Erase Flowchart

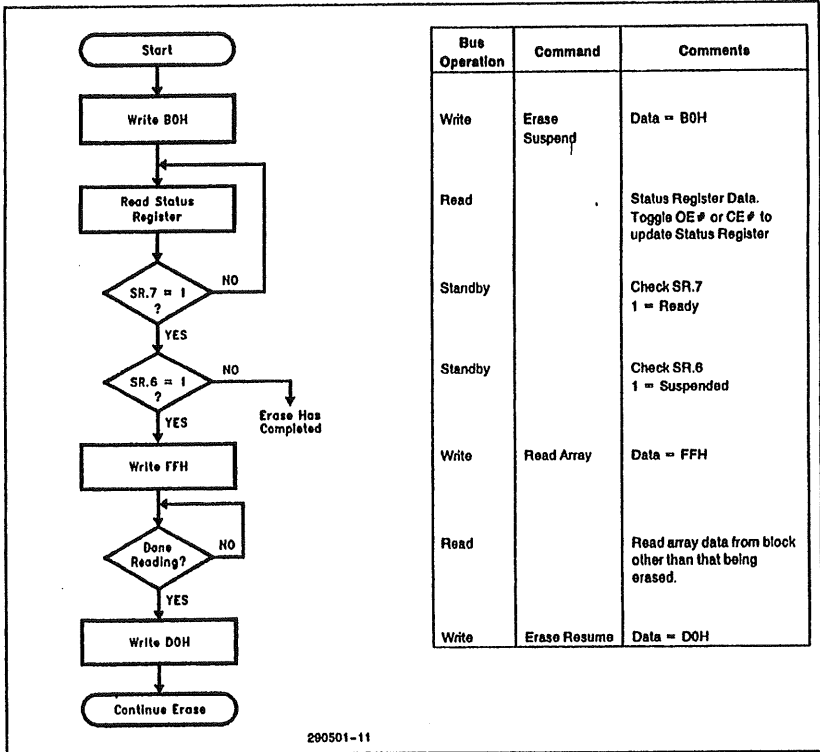


Figure 9. Erase Suspend/Resume Flowchart

### 3.4 Power Consumption

#### 3.4.1 ACTIVE POWER

With CE# at a logic-low level and RP# at a logic-high level, the device is placed in the active mode. The device I<sub>CC</sub> current is a maximum 85 mA at 10 MHz with TTL input signals.

#### 3.4.2 AUTOMATIC POWER SAVINGS

Automatic Power Savings (APS) is a low power feature during active mode of operation. The 4-Mbit family of products incorporate Power Reduction Control (PRC) circuitry which basically allows the device to put itself into a low current state when it is not being accessed. After data is read from the memory array, PRC logic controls the device's power consumption by entering the APS mode where

maximum I<sub>CC</sub> current is 3 mA and typical I<sub>CC</sub> current is 1 mA. The device stays in this static state with outputs valid until a new location is read.

#### 3.4.3 STANDBY POWER

With CE# at a logic-high level (V<sub>IH</sub>), and the CUI in read mode, the memory is placed in standby mode where the maximum I<sub>CC</sub> standby current is 100 μA with CMOS input signals. The standby operation disables much of the device's circuitry and substantially reduces device power consumption. The outputs (DQ[0:15] or DQ[0:7]) are placed in a high-impedance state independent of the status of the OE# signal. When the 4-Mbit boot block flash family is deselected during erase or program functions, the devices will continue to perform the erase or program function and consume program or erase active power until program or erase is completed.

## ADVANCE INFORMATION



**Intel 80C188EB Processor**



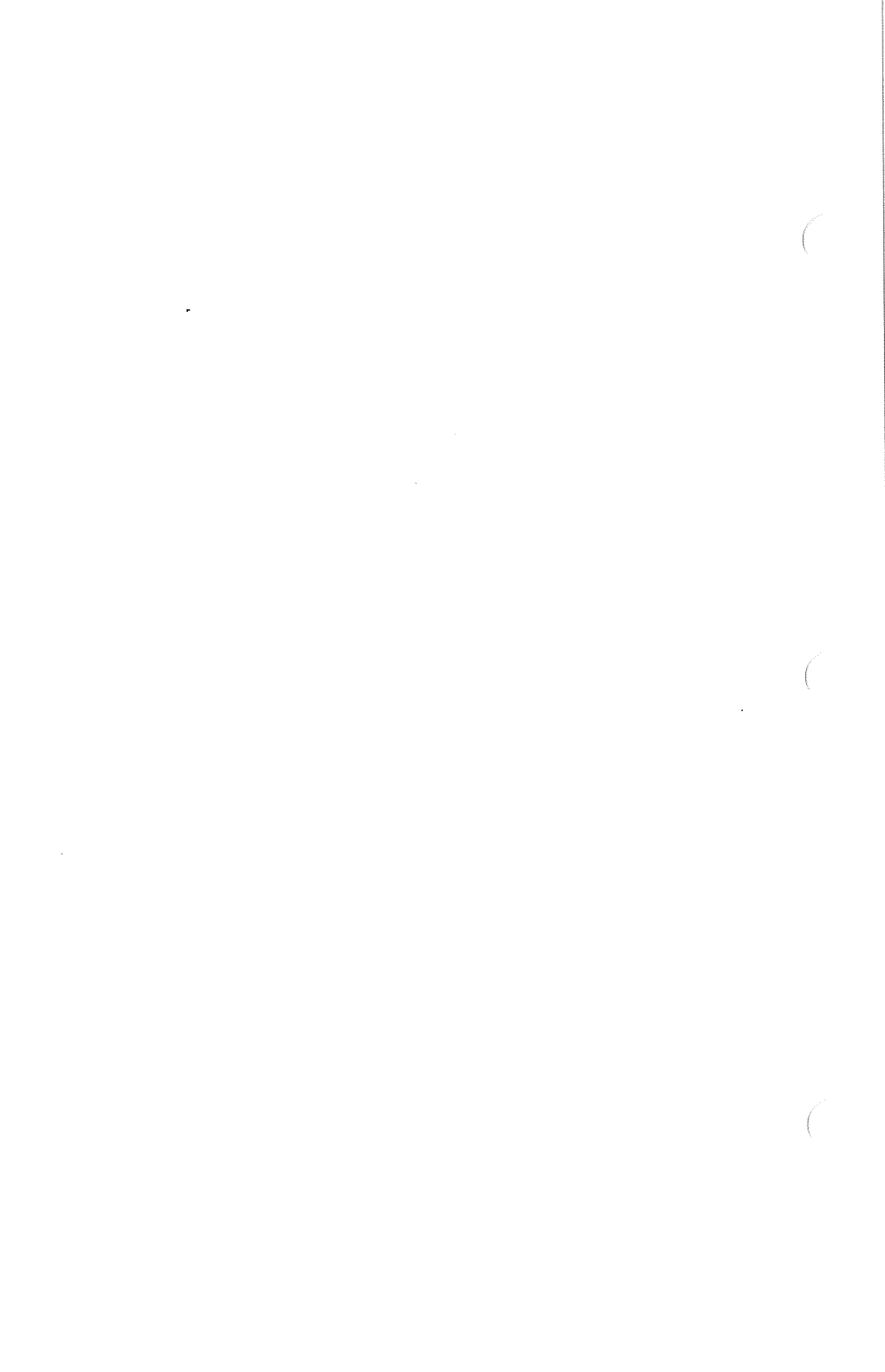


**intel.**<sup>®</sup>

**4**

**Peripheral Control  
Block**

**I**



## CHAPTER 4 PERIPHERAL CONTROL BLOCK

All integrated peripherals in the 80C186 Modular Core family are controlled by sets of registers within an integrated Peripheral Control Block (PCB). The peripheral control registers are physically located in the peripheral devices they control, but they are addressed as a single block of registers. The Peripheral Control Block encompasses 256 contiguous bytes and can be located on any 256-byte boundary of memory or I/O space. The PCB Relocation Register, which is also located within the Peripheral Control Block, controls the location of the PCB.

### 4.1 PERIPHERAL CONTROL REGISTERS

Each of the integrated peripherals' control and status registers is located at a fixed offset above the programmed base location of the Peripheral Control Block (see Table 4-2). These registers are described in the chapters that cover the associated peripheral. "Accessing the Peripheral Control Block" on page 4-4 discusses how the registers are accessed and outlines considerations for reading and writing them.

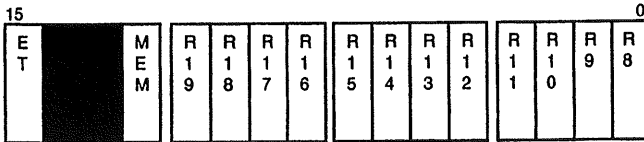
### 4.2 PCB RELOCATION REGISTER

In addition to control registers for the integrated peripherals, the Peripheral Control Block contains the PCB Relocation Register (Figure 4-1). The Relocation Register is located at a fixed offset within the Peripheral Control Block (Table 4-2). If the Peripheral Control Block is moved, the Relocation Register also moves.

The PCB Relocation Register allows the Peripheral Control Block to be relocated to any 256-byte boundary within memory or I/O space. The Memory I/O bit (MEM) selects either memory space or I/O space, and the R19:8 bits specify the starting (base) address of the PCB. The remaining bit, Escape Trap (ET), controls access to the math coprocessor interface.

"Setting the PCB Base Location" on page 4-6 describes how to set the base location and outlines some restrictions on the Peripheral Control Block location.

**Register Name:** PCB Relocation Register  
**Register Mnemonic:** RELREG  
**Register Function:** Relocates the PCB within memory or I/O space.



A1263-0A

Bit Mnemonic	Bit Name	Reset State	Function
ET	Escape Trap	0	The ET bit controls access to the math coprocessor. If ET is set, the CPU will trap (resulting in a Type 7 interrupt) when an ESC instruction is executed.  <b>NOTE:</b> The 8-bit bus version of the device automatically traps an ESC opcode to the Type 7 interrupt, regardless of the state of the ET bit.
MEM	Memory I/O	0	The MEM bit specifies the PCB location. Set MEM to locate the PCB in memory space, or clear it to locate the PCB in I/O space.
R19:8	PCB Base Address Upper Bits	0FFH	R19:8 define the upper address bits of the PCB base address. All lower bits are zero. R19:16 are ignored when the PCB is mapped to I/O space.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

**Figure 4-1. PCB Relocation Register**

**Table 4-2. Peripheral Control Block**

PCB Offset	Function	PCB Offset	Function	PCB Offset	Function	PCB Offset	Function
00H	Reserved	40H	T2CNT	80H	GCS0ST	C0H	Reserved
02H	EOI	42H	T2CMPA	82H	GCS0SP	C2H	Reserved
04H	POLL	44H	Reserved	84H	GCS1ST	C4H	Reserved
06H	POLLSTS	46H	T2CON	86H	GCS1SP	C6H	Reserved
08H	IMASK	48H	Reserved	88H	GCS2ST	C8H	Reserved
0AH	PRIMSK	4AH	Reserved	8AH	GCS2SP	CAH	Reserved
0CH	INSERV	4CH	Reserved	8CH	GCS3ST	CCH	Reserved
0EH	REQST	4EH	Reserved	8EH	GCS3SP	CEH	Reserved
10H	INSTS	50H	P1DIR	90H	GCS4ST	D0H	Reserved
12H	TCUCON	52H	P1PIN	92H	GCS4SP	D2H	Reserved
14H	SCUCON	54H	P1CON	94H	GCS5ST	D4H	Reserved
16H	I4CON	56H	P1LTCH	96H	GCS5SP	D6H	Reserved
18H	I0CON	58H	P2DIR	98H	GCS6ST	D8H	Reserved
1AH	I1CON	5AH	P2PIN	9AH	GCS6SP	DAH	Reserved
1CH	I2CON	5CH	P2CON	9CH	GCS7ST	DCH	Reserved
1EH	I3CON	5EH	P2LTCH	9EH	GCS7SP	DEH	Reserved
20H	Reserved	60H	B0CMP	A0H	LCSST	E0H	Reserved
22H	Reserved	62H	B0CNT	A2H	LCSST	E2H	Reserved
24H	Reserved	64H	S0CON	A4H	UCSST	E4H	Reserved
26H	Reserved	66H	S0STS	A6H	UCSSP	E6H	Reserved
28H	Reserved	68H	S0RBUF	A8H	RELREG	E8H	Reserved
2AH	Reserved	6AH	S0TBUF	AAH	Reserved	EAH	Reserved
2CH	Reserved	6CH	Reserved	ACH	Reserved	ECH	Reserved
2EH	Reserved	6EH	Reserved	AEH	Reserved	EEH	Reserved
30H	TOCNT	70H	B1CMP	B0H	RFBASE	F0H	Reserved
32H	TOCMPA	72H	B1CNT	B2H	RFTIME	F2H	Reserved
34H	TOCMPB	74H	S1CON	B4H	RFCON	F4H	Reserved
36H	TOCON	76H	S1STS	B6H	RFADDR	F6H	Reserved
38H	T1CNT	78H	S1RBUF	B8H	PWRCON	F8H	Reserved
3AH	T1CMPA	7AH	S1TBUF	BAH	Reserved	FAH	Reserved
3CH	T1CMPB	7CH	Reserved	BCH	STEPID	FCH	Reserved
3EH	T1CON	7EH	Reserved	BEH	Reserved	FEH	Reserved

### 4.3 RESERVED LOCATIONS

Many locations within the Peripheral Control Block are not assigned to any peripheral. Unused locations are reserved. Reading from these locations yields an undefined result. If reserved registers are written (for example, during a block MOV instruction) they must be set to 0H.

#### NOTE

Failure to follow this guideline could result in incompatibilities with future 80C186 Modular Core family products.

### 4.4 ACCESSING THE PERIPHERAL CONTROL BLOCK

All communication between integrated peripherals and the Modular CPU Core occurs over a special bus, called the *F-Bus*, which always carries 16-bit data. The Peripheral Control Block, like all integrated peripherals, is always accessed 16 bits at a time.

#### 4.4.1 Bus Cycles

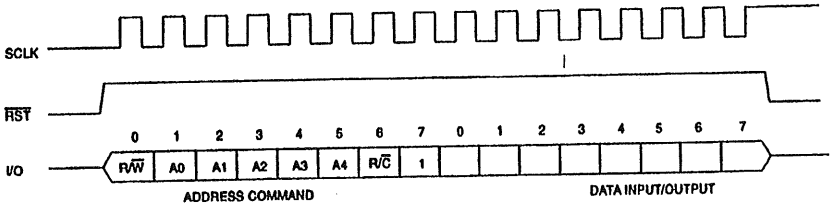
The processor runs an external bus cycle for any memory or I/O cycle accessing a location within the Peripheral Control Block. Address, data and control information is driven on the external pins as with an ordinary bus cycle. Information returned by an external device is ignored, even if the access does not correspond to the location of an integrated peripheral control register. This is also true for the 80C188 Modular Core family, except that word accesses made to integrated registers are performed in two bus cycles.

#### 4.4.2 READY Signals and Wait States

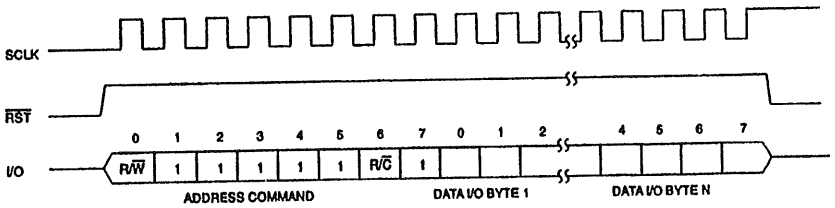
The processor generates an internal READY signal whenever an integrated peripheral is accessed. External READY is ignored. READY is also generated if an access is made to a location within the Peripheral Control Block that does not correspond to an integrated peripheral control register. For accesses to timer control and counting registers, the processor inserts one wait state. This is required to properly multiplex processor and counter element accesses to the timer control registers. For accesses to the remaining locations in the Peripheral Control Block, the processor does not insert wait states.

**DATA TRANSFER SUMMARY Figure 3**

**SINGLE BYTE TRANSFER**



**BURST MODE TRANSFER**



FUNCTION	BYTE N	SCLK n
CLOCK	8	72
RAM	31	256

REGISTER ADDRESS/DEFINITION Figure 4

REGISTER ADDRESS

A. CLOCK

	7	6	5	4	3	2	1	0	
SEC	1	0	0	0	0	0	0	RD/W	
MIN	1	0	0	0	0	0	1	RD/W	
HR	1	0	0	0	0	1	0	RD/W	
DATE	1	0	0	0	0	1	1	RD/W	
MONTH	1	0	0	0	1	0	0	RD/W	
DAY	1	0	0	0	1	0	1	RD/W	
YEAR	1	0	0	0	1	1	0	RD/W	
CONTROL	1	0	0	0	1	1	1	RD/W	
TRICKLE CHARGER	1	0	0	1	0	0	0	RD/W	
CLOCK BURST	1	0	1	1	1	1	1	RD/W	

B. RAM

RAM 0	1	1	0	0	0	0	0	RD/W
RAM 30	1	1	1	1	1	1	0	RD/W
RAM BURST	1	1	1	1	1	1	1	RD/W

REGISTER DEFINITION

00-69	CH	10 SEC	SEC
00-69	0	10 MIN	MIN
01-12 00-23	12/ 24	0	10 A/P
	HR		HR
01-28/29 01-30 01-31	0	0	10 DATE
			DATE
01-12	0	0	0
			10 M
			MONTH
01-07	0	0	0
			0
			DAY
00-69		10 YEAR	YEAR
	WP	0	0
		0	0
		0	0
		0	0
		0	0
		0	0
	TCS	TCS	TCS
		TCS	DS
			DS
			RS
			RS

RAM DATA 0							
RAM DATA 30							



**ABSOLUTE MAXIMUM RATINGS\***

Voltage on Any Pin Relative to Ground	-0.5V to +7.0V
Operating Temperature	0°C to 70°C
Storage Temperature	-55°C to +125°C
Soldering Temperature	260°C for 10 seconds

- \* This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

The Dallas Semiconductor DS1302 is built to the highest quality standards and manufactured for long term reliability. All Dallas Semiconductor devices are made using the same quality materials and manufacturing methods. However, standard versions of the DS1302 are not exposed to environmental stresses, such as burn-in, that some industrial applications require. Products which have successfully passed through this series of environmental stresses are marked IND or N, denoting their extended operating temperature and reliability rating. For specific reliability information on this product, please contact the factory in Dallas at (972) 371-4448.

**RECOMMENDED DC OPERATING CONDITIONS**

(0°C to 70°C)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage $V_{CC1}$ , $V_{CC2}$	$V_{CC1}$ , $V_{CC2}$	2.0		5.5	V	1, 11
Logic 1 Input	$V_{IH}$	2.0		$V_{CC}+0.3$	V	1
Logic 0 Input	$V_{IL}$	$V_{CC}=2.0V$	-0.3	+0.3	V	1
		$V_{CC}=5V$	-0.3	+0.8		

**DC ELECTRICAL CHARACTERISTICS**(0°C to 70°C;  $V_{CC} = 2.0$  to 5.5V\*)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Input Leakage	$I_{LI}$			+500	$\mu A$	6
I/O Leakage	$I_{LO}$			+500	$\mu A$	6
Logic 1 Output	$V_{OH}$	$V_{CC}=2.0V$	1.6		V	2
		$V_{CC}=5V$	2.4			
Logic 0 Output	$V_{OL}$	$V_{CC}=2.0V$		0.4	V	3
		$V_{CC}=5V$		0.4		
Active Supply Current	$I_{CC1A}$	$V_{CC1}=2.0V$		0.4	mA	5, 12
		$V_{CC1}=5V$		1.2		
Timekeeping Current	$I_{CC1T}$	$V_{CC1}=2.0V$		0.3	$\mu A$	4, 12
		$V_{CC1}=5V$		1		
Standby Current	$I_{CC1S}$	$V_{CC1}=2.0V$		100	nA	10, 12, 14
		$V_{CC1}=5V$		100		
Active Supply Current	$I_{CC2A}$	$V_{CC2}=2.0V$		0.425	mA	5, 13
		$V_{CC2}=5V$		1.28		

\*Unless otherwise noted.

**DC ELECTRICAL CHARACTERISTICS (cont'd)**(0°C to 70°C;  $V_{CC} = 2.0$  to 5.5V\*)

PARAMETER	SYMBOL		MIN	TYP	MAX	UNITS	NOTES
Timekeeping Current	$I_{CC2T}$	$V_{CC}=2.0V$			25.3	$\mu A$	4, 13
		$V_{CC}=5V$			81		
Standby Current	$I_{CC2S}$	$V_{CC}=2.0V$			25	$\mu A$	10, 13
		$V_{CC}=5V$			80		
Trickle Charge Resistors	R1			2		$K\Omega$	
	R2			4		$K\Omega$	
	R3			8		$K\Omega$	
Trickle Charger Diode Voltage Drop	$V_{TD}$			0.7		V	

\*Unless otherwise noted.

**CAPACITANCE** $(t_A = 25^\circ C)$ 

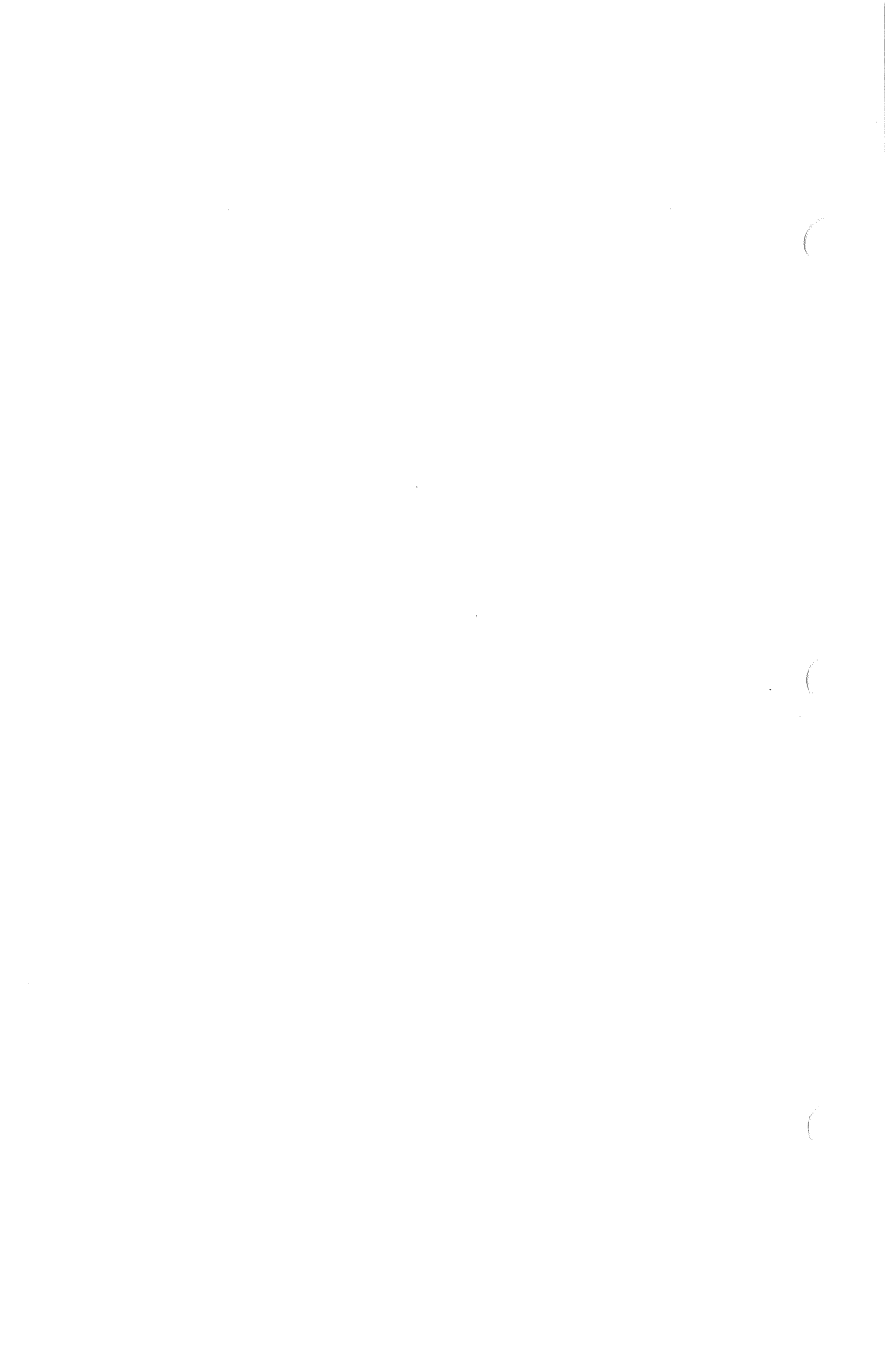
PARAMETER	SYMBOL	CONDITION	TYP	MAX	UNITS	NOTES
Input Capacitance	$C_I$		10		pF	
I/O Capacitance	$C_{I/O}$		15		pF	
Crystal Capacitance	$C_X$		6		pF	

**AC ELECTRICAL CHARACTERISTICS**(0°C to 70°C;  $V_{CC} = 2.0$  to 5.5V\*)

PARAMETER	SYMBOL		MIN	TYP	MAX	UNITS	NOTES
Data to CLK Setup	$t_{DC}$	$V_{CC}=2.0V$	200			ns	7
		$V_{CC}=5V$	50				
CLK to Data Hold	$t_{CDH}$	$V_{CC}=2.0V$	280			ns	7
		$V_{CC}=5V$	70				
CLK to Data Delay	$t_{CDD}$	$V_{CC}=2.0V$			800	ns	7, 8, 9
		$V_{CC}=5V$			200		
CLK Low Time	$t_{CL}$	$V_{CC}=2.0V$	1000			ns	7
		$V_{CC}=5V$	250				
CLK High Time	$t_{CH}$	$V_{CC}=2.0V$	1000			ns	7
		$V_{CC}=5V$	250				
CLK Frequency	$t_{CLK}$	$V_{CC}=2.0V$			0.5	MHz	7
		$V_{CC}=5V$		DC	2.0		
CLK Rise and Fall	$t_R, t_F$	$V_{CC}=2.0V$			2000	ns	
		$V_{CC}=5V$			500		
RST to CLK Setup	$t_{CC}$	$V_{CC}=2.0V$	4			$\mu s$	7
		$V_{CC}=5V$	1				

\*Unless otherwise noted.

**MAX197 12-Bit DAS**



# Multi-Range ( $\pm 10V$ , $\pm 5V$ , $+10V$ , $+5V$ ), Single $+5V$ , 12-Bit DAS with 8+4 Bus Interface

## Detailed Description

### Converter Operation

The MAX197, a multi-range, fault-tolerant ADC, uses successive approximation and internal input track/hold (T/H) circuitry to convert an analog signal to a 12-bit digital output. The parallel-output format provides easy interface to microprocessors ( $\mu$ Ps). Figure 3 shows the MAX197 in its simplest operational configuration.

### Analog-Input Track/Hold

In the internal acquisition control mode (control bit D5 set to 0), the T/H enters its tracking mode on WR's rising edge, and enters its hold mode when the internally timed (6 clock cycles) acquisition interval ends. A low impedance input source, which settles in less than  $1.5\mu s$ , is required to maintain conversion accuracy at the maximum conversion rate.

In the external acquisition control mode (D5 = 1), the T/H enters its tracking mode on the first WR rising edge and enters its hold mode when it detects the second WR rising edge with D5 = 0. See the *External Acquisition* section.

### Input Bandwidth

The ADC's input tracking circuitry has a 5MHz small-signal bandwidth. When using the internal acquisition

mode with an external clock frequency of 2MHz, a 100ksp/s throughput rate can be achieved. It is possible to digitize high-speed transient events and measure periodic signals with bandwidths exceeding the ADC's sampling rate by using undersampling techniques. To avoid high-frequency signals being aliased into the frequency band of interest, anti-alias filtering is recommended (MAX274/MAX275 continuous-time filters).

### Input Range and Protection

Figure 4 shows the equivalent input circuit. With  $V_{REF} = 4.096V$ , the MAX197 can be programmed for input ranges of  $\pm 10V$ ,  $\pm 5V$ ,  $0V$  to  $10V$ , or  $0V$  to  $5V$  by setting the appropriate control bits (D3, D4) in the control byte (see Tables 2 and 3). The full-scale input voltage depends on the voltage at REF (Table 1). When an external reference is applied at REFADJ, the voltage at REF is given by  $V_{REF} = 1.6384 \times V_{REFADJ}$  ( $2.4V < V_{REF} < 4.18V$ ).

Table 1. Full Scale and Zero Scale

RANGE (V)	ZERO SCALE (V)	-FULL SCALE	+FULL SCALE
0 to 5	0	—	$V_{REF} \times 1.2207$
0 to 10	0	—	$V_{REF} \times 2.4414$
$\pm 5$	—	$-V_{REF} \times 1.2207$	$V_{REF} \times 1.2207$
$\pm 10$	—	$-V_{REF} \times 2.4414$	$V_{REF} \times 2.4414$

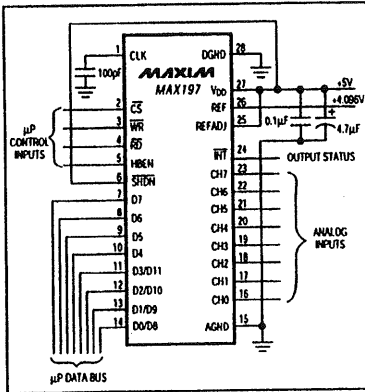


Figure 3. Operational Diagram

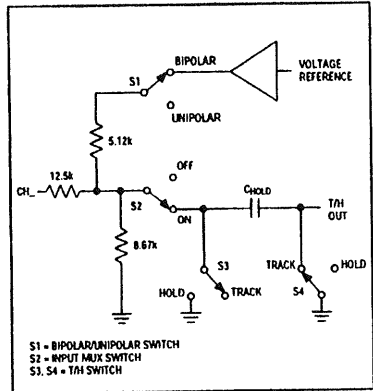


Figure 4. Equivalent Input Circuit

# Multi-Range ( $\pm 10V$ , $\pm 5V$ , $+10V$ , $+5V$ ), Single $+5V$ , 12-Bit DAS with 8+4 Bus Interface

**MAX197**

The input channels are overvoltage protected to  $\pm 16.5V$ . This protection is active even if the device is in power-down mode.

Even with  $V_{DD} = 0V$ , the input resistive network provides current-limiting that adequately protects the device.

### Digital Interface

Input data (control byte) and output data are multiplexed on a three-state parallel interface. This parallel I/O can easily be interfaced with a  $\mu P$ .  $\overline{CS}$ ,  $\overline{WR}$ , and  $\overline{RD}$  control the write and read operations.  $\overline{CS}$  is the standard chip-select signal, which enables a  $\mu P$  to address the MAX197 as an I/O port. When high, it disables the  $\overline{WR}$  and  $\overline{RD}$  inputs and forces the interface into a high-Z state.

### Input Format

The control byte is latched into the device, on pins D7–D0, during a write cycle. Table 2 shows the control-byte format.

### Output Data Format

The output data format is binary in unipolar mode and two's-complement binary in bipolar mode. When reading the output data,  $\overline{CS}$ , and  $\overline{RD}$  must be low. When  $\overline{HBEN}$  is low, the lower eight bits are read. When  $\overline{HBEN}$  is high, the upper four MSBs are available and the output data bits D4–D7 are either set low (in unipolar mode) or set to the value of the MSB (in bipolar mode) (Table 6).

**Table 2. Control-Byte Format**

D7 (MSB)	D6	D5	D4	D3	D2	D1	D0 (LSB)
PD1	PD0	ACQMOD	RNG	BIP	A2	A1	A0

BIT	NAME	DESCRIPTION
7, 6	PD1, PD0	These two bits select the clock and power-down modes (Table 4).
5	ACQMOD	0 = internally controlled acquisition (6 clock cycles), 1 = externally controlled acquisition
4	RNG	Selects the full-scale voltage magnitude at the input (Table 3).
3	BIP	Selects unipolar or bipolar conversion mode (Table 3).
2, 1, 0	A2, A1, A0	These are address bits for the input mux to select the "on" channel (Table 5).

**Table 3. Range and Polarity Selection**

BIP	RNG	INPUT RANGE (V)
0	0	0 to 5
0	1	0 to 10
1	0	$\pm 5$
1	1	$\pm 10$

**Table 4. Clock and Power-Down Selection**

PD1	PD0	DEVICE MODE
0	0	Normal Operation / External Clock Mode
0	1	Normal Operation / Internal Clock Mode
1	0	Standby Power-Down (STBYPD): clock mode is unaffected
1	1	Full Power-Down (FULLPD): clock mode is unaffected

**Table 5. Channel Selection**

A2	A1	A0	CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7
0	0	0	*							
0	0	1		*						
0	1	0			*					
0	1	1				*				
1	0	0					*			
1	0	1						*		
1	1	0							*	
1	1	1								*

## Multi-Range ( $\pm 10V$ , $\pm 5V$ , $+10V$ , $+5V$ ), Single $+5V$ , 12-Bit DAS with 8+4 Bus Interface

Table 6. Data-Bus Output

PIN	HBEN = LOW	HBEN = HIGH
D0	B0 (LSB)	B8
D1	B1	B9
D2	B2	B10
D3	B3	B11 (MSB)
D4	B4	B11 (BIP = 1) / 0 (BIP = 0)
D5	B5	B11 (BIP = 1) / 0 (BIP = 0)
D6	B6	B11 (BIP = 1) / 0 (BIP = 0)
D7	B7	B11 (BIP = 1) / 0 (BIP = 0)

### How to Start a Conversion

Conversions are initiated with a write operation, which selects the mux channel and configures the MAX197 for either unipolar or bipolar input range. A write pulse ( $WR + CS$ ) can either start an acquisition interval or initiate a combined acquisition plus conversion. The sampling interval occurs at the end of the acquisition interval. The ACQMOD bit in the input control byte offers two options for acquiring the signal: internal or external. The conversion period lasts for 12 clock cycles in either internal or external clock or acquisition mode.

Writing a new control byte during conversion cycle will abort conversion and start a new acquisition interval.

### Internal Acquisition

Select internal acquisition by writing the control byte with the ACQMOD bit cleared ( $ACQMOD = 0$ ). This causes the write pulse to initiate an acquisition interval whose duration is internally timed. Conversion starts when this six-clock-cycle acquisition interval ( $3\mu s$  when  $f_{CLK} = 2MHz$ ) ends. See Figure 5.

### External Acquisition

Use the external acquisition timing mode for precise control of the sampling aperture and/or independent control of acquisition and conversion times. The user controls acquisition and start-of-conversion with two separate write pulses. The first pulse, written with  $ACQMOD = 1$ , starts an acquisition interval of indeterminate length. The second write pulse, written with  $ACQMOD = 0$ , terminates acquisition and starts conversion on  $WR$ 's rising edge (Figure 6). However, if the second control byte contains  $ACQMOD = 1$ , an indefinite acquisition interval is restarted.

The address bits for the input mux must have the same values on the first and second write pulses. Power-down mode bits ( $PD0$ ,  $PD1$ ) can assume new values on the second write pulse (see *Power-Down Mode*).

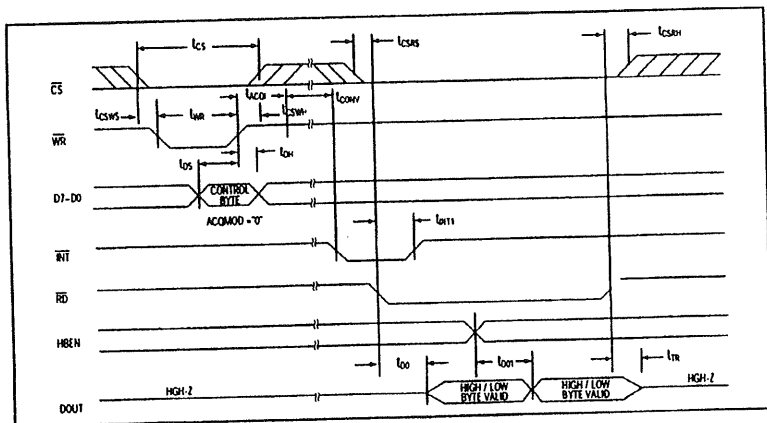


Figure 5. Conversion Timing Using Internal Acquisition Mode

# Multi-Range ( $\pm 10V$ , $\pm 5V$ , $+10V$ , $+5V$ ), Single $+5V$ , 12-Bit DAS with 8+4 Bus Interface

MAX197

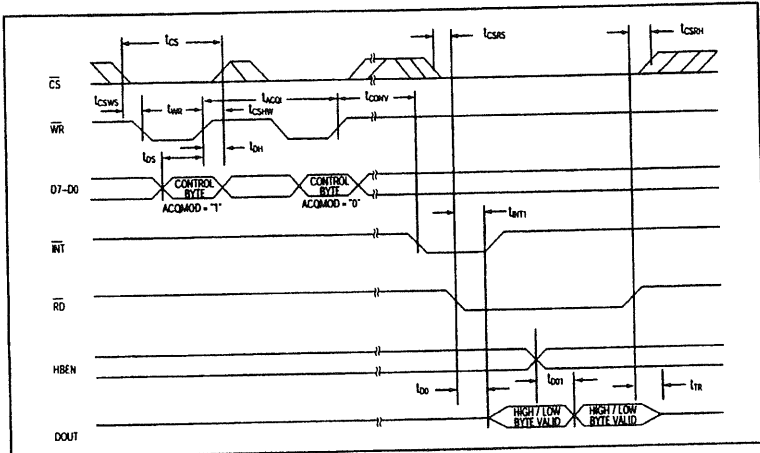


Figure 6. Conversion Timing Using External Acquisition Mode

### How to Read a Conversion

A standard interrupt signal, INT, is provided to allow the device to flag the  $\mu P$  when the conversion has ended and a valid result is available. INT goes low when conversion is complete and the output data is ready (Figures 5 and 6). It returns high on the first read cycle or if a new control byte is written.

### Clock Modes

The MAX197 operates with either an internal or an external clock. Control bits (D6, D7) select either internal or external clock mode. Once the desired clock mode is selected, changing these bits to program power-down will not affect the clock mode. In each mode, internal or external acquisition can be used. At power-up, external clock mode is selected.

### Internal Clock Mode

Select internal clock mode to free the  $\mu P$  from the burden of running the SAR conversion clock. To select this mode, write the control byte with D7 = 0 and D6 = 1. A 100pF capacitor between the CLK pin and ground sets this frequency to 1.56MHz nominal. Figure 7

shows a linear relationship between the internal clock period and the value of the external capacitor used.

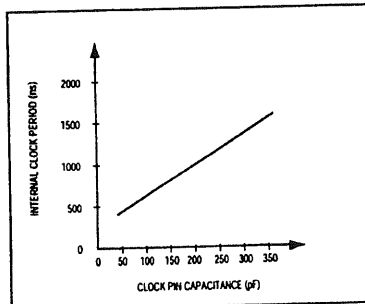
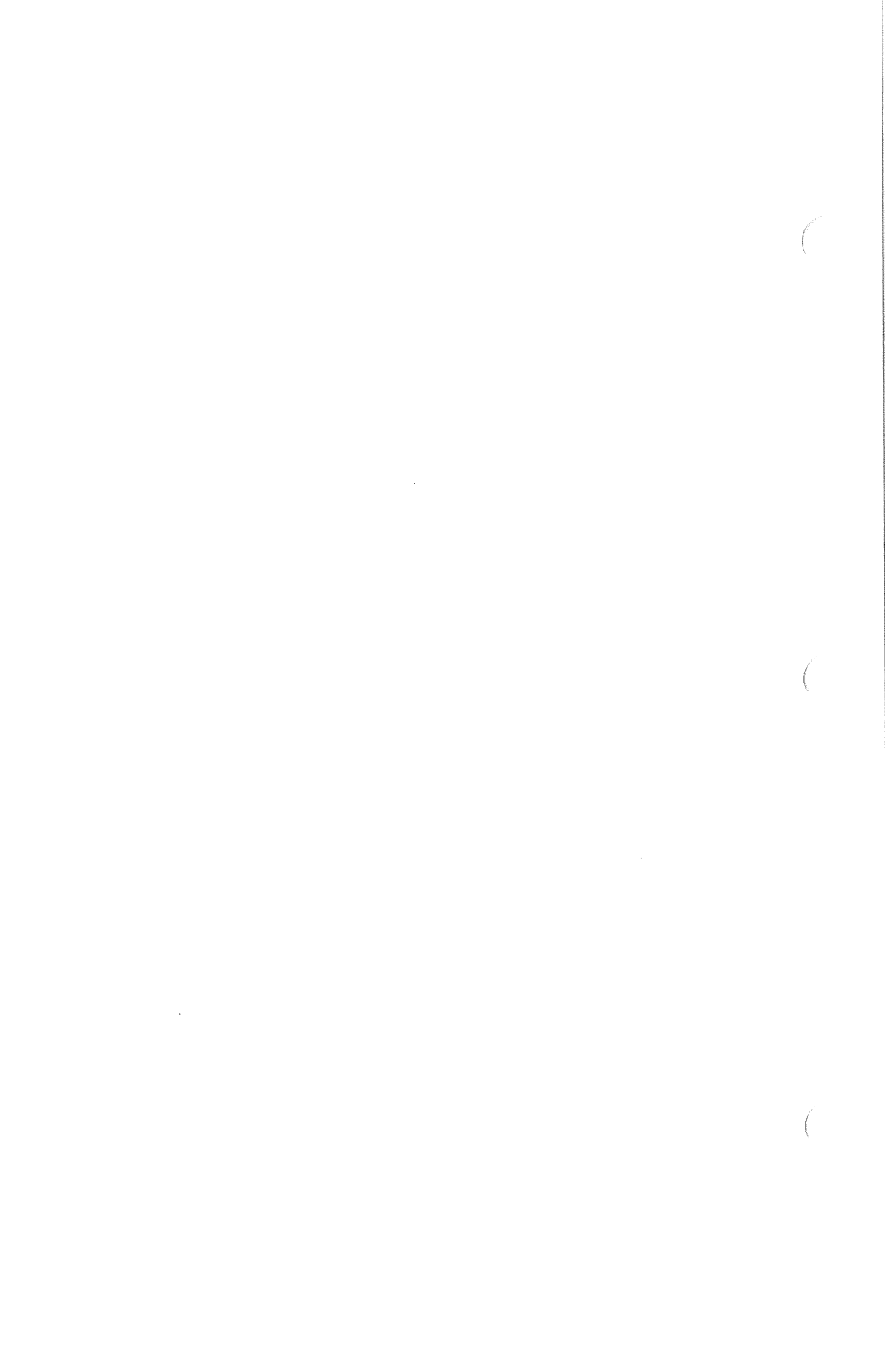


Figure 7. Internal Clock Period vs. Clock Pin Capacitance



**MAX525 12-Bit D/A Converter**



# MAXIM

## Low-Power, Quad, 12-Bit Voltage-Output DAC with Serial Interface

MAX525

### General Description

The MAX525 combines four low-power, voltage-output, 12-bit digital-to-analog converters (DACs) and four precision output amplifiers in a space-saving, 20-pin package. In addition to the four voltage outputs, each amplifier's negative input is also available to the user. This facilitates specific gain configurations, remote sensing, and high output drive capacity, making the MAX525 ideal for industrial-process-control applications. Other features include software shutdown, hardware shutdown lockout, an active-low reset which clears all registers and DACs to zero, a user-programmable logic output, and a serial-data output.

Each DAC has a double-buffered input organized as an input register followed by a DAC register. A 16-bit serial word loads data into each input/DAC register. The serial interface is compatible with SPI™/QSPI™ and Microwire™. It allows the input and DAC registers to be updated independently or simultaneously with a single software command. The DAC registers can be simultaneously updated via the 3-wire serial interface. All logic inputs are TTL/CMOS-logic compatible.

### Applications

Industrial Process Controls  
Automatic Test Equipment  
Digital Offset and Gain Adjustment  
Motion Control  
Remote Industrial Controls  
Microprocessor-Controlled Systems

### Features

- ◆ Four 12-Bit DACs with Configurable Output Amplifiers
- ◆ +5V Single-Supply Operation
- ◆ Low Supply Current: 0.85mA Normal Operation  
10µA Shutdown Mode
- ◆ Available in 20-Pin SSOP
- ◆ Power-On Reset Clears all Registers and DACs to Zero
- ◆ Capable of Recalling Last State Prior to Shutdown
- ◆ SPI/QSPI and Microwire Compatible
- ◆ Simultaneous or Independent Control of DACs via 3-Wire Serial Interface
- ◆ User-Programmable Digital Output

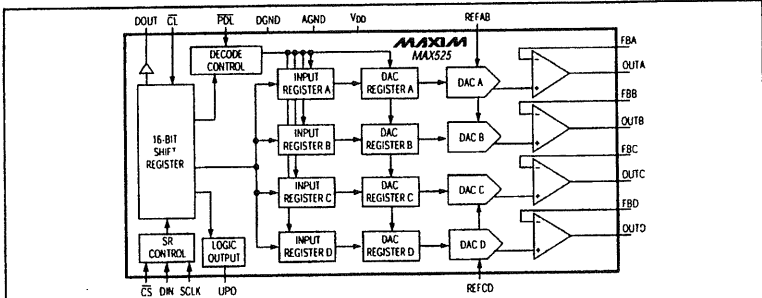
### Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE	INL (LSB)
MAX525ACPP	0 °C to +70 °C	20 Plastic DIP	±1/2
MAX525BCPP	0 °C to +70 °C	20 Plastic DIP	±1
MAX525ACAP	0 °C to +70 °C	20 SSOP	±1/2
MAX525BCAP	0 °C to +70 °C	20 SSOP	±1

Ordering information continued on last page.

Pin Configuration appears at end of data sheet.

### Functional Diagram



SPI and QSPI are trademarks of Motorola, Inc. Microwire is a trademark of National Semiconductor Corp.

MAXIM

Maxim Integrated Products 1

For free samples & the latest literature: <http://www.maxim-ic.com>, or phone 1-800-998-8800.  
For small orders, phone 408-737-7600 ext. 3468.

# Low-Power, Quad, 12-Bit Voltage-Output DAC with Serial Interface

## ABSOLUTE MAXIMUM RATINGS

V <sub>DD</sub> to AGND	.....-0.3V to +6V
V <sub>DD</sub> to DGND	.....-0.3V to +6V
AGND to DGND	.....±0.3V
REFAB, REFCD to AGND	.....-0.3V to (V <sub>DD</sub> + 0.3V)
OUT <sub>1</sub> , FB <sub>1</sub> to AGND	.....-0.3V to (V <sub>DD</sub> + 0.3V)
Digital inputs to DGND	.....-0.3V to +6V
DO <sub>OUT</sub> , LPO to DGND	.....-0.3V to (V <sub>DD</sub> + 0.3V)
Continuous Current into Any Pin	.....±20mA
Continuous Power Dissipation (T <sub>A</sub> = +70 °C)	.....640mW
Plastic DIP (derate 8.00mW/°C above +70 °C)	.....640mW
SSOP (derate 8.00mW/°C above +70 °C)	.....640mW
CERDIP (derate 11.11mW/°C above +70 °C)	.....889mW

## Operating Temperature Ranges

MAX525_C_P	.....0 °C to +70 °C
MAX525_E_P	.....40 °C to +85 °C
MAX525_MJP	.....-55 °C to +125 °C
Storage Temperature Range	.....-65 °C to +150 °C
Lead Temperature (soldering, 10sec)	.....+300 °C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## ELECTRICAL CHARACTERISTICS

(V<sub>DD</sub> = +5V ±10%, AGND = DGND = 0V, REFAB = REFCD = 2.5V, R<sub>L</sub> = 5kΩ, C<sub>L</sub> = 100pF, T<sub>A</sub> = T<sub>MIN</sub> to T<sub>MAX</sub>, unless otherwise noted. Typical values are at T<sub>A</sub> = +25 °C. Output buffer connected in unity-gain configuration (Figure 9).)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
<b>STATIC PERFORMANCE—ANALOG SECTION</b>						
Resolution	N		12			Bits
Integral Nonlinearity (Note 1)	INL	MAX525A		±0.25	±0.5	LSB
		MAX525B			±1.0	
Differential Nonlinearity	DNL	Guaranteed monotonic			±1.0	LSB
Offset Error	Vos				±6.0	mV
Offset-Error Tempco				6		ppm/°C
Gain Error (Note 1)	GE			-0.8	±2.0	LSB
Gain-Error Tempco				1		ppm/°C
Power-Supply Rejection Ratio	PSRR	4.5V ≤ V <sub>DD</sub> ≤ 5.5V		100	600	μV/V
<b>MATCHING PERFORMANCE (T<sub>A</sub> = +25 °C)</b>						
Gain Error	GE			-0.8	±2.0	LSB
Offset Error				±1.0	±6.0	mV
Integral Nonlinearity	INL			±0.35	±1.0	LSB
<b>REFERENCE INPUT</b>						
Reference Input Range	V <sub>REF</sub>		0		V <sub>DD</sub> - 1.4	V
Reference Input Resistance	R <sub>REF</sub>	Code-dependent, minimum at code 555 hex	10			kΩ
Reference Current in Shutdown				0.01	±1	μA

## Low-Power, Quad, 12-Bit Voltage-Output DAC with Serial Interface

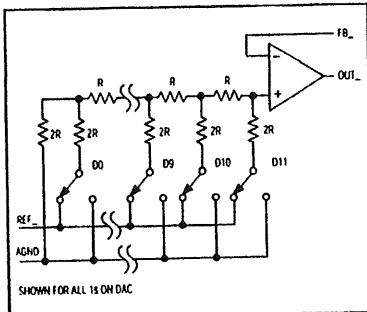


Figure 1. Simplified DAC Circuit Diagram

### Detailed Description

The MAX525 contains four 12-bit, voltage-output digital-to-analog converters (DACs) that are easily addressed using a simple 3-wire serial interface. It includes a 16-bit data-in/data-out shift register, and each DAC has a doubled-buffered input composed of an input register and a DAC register (see *Functional Diagram*). In addition to the four voltage outputs, each amplifier's negative input is available to the user.

The DACs are inverted R-2R ladder networks that convert 12-bit digital inputs into equivalent analog output voltages in proportion to the applied reference voltage inputs. DACs A and B share the REFAB reference input, while DACs C and D share the REFCD reference input. The two reference inputs allow different full-scale output voltage ranges for each pair of DACs. Figure 1 shows a simplified circuit diagram of one of the four DACs.

#### Reference Inputs

The two reference inputs accept positive DC and AC signals. The voltage at each reference input sets the full-scale output voltage for its two corresponding DACs. The reference input voltage range is 0V to (VDD - 1.4V). The output voltages (VOUT<sub>n</sub>) are represented by a digitally programmable voltage source as:

$$VOUT_n = (VREF \times NB / 4096) \times \text{Gain}$$

where NB is the numeric value of the DAC's binary input code (0 to 4095), VREF is the reference voltage, and Gain is the externally set voltage gain.

The impedance at each reference input is code-dependent, ranging from a low value of 10kΩ when both DACs connected to the reference have an input code of 555 hex, to a high value exceeding several gigohms (leakage currents) with an input code of 000 hex. Because the input impedance at the reference pins is code-dependent, load regulation of the reference source is important.

The REFAB and REFCD reference inputs have a 10kΩ guaranteed minimum input impedance. When the two reference inputs are driven from the same source, the effective minimum impedance is 5kΩ. A voltage reference with a load regulation of 6ppm/mA, such as the MAX873, would typically deviate by 0.025LSB (0.061LSB worst case) when driving both MAX525 reference inputs simultaneously at 2.5V. Driving the REFAB and REFCD pins separately improves reference accuracy.

In shutdown mode, the MAX525's REFAB and REFCD inputs enter a high-impedance state with a typical input leakage current of 0.01μA.

The reference input capacitance is also code dependent and typically ranges from 20pF with an input code of all 0s to 100pF with an input code of all 1s.

#### Output Amplifiers

All MAX525 DAC outputs are internally buffered by precision amplifiers with a typical slew rate of 0.6V/μs. Access to the inverting input of each output amplifier provides the user greater flexibility in output gain setting/signal conditioning (see the *Applications Information* section).

With a full-scale transition at the MAX525 output, the typical settling time to  $\pm 1/2$ LSB is 12μs when loaded with 5kΩ in parallel with 100pF (loads less than 2kΩ degrade performance).

The MAX525 output amplifier's output dynamic responses and settling performances are shown in the *Typical Operating Characteristics*.

#### Power-Down Mode

The MAX525 features a software-programmable shutdown that reduces supply current to a typical value of 10μA. The power-down lockout (PDL) pin must be high to enable the shutdown mode. Writing 1100XXXXXXX as the input-control word puts the MAX525 in power-down mode (Table 1).

# Low-Power, Quad, 12-Bit Voltage-Output DAC with Serial Interface

**MAX525**

In power-down mode, the MAX525 output amplifiers and the reference inputs enter a high-impedance state. The serial interface remains active. Data in the input registers is retained in power-down, allowing the MAX525 to recall the output states prior to entering shutdown. Start up from power-down either by recalling the previous configuration or by updating the DACs with new data. When powering up the device or bringing it out of shutdown, allow 15µs for the outputs to stabilize.

### Serial-Interface Configurations

The MAX525's 3-wire serial interface is compatible with both Microwire™ (Figure 2) and SPI™/QSPI™ (Figure 3). The serial input word consists of two address bits and two control bits followed by 12 data bits (MSB first), as shown in Figure 4. The 4-bit address/control code determines the MAX525's response outlined in Table 1. The connection between DOUT and the serial-interface port is not necessary, but may be used for data echo. Data held in the MAX525's shift register can be shifted out of DOUT and returned to the microprocessor (µP) for data verification.

The MAX525's digital inputs are double buffered. Depending on the command issued through the serial interface, the input register(s), the DAC register(s) can be loaded directly, or all four DAC registers can be updated simultaneously from the input registers (Table 1)

### Serial-Interface Description

The MAX525 requires 16 bits of serial data. Table 1 lists the serial-interface programming commands. For certain commands, the 12 data bits are "don't cares." Data is sent MSB first and can be sent in two 8-bit packets or one 16-bit word (CS must remain low until 16 bits are transferred). The serial data is composed of two DAC address bits (A1, A0) and two control bits (C1, C0), followed by the 12 data bits D11...D0 (Figure 4). The 4-bit address/control code determines:

- The register(s) to be updated
- The clock edge on which data is to be clocked out via the serial-data output (DOUT)
- The state of the user-programmable logic output (UPO)
- If the part is to go into shutdown mode (assuming PDL is high)
- How the part is configured when coming out of shutdown mode.

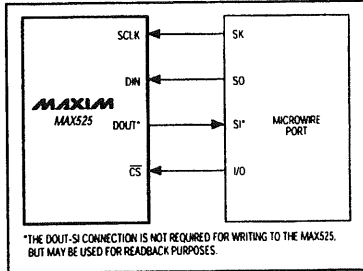


Figure 2. Connections for Microwire

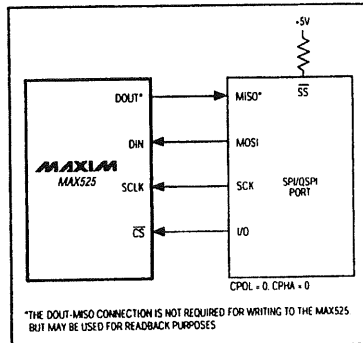


Figure 3. Connections for SPI/QSPI

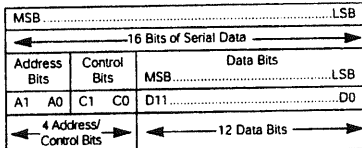


Figure 4. Serial-Data Format

# Low-Power, Quad, 12-Bit Voltage-Output DAC with Serial Interface

MAX525

**Table 1. Serial-Interface Programming Commands**

16-BIT SERIAL WORD					FUNCTION
A1	A0	C1	C0	D11.....D0 MSB                  LSB	
0	0	0	1	12-bit DAC data	Load input register A; DAC registers unchanged.
0	1	0	1	12-bit DAC data	Load input register B; DAC registers unchanged.
1	0	0	1	12-bit DAC data	Load input register C; DAC registers unchanged.
1	1	0	1	12-bit DAC data	Load input register D; DAC registers unchanged.
0	0	1	1	12-bit DAC data	Load input register A; all DAC registers updated.
0	1	1	1	12-bit DAC data	Load input register B; all DAC registers updated.
1	0	1	1	12-bit DAC data	Load input register C; all DAC registers updated.
1	1	1	1	12-bit DAC data	Load input register D; all DAC registers updated.
0	1	0	0	XXXXXXXXXXXX	Update all DAC registers from their respective input registers (start-up)
1	0	0	0	12-bit DAC data	Load all DAC registers from shift register (start-up)
1	1	0	0	XXXXXXXXXXXX	Shutdown (provided PDL = 1)
0	0	1	0	XXXXXXXXXXXX	UPO goes low (default)
0	1	1	0	XXXXXXXXXXXX	UPO goes high
0	0	0	0	XXXXXXXXXXXX	No operation (NOP) to DAC registers
1	1	1	0	XXXXXXXXXXXX	Mode 1, DOUT clocked out on SCLK's rising edge. All DAC registers updated.
1	0	1	0	XXXXXXXXXXXX	Mode 0, DOUT clocked out on SCLK's falling edge. All DAC registers updated (default).

\*X = Don't care

Figure 5 shows the serial-interface timing requirements. The chip-select pin ( $\overline{CS}$ ) must be low to enable the DAC's serial interface. When  $\overline{CS}$  is high, the interface control circuitry is disabled.  $\overline{CS}$  must go low at least  $t_{CSS}$  before the rising serial clock (SCLK) edge to properly clock in the first bit. When  $\overline{CS}$  is low, data is clocked into the internal shift register via the serial-data input pin (DIN) on SCLK's rising edge. The maximum guaranteed clock frequency is 10MHz. Data is latched into the appropriate MAX525 input/DAC registers on  $\overline{CS}$ 's rising edge.

The programming command Load-All-DACs-From-Shift-Register allows all input and DAC registers to be simultaneously loaded with the same digital code from the input shift register. The no operation (NOP) command leaves the register contents unaffected and is useful when the MAX525 is configured in a daisy chain (see the *Daisy Chaining Devices* section). The command to

change the clock edge on which serial data is shifted out of DOUT also loads data from all input registers to their respective DAC registers.

### Serial-Data Output (DOUT)

The serial-data output, DOUT, is the internal shift register's output. The MAX525 can be programmed so that data is clocked out of DOUT on SCLK's rising edge (Mode 1) or falling edge (Mode 0). In Mode 0, output data at DOUT lags input data at DIN by 16.5 clock cycles, maintaining compatibility with Microwire™, SPI™/QSPI™, and other serial interfaces. In Mode 1, output data lags input data by 16 clock cycles. On power-up, DOUT defaults to Mode 0 timing.

### User-Programmable Logic Output (UPO)

The user-programmable logic output, UPO, allows an external device to be controlled via the MAX525 serial interface (Table 1).





**Table 4-2. Peripheral Control Block**

PCB Offset	Function	PCB Offset	Function	PCB Offset	Function	PCB Offset	Function
00H	Reserved	40H	T2CNT	80H	GCS0ST	C0H	Reserved
02H	EOI	42H	T2CMPA	82H	GCS0SP	C2H	Reserved
04H	POLL	44H	Reserved	84H	GCS1ST	C4H	Reserved
06H	POLLSTS	46H	T2CON	86H	GCS1SP	C6H	Reserved
08H	IMASK	48H	Reserved	88H	GCS2ST	C8H	Reserved
0AH	PRIMSK	4AH	Reserved	8AH	GCS2SP	CAH	Reserved
0CH	INSERV	4CH	Reserved	8CH	GCS3ST	CCH	Reserved
0EH	REQST	4EH	Reserved	8EH	GCS3SP	CEH	Reserved
10H	INSTS	50H	P1DIR	90H	GCS4ST	D0H	Reserved
12H	TCUCON	52H	P1PIN	92H	GCS4SP	D2H	Reserved
14H	SCUCON	54H	P1CON	94H	GCS5ST	D4H	Reserved
16H	I4CON	56H	P1LTCH	96H	GCS5SP	D6H	Reserved
18H	I0CON	58H	P2DIR	98H	GCS6ST	D8H	Reserved
1AH	I1CON	5AH	P2PIN	9AH	GCS6SP	DAH	Reserved
1CH	I2CON	5CH	P2CON	9CH	GCS7ST	DCH	Reserved
1EH	I3CON	5EH	P2LTCH	9EH	GCS7SP	DEH	Reserved
20H	Reserved	60H	B0CMP	A0H	LCSST	E0H	Reserved
22H	Reserved	62H	B0CNT	A2H	LCSSP	E2H	Reserved
24H	Reserved	64H	S0CON	A4H	UCSST	E4H	Reserved
26H	Reserved	66H	S0STS	A6H	UCSSP	E6H	Reserved
28H	Reserved	68H	S0RBUF	A8H	RELREG	E8H	Reserved
2AH	Reserved	6AH	S0TBUF	AAH	Reserved	EAH	Reserved
2CH	Reserved	6CH	Reserved	ACH	Reserved	ECH	Reserved
2EH	Reserved	6EH	Reserved	AEH	Reserved	EEH	Reserved
30H	T0CNT	70H	B1CMP	B0H	RFBASE	F0H	Reserved
32H	T0CMPA	72H	B1CNT	B2H	RFTIME	F2H	Reserved
34H	T0CMPB	74H	S1CON	B4H	RFCON	F4H	Reserved
36H	T0CON	76H	S1STS	B6H	RFADDR	F6H	Reserved
38H	T1CNT	78H	S1RBUF	B8H	PWRCON	F8H	Reserved
3AH	T1CMPA	7AH	S1TBUF	BAH	Reserved	FAH	Reserved
3CH	T1CMPB	7CH	Reserved	BCH	STEPID	FCH	Reserved
3EH	T1CON	7EH	Reserved	BEH	Reserved	FEH	Reserved

### 4.3 RESERVED LOCATIONS

Many locations within the Peripheral Control Block are not assigned to any peripheral. Unused locations are reserved. Reading from these locations yields an undefined result. If reserved registers are written (for example, during a block MOV instruction) they must be set to 0H.

#### NOTE

Failure to follow this guideline could result in incompatibilities with future 80C186 Modular Core family products.

### 4.4 ACCESSING THE PERIPHERAL CONTROL BLOCK

All communication between integrated peripherals and the Modular CPU Core occurs over a special bus, called the *F-Bus*, which always carries 16-bit data. The Peripheral Control Block, like all integrated peripherals, is always accessed 16 bits at a time.

#### 4.4.1 Bus Cycles

The processor runs an external bus cycle for any memory or I/O cycle accessing a location within the Peripheral Control Block. Address, data and control information is driven on the external pins as with an ordinary bus cycle. Information returned by an external device is ignored, even if the access does not correspond to the location of an integrated peripheral control register. This is also true for the 80C188 Modular Core family, except that word accesses made to integrated registers are performed in two bus cycles.

#### 4.4.2 READY Signals and Wait States

The processor generates an internal READY signal whenever an integrated peripheral is accessed. External READY is ignored. READY is also generated if an access is made to a location within the Peripheral Control Block that does not correspond to an integrated peripheral control register. For accesses to timer control and counting registers, the processor inserts one wait state. This is required to properly multiplex processor and counter element accesses to the timer control registers. For accesses to the remaining locations in the Peripheral Control Block, the processor does not insert wait states.



### 4.4.3 F-Bus Operation

The F-Bus functions differently than the external data bus for byte and word accesses. All write transfers on the F-Bus occur as words, regardless of how they are encoded. For example, the instruction `OUT DX, AL` (`DX` is even) will write the entire `AX` register to the Peripheral Control Block register at location `[DX]`. If `DX` were an odd location, `AL` would be placed in `[DX]` and `AH` would be placed at `[DX-1]`. A word operation to an odd address would write `[DX]` and `[DX-1]` with `AL` and `AH`, respectively. This differs from normal external bus operation where unaligned word writes modify `[DX]` and `[DX+1]`. In summary, do not use odd-aligned byte or word writes to the PCB.

Aligned word reads work normally. Unaligned word reads work differently. For example, `IN AX, DX` (`DX` is odd) will transfer `[DX]` into `AL` and `[DX-1]` into `AH`. Byte reads from even or odd addresses work normally, but only a byte will be read. For example, `IN AL, DX` will not transfer `[DX]` into `AX` (only `AL` is modified).

No problems will arise if the following recommendations are adhered to.

<b>Word reads</b>	Aligned word reads of the PCB work normally. Access only even-aligned words with <code>IN AX, DX</code> or <code>MOV word register, even PCB address</code> .
<b>Byte reads</b>	Byte reads of the PCB work normally. Beware of reading word-wide PCB registers that may change value between successive reads (e.g., timer count value).
<b>Word writes</b>	<p>Always write even-aligned words to the PCB. Writing an odd-aligned word will give unexpected results.</p> <p>For the 80C186 Modular Core, use either</p> <ul style="list-style-type: none"> <li>- <code>OUT DX, AX</code> or</li> <li>- <code>OUT DX, AL</code> or</li> <li>- <code>MOV even PCB address, word register</code>.</li> </ul> <p>For the 80C188 Modular Core, using <code>OUT DX, AX</code> will perform an unnecessary bus cycle and is not recommended. Use either</p> <ul style="list-style-type: none"> <li>- <code>OUT DX, AL</code> or</li> <li>- <code>MOV even-aligned byte PCB address, byte register low byte</code>.</li> </ul>
<b>Byte writes</b>	Always use even-aligned byte writes to the PCB. Even-aligned byte writes will modify the entire word PCB location. Do not perform unaligned byte writes to the PCB.

**4.4.3.1 Writing the PCB Relocation Register**

Whenever mapping the Peripheral Control Block to another location, the user should program the Relocation Register with a byte write (i.e., OUT DX, AL). Internally, the Relocation Register is written with 16 bits of the AX register, while externally the Bus Interface Unit runs a single 8-bit bus cycle. If a word instruction (i.e., OUT DX, AX) is used with an 80C188 Modular Core family member, the Relocation Register is written on the first bus cycle. The Bus Interface Unit then runs an unnecessary second bus cycle. The address of the second bus cycle is no longer within the control block, since the Peripheral Control Block was moved on the first cycle. External READY- must now be generated to complete the cycle. For this reason, we recommend byte operations for the Relocation Register.

**4.4.3.2 Accessing the Peripheral Control Registers**

Byte instructions should be used for the registers in the Peripheral Control Block of an 80C188 Modular Core family member. This requires half the bus cycles of word operations. Byte operations are valid only for even-addressed writes to the Peripheral Control Block. A word read (e.g., IN AX, DX) must be performed to read a 16-bit Peripheral Control Block register.

**4.4.3.3 Accessing Reserved Locations**

Unused locations are reserved. If a write is made to these locations, a bus cycle occurs, but data is not stored. If a subsequent read is made to the same location, the value written is not read back. If reserved registers are written (for example, during a block MOV instruction) they must be set to 0H.

**NOTE**

Failure to follow this guideline could result in incompatibilities with future 80C186 Modular Core family products.

**4.5 SETTING THE PCB BASE LOCATION**

Upon reset, the PCB Relocation Register (see Figure 4-1 on page 4-2) contains the value 00FFH, which causes the Peripheral Control Block to be located at the top of I/O space (0FF00H to 0FFFFH). Writing the PCB Relocation Register allows the user to change that location.



As an example, to relocate the Peripheral Control Block to the memory range 10000-100FFH, the user would program the PCB Relocation Register with the value 1100H. Since the Relocation Register is part of the Peripheral Control Block, it relocates to word 10000H plus its fixed offset.

**NOTE**

Due to an internal condition, external ready is ignored if the device is configured in Cascade mode and the Peripheral Control Block (PCB) is located at 0000H in I/O space. In this case, wait states **cannot** be added to interrupt acknowledge bus cycles. However, you **can** add wait states to interrupt acknowledge cycles if the PCB is located at any other address.

**4.5.1 Considerations for the 80C187 Math Coprocessor Interface**

Systems using the 80C187 math coprocessor interface must **not** relocate the Peripheral Control Block to location 0000H in I/O space. The 80C187 interface uses I/O locations 0F8H through 0FFH. If the Peripheral Control Block resides in these locations, the processor communicates with the Peripheral Control Block, **not** the 80C187 interface circuitry.

**NOTE**

If the PCB is located at 0000H in I/O space and access to the math coprocessor interface is enabled (the Escape Trap bit is clear), a numerics (ESC) instruction causes indeterminate system operation.

The 8-bit bus version of the device automatically traps an ESC instruction to the Type 7 interrupt, regardless of the state of the Escape Trap (ET) bit.

For details on the math coprocessor interface, see Chapter 12, "Math Coprocessing."



**intel®**

**6**

## **Chip-Select Unit**

**1**





## CHAPTER 6 CHIP-SELECT UNIT

Every system requires some form of component-selection mechanism to enable the CPU to access a specific memory or peripheral device. The signal that selects the memory or peripheral device is referred to as a chip-select. Besides selecting a specific device, each chip-select can be used to control the number of wait states inserted into the bus cycle. Devices that are too slow to keep up with the maximum bus bandwidth can use wait states to slow the bus down.

### 6.1 COMMON METHODS FOR GENERATING CHIP-SELECTS

One method of generating chip-selects uses latched address signals directly. An example interface is shown in Figure 6-1(A). In the example, an inverted A16 is connected to an SRAM device with an active-low chip-select. Any bus cycle with an address between 10000H and 1FFFFH (A16 = 1) enables the SRAM device. Also note that any bus cycle with an address starting at 30000H, 50000H, 70000H and so on also selects the SRAM device.

Decoding more address bits solves the problem of a chip-select being active over multiple address ranges. In Figure 6-1(B), a one-of-eight decoder is connected to the uppermost address bits. Each decoded output is active for one-eighth of the 1 Mbyte address space. However, each chip-select has a fixed starting address and range. Future system memory changes could require circuit changes to accommodate the additional memory.

### 6.2 CHIP-SELECT UNIT FEATURES AND BENEFITS

The Chip-Select Unit overcomes limitations of the designs shown in Figure 6-1 and has the following features:

- Ten chip-select outputs
- Programmable start and stop addresses
- Memory or I/O bus cycle decoder
- Programmable wait-state generator
- Provision to disable a chip-select
- Provision to override bus ready

Figure 6-2 illustrates the logic blocks that generate a chip-select. Each chip-select has a duplicate set of logic.

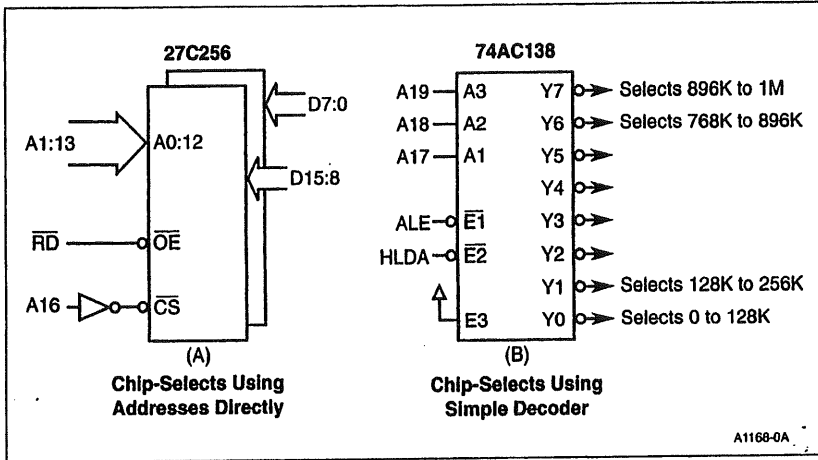


Figure 6-1. Common Chip-Select Generation Methods

### 6.3 CHIP-SELECT UNIT FUNCTIONAL OVERVIEW

The Chip-Select Unit (CSU) decodes bus cycle address and status information and enables the appropriate chip-select. Figure 6-3 illustrates the timing of a chip-select during a bus cycle. Note that the chip-select goes active in the same bus state as address goes active, eliminating any delay through address latches and decoder circuits. The Chip-Select Unit activates a chip-select for bus cycles initiated by the CPU or Refresh Control Unit.

Any of the ten chip-selects can map into either memory or I/O address space. A memory-mapped chip-select can start and end on any 1 Kbyte address location. An I/O-mapped chip-select can start and end on any 64 byte address location. The chip-selects typically associate with memory and peripheral devices as follows:

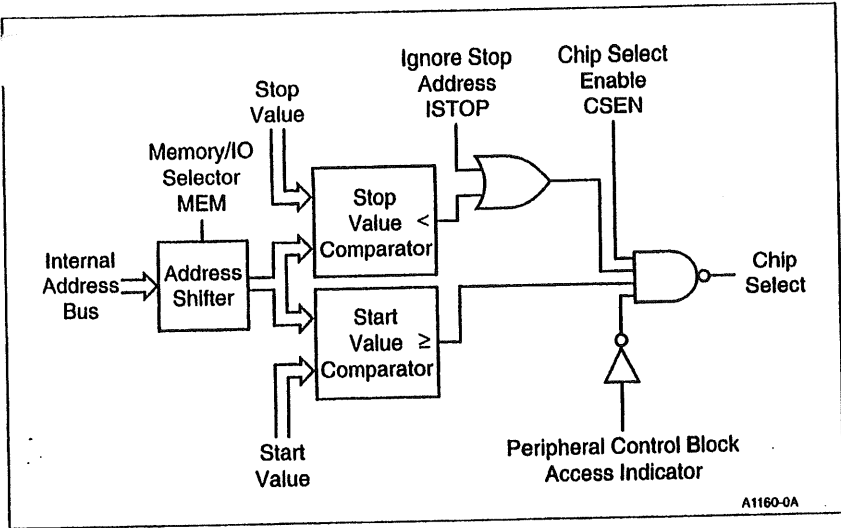


Figure 6-2. Chip-Select Block Diagram

- $\overline{CS}$  Mapped to the upper memory address space; selects the BOOT memory device (EPROM or Flash memory types).
- $\overline{LCS}$  Mapped to the lower memory address space; selects a static memory (SRAM) device that stores the interrupt vector table, local stack and data and scratch pad data.
- $\overline{GCS7:0}$  Mapped to memory or I/O address space; selects additional SRAM memory, DRAM memory, local peripherals, system bus, etc.

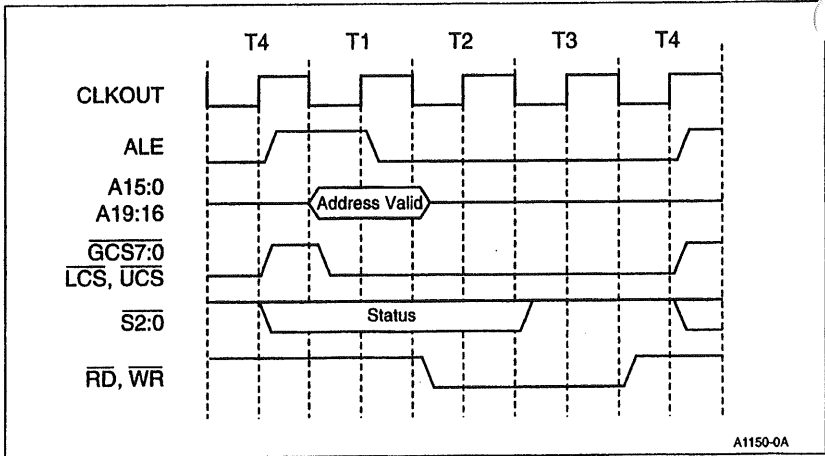


Figure 6-3. Chip-Select Relative Timings

A chip-select goes active when it meets all of the following criteria:

1. The chip-select is enabled.
2. The bus cycle status matches the programmed type (memory or I/O).
3. The bus cycle address is equal to or greater than the start address value.
4. The bus cycle address is less than the stop address value or the stop address is ignored.
5. The bus cycle is **not** accessing the Peripheral Control Block.

A memory address applies to memory read, memory write and instruction prefetch bus cycles. An I/O address applies to I/O read and I/O write bus cycles. Interrupt acknowledge and HALT bus cycles never activate a chip-select, regardless of the address generated.

After power-on or system reset, only the  $\overline{UCS}$  chip-select is initialized and active (see Figure 6-4).

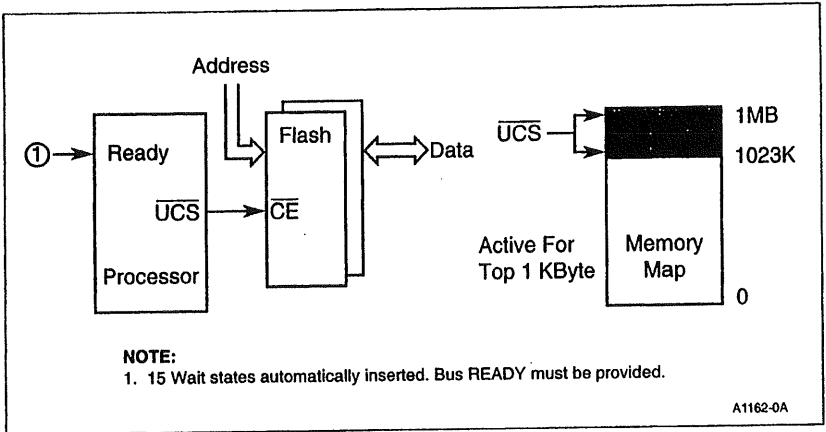


Figure 6-4. UCS Reset Configuration

#### 6.4 PROGRAMMING

Two registers, START and STOP, determine the operating characteristics of each chip-select. The Peripheral Control Block defines the location of the Chip-Select Unit registers. lists the registers and their associated programming names.

Table 6-1. Chip-Select Unit Registers

START Register Mnemonic	STOP Register Mnemonic	Chip-Select Affected
GCS0ST	GCS0SP	GCS0
GCS1ST	GCS1SP	GCS1
GCS2ST	GCS2SP	GCS2
GCS3ST	GCS3SP	GCS3
GCS4ST	GCS4SP	GCS4
GCS5ST	GCS5SP	GCS5
GCS6ST	GCS6SP	GCS6
GCS7ST	GCS7SP	GCS7
UCSST	UCSSP	UCS
LCSSST	LCSSP	LCS

The **START** register (Figure 6-5) defines the starting address and the wait state requirements. The **STOP** register (Figure 6-6) defines the ending address and the bus ready, bus cycle and enable requirements.

### **6.4.1 Initialization Sequence**

Chip-selects do not have to be initialized in any specific order. However, the following guidelines help prevent a system failure.

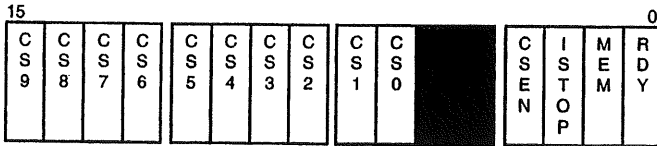
1. Initialize local memory chip-selects
2. Initialize local peripheral chip-selects
3. Perform local diagnostics
4. Initialize off-board memory and peripheral chip-selects
5. Complete system diagnostics

An unmasked interrupt or NMI must not occur until the interrupt vector addresses have been written to memory. Failure to prevent an interrupt from occurring during initialization will cause a system failure. Use external logic to generate the chip-select if interrupts cannot be masked prior to initialization.





**Register Name:** Chip-Select Stop Register  
**Register Mnemonic:** UCSSP, LCSSP, GCSxSP (x=0-7)  
**Register Function:** Defines chip-select stop address and other control functions.



A1164-0A

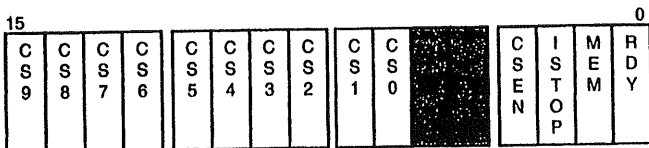
Bit Mnemonic	Bit Name	Reset State	Function
CS9:0	Stop Address	3FFH	Defines the ending address for the chip-select. CS9:0 are compared with the A19:10 (memory bus cycles) or A15:6 (I/O bus cycles) address bits. A less than result enables the chip-select. CS9:0 are ignored if ISTOP is set.
CSEN	Chip-Select Enable	0 (Note)	Disables the chip-select when cleared. Setting CSEN enables the chip-select.
ISTOP	Ignore Stop Address	0 (Note)	Setting this bit disables stop address checking, which automatically sets the ending address at 0FFFFH (memory) or 0FFFFH (I/O). When ISTOP is cleared, the stop address requirements must be met to enable the chip-select.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products. The reset state of CSEN and ISTOP is '1' for the UCSSP register.

**Figure 6-6. STOP Register Definition**



**Register Name:** Chip-Select Stop Register  
**Register Mnemonic:** UCSSP, LCSSP, GCSxSP (x=0-7)  
**Register Function:** Defines chip-select stop address and other control functions.



A1164-0A

Bit Mnemonic	Bit Name	Reset State	Function
MEM	Bus Cycle Selector	1	When MEM is set, the chip-select goes active for memory bus cycles. Clearing MEM activates the chip-select for I/O bus cycles. MEM defines which address bits are used by the start and stop address comparators. When MEM is cleared, address bits A15:6 are routed to the comparators. When MEM is set, address bits A19:10 are routed to the comparators.
RDY	Bus Ready Enable	1	Setting RDY requires that bus ready be active to complete a bus cycle. Bus ready is ignored when RDY is cleared. RDY must be set to extend wait states beyond the number determined by WS3:0.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products. The reset state of CSEN and ISTOP is '1' for the UCSSP register.

**Figure 6-6. STOP Register Definition (Continued)**

The correct sequence to program a non-enabled chip-select is as follows. (If the chip-select is already enabled, either reverse the sequence or disable the chip-select before reprogramming it.)

1. Program the START register
2. Program the STOP register



**6.4.2 Start Address**

The START register of each chip-select defines its starting (base) address. The start address value is compared to the ten most-significant address bits of the bus cycle. A bus cycle whose ten most-significant address bits are equal to or greater than the start address value causes the chip-select to go active. Table 6-3 defines the address bits that are compared with the start address value for memory and I/O bus cycles.

It is **not** possible to have a chip-select start on any arbitrary byte boundary. A chip-select configured for memory accesses can start only on multiples of 1 Kbyte. A chip-select configured for I/O accesses can start only on multiples of 64 bytes. The equations below calculate the physical start address for a given start address value.

For memory accesses:  $\text{Start Value (Decimal)} \times 1024 = \text{Physical Start Address (Decimal)}$

For I/O accesses:  $\text{Start Value (Decimal)} \times 64 = \text{Physical Start Address (Decimal)}$

**Table 6-3. Memory and I/O Compare Addresses**

Address Space	Address Range	Number of Bits	Comparator Input	Resolution
Memory	1 Mbyte	20	A19:A10	1 Kbyte
I/O	64 Kbyte	16	A15:A6	64 Bytes

**6.4.3 Stop Address**

The STOP register of each chip-select defines its ending address. The stop address value is compared to the ten most-significant address bits of the bus cycle. A bus cycle whose ten most-significant bits of address are less than the stop address value causes the chip-select to go active. Table 6-3 defines the address bits that are compared with the stop address value for memory and I/O bus cycles.

It is **not** possible to have a chip-select end on any arbitrary byte boundary. A chip-select configured for memory accesses can end only on multiples of 1 Kbyte. A chip-select configured for I/O accesses can end only on multiples of 64 bytes. The equations below define the ending address for the chip-select.

For memory accesses:  $(\text{Stop Value (Decimal)} \times 1024) - 1 = \text{Physical Ending Address (Decimal)}$

For I/O accesses:  $(\text{Stop Value (Decimal)} \times 64) - 1 = \text{Physical Ending Address (Decimal)}$



In the previous equations, a stop value of 1023 (03FFH) results in a physical ending address of 0FFBFFH (memory) or 0FFBFH (I/O). These addresses do **not** represent the top of the memory or I/O address space. To have a chip-select enabled to the end of the physical address space, the ISTOP control bit must be set. The ISTOP control bit overrides the stop address comparator output (see Figure 6-2 on page 6-3).

#### 6.4.4 Enabling and Disabling Chip-Selects

The ability to enable or disable a chip-select is important when multiple memory devices share (or can share) the same physical address space. Examples of where two or more devices would occupy the same address space include shadowed memory, bank switching and paging.

The STOP register holds the CSEN control bit, which determines whether the chip-select should go active. A chip-select never goes active if its CSEN control bit is cleared.

Chip-selects can be disabled by programming the stop address value less than the start address value or by programming the start address value greater than the stop address value. However, the ISTOP control bit cannot be set when chip-selects are disabled in this manner.

#### 6.4.5 Bus Wait State and Ready Control

Normally, the bus ready input must be inactive at the appropriate time to insert wait states into the bus cycle. The Chip-Select Unit can ignore the state of the bus ready input to extend and complete the bus cycle automatically. Most memory and peripheral devices operate properly using fifteen or fewer wait states. However, accessing such devices as a dual-port memory, an expansion bus interface, a system bus interface or remote peripheral devices can require more than fifteen wait states to complete a bus cycle.

The START register holds a four-bit value (WS3:0) that defines the number of wait states to insert into the bus cycle. Figure 6-7 shows a simplified logic diagram of the wait state and ready control functions.

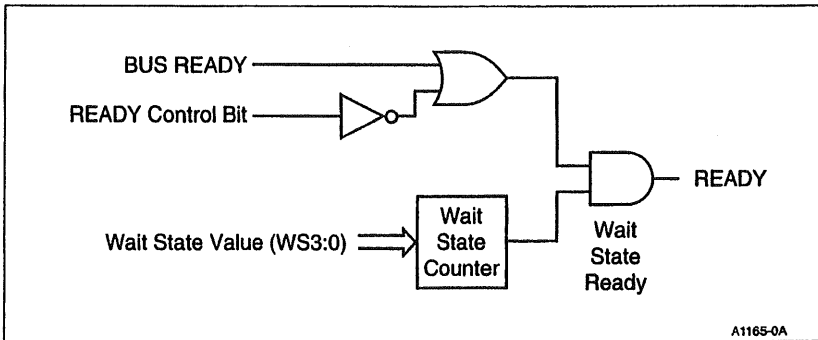


Figure 6-7. Wait State and Ready Control Functions

The STOP register defines the RDY control bit to extend bus cycles beyond fifteen wait states. The RDY control bit determines whether the bus cycle should complete normally (i.e., require bus ready) or unconditionally (i.e., ignore bus ready). Chip-selects connected to devices requiring fifteen wait states or fewer can program RDY inactive to automatically complete the bus cycle. Devices that may require more than fifteen wait states must program RDY active.

A bus cycle with wait states automatically inserted cannot be shortened. A bus cycle that ignores bus ready cannot be lengthened.

#### 6.4.6 Overlapping Chip-Selects

The Chip-Select Unit activates all enabled chip-selects programmed to cover the same physical address space. This is true if any portion of the chip-selects' address ranges overlap (i.e., chip-selects' ranges do not need to overlap completely to all go active). There are various reasons for overlapping chip-selects. For example, a system might have a need for overlapping a portion of read-only memory with read/write memory or copying data to two devices simultaneously.

If overlapping chip-selects do not have identical wait state and bus ready programming, the Chip-Select Unit will adjust itself based on the criteria shown in Figure 6-8.

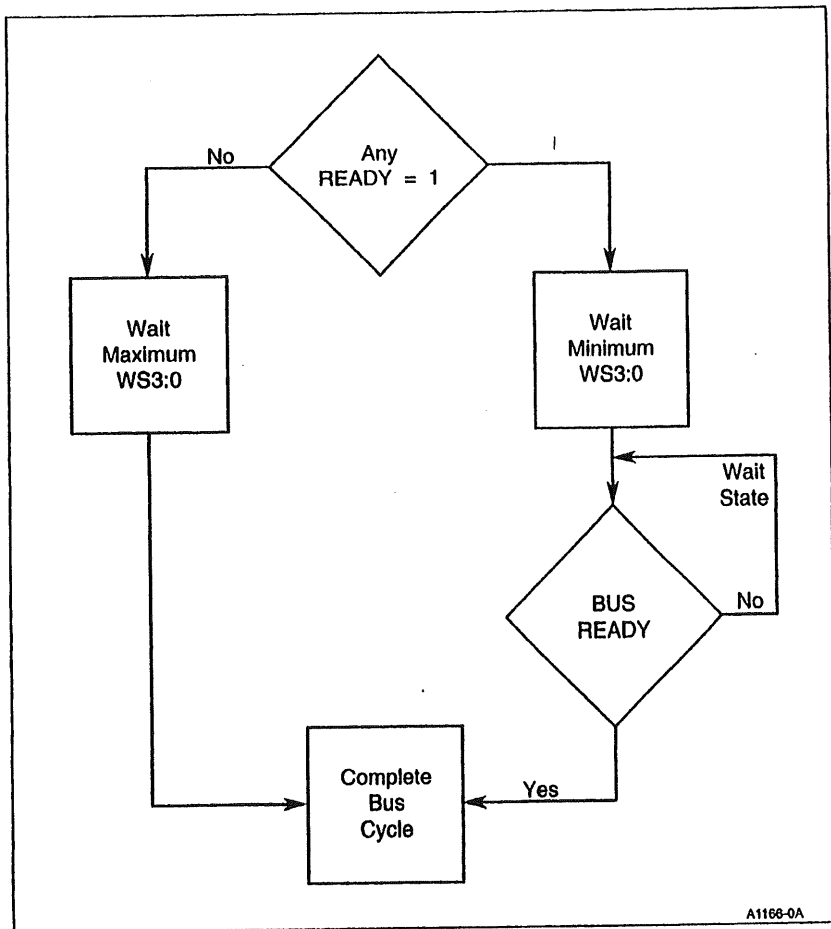


Figure 6-8. Overlapping Chip-Selects

Table lists example wait state and bus ready requirements for overlapping chip-selects and the resulting requirements for accesses to the overlapped region.

Table 6-4. Example Adjustments for Overlapping Chip-Selects

Chip-Select X		Chip-Select Y		Overlapped Region Access	
Wait States	Bus Ready	Wait States	Bus Ready	Wait States	Bus Ready
3	ignored	9	ignored	9	ignored
5	required	0	ignored	0	required
2	required	2	required	2	required

Be cautious when overlapping chip-selects with different wait state or bus ready programming. The following two conditions require special attention to ensure proper system operation:

1. When all overlapping chip-selects ignore bus ready but have different wait states, verify that each chip-select still works properly using the highest wait state value. A system failure may result when too few or too many wait states occur in the bus cycle.
2. If one or more of the overlapping chip-selects requires bus ready, verify that all chip-selects that ignore bus ready still work properly using both the smallest wait state value and the longest possible bus cycle. A system failure may result when too few or too many wait states occur in the bus cycle.

#### 6.4.7 Memory or I/O Bus Cycle Decoding

The Chip-Select Unit decodes bus cycle status and address information to determine whether a chip-select goes active. The MEM control bit in the STOP register defines whether memory or I/O address space is decoded. Memory address space accesses consist of memory read, memory write and instruction prefetch bus cycles. I/O address space accesses consist of I/O read and I/O write bus cycles.

Chip-selects go active for bus cycles initiated by the CPU and Refresh Control Unit.

#### 6.4.8 Programming Considerations

When programming chip-selects active for I/O bus cycles, remember that eight bytes of I/O are reserved by Intel. These eight bytes (locations 00F8H through 00FFH) control the interface to an 80C187 math coprocessor. A chip-select can overlap this reserved space provided there is no intention of using the 80C187. However, to avoid possible future compatibility issues, Intel recommends that no chip-select start at I/O address location 00C0H.

The  $\overline{\text{GCS}}$  chip-select outputs are multiplexed with output port functions. The register that controls the multiplexed outputs resides in the I/O Port Unit. (See Table 11-1 on page 11-7 and Figure 11-5 on page 11-8.)

## 6.5 CHIP-SELECTS AND BUS HOLD

The Chip-Select Unit decodes only internally generated address and bus state information. An external bus master cannot make use of the Chip-Select Unit. During HLDA, all chip-selects remain inactive.

The circuit shown in Figure 6-9 allows an external bus master to access a device during bus HOLD.

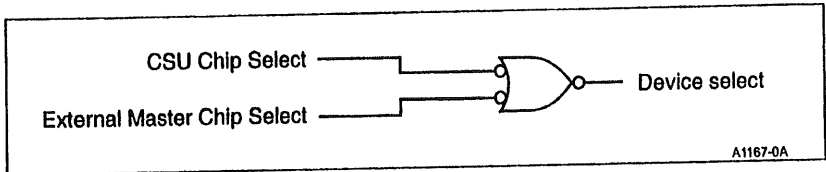


Figure 6-9. Using Chip-Selects During HOLD

## 6.6 EXAMPLES

The following sections provide examples of programming the Chip-Select Unit to meet the needs of a particular application. The examples do not go into hardware analysis or design issues.

### 6.6.1 Example 1: Typical System Configuration

Figure 6-10 illustrates a block diagram of a typical system design with a 128 Kbyte EPROM and a 32 Kbyte SRAM. The peripherals are mapped to I/O address space. Example 6.1 shows a program template for initializing the Chip-Select Unit.

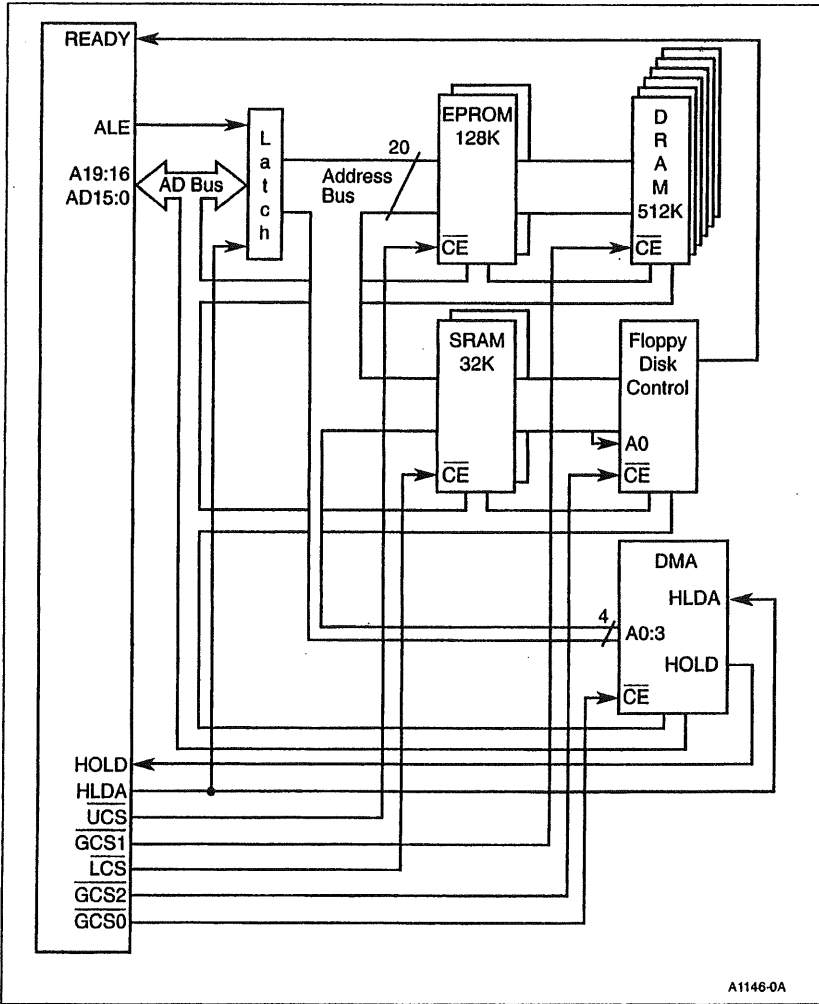


Figure 6-10. Typical System



**intel**<sup>®</sup>

**7**

# **Refresh Control Unit**

**1**



# CHAPTER 7 REFRESH CONTROL UNIT

The Refresh Control Unit (RCU) simplifies dynamic memory controller design with its integrated address and clock counters. Figure 7-1 shows the relationship between the Bus Interface Unit and the Refresh Control Unit. Integrating the Refresh Control Unit into the processor allows an external DRAM controller to use chip-selects, wait state logic and status lines.

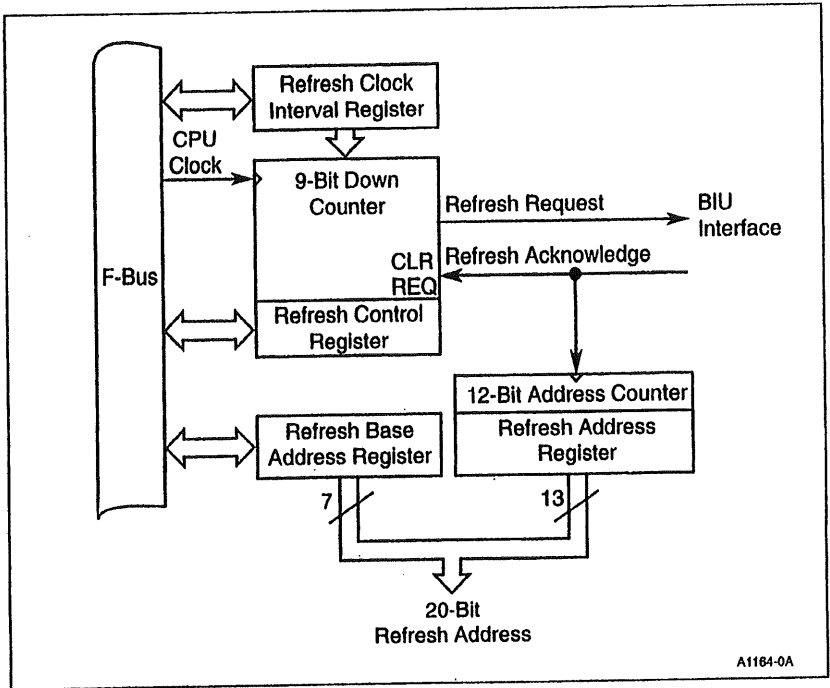


Figure 7-1. Refresh Control Unit Block Diagram

## 7.1 THE ROLE OF THE REFRESH CONTROL UNIT

Like a DMA controller, the Refresh Control Unit runs bus cycles independent of CPU execution. Unlike a DMA controller, however, the Refresh Control Unit does not run bus cycle bursts nor does it transfer data. The DRAM refresh process refreshes individual DRAM rows in “dummy read” cycles, while cycling through all necessary addresses.

The microprocessor interface to DRAMs is more complicated than other memory interfaces. A complete DRAM controller requires circuitry beyond that provided by the processor even in the simplest configurations. This circuitry must respond correctly to reads, writes and DRAM refresh cycles. The external DRAM controller generates the Row Address Strobe ( $\overline{RAS}$ ), Column Address Strobe ( $\overline{CAS}$ ) and other DRAM control signals.

Pseudo-static RAMs use dynamic memory cells but generate address strobes and refresh addresses internally. The address counters still need external timing pulses. These pulses are easy to derive from the processor's bus control signals. Pseudo-static RAMs do not need a full DRAM controller.

## 7.2 REFRESH CONTROL UNIT CAPABILITIES

A 12-bit address counter forms the refresh addresses, supporting any dynamic memory devices with up to 12 rows of memory cells (12 refresh address bits). This includes all practical DRAM sizes for the processor's 1 Mbyte address space.

## 7.3 REFRESH CONTROL UNIT OPERATION

Figure 7-2 illustrates Refresh Control Unit counting, address generation and BIU bus cycle generation in flowchart form.

The nine-bit down-counter loads from the Refresh Interval Register on the falling edge of CLK-OUT. Once loaded, it decrements every falling CLKOUT edge until it reaches one. Then the down-counter reloads and starts counting again, simultaneously triggering a refresh request. Once enabled, the DRAM refresh process continues indefinitely until the user reprograms the Refresh Control Unit, a reset occurs, or the processor enters Powerdown mode.

The refresh request remains active until the bus becomes available. When the bus is free, the BIU will run its “dummy read” cycle. Refresh bus requests have higher priority than most CPU bus cycles, all DMA bus cycles and all interrupt vectoring sequences. Refresh bus cycles also have a higher priority than the HOLD/ $\overline{HLDA}$  bus arbitration protocol (see “Refresh Operation and Bus HOLD” on page 7-13).

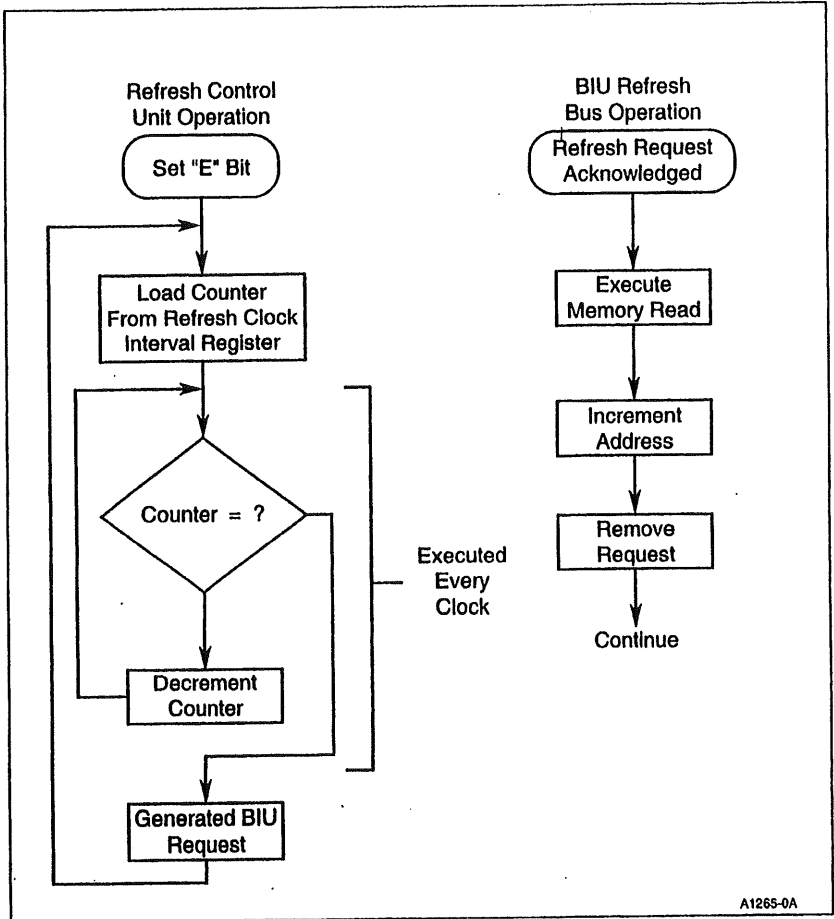


Figure 7-2. Refresh Control Unit Operation Flow Chart

The nine-bit refresh clock counter does not wait until the BIU services the refresh request to continue counting. This operation ensures that refresh requests occur at the correct interval. Otherwise, the time between refresh requests would be a function of varying bus activity. When the BIU services the refresh request, it clears the request and increments the refresh address.

The BIU does not queue DRAM refresh requests. If the Refresh Control Unit generates another request before the BIU handles the present request, the BIU loses the present request. However, the address associated with the request is not lost. The refresh address changes only after the BIU runs a refresh bus cycle. If a DRAM refresh cycle is excessively delayed, there is still a chance that the processor will successfully refresh the corresponding row of cells in the DRAM, retaining the data.

### 7.4 REFRESH ADDRESSES

Figure 7-3 shows the physical address generated during a refresh bus cycle. This figure applies to both the 8-bit and 16-bit data bus microprocessor versions. Refresh address bits RA19:13 come from the Refresh Base Address Register. (See "Refresh Base Address Register" on page 7-7.)

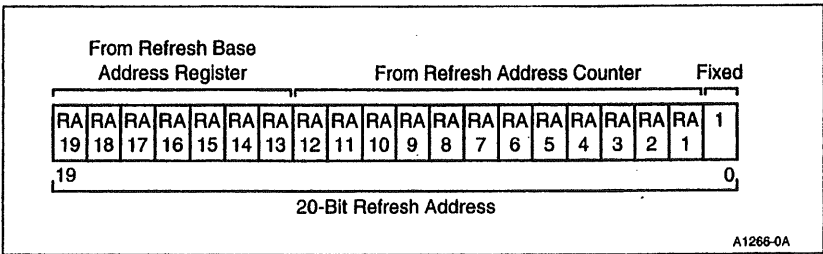


Figure 7-3. Refresh Address Formation

A linear-feedback shift counter generates address bits RA12:1 and RA0 is always one. The counter does not count linearly from 0 through FFFH. However, the counting algorithm cycles uniquely through all possible 12-bit values. It matters only that each row of DRAM memory cells is refreshed at a specific interval. The order of the rows is unimportant.

Address bit A0 is fixed at one during all refresh operations. In applications based on a 16-bit data bus processor, A0 typically selects memory devices placed on the low (even) half of the bus. Applications based on an 8-bit data bus processor typically use A0 as a true address bit. The DRAM controller must **not** route A0 to row address pins on the DRAMs.

## 7.5 REFRESH BUS CYCLES

Refresh bus cycles look exactly like ordinary memory read bus cycles except for the control signals listed in Table 7-1. These signals can be ANDed in a DRAM controller to detect a refresh bus cycle. The 16-bit bus processor drives both the  $\overline{\text{BHE}}$  and A0 pins high during refresh cycles. The 8-bit bus version replaces the  $\overline{\text{BHE}}$  pin with  $\overline{\text{RFSH}}$ , which has the same timings. The 8-bit bus processor drives  $\overline{\text{RFSH}}$  low and A0 high during refresh cycles..

Table 7-1. Identification of Refresh Bus Cycles

Data Bus Width	$\overline{\text{BHE}}/\overline{\text{RFSH}}$	A0
16-Bit Device	1	1
8-Bit Device	0	1

## 7.6 GUIDELINES FOR DESIGNING DRAM CONTROLLERS

The basic DRAM access method consists of four phases:

1. The DRAM controller supplies a row address to the DRAMs.
2. The DRAM controller asserts a Row Address Strobe ( $\overline{\text{RAS}}$ ), which latches the row address inside the DRAMs.
3. The DRAM controller supplies a column address to the DRAMs.
4. The DRAM controller asserts a Column Address Strobe ( $\overline{\text{CAS}}$ ), which latches the column address inside the DRAMs.

Most 80C186 Modular Core family DRAM interfaces use only this method. Others are not discussed here.

The DRAM controller's purpose is to use the processor's address, status and control lines to generate the multiplexed addresses and strobes. These signals must be appropriate for three bus cycle types: read, write and refresh. They must also meet specific pulse width, setup and hold timing requirements. DRAM interface designs need special attention to transmission line effects, since DRAMs represent significant loads on the bus.

DRAM controllers may be either clocked or unclocked. An unclocked DRAM controller requires a tapped digital delay line to derive the proper timings.

Clocked DRAM controllers may use either discrete or programmable logic devices. A state machine design is appropriate, especially if the circuit must provide wait state control (beyond that possible with the processor's Chip-Select Unit). Because of the microprocessor's four-clock bus, clocking some logic elements on each CLKOUT phase is advantageous (see Figure 7-4).

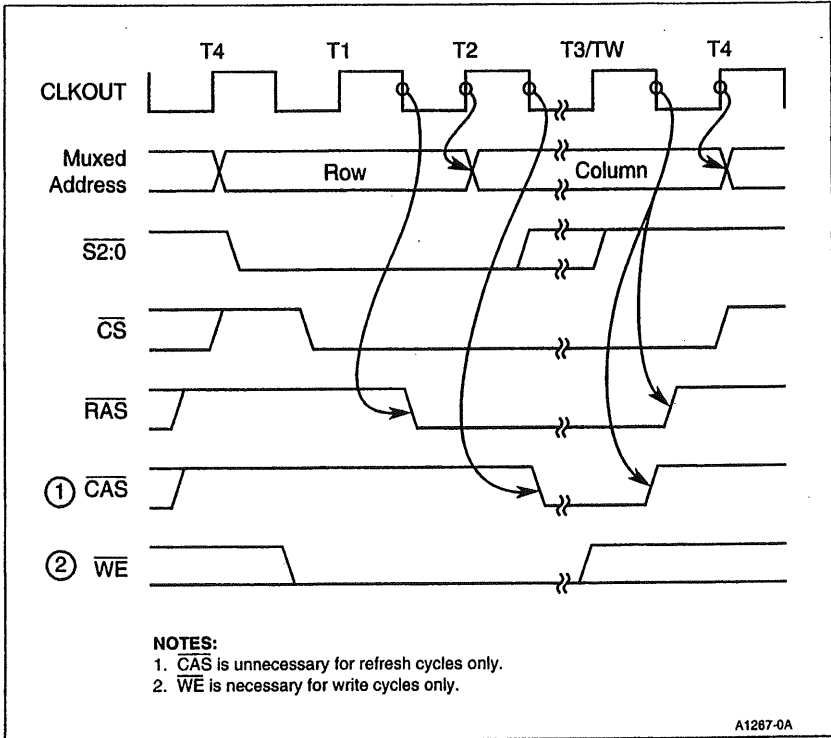


Figure 7-4. Suggested DRAM Control Signal Timing Relationships

The cycle begins with presentation of the row address.  $\overline{RAS}$  should go active on the falling edge of T2. At the rising edge of T2, the address lines should switch to a column address.  $\overline{CAS}$  goes active on the falling edge of T3. Refresh cycles do not require  $\overline{CAS}$ . When  $\overline{CAS}$  is present, the "dummy read" cycle becomes a true read cycle (the DRAM drives the bus), and the DRAM row still gets refreshed.

Both  $\overline{RAS}$  and  $\overline{CAS}$  stay active during any wait states. They go inactive on the falling edge of T4. At the rising edge of T4, the address multiplexer shifts to its original selection (row addressing), preparing for the next DRAM access.



## 7.7 PROGRAMMING THE REFRESH CONTROL UNIT

Given a specific processor operating frequency and information about the DRAMs in the system, the user can program the Refresh Control Unit registers.

### 7.7.1 Calculating the Refresh Interval

DRAM data sheets show DRAM refresh requirements as a number of refresh cycles necessary and the maximum period to run the cycles. (The number of refresh cycles is the same as the number of rows.) You must compensate for bus latency — the time it takes for the Refresh Control Unit to gain control of the bus. This is typically 1–5%, but if an external bus master will be extremely slow to release the bus, increase the overhead percentage. At standard operating frequencies, DRAM refresh bus overhead totals 2–3% of the total bus bandwidth.

Given this information and the CPU operating frequency, use the formula in Figure 7-5 to determine the correct value for the RFTIME Register value.

$\frac{R_{\text{PERIOD}} \times F_{\text{CPU}}}{\text{Rows} + (\text{Rows} \times \text{Overhead}\%)} = \text{RFTIME Register Value}$	
$R_{\text{PERIOD}}$	= Maximum refresh period specified by DRAM manufacturer (in $\mu\text{s}$ ).
$F_{\text{CPU}}$	= Operating frequency (in MHz).
ROWS	= Total number of rows to be refreshed.
Overhead %	= Derating factor to compensate for missed refresh requests (typically 1 – 5 %).

**Figure 7-5. Formula for Calculating Refresh Interval for RFTIME Register**

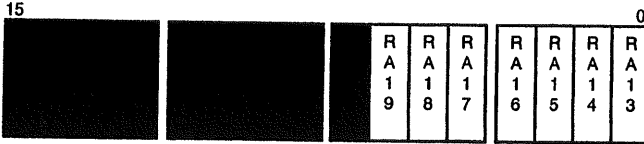
### 7.7.2 Refresh Control Unit Registers

Three contiguous Peripheral Control Block registers operate the Refresh Control Unit: the Refresh Base Address Register, Refresh Clock Interval Register and the Refresh Control Register. A fourth register, the Refresh Address Register, permits examination of the refresh address bits generated by the Refresh Control Unit.

#### 7.7.2.1 Refresh Base Address Register

The Refresh Base Address Register (Figure 7-6) programs the base (upper seven bits) of the refresh address. Seven-bit mapping places the refresh address at any 4 Kbyte boundary within the 1 Mbyte address space. When the partial refresh address from the 12-bit address counter (see Figure 7-1 and “Refresh Control Unit Capabilities” on page 7-2) passes FFFH, the Refresh Control Unit does **not** increment the refresh base address. Setting the base address ensures that the address driven during a refresh bus cycle activates the DRAM chip select.

**Register Name:** Refresh Base Address Register  
**Register Mnemonic:** RFBASE  
**Register Function:** Determines upper 7 bits of refresh address.



A1008-0A

Bit Mnemonic	Bit Name	Reset State	Function
RA19:13	Refresh Base	00H	Uppermost address bits for DRAM refresh cycles.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 7-6. Refresh Base Address Register

7.7.2.2 Refresh Clock Interval Register

The Refresh Clock Interval Register (Figure 7-7) defines the time between refresh requests. The higher the value, the longer the time between requests. The down-counter decrements every falling CLKOUT edge, regardless of core activity. When the counter reaches one, the Refresh Control Unit generates a refresh request, and the counter reloads the value from the register.

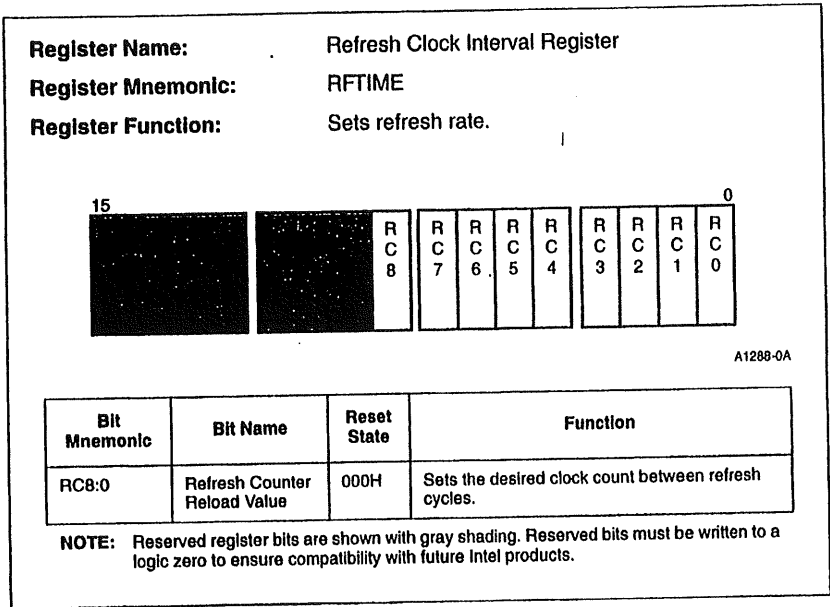
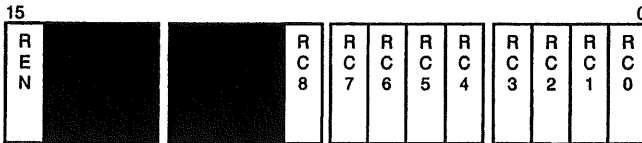


Figure 7-7. Refresh Clock Interval Register

7.7.2.3 Refresh Control Register

Figure 7-8 shows the Refresh Control Register. The user may read or write the REN bit at any time to turn the Refresh Control Unit on or off. The lower nine bits contain the current nine-bit down-counter value. The user cannot program these bits. Disabling the Refresh Control Unit clears both the counter and the corresponding counter bits in the control register.

**Register Name:** Refresh Control Register  
**Register Mnemonic:** RFCON  
**Register Function:** Controls Refresh Unit operation.



A1311-0A

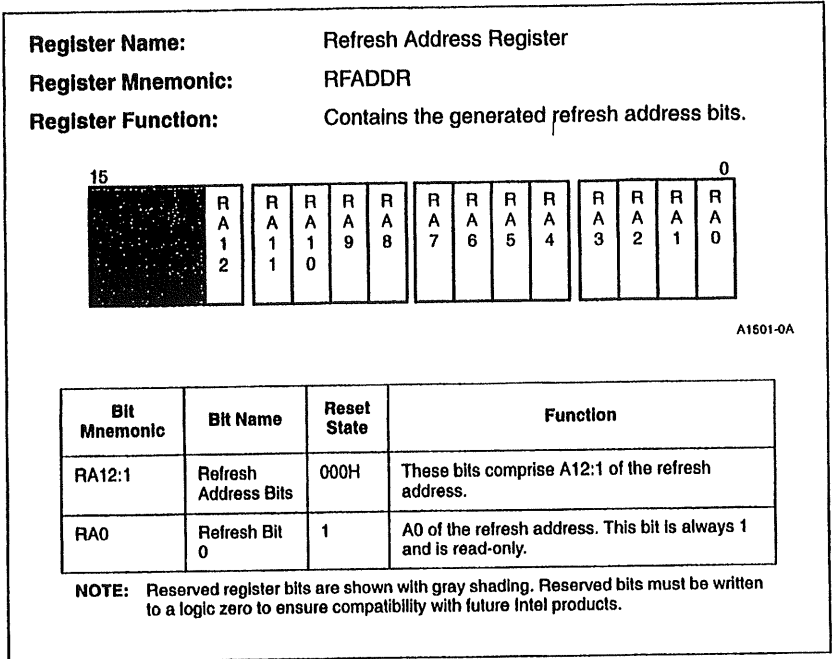
Bit Mnemonic	Bit Name	Reset State	Function
REN	Refresh Control Unit Enable	0	Setting REN enables the Refresh Unit. Clearing REN disables the Refresh Unit.
RC8:0	Refresh Counter	000H	These bits contain the present value of the down-counter that triggers refresh requests.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

**Figure 7-8. Refresh Control Register**

**7.7.2.4 Refresh Address Register**

The Refresh Address Register (Figure 7-9) contains address bits RA12:1, which will appear on the bus as A12:1 on the next refresh bus cycle. Bit 0 is fixed as a one in the register and in all refresh addresses.



**Figure 7-9. Refresh Address Register**

### 7.7.3 Programming Example

Example 7-1 contains sample code to initialize the Refresh Control Unit.

```

$mod186
name          example_80C186_RCU_code

; FUNCTION: This function initializes the DRAM Refresh
; Control Unit to refresh the DRAM starting at dram_addr
; at clock_time intervals.

; SYNTAX:
; extern void far config_rcu(int dram_addr, int clock_time);

; INPUTS:      dram_addr - Base address of DRAM to refresh
;              clock_time - DRAM refresh rate

; OUTPUTS:     None

;              NOTE: Parameters are passed on the stack as
;              required by high-level languages.

RFBASE        equ  xxxxxh                ;substitute register offset
RFTIME        equ  xxxxxh
RFCON         equ  xxxxxh
RFADDR        equ  xxxxxh                ;not used
Enable        equ  8000h                 ;enable bit

lib_80186     segment public 'code'
              assume cs:lib_80186

              public _config_rcu
_config_rcu   proc far

              push bp                      ;save caller's bp
              mov  bp, sp                  ;get current top of stack

_clock_time   equ  word ptr[bp+6]         ;get parameters off
_dram_addr    equ  word ptr[bp+8]         ;the stack

              push ax                      ;save registers that
              push cx                      ;will be modified
              push dx
              push di

```

Example 7-1. Initializing the Refresh Control Unit

```

        mov dx, RFBASE                ;set upper 7 address bits
        mov ax, _dram_addr
        out dx, al

        mov dx, RFTIME                ;set clock pre_scaler
        mov ax, _clock_time
        out dx, al

        mov dx, RFCON                 ;Enable RCU
        mov ax, Enable
        out dx, al

        mov cx, 8                     ;8 dummy cycles are
        xor di, di                     ;required by DRAMs
                                         ;before actual use

_exercise_ram:
        mov word ptr [di], 0
        loop _exercise_ram

        pop di                         ;restore saved registers
        pop dx
        pop cx
        pop ax
        pop bp                         ;restore caller's bp

        ret
    _config_rcu
    endp
lib_80186
    ends
    end

```

**Example 7-1. Initializing the Refresh Control Unit (Continued)**

## 7.8 REFRESH OPERATION AND BUS HOLD

When another bus master controls the bus, the processor keeps  $\overline{HLDA}$  active as long as the HOLD input remains active. If the Refresh Control Unit generates a refresh request during bus hold, the processor drives the  $\overline{HLDA}$  signal inactive, indicating to the current bus master that it wishes to regain bus control (see Figure 7-10). The BIU begins a refresh bus cycle only after the alternate master removes HOLD. The user must design the system so that the processor can regain bus control. If the alternate master asserts HOLD after the processor starts the refresh cycle, the CPU will relinquish control by asserting  $\overline{HLDA}$  when the refresh cycle is complete.

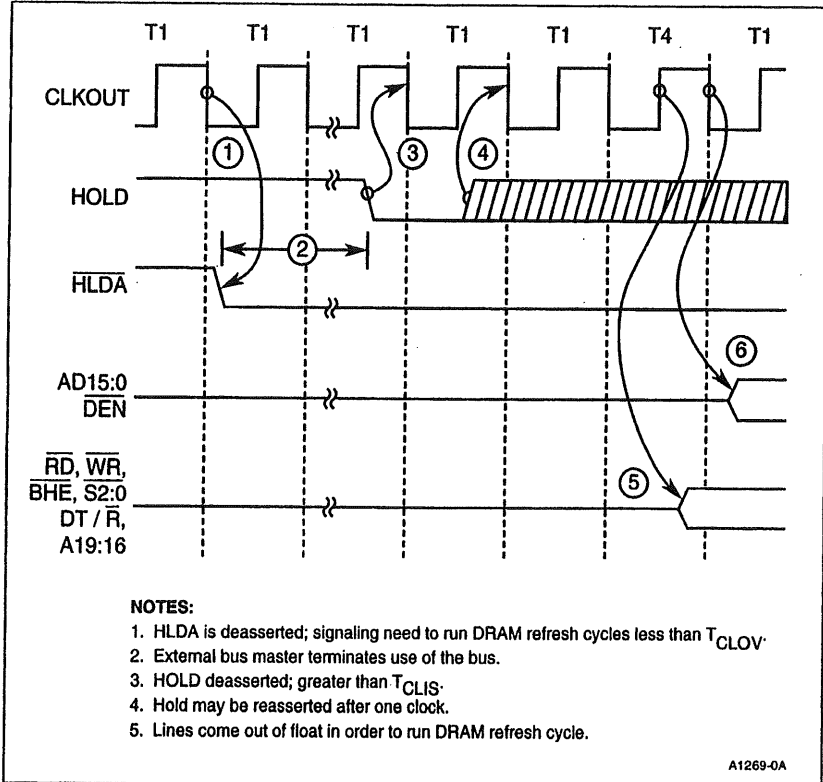


Figure 7-10. Regaining Bus Control to Run a DRAM Refresh Bus Cycle

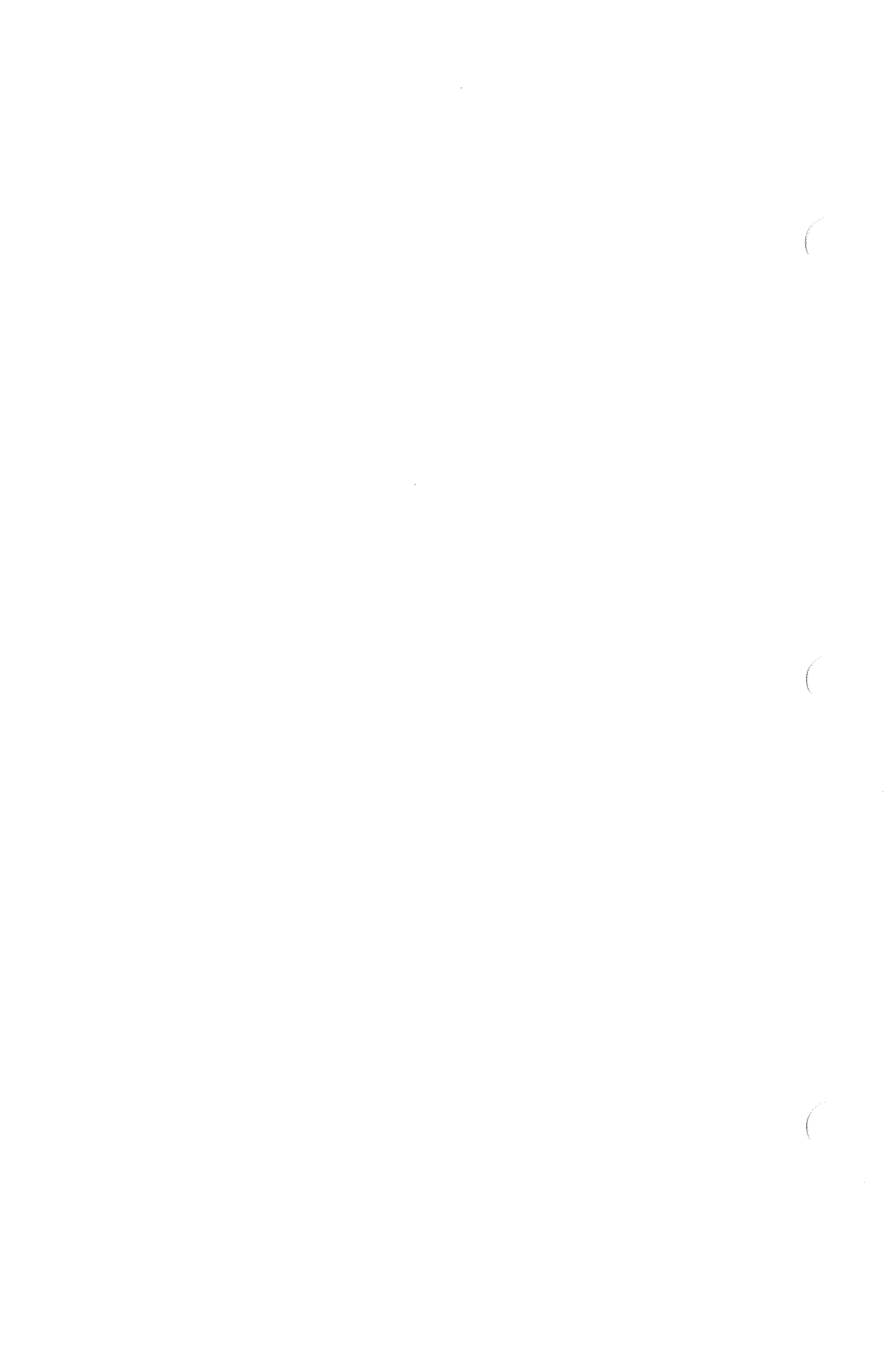


**intel.**

**8**

**Interrupt Control  
Unit**

**I**



## 2.3 INTERRUPTS AND EXCEPTION HANDLING

Interrupts and exceptions alter program execution in response to an external event or an error condition. An interrupt handles asynchronous external events, for example an NMI. Exceptions result directly from the execution of an instruction, usually an instruction fault. The user can cause a software interrupt by executing an "INTn" instruction. The CPU processes software interrupts in the same way that it handles exceptions.

The 80C186 Modular Core responds to interrupts and exceptions in the same way for all devices within the 80C186 Modular Core family. However, devices within the family may have different Interrupt Control Units. The Interrupt Control Unit handles all external interrupt sources and presents them to the 80C186 Modular Core via one maskable interrupt request (see Figure 2-24). This discussion covers only those areas of interrupts and exceptions that are common to the 80C186 Modular Core family. The Interrupt Control Unit is proliferation-dependent; see Chapter 8, "Interrupt Control Unit," for additional information.

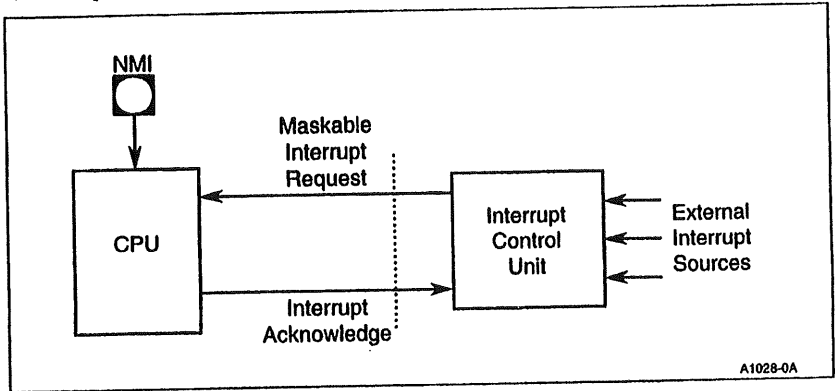


Figure 2-24. Interrupt Control Unit

### 2.3.1 Interrupt/Exception Processing

The 80C186 Modular Core can service up to 256 different interrupts and exceptions. A 256-entry Interrupt Vector Table (Figure 2-25) contains the pointers to interrupt service routines. Each entry consists of four bytes, which contain the Code Segment (CS) and Instruction Pointer (IP) of the first instruction in the interrupt service routine. Each interrupt or exception is given a type number, 0 through 255, corresponding to its position in the Interrupt Vector Table. Note that interrupt types 0–31 are reserved for Intel and should not be used by an application program.

OVERVIEW OF THE 80C186 FAMILY ARCHITECTURE

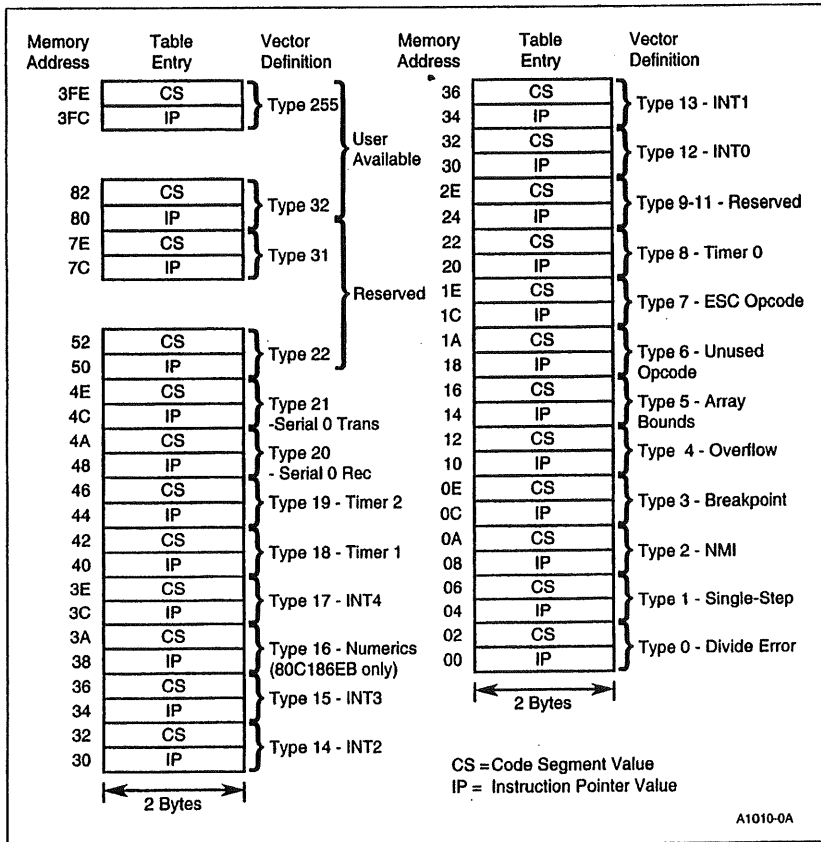


Figure 2-25. Interrupt Vector Table

When an interrupt is acknowledged, a common event sequence (Figure 2-26) allows the processor to execute the interrupt service routine.

1. The processor saves a partial machine status by pushing the Processor Status Word onto the stack.
2. The Trap Flag bit and Interrupt Enable bit are cleared in the Processor Status Word. This prevents maskable interrupts or single step exceptions from interrupting the processor during the interrupt service routine.

3. The current CS and IP are pushed onto the stack.
4. The CPU fetches the new CS and IP for the interrupt vector routine from the Interrupt Vector Table and begins executing from that point.

The CPU is now executing the interrupt service routine. The programmer must save (usually by pushing onto the stack) all registers used in the interrupt service routine; otherwise, their contents will be lost. To allow nesting of maskable interrupts, the programmer must set the Interrupt Enable bit in the Processor Status Word.

When exiting an interrupt service routine, the programmer must restore (usually by popping off the stack) the saved registers and execute an IRET instruction, which performs the following steps.

1. Loads the return CS and IP by popping them off the stack.
2. Pops and restores the old Processor Status Word from the stack.

The CPU now executes from the point at which the interrupt or exception occurred.

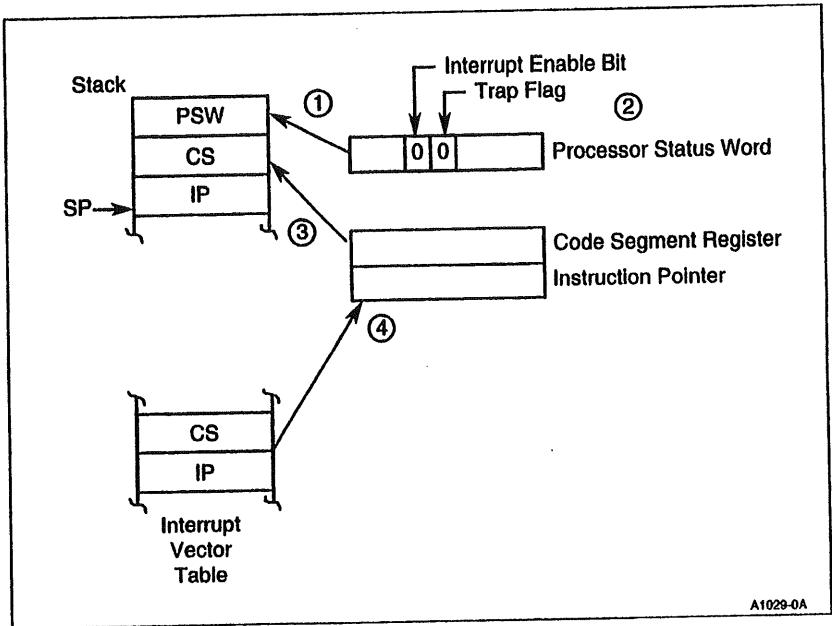


Figure 2-26. Interrupt Sequence

### 2.3.1.1 Non-Maskable Interrupts

The Non-Maskable Interrupt (NMI) is the highest priority interrupt. It is usually reserved for a catastrophic event such as impending power failure. An NMI cannot be prevented (or masked) by software. When the NMI input is asserted, the interrupt processing sequence begins after execution of the current instruction completes (see "Interrupt Latency" on page 2-44). The CPU automatically generates a type 2 interrupt vector.

The NMI input is asynchronous. Setup and hold times are given only to guarantee recognition on a specific clock edge. To be recognized, NMI must be asserted for at least one CLKOUT period and meet the correct setup and hold times. NMI is edge-triggered and level-latched. Multiple NMI requests cause multiple NMI service routines to be executed. NMI can be nested in this manner an infinite number of times.

### 2.3.1.2 Maskable Interrupts

Maskable interrupts are the most common way to service external hardware interrupts. Software can globally enable or disable maskable interrupts. This is done by setting or clearing the Interrupt Enable bit in the Processor Status Word.

The Interrupt Control Unit processes the multiple sources of maskable interrupts and presents them to the core via a single maskable interrupt input. The Interrupt Control Unit provides the interrupt vector type to the 80C186 Modular Core. The Interrupt Control Unit differs among members of the 80C186 Modular Core family; see Chapter 8, "Interrupt Control Unit," for information.

### 2.3.1.3 Exceptions

Exceptions occur when an unusual condition prevents further instruction processing until the exception is corrected. The CPU handles software interrupts and exceptions in the same way. The interrupt type for an exception is either predefined or supplied by the instruction.

Exceptions are classified as either faults or traps, depending on when the exception is detected and whether the instruction that caused the exception can be restarted. Faults are detected and serviced **before** the faulting instruction can be executed. The return address pushed onto the stack in the interrupt processing instruction points to the beginning of the faulting instruction. This allows the instruction to be restarted. Traps are detected and serviced **immediately after** the instruction that caused the trap. The return address pushed onto the stack during the interrupt processing points to the instruction following the trapping instruction.

#### Divide Error — Type 0

A Divide Error trap is invoked when the quotient of an attempted division exceeds the maximum value of the destination. A divide-by-zero is a common example.



### Single Step — Type 1

The Single Step trap occurs after the CPU executes one instruction with the Trap Flag (TF) bit set in the Processor Status Word. This allows programs to execute one instruction at a time. Interrupts are not generated after prefix instructions (e.g., REP), after instructions that modify segment registers (e.g., POP DS) or after the WAIT instruction. Vectoring to the single-step interrupt service routine clears the Trap Flag bit. An IRET instruction in the interrupt service routine restores the Trap Flag bit to logic "1" and transfers control to the next instruction to be single-stepped.

### Breakpoint Interrupt — Type 3

The Breakpoint Interrupt is a single-byte version of the INT instruction. It is commonly used by software debuggers to set breakpoints in RAM. Because the instruction is only one byte long, it can substitute for any instruction.

### Interrupt on Overflow — Type 4

The Interrupt on Overflow trap occurs if the Overflow Flag (OF) bit is set in the Processor Status Word and the INTO instruction is executed. Interrupt on Overflow is a common method for handling arithmetic overflows conditionally.

### Array Bounds Check — Type 5

An Array Bounds trap occurs when the array index is outside the array bounds during execution of the BOUND instruction (see Appendix A, "80C186 Instruction Set Additions and Extensions").

### Invalid Opcode — Type 6

Execution of an undefined opcode causes an Invalid Opcode trap.

### Escape Opcode — Type 7

The Escape Opcode fault is used for floating point emulation. With 80C186 Modular Core family members, this fault is enabled by setting the Escape Trap (ET) bit in the Relocation Register (see Chapter 4, "Peripheral Control Block"). When a floating point instruction is executed with the Escape Trap bit set, the Escape Opcode fault occurs, and the Escape Opcode service routine emulates the floating point instruction. If the Escape Trap bit is cleared, the CPU sends the floating point instruction to an external 80C187.

80C188 Modular Core Family members do not support the 80C187 interface and always generate the Escape Opcode Fault.



### Numerics Coprocessor Fault — Type 16

The Numerics Coprocessor fault is caused by an external 80C187 numerics coprocessor. The 80C187 reports the exception by asserting the **ERROR** pin. The 80C186 Modular Core checks the **ERROR** pin only when executing a numerics instruction. A Numerics Coprocessor Fault indicates that the **previous** numerics instruction caused the exception. The 80C187 saves the address of the floating point instruction that caused the exception. The return address pushed onto the stack during the interrupt processing points to the numerics instruction that detected the exception. This way, the last numerics instruction can be restarted.

### 2.3.2 Software Interrupts

A Software Interrupt is caused by executing an “INT $n$ ” instruction. The  $n$  parameter corresponds to the specific interrupt type to be executed. The interrupt type can be any number between 0 and 255. If the  $n$  parameter corresponds to an interrupt type associated with a hardware interrupt (NMI, Timers), the vectors are fetched and the routine is executed, but the corresponding bits in the Interrupt Status register **are not altered**.

The CPU processes software interrupts and exceptions in the same way. Software interrupts, exceptions and traps cannot be masked.

### 2.3.3 Interrupt Latency

Interrupt latency is the amount of time it takes for the CPU to recognize the existence of an interrupt. The CPU generally recognizes interrupts only between instructions or on instruction boundaries. Therefore, the current instruction must finish executing before an interrupt can be recognized.

The worst-case 80C186 instruction execution time is an integer divide instruction with segment override prefix. The instruction takes 69 clocks, assuming an 80C186 Modular Core family member and a zero wait-state external bus. The execution time for an 80C188 Modular Core family member may be longer, depending on the queue.

This is one factor in determining interrupt latency. In addition, the following are also factors in determining maximum latency:

1. The CPU does not recognize the Maskable Interrupt unless the Interrupt Enable bit is set.
2. The CPU does not recognize interrupts during HOLD.
3. Once communication is completely established with an 80C187, the CPU does not recognize interrupts until the numerics instruction is finished.



The CPU can recognize interrupts only on valid instruction boundaries. A valid instruction boundary usually occurs when the current instruction finishes. The following is a list of exceptions:

1. MOVs and POPs referencing a segment register delay the servicing of interrupts until after the following instruction. The delay allows a 32-bit load to the SS and SP without an interrupt occurring between the two loads.
2. The CPU allows interrupts between repeated string instructions. If multiple prefixes precede a string instruction and the instruction is interrupted, only the one prefix preceding the string primitive is restored.
3. The CPU can be interrupted during a WAIT instruction. The CPU will return to the WAIT instruction.

### **2.3.4 Interrupt Response Time**

Interrupt response time is the time from the CPU recognizing an interrupt until the first instruction in the service routine is executed. Interrupt response time is less for interrupts or exceptions which supply their own vector type. The maskable interrupt has a longer response time because the vector type must be supplied by the Interrupt Control Unit (see Chapter 8, "Interrupt Control Unit").

Figure 2-27 shows the events that dictate interrupt response time for the interrupts that supply their type. Note that an on-chip bus master, such as the DRAM Refresh Unit, can make use of idle bus cycles. This can increase interrupt response time.

	Clocks
Idle	5
Read IP	4
Idle	5
Read CS	4
Idle	4
Push Flags	4
Idle	3
Push CS	4
Push IP	4
Idle	5
First Instruction Fetch From Interrupt Routine <div style="display: inline-block; vertical-align: middle;">                         .....&gt;                     </div>	
<b>Total 42</b>	

A1030-0A

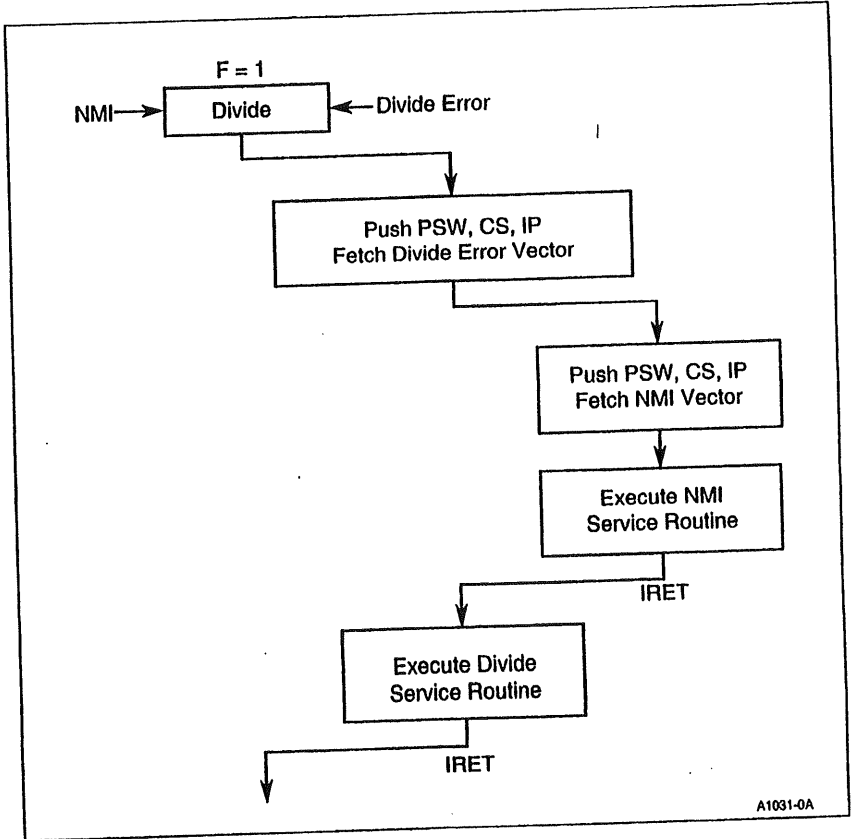
**Figure 2-27. Interrupt Response Factors**

### 2.3.5 Interrupt and Exception Priority

Interrupts can be recognized only on valid instruction boundaries. If an NMI and a maskable interrupt are both recognized on the same instruction boundary, NMI has precedence. The maskable interrupt will not be recognized until the Interrupt Enable bit is set and it is the highest priority.

Only the single step exception can occur concurrently with another exception. At most, two exceptions can occur at the same instruction boundary and one of those exceptions must be the single step. Single step is a special case; it is discussed on page 2-47. Ignoring single step (for now), only one exception can occur at any given instruction boundary.

An exception has priority over both NMI and the maskable interrupt. However, a pending NMI can interrupt the CPU at any valid instruction boundary. Therefore, NMI can interrupt an exception service routine. If an exception and NMI occur simultaneously, the exception vector is taken, then is followed immediately by the NMI vector (see Figure 2-28). While the exception has higher priority at the instruction boundary, the NMI interrupt service routine is executed first.



**Figure 2-28. Simultaneous NMI and Exception**

Single step priority is a special case. If an interrupt (NMI or maskable) occurs at the same instruction boundary as a single step, the interrupt vector is taken first, then is followed immediately by the single step vector. However, the single step service routine is executed before the interrupt service routine (see Figure 2-29). If the single step service routine re-enables single step by executing the IRET, the interrupt service routine will also be single stepped. This can severely limit the real-time response of the CPU to an interrupt.

To prevent the single-step routine from executing before a maskable interrupt, disable interrupts while single stepping an instruction, then enable interrupts in the single step service routine. The maskable interrupt is serviced from within the single step service routine and that interrupt service routine is not single-stepped. To prevent single stepping before an NMI, the single-step service routine must compare the return address on the stack to the NMI vector. If they are the same, return to the NMI service routine immediately without executing the single step service routine.

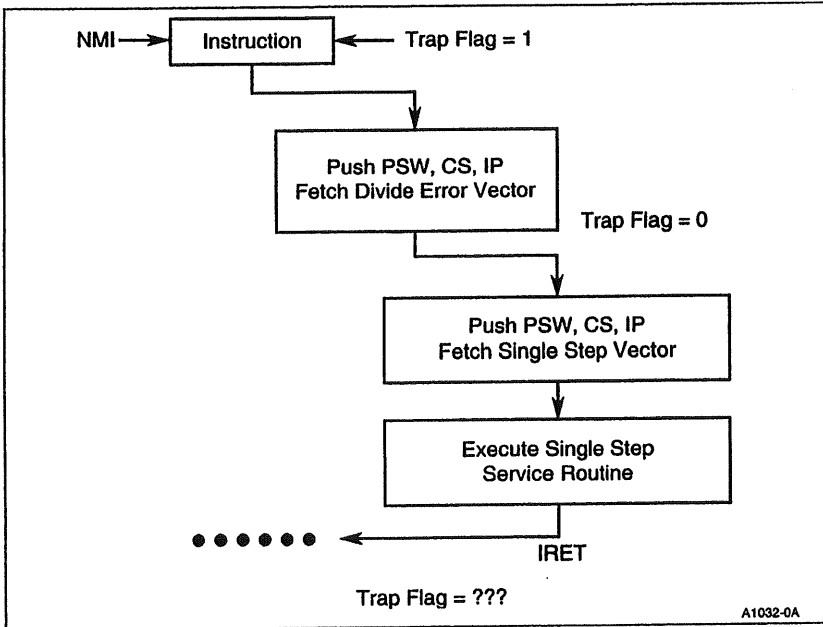


Figure 2-29. Simultaneous NMI and Single Step Interrupts

The most complicated case is when an NMI, a maskable interrupt, a single step and another exception are pending on the same instruction boundary. Figure 2-30 shows how this case is prioritized by the CPU. Note that if the single-step routine sets the Trap Flag (TF) bit before executing the IRET instruction, the NMI routine will also be single stepped.

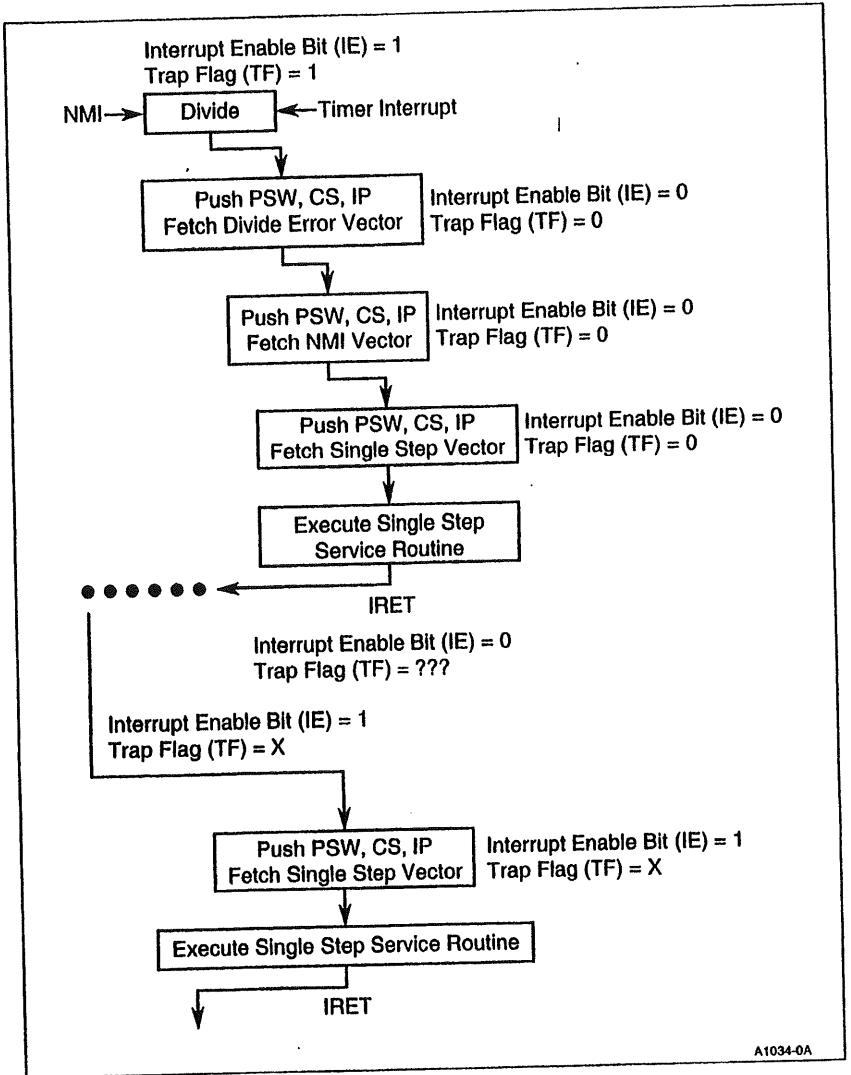


Figure 2-30. Simultaneous NMI, Single Step and Maskable Interrupt



## CHAPTER 8 INTERRUPT CONTROL UNIT

The 80C186 Modular Core has a single maskable interrupt input. (See “Interrupts and Exception Handling” on page 2-39.) The Interrupt Control Unit (ICU) expands the interrupt capabilities beyond a single input. It receives and processes maskable interrupts from multiple sources and presents them to the CPU through the maskable interrupt input. Interrupts can originate from the on-chip peripherals and from five external interrupt pins. The Interrupt Control Unit synchronizes and prioritizes the interrupts and provides the interrupt type vector to the CPU. (See Figure 8-1.)

The Interrupt Control Unit has the following features:

- Programmable priority of each interrupt source
- Individual masking of each interrupt source
- Nesting of interrupt sources
- Support for polled operation
- Support for cascading external 8259A modules to expand external interrupt sources

### 8.1 FUNCTIONAL OVERVIEW

All microcomputer systems must communicate in some way with the external world. A typical system might have a keyboard, a disk drive and a communications port, all requiring CPU attention at different times. There are two distinct ways to process peripheral I/O requests: polling and interrupts.

Polling requires that the CPU check each peripheral device in the system periodically to see whether it requires servicing. It would not be unusual to poll a low-speed peripheral (a serial port, for instance) thousands of times before it required servicing. In most cases, the use of polling has a detrimental effect on system throughput. Any time used to check the peripherals is time spent away from the main processing tasks.

Interrupts eliminate the need for polling by signalling the CPU that a peripheral device requires servicing. The CPU then stops executing the main task, saves its state and transfers execution to the peripheral-servicing code (the *interrupt handler*). At the end of the interrupt handler, the CPU's original state is restored and execution continues at the point of interruption in the main task.

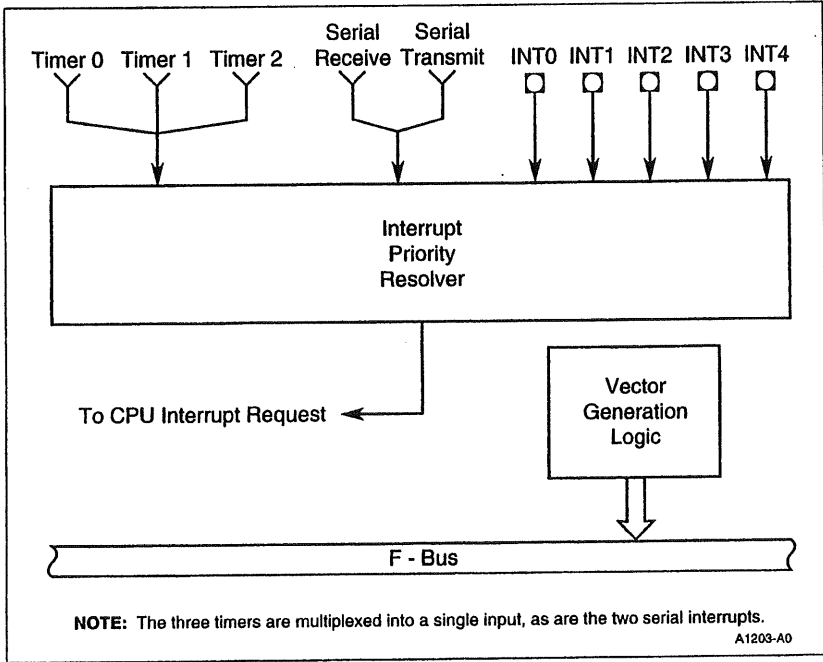


Figure 8-1. Interrupt Control Unit Block Diagram

### 8.1.1 Generic Functions

Several functions of the Interrupt Control Unit are common among most interrupt controllers. This section describes how those generic functions are implemented in the Interrupt Control Unit.

#### 8.1.1.1 Interrupt Masking

There are circumstances in which a programmer may need to disable an interrupt source temporarily (for example, while executing a time-critical section of code or servicing a high-priority task). This temporary disabling is called interrupt masking. All interrupts from the Interrupt Control Unit can be masked either globally or individually.



The Interrupt Enable bit in the Processor Status Word globally enables or disables the maskable interrupt request from the Interrupt Control Unit. The programmer controls the Interrupt Enable bit with the STI (set interrupt) and CLI (clear interrupt) instructions.

Besides being globally enabled or disabled by the Interrupt Enable bit, each interrupt source can be individually enabled or disabled. The Interrupt Mask register has a single bit for each interrupt source. The programming can selectively mask (disable) or unmask (enable) each interrupt source by setting or clearing the corresponding bit in the Interrupt Mask register.

### 8.1.1.2 Interrupt Priority

One critical function of the Interrupt Control Unit is to prioritize interrupt requests. When multiple interrupts are pending, priority determines which interrupt request is serviced first. In many systems, an interrupt handler may itself be interrupted by another interrupt source. This is known as *interrupt nesting*. With interrupt nesting, priority determines whether an interrupt source can preempt an interrupt handler that is currently executing.

Each interrupt source is assigned a priority between zero (highest) and seven (lowest). After reset, the interrupts default to the priorities shown in Table 8-1. Because the timers share an interrupt source, they also share a priority. Within the assigned priority, each has a relative priority (Timer 0 has the highest relative priority and Timer 2 has the lowest). The serial channel 0 receive and transmit interrupts also share a priority. Within the assigned priority, the receive interrupt has the higher relative priority.

**Table 8-1. Default Interrupt Priorities**

Interrupt Name	Relative Priority
Timer 0	0 (a)
Timer 1	0 (b)
Timer 2	0 (c)
Serial Channel 0 Receive	1(a)
Serial Channel 0 Transmit	1(b)

The priority of each source is programmable. The Interrupt Control register enables the programmer to assign each source a priority that differs from the default. The priority must still be between zero (highest) and seven (lowest). Interrupt sources can be programmed to share a priority. The Interrupt Control Unit uses the default priorities (see Table 8-1) within the shared priority level to determine which interrupt to service first. For example, assume that INTO and INT1 are both programmed to priority seven. Because INTO has the higher default priority, it is serviced first.

## INTERRUPT CONTROL UNIT

Interrupt sources can be masked on the basis of their priority. The Priority Mask register masks all interrupts with priorities lower than its programmed value. After reset, the Priority Mask register contains priority seven, which effectively enables all interrupts. The programmer can then program the register with any valid priority level.

### 8.1.1.3 Interrupt Nesting

When entering an interrupt handler, the CPU pushes the Processor Status Word onto the stack and clears the Interrupt Enable bit. The processor enters all interrupt handlers with maskable interrupts disabled. Maskable interrupts remain disabled until either the IRET instruction restores the Interrupt Enable bit or the programmer explicitly enables interrupts. Enabling maskable interrupts within an interrupt handler allows interrupts to be nested. Otherwise, interrupts are processed sequentially; one interrupt handler must finish before another executes.

The simplest way to use the Interrupt Control Unit is without nesting. The operation and servicing of all sources of maskable interrupts is straightforward. However, the application tradeoff is that an interrupt handler will finish executing even if a higher priority interrupt occurs. This can add considerable latency to the higher priority interrupt.

In the simplest terms, the Interrupt Control Unit asserts the maskable interrupt request to the CPU, waits for the interrupt acknowledge, then presents the interrupt type of the highest priority unmasked interrupt to the CPU. The CPU then executes the interrupt handler for that interrupt. Because the interrupt handler never sets the Interrupt Enable bit, it can never be interrupted.

The function of the Interrupt Control Unit is more complicated with interrupt nesting. In this case, an interrupt can occur during execution of an interrupt handler. That is, one interrupt can *preempt* another. Two rules apply for interrupt nesting:

- An interrupt source cannot preempt interrupts of higher priority.
- An interrupt source cannot preempt itself. The interrupt handler must finish executing before the interrupt is serviced again. (Special Fully Nested Mode is an exception. See “Special Fully Nested Mode” on page 8-8.)

## 8.2 FUNCTIONAL OPERATION

This section covers the process in which the Interrupt Control Unit receives interrupts and asserts the maskable interrupt request to the CPU.

### 8.2.1 Typical Interrupt Sequence

When the Interrupt Control Unit first detects an interrupt, it sets the corresponding bit in the Interrupt Request register to indicate that the interrupt is pending. The Interrupt Control Unit checks all pending interrupt sources. If the interrupt is unmasked and meets the priority criteria (see "Priority Resolution" on page 8-5), the Interrupt Control Unit asserts the maskable interrupt request to the CPU, then waits for the interrupt acknowledge.

When the Interrupt Control Unit receives the interrupt acknowledge, it passes the interrupt type to the CPU. At that point, the CPU begins the interrupt processing sequence. (See "Interrupt/Exception Processing" on page 2-39 for details.) The Interrupt Control Unit always passes the vector that has the highest priority at the time the acknowledge is received. If a higher priority interrupt occurs before the interrupt acknowledge, the higher priority interrupt has precedence.

When it receives the interrupt acknowledge, the Interrupt Control Unit clears the corresponding bit in the Interrupt Request register and sets the corresponding bit in the In-Service register. The In-Service register keeps track of which interrupt handlers are being processed. At the end of an interrupt handler, the programmer must issue an End-of-Interrupt (EOI) command to explicitly clear the In-Service register bit. If the bit remains set, the Interrupt Control Unit cannot process any additional interrupts from that source.

### 8.2.2 Priority Resolution

The decision to assert the maskable interrupt request to the CPU is somewhat complicated. The complexity is needed to support interrupt nesting. First, an interrupt occurs and the corresponding Interrupt Request register bit is set. The Interrupt Control Unit then asserts the maskable interrupt request to the CPU, if the pending interrupt satisfies these requirements:

1. its Interrupt Mask bit is cleared (it is unmasked)
2. its priority is higher than the value in the Priority Mask register
3. its In-Service bit is cleared
4. its priority is equal to or greater than that of any interrupt whose In-Service bit is set

The In-Service register keeps track of interrupt handler execution. The Interrupt Control Unit uses this information to decide whether another interrupt source has sufficient priority to preempt an interrupt handler that is executing.

**8.2.2.1 Priority Resolution Example**

This example illustrates priority resolution. Assume these initial conditions:

- the Interrupt Control Unit has been initialized
- no interrupts are pending
- no In-Service bits are set
- the Interrupt Enable bit is set
- all interrupts are unmasked
- the default priority scheme is being used
- the Priority Mask register is set to the lowest priority (seven)

The example uses two external interrupt sources, INT0 and INT3, to describe the process.

1. A low-to-high transition on INT0 sets its Interrupt Request bit. The interrupt is now pending.
2. Because INT0 is the only pending interrupt, it meets all the priority criteria. The Interrupt Control Unit asserts the interrupt request to the CPU and waits for an acknowledge.
3. The CPU acknowledges the interrupt.
4. The Interrupt Control Unit passes the interrupt type (in this case, type 12) to the CPU.
5. The Interrupt Control Unit clears the INT0 bit in the Interrupt Request register and sets the INT0 bit in the In-Service register.
6. The CPU executes the interrupt processing sequence and begins executing the interrupt handler for INT0.
7. During execution of the INT0 interrupt handler, a low-to-high transition on INT3 sets its Interrupt Request bit.
8. The Interrupt Control Unit determines that INT3 has a lower priority than INT0, which is currently executing (INT0's In-Service bit is set). INT3 does not meet the priority criteria, so no interrupt request is sent to the CPU. (If INT3 were programmed with a higher priority than INT0, the request would be sent.) INT3 remains pending in the Interrupt Request register.
9. The INT0 interrupt handler completes and sends an EOI command to clear the INT0 bit in the In-Service register.
10. INT3 is still pending and now meets all the priority criteria. The Interrupt Control Unit asserts the interrupt request to the CPU and the process begins again.

### 8.2.2.2 Interrupts That Share a Single Source

Multiple interrupt requests can share a single interrupt input to the Interrupt Control Unit. (For example, the three timers share a single input.) Although these interrupts share an input, each has its own interrupt vector. (For example, when a Timer 0 interrupt occurs, the Timer 0 interrupt handler is executed.) This section uses the three timers as an example to describe how these interrupts are prioritized and serviced.

The Interrupt Status register acts as a second-level request register to process the timer interrupts. It contains a bit for each timer interrupt. When a timer interrupt occurs, both the individual Interrupt Status register bit and the shared Interrupt Request register bit are set. From this point, the interrupt is processed like any other interrupt source.

When the shared interrupt is acknowledged, the timer interrupt with the highest priority (see Table 8-1 on page 8-3) at that time is serviced first and that timer's Interrupt Status bit is cleared. If no other timer Interrupt Status bits are set, the shared Interrupt Request bit is also cleared. If other timer interrupts are pending, the Interrupt Request bit remains set.

When the timer interrupt is acknowledged, the shared In-Service bit is set. No other timer interrupts can occur when the In-Service bit is set. If a second timer interrupt occurs while another timer interrupt is being serviced, the second interrupt remains pending until the interrupt handler for the first interrupt finishes and clears the In-Service bit. (This is true even if the second interrupt has a higher priority than the first.)

### 8.2.3 Cascading with External 8259As

For applications that require more external interrupt pins than the number provided on the Interrupt Control Unit, external 8259A modules can be used to increase the number of external interrupt pins. The cascade mode of the Interrupt Control Unit supports the external 8259As. The  $\overline{INT2}/\overline{INTA0}$  and  $\overline{INT3}/\overline{INTA1}$  pins can serve either of two functions. Outside cascade mode, they serve as external interrupt inputs. In cascade mode, they serve as interrupt acknowledge outputs.  $\overline{INTA0}$  is the acknowledge for  $\overline{INT0}$ , and  $\overline{INTA1}$  is the acknowledge for  $\overline{INT1}$ . (See Figure 8-2.)

The  $\overline{INT2}/\overline{INTA0}$  and  $\overline{INT3}/\overline{INTA1}$  pins are inputs after reset until the pins are configured as outputs. The pullup resistors ensure that the  $\overline{INTA}$  pins never float (which would cause a spurious interrupt acknowledge to the 8259A). The value of the resistors must be high enough to prevent excessive loading on the pins.

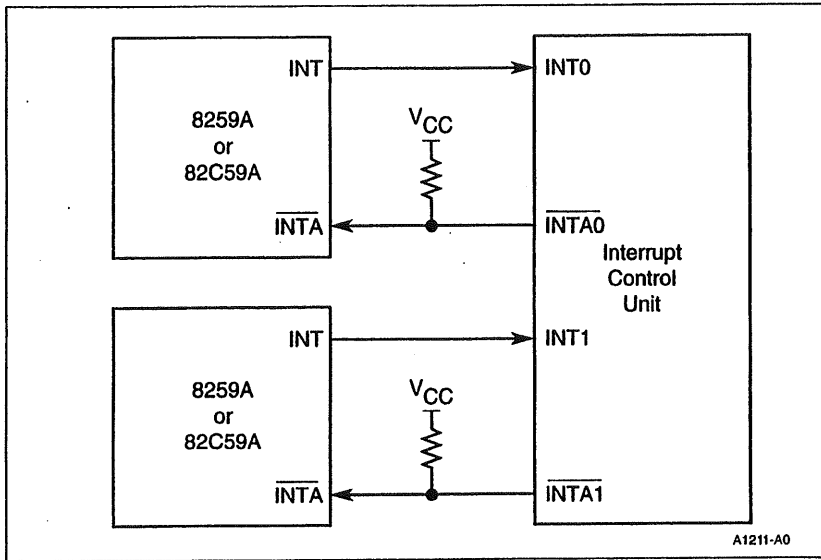


Figure 8-2. Using External 8259A Modules in Cascade Mode

### 8.2.3.1 Special Fully Nested Mode

Special fully nested mode is an optional feature normally used with cascade mode. It is applicable only to INT0 and INT1. In special fully nested mode, an interrupt request is serviced even if its In-Service bit is set.

In cascade mode, an 8259A controls up to eight external interrupts that share a single interrupt input pin. Special fully nested mode allows the 8259A's priority structure to be maintained. For example, assume that the CPU is servicing a low-priority interrupt from the 8259A. While the interrupt handler is executing, the 8259A receives a higher priority interrupt from one of its sources. The 8259A applies its own priority criteria to that interrupt and asserts its interrupt to the Interrupt Control Unit. Special fully nested mode allows the higher priority interrupt to be serviced even though the In-Service bit for that source is already set. A higher priority interrupt has preempted a lower priority interrupt, and interrupt nesting is fully maintained.

Special fully nested mode can also be used without cascade mode. In this case, it allows a single external interrupt pin (either INT0 or INT1) to preempt itself.

### 8.2.4 Interrupt Acknowledge Sequence

During the interrupt acknowledge sequence, the Interrupt Control Unit passes the interrupt type to the CPU. The CPU then multiplies the interrupt type by four to derive the interrupt vector address in the interrupt vector table. (“Interrupt/Exception Processing” on page 2-39 describes the interrupt acknowledge sequence and Figure 2-25 on page 2-40 illustrates the interrupt vector table.)

The interrupt types for all sources are fixed and unalterable (see Table 8-2). The Interrupt Control Unit passes these types to the CPU internally. The first external indication of the interrupt acknowledge sequence is the CPU fetch from the interrupt vector table.

In cascade mode, the external 8259A supplies the interrupt type. In this case, the CPU runs an external interrupt acknowledge cycle to fetch the interrupt type from the 8259A (see “Interrupt Acknowledge Bus Cycle” on page 3-25).

**Table 8-2. Fixed Interrupt Types**

Interrupt Name	Interrupt Type
Timer 0	8
Timer 1	18
Timer 2	19
Serial Channel 0 Receive	20
Serial Channel 0 Transmit	21
INT4	17
INT0	12
INT1	13
INT2	14
INT3	15

### 8.2.5 Polling

In some applications, it is desirable to poll the Interrupt Control Unit. The CPU polls the Interrupt Control Unit for any pending interrupts, and software can service interrupts whenever it is convenient. The Poll and Poll Status registers support polling.

Software reads the Poll register to get the type of the highest priority pending interrupt, then calls the corresponding interrupt handler. Reading the Poll register also acknowledges the interrupt. This clears the Interrupt Request bit and sets the In-Service bit for the interrupt. The Poll Status register has the same format as the Poll register, but reading the Poll Status register does **not** acknowledge the interrupt.

**INTERRUPT CONTROL UNIT****8.2.6 Edge and Level Triggering**

The external interrupts (INT4:0) can be programmed for either edge or level triggering (see "Interrupt Control Registers" on page 8-12). Both types of triggering are active high. An edge-triggered interrupt is generated by a zero-to-one transition on an external interrupt pin. The pin must remain high until after the CPU acknowledges the interrupt, then must go low to reset the edge-detection circuitry. (See the current data sheet for timing requirements.) The edge-detection circuitry must be reset to enable further interrupts to occur.

A level-triggered interrupt is generated by a valid logic one on the external interrupt pin. The pin must remain high until after the CPU acknowledges the interrupt. Unlike edge-triggered interrupts, level-triggered interrupts will continue to occur if the pin remains high. A level-triggered external interrupt pin must go low before the EOI command to prevent another interrupt.

**NOTE**

When external 8259As are cascaded into the Interrupt Control Unit, INT0 and INT1 must be programmed for level-triggered interrupts.

**8.2.7 Additional Latency and Response Time**

The Interrupt Control Unit adds 5 clocks to the interrupt latency of the CPU. Cascade mode adds 13 clocks to the interrupt response time because the CPU must run the interrupt acknowledge bus cycles. (See Figure 8-3 on page 8-11 and Figure 2-27 on page 2-46.)



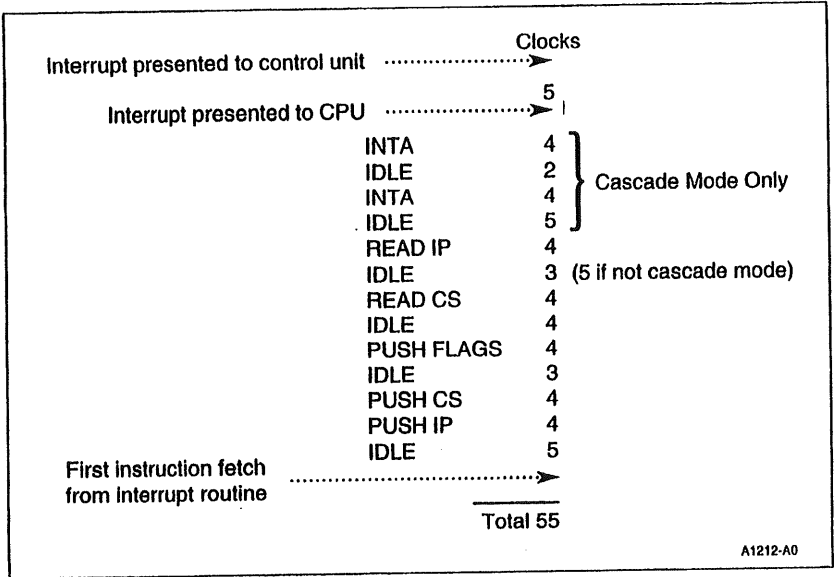


Figure 8-3. Interrupt Control Unit Latency and Response Time

### 8.3 PROGRAMMING THE INTERRUPT CONTROL UNIT

Table 8-3 lists the Interrupt Control Unit registers with their Peripheral Control Block offset addresses. The remainder of this section describes the functions of the registers.

Table 8-3. Interrupt Control Unit Registers

Register Name	Offset Address
INT3 Control	1EH
INT2 Control	1CH
INT1 Control	1AH
INT0 Control	18H
INT4 Control	16H
Serial Control	14H
Timer Control	12H
Interrupt Status	10H
Interrupt Request	0EH

Table 8-3. Interrupt Control Unit Registers (Continued)

Register Name	Offset Address
In-Service	0CH
Priority Mask	0AH
Interrupt Mask	08H
Poll Status	06H
Poll	04H
EOI	02H

### 8.3.1 Interrupt Control Registers

Each interrupt source has its own Interrupt Control register. The Interrupt Control register allows you to define the behavior of each interrupt source. Figure 8-4 shows the registers for the timers and serial channel, Figure 8-5 shows the registers for INT4:2, and Figure 8-6 shows the registers for INT0 and INT1.

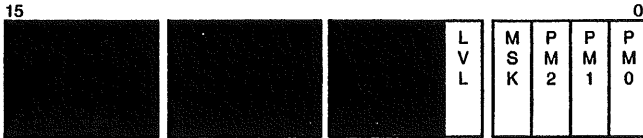
All Interrupt Control registers have a three-bit field (PM2:0) that defines the priority level for the interrupt source and a mask bit (MSK) that enables or disables the interrupt source. The mask bit is the same as the one in the Interrupt Mask register. Modifying a bit in either register also modifies that same bit in the other register.

The Interrupt Control registers for the external interrupt pins also have a bit (LVL) that selects level-triggered or edge-triggered mode for that interrupt. (See "Edge and Level Triggering" on page 8-10.)

The Interrupt Control registers for the cascadable external interrupt pins (INT0 and INT1) have two additional bits to support the external 8259As. The CAS bit enables cascade mode, and the SFNM bit enables special fully nested mode.



**Register Name:** Interrupt Control Register (non-cascadable pins)  
**Register Mnemonic:** I2CON, I3CON, I4CON  
**Register Function:** Control register for the non-cascadable external internal interrupt pins



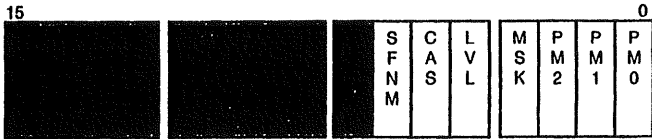
A1214-A0

Bit Mnemonic	Bit Name	Reset State	Function
LVL	Level-trigger	0	Selects the interrupt triggering mode: 0 = edge triggering 1 = level triggering.
MSK	Interrupt Mask	1	Clear to enable interrupts from this source.
PM2:0	Priority Level	111	Defines the priority level for this source.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

**Figure 8-5. Interrupt Control Register for Noncascadable External Pins**

**Register Name:** Interrupt Control Register (cascadable pins)  
**Register Mnemonic:** I0CON, I1CON  
**Register Function:** Control register for the cascadable external interrupt pins



A1215-A0

Bit Mnemonic	Bit Name	Reset State	Function
SFNM	Special Fully Nested Mode	0	Set to enable special fully nested mode.
CAS	Cascade Mode	0	Set to enable cascade mode.
LVL	Level-trigger	0	Selects the interrupt triggering mode: 0 = edge triggering 1 = level triggering. The LVL bit must be set when external 8259As are cascaded into the Interrupt Control Unit.
MSK	Interrupt Mask	1	Clear to enable interrupts from this source.
PM2:0	Priority Level	111	Defines the priority level for this source.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

**Figure 8-6. Interrupt Control Register for Cascadable Interrupt Pins**

### 8.3.2 Interrupt Request Register

The Interrupt Request register (Figure 8-7) has one bit for each interrupt source. When a source requests an interrupt, its Interrupt Request bit is set (without regard to whether the interrupt is masked). The Interrupt Request bit is cleared when the interrupt is acknowledged. An external interrupt pin must remain asserted until its interrupt is acknowledged. Otherwise, the Interrupt Request bit will be cleared, but the interrupt will not be serviced.

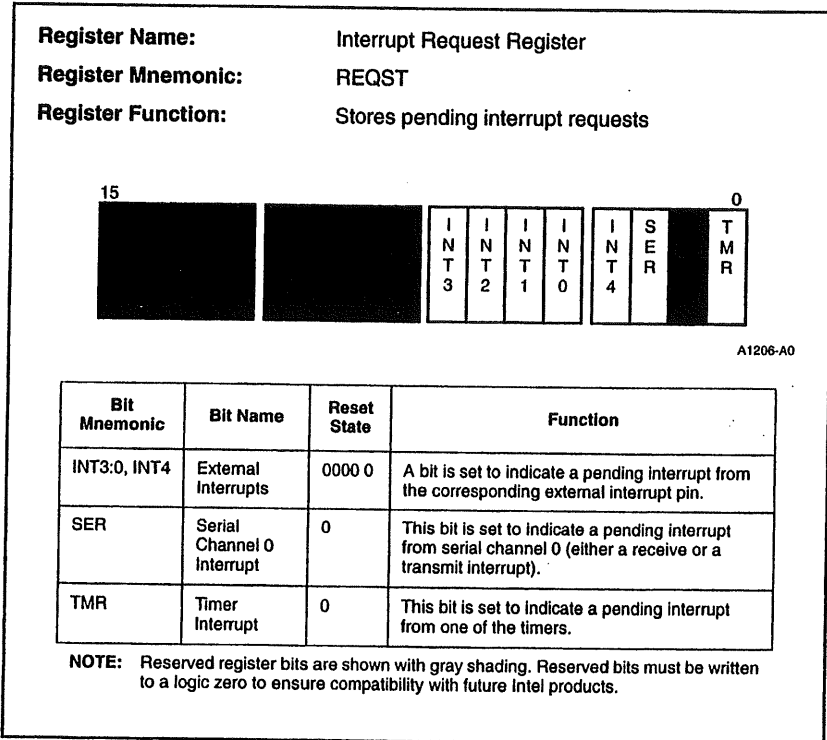
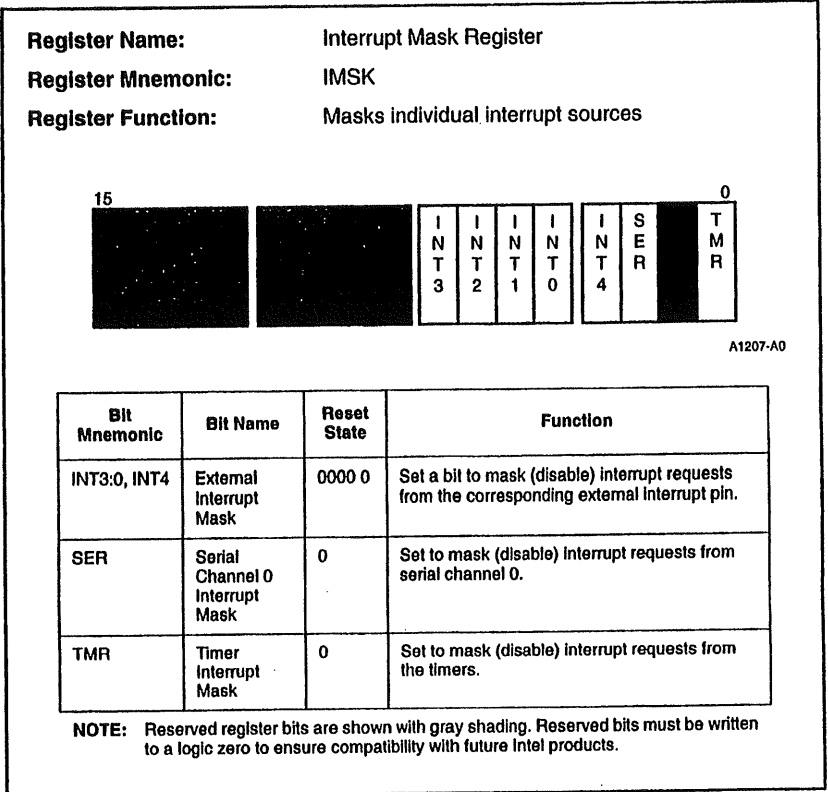


Figure 8-7. Interrupt Request Register

**8.3.3 Interrupt Mask Register**

The Interrupt Mask register (Figure 8-8) contains a mask bit for each interrupt source. This register allows you to mask (disable) individual interrupts. Set a mask bit to disable interrupts from the corresponding source. The mask bit is the same as the one in the Interrupt Control register. Modifying a bit in either register also modifies that same bit in the other register.



**Figure 8-8. Interrupt Mask Register**

### 8.3.4 Priority Mask Register

The Priority Mask register (Figure 8-9) contains a three-level field that holds a priority value. This register allows you to mask interrupts based on their priority levels. Write a priority value to the PM2:0 field to specify the lowest priority interrupt to be serviced. This disables (masks) any interrupt source whose priority is lower than the PM2:0 value. After reset, the Priority Mask register is set to the lowest priority (seven), which enables all interrupts of any priority.

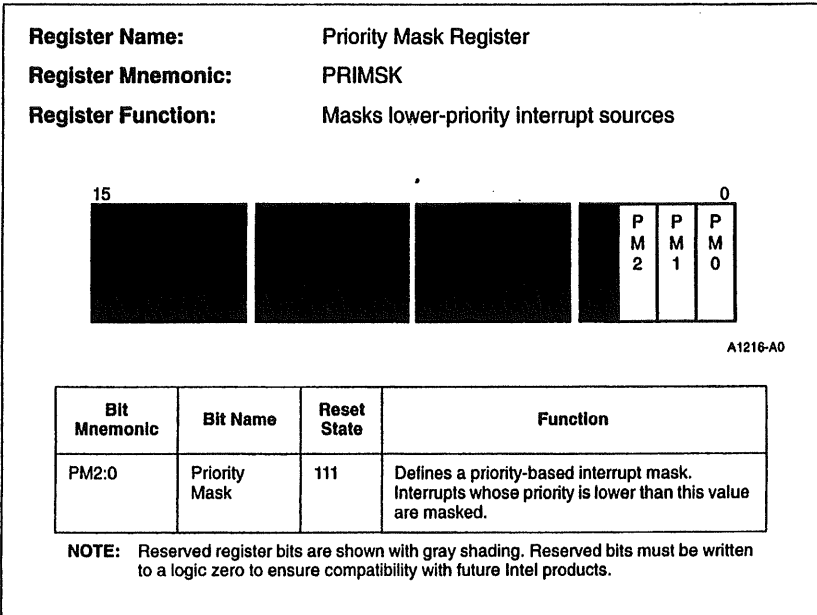


Figure 8-9. Priority Mask Register

### 8.3.5 In-Service Register

The In-Service register has a bit for each interrupt source. The bits indicate which source's interrupt handlers are currently executing. The In-Service bit is set when an interrupt is acknowledged; the interrupt handler must clear it with an End-of-Interrupt (EOI) command. The Interrupt Control Unit uses the In-Service register to support interrupt nesting.



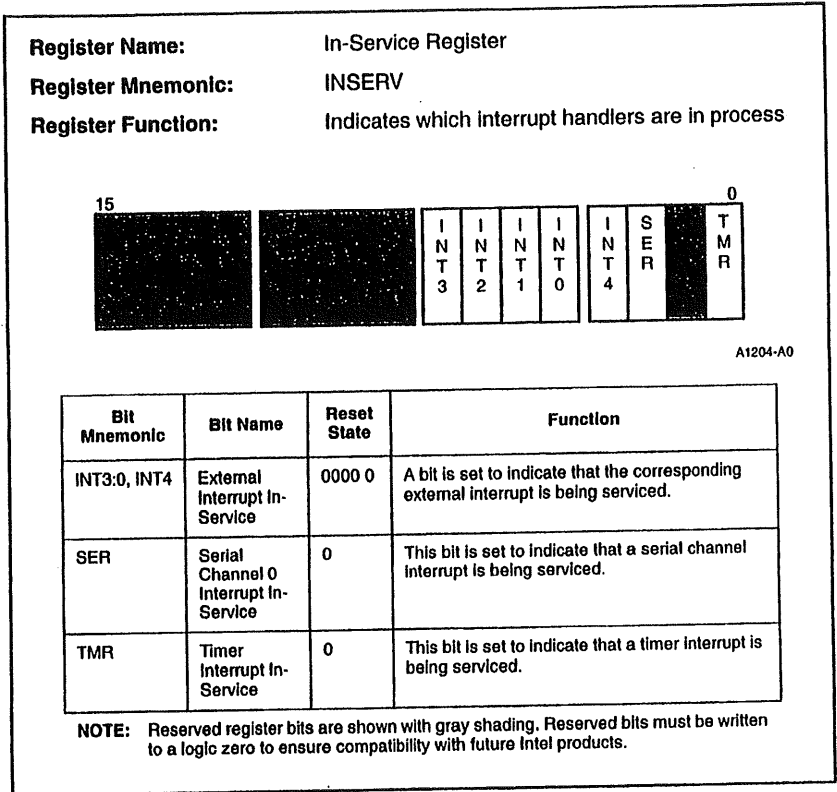


Figure 8-10. In-Service Register

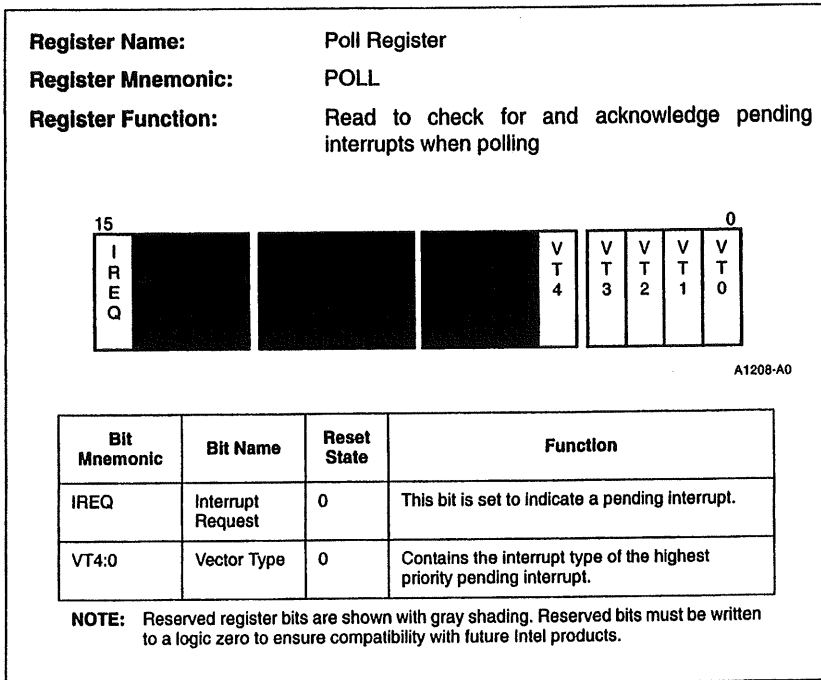
### 8.3.6 Poll and Poll Status Registers

The Poll and Poll Status registers allow you to poll the Interrupt Control Unit and service interrupts through software. You can read these registers to determine whether an interrupt is pending and, if so, the interrupt type. The registers contain identical information, but reading them produces different results.

## INTERRUPT CONTROL UNIT

Reading the Poll register (Figure 8-11) acknowledges the pending interrupt, just as if the CPU had started the interrupt vectoring sequence. The Interrupt Control Unit updates the Interrupt Request, In-Service, Poll, and Poll Status registers, as it does in the normal interrupt acknowledge sequence. However, the processor does not run an interrupt acknowledge sequence or fetch the vector from the vector table. Instead, software must read the interrupt type and execute the proper routine to service the pending interrupt.

Reading the Poll Status register (Figure 8-12) does not acknowledge the pending interrupt or update the registers.



**Figure 8-11. Poll Register**

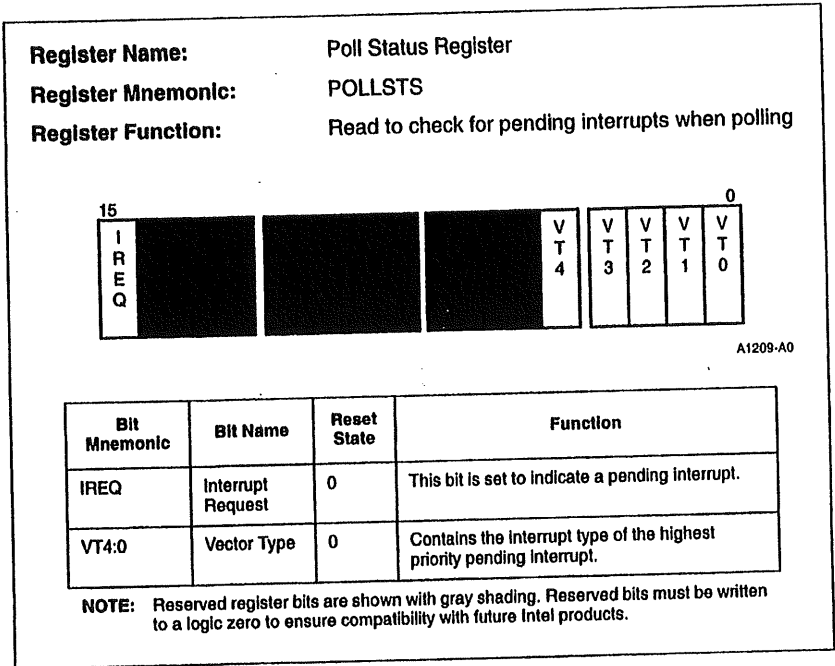


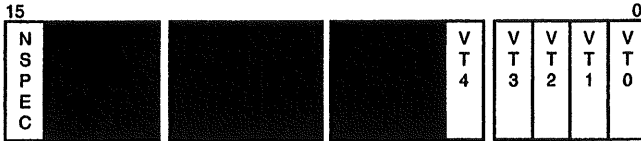
Figure 8-12. Poll Status Register

### 8.3.7 End-of-Interrupt (EOI) Register

The End-of-Interrupt register (Figure 8-13) issues an End-of-Interrupt (EOI) command to the Interrupt Control Unit, which clears the In-Service bit for the associated interrupt. An interrupt handler typically ends with an EOI command. There are two types of EOI commands: nonspecific and specific. A nonspecific EOI simply clears the In-Service bit of the highest priority interrupt. To issue a nonspecific EOI command, set the NSPEC bit. (Write 8000H to the EOI register.)

A specific EOI clears a particular In-Service bit. To issue a specific EOI command, clear the NSPEC bit and write the VT4:0 bits with the interrupt type of the interrupt whose In-Service bit you wish to clear. For example, to clear the In-Service bit for INT2, write 000BH to the EOI register. The timer interrupts share an In-Service bit. To clear the In-Service bit for any timer interrupt with a specific EOI, write 0008H (interrupt type 8) to the EOI register.

**Register Name:** End-of-Interrupt Register  
**Register Mnemonic:** EOI  
**Register Function:** Used to issue an EOI command



A1210-A0

Bit Mnemonic	Bit Name	Reset State	Function
NSPEC	Nonspecific EOI	0	Set to issue a nonspecific EOI.
VT4:0	Interrupt Type	0 0000	Write with the interrupt type of the interrupt whose In-Service bit is to be cleared.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

**Figure 8-13. End-of-Interrupt Register**

### 8.3.8 Interrupt Status Register

The Interrupt Status register (Figure 8-14) contains one bit for each interrupt that shares an interrupt source and one bit for the nonmaskable interrupt (NMI). A bit is set to indicate a pending interrupt and is cleared when the interrupt request is acknowledged. Any number of bits can be set at any one time.

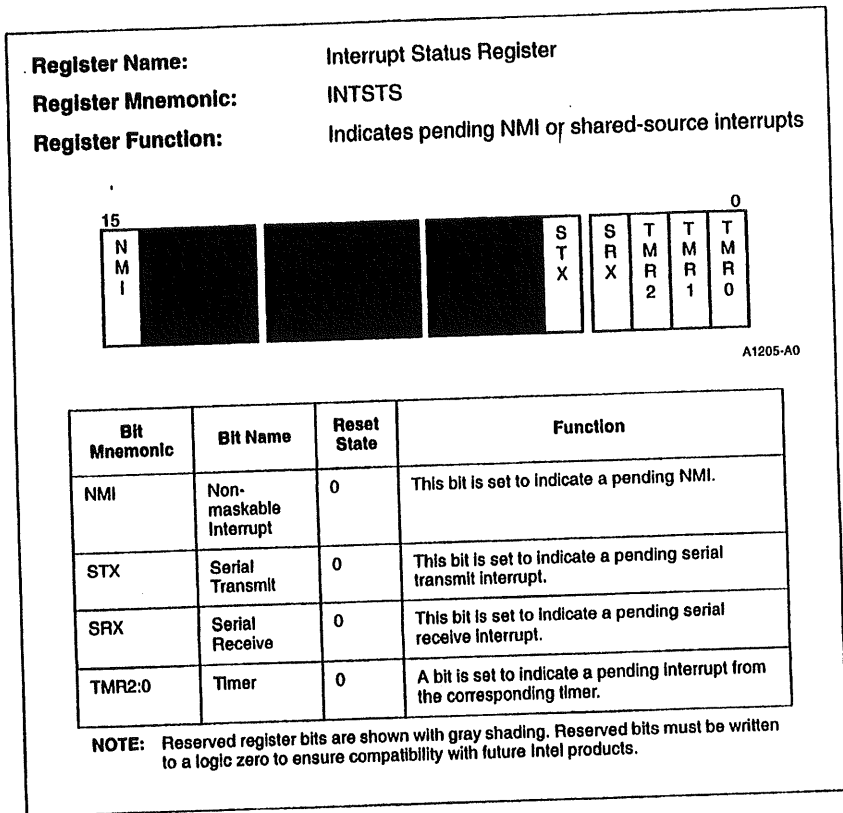


Figure 8-14. Interrupt Status Register

### 8.3.9 Initializing the Interrupt Control Unit

Follow these steps to initialize the Interrupt Control Unit.

1. Determine which interrupt sources you want to use.
2. Determine whether to use the default priority scheme or devise your own.

## INTERRUPT CONTROL UNIT

3. Program the Interrupt Control register for each interrupt source.
  - For external interrupt pins, select edge or level triggering.
  - For INT0 or INT1, enable cascade mode, special fully nested mode, or both, if you wish to use them.
  - If you are using a custom priority scheme, program the priority level for each interrupt source.
4. Program the Priority Mask with a priority mask level, if you wish to mask interrupts based on priority. (The default is level seven, which enables all interrupt levels.)
5. Set the mask bit in the Interrupt Mask register for any interrupts that you wish to disable.

Example 8-1 shows sample code to initialize the Interrupt Control Unit.

```

$mod186
name          example_80C186_ICU_initialization
;
;This routine configures the interrupt controller to provide two cascaded
;interrupt inputs (through an external 8259A connected to INT0 and INTA0#)
;and two direct interrupt inputs connected to INT1 and INT3. The default
;priorities are used.
;
;The example assumes that the register addresses have been properly defined.
;
code          segment
              assume cs:code
set_int_     proc near
              push dx
              push ax
              mov  ax,0110111B          ;cascade mode, priority seven
              mov  dx,I0CON            ;INT0 control register
              out  dx,ax
              mov  ax,01001101B       ;unmask INT1 and INT3
              mov  dx,IMASK
              out  dx,ax
              pop  ax
              pop  dx
              ret
set_int_     endp
code         ends
end

```

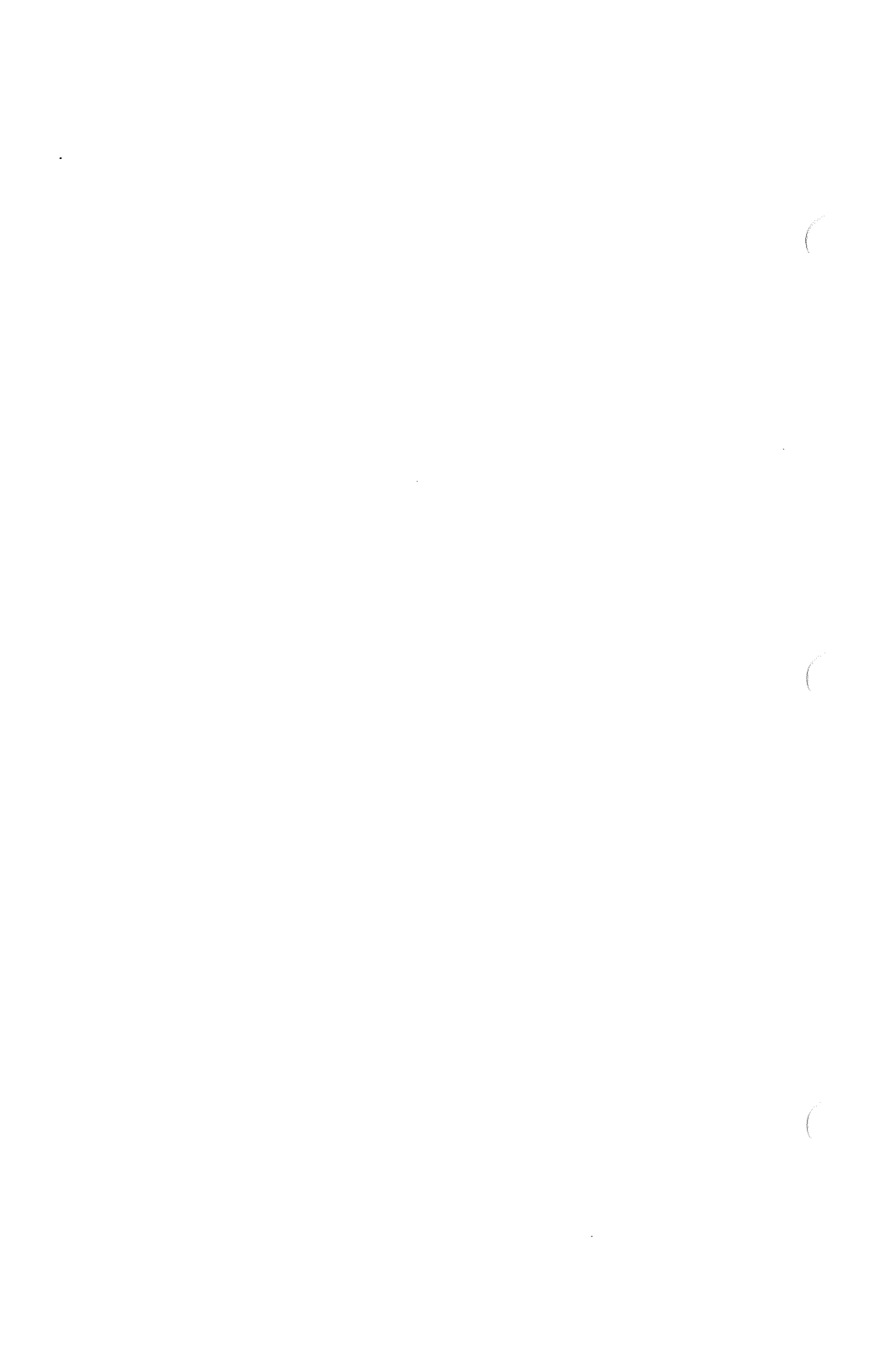
**Example 8-1. Initializing the Interrupt Control Unit**

**intel.**<sup>®</sup>

**9**

# **Timer/Counter Unit**

**I**





## CHAPTER 9 TIMER/COUNTER UNIT

The Timer/Counter Unit can be used in many applications. Some of these applications include a real-time clock, a square-wave generator and a digital one-shot. All of these can be implemented in a system design. A real-time clock can be used to update time-dependent memory variables. A square-wave generator can be used to provide a system clock tick for peripheral devices. (See "Timer/Counter Unit Application Examples" on page 9-17 for code examples that configure the Timer/Counter Unit for these applications.)

### 9.1 FUNCTIONAL OVERVIEW

The Timer/Counter Unit is composed of three independent 16-bit timers (see Figure 9-1). The operation of these timers is independent of the CPU. The internal Timer/Counter Unit can be modeled as a single counter element, time-multiplexed to three register banks. The register banks are dual-ported between the counter element and the CPU. During a given bus cycle, the counter element and CPU can both access the register banks; these accesses are synchronized.

The Timer/Counter Unit is serviced over four clock periods, one timer during each clock, with an idle clock at the end (see Figure 9-2). No connection exists between the counter element's sequencing through timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are asynchronous. This time-multiplexed scheme results in a delay of  $2\frac{1}{2}$  to  $6\frac{1}{2}$  CLKOUT periods from timer input to timer output.

Each timer keeps its own running count and has a user-defined maximum count value. Timers 0 and 1 can use one maximum count value (single maximum count mode) or two alternating maximum count values (dual maximum count mode). Timer 2 can use only one maximum count value. The control register for each timer determines the counting mode to be used. When a timer is serviced, its present count value is incremented and compared to the maximum count for that timer. If these two values match, the count value resets to zero. The timers can be configured either to stop after a single cycle or to run continuously.

Timers 0 and 1 are functionally identical. Figure 9-3 illustrates their operation. Each has a latched, synchronized input pin and a single output pin. Each timer can be clocked internally or externally. Internally, the timer can either increment at  $\frac{1}{4}$  CLKOUT frequency or be prescaled by Timer 2. A timer that is prescaled by Timer 2 increments when Timer 2 reaches its maximum count value.

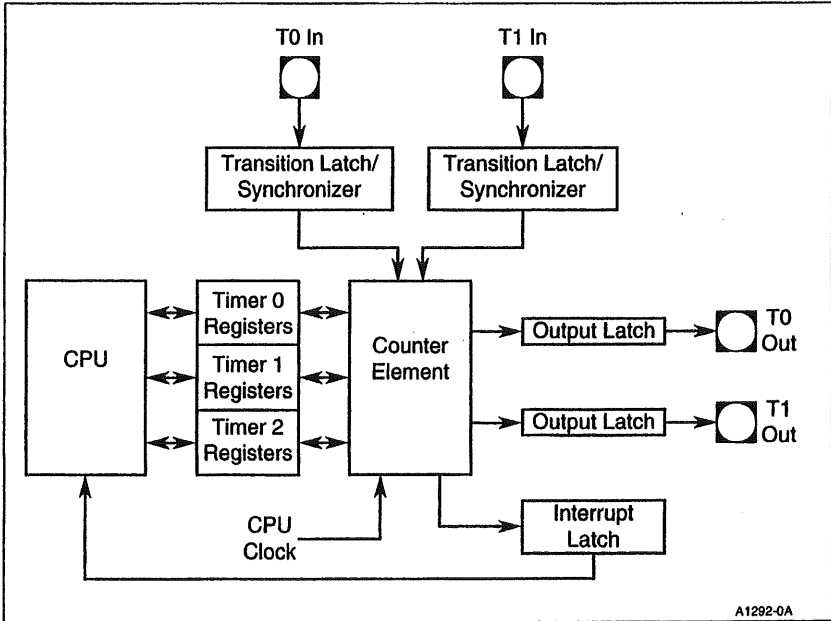


Figure 9-1. Timer/Counter Unit Block Diagram



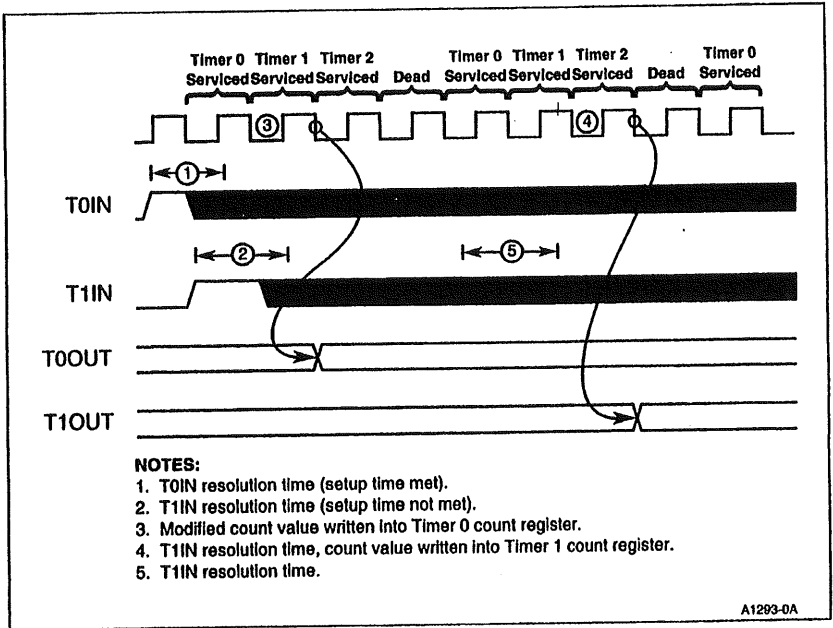
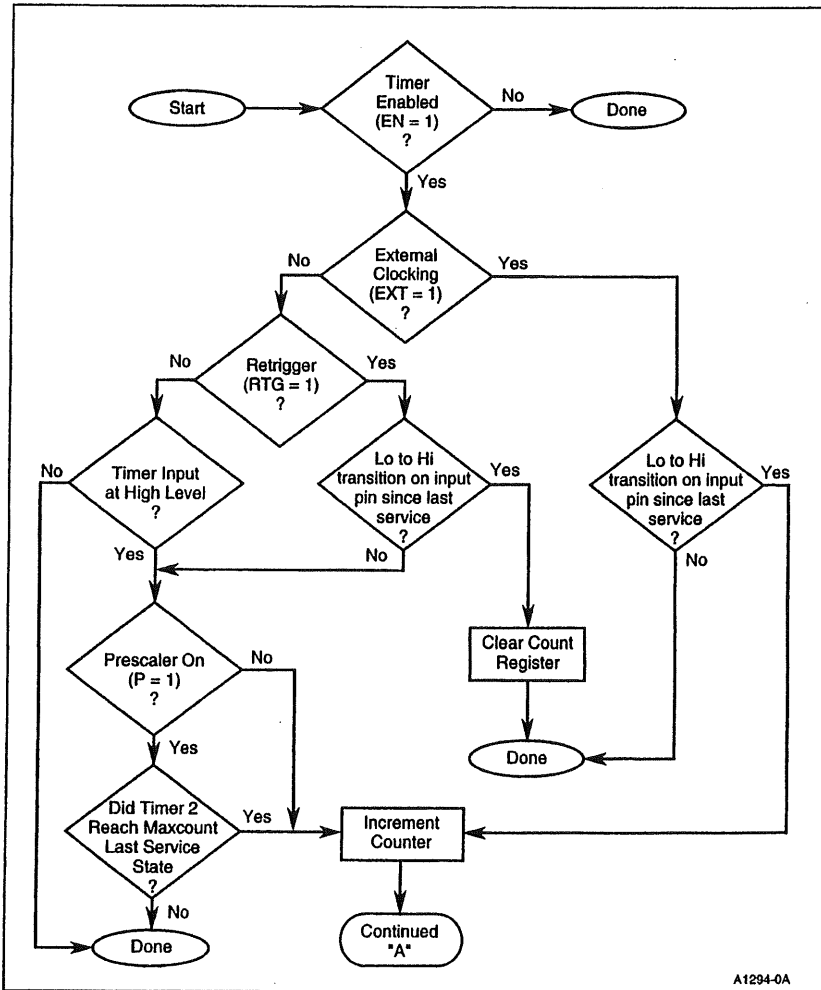


Figure 9-2. Counter Element Multiplexing and Timer Input Synchronization



A1294-0A

Figure 9-3. Timers 0 and 1 Flow Chart

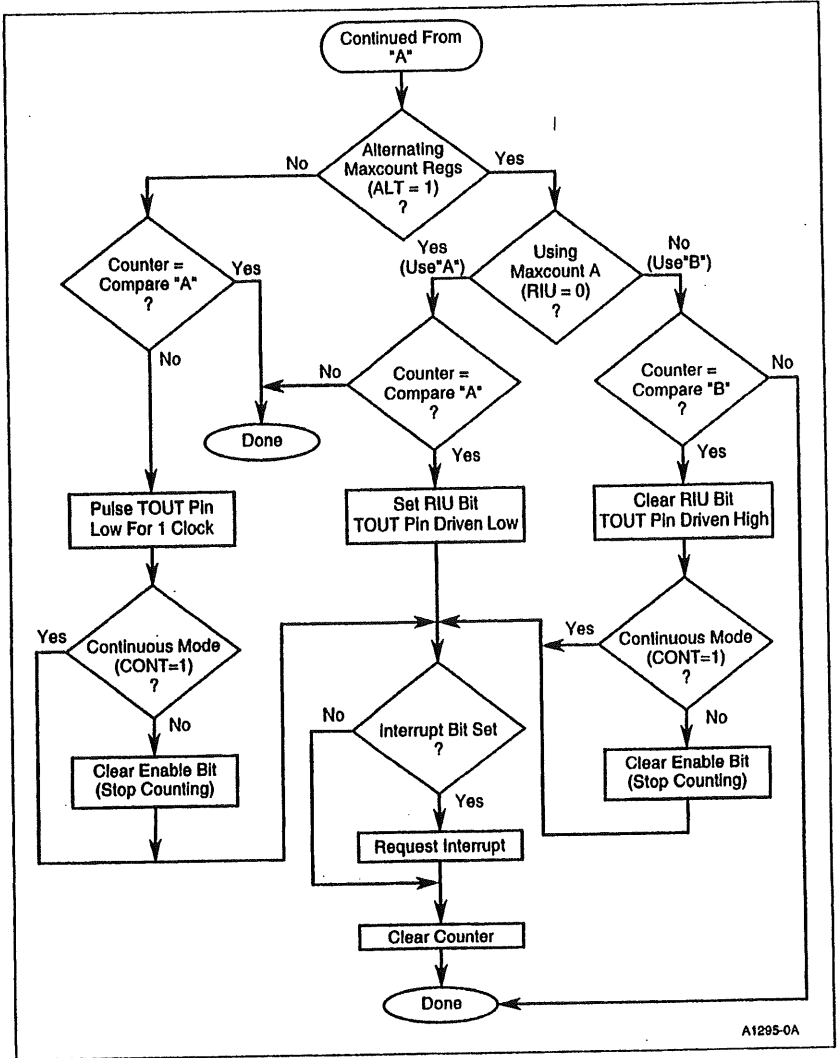


Figure 9-3. Timers 0 and 1 Flow Chart (Continued)

When configured for internal clocking, the Timer/Counter Unit uses the input pins either to enable timer counting or to retrigger the associated timer. Externally, a timer increments on low-to-high transitions on its input pin (up to  $\frac{1}{4}$  CLKOUT frequency).

Timers 0 and 1 each have a single output pin. Timer output can be either a single pulse, indicating the end of a timing cycle, or a variable duty cycle wave. These two output options correspond to single maximum count mode and dual maximum count mode, respectively (Figure 9-4). Interrupts can be generated at the end of every timing cycle.

Timer 2 has no input or output pins and can be operated only in single maximum count mode (Figure 9-4). It can be used as a free-running clock and as a prescaler to Timers 0 and 1. Timer 2 can be clocked only internally, at  $\frac{1}{4}$  CLKOUT frequency. Timer 2 can also generate interrupts at the end of every timing cycle.

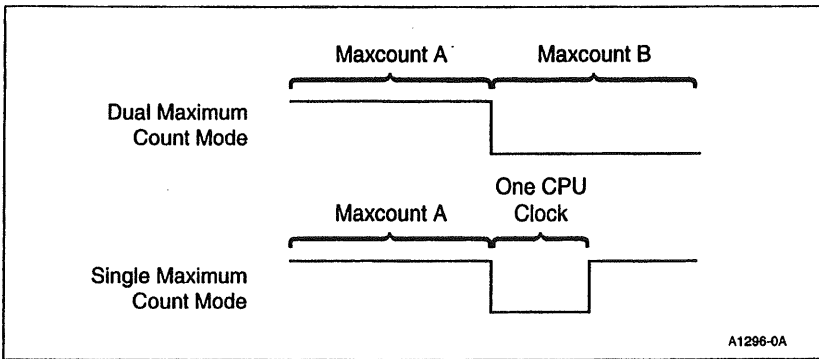
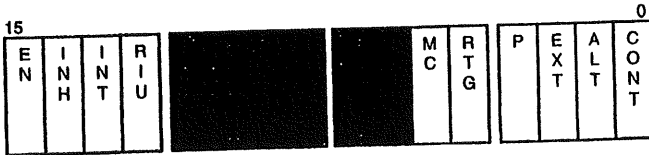


Figure 9-4. Timer/Counter Unit Output Modes

## 9.2 PROGRAMMING THE TIMER/COUNTER UNIT

Each timer has three registers: a Timer Control register (Figure 9-5 and Figure 9-6), a Timer Count register (Figure 9-7) and a Timer Maxcount Compare register (Figure 9-8). Timers 0 and 1 also have access to an additional Maxcount Compare register. The Timer Control register controls timer operation. The Timer Count register holds the current timer count value, and the Maxcount Compare register holds the maximum timer count value.

**Register Name:** Timer 0 and 1 Control Registers  
**Register Mnemonic:** T0CON, T1CON  
**Register Function:** Defines Timer 0 and 1 operation.



A1297-0A

Bit Mnemonic	Bit Name	Reset State	Function
EN	Enable	0	Set to enable the timer. This bit can be written only when the INH bit is set.
INH	Inhibit	X	Set to enable writes to the EN bit. Clear to ignore writes to the EN bit. The INH bit is not stored; it always reads as zero.
INT	Interrupt	X	Set to generate an interrupt request when the Count register equals a Maximum Count register. Clear to disable interrupt requests.
RIU	Register In Use	X	Indicates which compare register is in use. When set, the current compare register is Maxcount Compare B; when clear, it is Maxcount Compare A.
MC	Maximum Count	X	This bit is set when the counter reaches a maximum count. <b>The MC bit must be cleared by writing to the Timer Control register. This is not done automatically.</b> If MC is clear, the counter has not reached a maximum count.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 9-5. Timer 0 and Timer 1 Control Registers





**Register Name:** Timer 2 Control Register  
**Register Mnemonic:** T2CON  
**Register Function:** Defines Timer 2 operation.



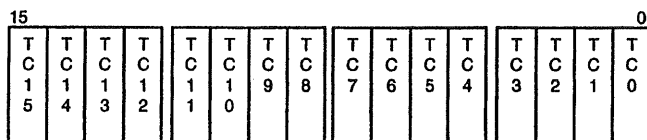
A1298-0A

Bit Mnemonic	Bit Name	Reset State	Function
EN	Enable	0	Set to enable the timer. This bit can be written only when the INH bit is set.
INH	Inhibit	X	Set to enable writes to the EN bit. Clear to ignore writes to the EN bit. The INH bit is not stored; it always reads as zero.
INT	Interrupt	X	Set to generate an interrupt request when the Count register equals a Maximum Count register. Clear to disable interrupt requests.
MC	Maximum Count	X	This bit is set when the counter reaches a maximum count. <b>The MC bit must be cleared by writing to the Timer Control register. This is not done automatically.</b> If MC is clear, the counter has not reached a maximum count.
CONT	Continuous Mode	X	Set to cause the timer to run continuously. Clear to disable the counter (clear the EN bit) after each counting sequence.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 9-6. Timer 2 Control Register

**Register Name:** Timer Count Register  
**Register Mnemonic:** T0CNT, T1CNT, T2CNT  
**Register Function:** Contains the current timer count.



A1289-0A

Bit Mnemonic	Bit Name	Reset State	Function
TC15:0	Timer Count Value	XXXXH	Contains the current count of the associated timer.

**Figure 9-7. Timer Count Registers**

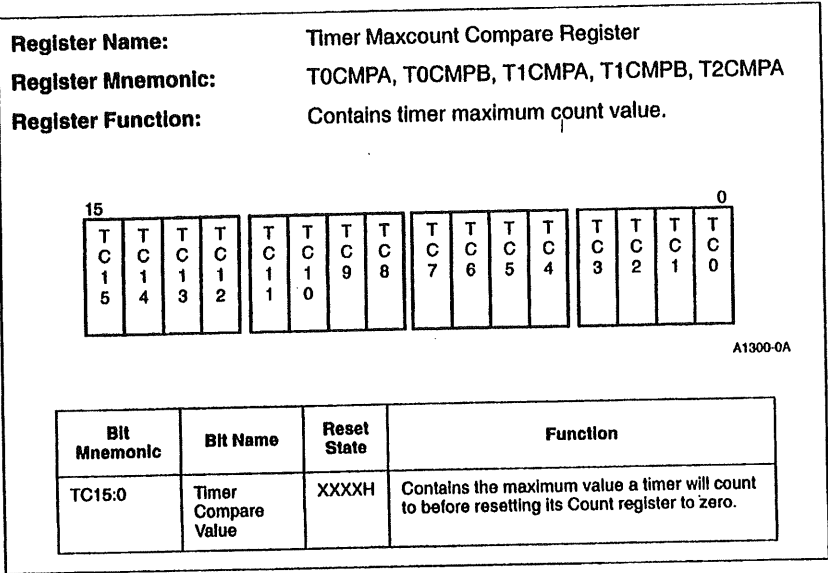


Figure 9-8. Timer Maxcount Compare Registers

### 9.2.1 Initialization Sequence

When initializing the Timer/Counter Unit, the following sequence is suggested:

1. If timer interrupts will be used, program interrupt vectors into the Interrupt Vector Table.
2. Clear the Timer Count register. This must be done before the timer is enabled because the count register is undefined at reset. Clearing the count register ensures that counting begins at zero.
3. Write the desired maximum count value to the Timer Maxcount Compare register. For dual maximum count mode, write a value to both Maxcount Compare A and B.
4. Program the Timer Control register to enable the timer. When using Timer 2 to prescale another timer, enable Timer 2 last. If Timer 2 is enabled first, it will be at an unknown point in its timing cycle when the timer to be prescaled is enabled. This results in an unpredictable duration of the first timing cycle for the prescaled timer.

### 9.2.2 Clock Sources

The 16-bit Timer Count register increments once for each timer event. A timer event can be a low-to-high transition on a timer input pin (Timers 0 and 1), a pulse generated every fourth CPU clock (all timers) or a timeout of Timer 2 (Timers 0 and 1). Up to 65536 ( $2^{16}$ ) events can be counted.

Timers 0 and 1 can be programmed to count low-to-high transitions on their input pins as timer events by setting the External (EXT) bit in their control registers. Transitions on the external pin are synchronized to the CPU clock before being presented to the timer circuitry. The timer counts **transitions** on this pin. The input signal must go low, then high, to cause the timer to increment. The maximum count-rate for the timers is  $\frac{1}{4}$  the CPU clock rate (measured at CLKOUT) because the timers are serviced only once every four clocks.

All timers can use transitions of the CPU clock as timer events. For internal clocking, the timer increments every fourth CPU clock due to the counter element's time-multiplexed servicing scheme. Timer 2 can use only the internal clock as a timer event.

Timers 0 and 1 can also use Timer 2 reaching its maximum count as a timer event. In this configuration, Timer 0 or Timer 1 increments each time Timer 2 reaches its maximum count. See Table 9-1 for a summary of clock sources for Timers 0 and 1. Timer 2 must be initialized and running in order to increment values in other timer/counters.

**Table 9-1. Timer 0 and 1 Clock Sources**

EXT	P	Clock Source
0	0	Timer clocked internally at $\frac{1}{4}$ CLKOUT frequency.
0	1	Timer clocked internally, prescaled by Timer 2.
1	X	Timer clocked externally at up to $\frac{1}{4}$ CLKOUT frequency.

### 9.2.3 Counting Modes

All timers have a Timer Count register and a Maxcount Compare A register. Timers 0 and 1 also have access to a second Maxcount Compare B register. Whenever the contents of the Timer Count register equal the contents of the Maxcount Compare register, the count register resets to zero. The maximum count value will never be stored in the count register. This is because the counter element increments, compares and resets a timer in one clock cycle. Therefore, the maximum value is never written back to the count register. The Maxcount Compare register can be written at any time during timer operation.

The timer counting from its initial count (usually zero) to its maximum count (either Maxcount Compare A or B) and resetting to zero defines one timing cycle. A Maxcount Compare value of 0 implies a maximum count of 65536, a Maxcount Compare value of 1 implies a maximum count of 1, etc.

Only equivalence between the Timer Count and Maxcount Compare registers is checked. The count does not reset to zero if its value is greater than the maximum count. If the count value exceeds the Maxcount Compare value, the timer counts to 0FFFFH, increments to zero, then counts to the value in the Maxcount Compare register. Upon reaching a maximum count value, the Maximum Count (MC) bit in the Timer Control register sets. **The MC bit must be cleared by writing to the Timer Control register. This is not done automatically.** |

The Timer/Counter Unit can be configured to execute different counting sequences. The timers can operate in single maximum count mode (all timers) or dual maximum count mode (Timers 0 and 1 only). They can also be programmed to run continuously in either of these modes. The Alternate (ALT) bit in the Timer Control register determines the counting modes used by Timers 0 and 1.

All timers can use single maximum count mode, where only Maxcount Compare A is used. The timer will count to the value contained in Maxcount Compare A and reset to zero. Timer 2 can operate only in this mode.

Timers 0 and 1 can also use dual maximum count mode. In this mode, Maxcount Compare A and Maxcount Compare B are both used. The timer counts to the value contained in Maxcount Compare A, resets to zero, counts to the value contained in Maxcount Compare B, and resets to zero again. The Register In Use (RIU) bit in the Timer Control register indicates which Maxcount Compare register is currently in use.

The timers can be programmed to run continuously in single maximum count and dual maximum count modes. The Continuous (CONT) bit in the Timer Control register determines whether a timer is disabled after a single counting sequence.

### 9.2.3.1 Retriggering

The timer input pins affect timer counting in three ways (see Table 9-2). The programming of the External (EXT) and Retrigger (RTG) bits in the Timer Control register determines how the input signals are used. When the timers are clocked internally, the RTG bit determines whether the input pin enables timer counting or retriggers the current timing cycle.

**Table 9-2. Timer Retriggering**

EXT	RTG	Timer Operation
0	0	Timer counts internal events, if input pin remains high.
0	1	Timer counts internal events; count resets to zero on every low-to-high transition on the input pin.
1	X	Timer input acts as clock source.

When the EXT and RTG bits are clear, the timer counts internal timer events. In this mode, the input is level-sensitive, not edge-sensitive. A low-to-high transition on the timer input is not required for operation. The input pin acts as an external enable. If the input is high, the timer will count through its sequence, provided the timer remains enabled.

When the EXT bit is clear and the RTG bit is set, every low-to-high transition on the timer input pin causes the Count register to reset to zero. After the timer is enabled, counting begins only after the first low-to-high transition on the input pin. If another low-to-high transition occurs before the end of the timer cycle, the timer count resets to zero and the timer cycle begins again. In dual maximum count mode, the Register In Use (RIU) bit does not clear when a low-to-high transition occurs. For example, if the timer retriggers while Maxcount Compare B is in use, the timer resets to zero and counts to maximum count B before the RIU bit clears. **In dual maximum count mode, the timer retriggering extends the use of the current Maxcount Compare register.**

#### 9.2.4 Pulsed and Variable Duty Cycle Output

Timers 0 and 1 each have an output pin that can perform two functions. First, the output can be a single pulse, indicating the end of a timing cycle (single maximum count mode). Second, the output can be a level, indicating the Maxcount Compare register currently in use (dual maximum count mode). The output occurs one clock after the counter element services the timer when the maximum count is reached (see Figure 9-9).

With external clocking, the time between a transition on a timer input and the corresponding transition of the timer output varies from  $2\frac{1}{2}$  to  $6\frac{1}{2}$  clocks. This delay occurs due to the time-multiplexed servicing scheme of the Timer/Counter Unit. The exact timing depends on when the input occurs relative to the counter element's servicing of the timer. Figure 9-2 on page 9-3 shows the two extremes in timer output delay. Timer 0 demonstrates the best possible case, where the input occurs immediately before the timer is serviced. Timer 1 demonstrates the worst possible case, where the input is latched, but the setup time is not met and the input is not recognized until the counter element services the timer again.

In single maximum count mode, the timer output pin goes low for one CPU clock period (see Figure 9-4 on page 9-6). This occurs when the count value equals the Maxcount Compare A value. If programmed to run continuously, the timer generates periodic pulses.



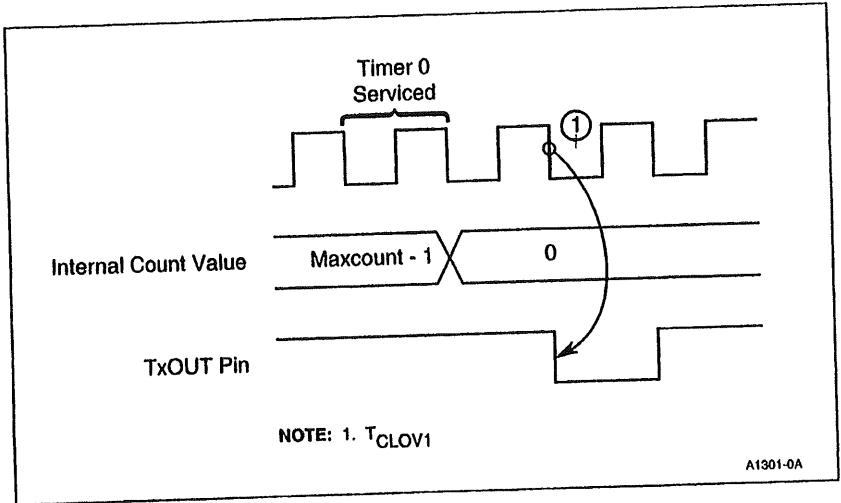


Figure 9-9. TxOUT Signal Timing

In dual maximum count mode, the timer output pin indicates which Maxcount Compare register is currently in use. A low output indicates Maxcount Compare B, and a high output indicates Maxcount Compare A (see Figure 9-4 on page 9-6). If programmed to run continuously, a repetitive waveform can be generated. For example, if Maxcount Compare A contains 10, Maxcount Compare B contains 20, and CLKOUT is 12.5 MHz, the timer generates a 33 percent duty cycle waveform at 104 KHz. The output pin always goes high at the end of the counting sequence (even if the timer is not programmed to run continuously).

### 9.2.5 Enabling/Disabling Counters

Each timer has an Enable (EN) bit in its Control register to allow or prevent timer counting. The Inhibit ( $\overline{INH}$ ) bit controls write accesses to the EN bit. Timers 0 and 1 can be programmed to use their input pins as enable functions also. If a timer is disabled, the count register does not increment when the counter element services the timer.

The Enable bit can be altered by programming or the timers can be programmed to disable themselves at the end of a counting sequence with the Continuous (CONT) bit. If the timer is not programmed for continuous operation, the Enable bit automatically clears at the end of a counting sequence. In single maximum count mode, this occurs after Maxcount Compare A is reached. In dual maximum count mode, this occurs after Maxcount Compare B is reached (Timers 0 and 1 only).

## TIMER/COUNTER UNIT

The input pins for Timers 0 and 1 provide an alternate method for enabling and disabling timer counting. When using internal clocking, the input pin can be programmed either to enable the timer or to reset the timer count, depending on the state of the Retrigger (RTG) bit in the control register. When used as an enable function, the input pin either allows (input high) or prevents (input low) timer counting. To ensure recognition of an input level, it must be valid for four CPU clocks. This is due to the counter element's time-multiplexed servicing scheme for the timers.

### 9.2.6 Timer Interrupts

All timers can generate internal interrupt requests. Although all three timers share a single interrupt request to the CPU, each has its own vector location and internal priority. Timer 0 has the highest interrupt priority and Timer 2 has the lowest.

Timer Interrupts are enabled or disabled by the Interrupt (INT) bit in the Timer Control register. If enabled, an interrupt is generated every time a maximum count value is reached. In dual maximum count mode, an interrupt is generated each time the value in Maxcount Compare A or Maxcount Compare B is reached. If the interrupt is disabled after a request has been generated, but before a pending interrupt is serviced, the interrupt request remains active (the Interrupt Controller latches the request). If a timer generates a second interrupt request before the CPU services the first interrupt request, the first request is lost.

### 9.2.7 Programming Considerations

Timer registers can be read or written whether the timer is operating or not. Since processor accesses to timer registers are synchronized with counter element accesses, a half-modified count register will never be read.

When Timer 0 and Timer 1 use an internal clock source, the input pin must be high to enable counting.

## 9.3 TIMING

Certain timing considerations need to be made with the Timer/Counter Unit. These include input setup and hold times, synchronization and operating frequency.

### 9.3.1 Input Setup and Hold Timings

To ensure recognition, setup and hold times must be met with respect to CPU clock edges. The timer input signal must be valid  $T_{CHIS}$  before the rising edge of CLKOUT and must remain valid  $T_{CHIH}$  after the same rising edge. If these timing requirements are not met, the input will not be recognized until the next clock edge.



### 9.3.2 Synchronization and Maximum Frequency

All timer inputs are latched and synchronized with the CPU clock. Because of the internal logic required to synchronize the external signals, and the multiplexing of the counter element, the Timer/Counter Unit can operate only up to  $\frac{1}{4}$  of the CLKOUT frequency. Clocking at greater frequencies will result in missed clocks.

#### 9.3.2.1 Timer/Counter Unit Application Examples

The following examples are possible applications of the Timer/Counter Unit. They include a real-time clock, a square wave generator and a digital one-shot.

### 9.3.3 Real-Time Clock

Example 9-1 contains sample code to configure Timer 2 to generate an interrupt request every 10 milliseconds. The CPU then increments memory-based clock variables.

### 9.3.4 Square-Wave Generator

A square-wave generator can be useful to act as a system clock tick. Example 9-2 illustrates how to configure Timer 1 to operate this way.

### 9.3.5 Digital One-Shot

Example 9-3 configures Timer 1 to act as a digital one-shot.

```

$mod186
name example_80186_family_timer_code

;FUNCTION:  This function sets up the timer and interrupt controller
;           to cause the timer to generate an interrupt every
;           10 milliseconds and to service interrupts to
;           implement a real time clock.
;
;           Timer 2 is used in this example because no input or
;           output signals are required.
;
;SYNTAX:    extern void far set_time(hour, minute, second, T2Compare)
;
;INPUTS:    hour - hour to set time to.
;           minute - minute to set time to.
;           second - second to set time to.
;           T2Compare - T2CMPA value (see note below)
;
;OUTPUTS:   None

;NOTE:      Parameters are passed on the stack as required by
;           high-level languages
;
;           For a CLKOUT of 16Mhz,
;
;           f(timer2)           = 16Mhz/4
;                               = 4Mhz
;                               = 0.25us for T2CMPA = 1
;
;           T2CMPA(10ms)       = 10ms/0.25us
;                               = 10e-3/0.25e-6
;                               = 40000

;substitute register offsets
T2CON   equ xxxxh           ;Timer 2 Control register
T2CMPA  equ xxxxh           ;Timer 2 Compare register
T2CNT   equ xxxxh           ;Timer 2 Counter register
TCUCON  equ xxxxh           ;Int. Control register
EOI     equ xxxxh           ;End Of Interrupt register
INTSTS  equ xxxxh           ;Interrupt Status register
timer_2_int equ 19         ;timer 2:vector type 19

data segment public 'data'
    public _hour, _minute, _second, _msec

    _hour      db ?
    _minute    db ?
    _second    db ?
    _msec      db ?

data ends

```

Example 9-1. Configuring a Real-Time Clock

```

lib_80186 segment public 'code'
    assume cs:lib_80186, ds:data

public _set_time
_set_time proc far

    push    bp                ;save caller's bp
    mov     bp, sp           ;get current top of stack

    hour    equ word ptr[bp+6] ;get parameters off stack
    minute  equ word ptr[bp+8]
    second  equ word ptr[bp+10]
    T2Compare equ word ptr[bp+12]

    push    ax                ;save registers used
    push    dx
    push    si

    push    ds                ;set interrupt vector
    xor     ax, ax
    mov     ds, ax
    mov     si, 4*timer_2_int
    mov     word ptr ds:[si], offset

timer_2_interrupt_routine
    inc     si
    inc     si
    mov     ds:[si], cs
    pop     ds

    mov     ax, hour          ;set time
    mov     _hour, al
    mov     ax, minute
    mov     _minute, al
    mov     ax, second
    mov     _second, al
    mov     _msec, 0

    mov     dx, T2CNT         ;clear Count register
    xor     ax, ax
    out     dx, al

    mov     dx, T2CMPA       ;set maximum count value
    mov     ax, T2Compare    ;see note in header above
    out     dx, al
    mov     dx, T2CON        ;set up the control word:
    mov     ax, 0E001H       ;enable counting,
    out     dx, al          ;generate interrupt on MC,
                            ;continuous counting

    mov     dx, TCUCON       ;set up interrupt controller
    xor     ax, ax          ;unmask highest priority interrupt
    out     dx, al

```

**Example 9-1. Configuring a Real-Time Clock (Continued)**

```

    sti                    ;enable interrupts

    pop    si              ;restore saved registers
    pop    dx
    pop    ax
    pop    bp              ;restore caller's bp
    ret
_set_time endp

timer_2_interrupt_routine proc far

    push   ax              ;save registers used
    push   dx
    cmp    _msec, 99      ;has 1 sec passed?
    jae    bump_second    ;if above or equal...
    inc    _msec
    jmp    short reset_int_ctl

bump_second:
    mov    _msec, 0       ;reset millisecond
    cmp    _minute, 59    ;has 1 minute passed?
    jae    bump_minute
    inc    _second
    jmp    short reset_int_ctl

bump_minute:
    mov    _second, 0     ;reset second
    cmp    _minute, 59    ;has 1 hour passed?
    jae    bump_hour
    inc    _minute
    jmp    short reset_int_ctl

bump_hour:
    mov    _minute, 0     ;reset minute
    cmp    _hour, 12      ;have 12 hours passed?
    jae    reset_hour
    inc    _hour
    jmp    reset_int_ctl

reset_hour:
    mov    _hour, 1       ;reset hour

reset_int_ctl:
    mov    dx, EOI
    mov    ax, 8000h      ;non-specific end of interrupt
    out    dx, al
    pop    dx
    pop    ax
    iret

timer_2_interrupt_routine endp

lib_80186    ends
end

```

Example 9-1. Configuring a Real-Time Clock (Continued)

```

$mod186
name      example_timer1_square_wave_code

;FUNCTION: This function generates a square wave of given
;          frequency and duty cycle on Timer 1 output pin.
;
; SYNTAX:  extern void far clock(int mark, int space)
;
; INPUTS:  mark - This is the mark (1) time.
;          space - This is the space (0) time.
;
;          The register compare value for a given time can be
;          easily calculated from the formula below.
;
;          CompareValue = (req_pulse_width*f)/4
;
; OUTPUTS: None
;
; NOTE:    Parameters are passed on the stack as required by
;          high-level Languages
;
          TICMPA equ xxxxH      ;substitute register offsets
          TICMPB equ xxxxH
          TICNT  equ xxxxH
          TICON  equ xxxxH

lib_80186 segment public 'code'
          assume cs:lib_80186

public   _clock
_clock  proc far

          push  bp                ;save caller's bp
          mov   bp, sp            ;get current top of stack
          _space equ word ptr [bp+6] ;get parameters off the stack
          _mark  equ word ptr [bp+8]

          push  ax                ;save registers that will be
          push  bx                ;modified
          push  dx

          mov   dx, TICMPA        ;set mark time
          mov   ax, _mark
          out   dx, al

          mov   dx, TICMPB        ;set space time
          mov   ax, _space
          out   dx, al

          mov   dx, TICNT        ;Clear Timer 1 Counter
          xor   ax, ax
          out   dx, al

          mov   dx, TICON        ;start Timer 1
          mov   ax, C003H
          out   dx, al

```

Example 9-2. Configuring a Square-Wave Generator

```

    pop    dx                ;restore saved registers
    pop    bx
    pop    ax

    pop    bp                ;restore caller's bp
    ret
_clock   endp
lib_80186 ends
end

```

**Example 9-2. Configuring a Square-Wave Generator (Continued)**

```

$mod186
name example_timer1_1_shot_code

; FUNCTION: This function generates an active-low one-shot pulse
;           on Timer 1 output pin.
;
; SYNTAX:   extern void far one_shot(int CMPB);
;
; INPUTS:   CMPB - This is the TICMPB value required to generate a
;           pulse of a given pulse width. This value is calculated
;           from the formula below.
;
;            $CMPB = (req\_pulse\_width * f) / 4$ 
;
; OUTPUTS:  None
;
; NOTE:     Parameters are passed on the stack as required by
;           high-level languages
;
    TICNT equ xxxxxH        ;substitute register offsets
    TICMPA equ xxxxxH
    TICMPB equ xxxxxH
    TICON equ xxxxxH
    MaxCount equ 0020H

lib_80186 segment public 'code'
    assume cs:lib_80186

public    _one_shot
_one_shot proc far

    push  bp                ;save caller's bp
    mov   bp, sp           ;get current top of stack

```

**Example 9-3. Configuring a Digital One-Shot**

```

_CMPB equ word ptr[bp+6] ;get parameter off the stack

push ax ;save registers that will be
push dx ;modified
mov dx, T1CNT ;Clear Timer 1 Counter
xor ax, ax
out dx, al
mov dx, T1CMPA ;set time before t_shot to 0
mov ax, 1
out dx, al
mov dx, T1CMPB ;set pulse time
mov ax, _CMPB
out dx, al
mov dx, T1CON
mov ax, C002H ;start Timer 1
out dx, al

CountDown: in ax, dx ;read in T1CON
test ax, MaxCount ;max count occurred?
jz CountDown ;no: then wait
and ax, not MaxCount ;clear max count bit
out dx, al ;update T1CON

pop dx ;restore saved registers
pop ax
pop bp ;restore caller's bp
ret

_one_shot endp
lib_80186 ends
end

```

**Example 9-3. Configuring a Digital One-Shot (Continued)**





intel®

10

**Serial  
Communications  
Unit**

**I**





## CHAPTER 10 SERIAL COMMUNICATIONS UNIT

### 10.1 INTRODUCTION

The Serial Communications Unit is composed of two identical serial ports, or channels. Each serial port is independent of the other. This chapter describes the operation of a single serial port.

The serial port implements several industry-standard asynchronous communications protocols, and it readily interfaces to many different processors over a standard serial interface. Several processors and systems can be connected to a common serial bus using a multiprocessor protocol. The serial port also implements a simple synchronous protocol. The synchronous protocol is most commonly used to expand the number of I/O pins with shift registers.

#### Features:

- Full duplex operation
- Programmable seven, eight or nine data bits in asynchronous mode
- Independent baud rate generator
- Maximum baud rate of 1/16 the processor clock
- Double-buffered transmit and receive
- Clear-to-Send feature for transmission
- Break character transmission and detection
- Programmable even, odd or no parity
- Detects both framing and overrun errors
- Supports interrupt on transmit and receive

#### 10.1.1 Asynchronous Communications

Asynchronous communications protocols allow different devices to communicate without a common reference clock. The devices communicate at a common baud rate, or bits per second. Data is transmitted and received in *frames*. A *frame* is a sequence of bits shifted serially onto or off the communications line.

Each asynchronous frame consists of a start bit (always a logic zero), followed by the data bits and a terminating stop bit. The serial port can transmit and receive seven, eight or nine data bits. The last data bit can optionally be replaced by an even or odd parity bit. Figure 10-1 shows a typical 10-bit frame.

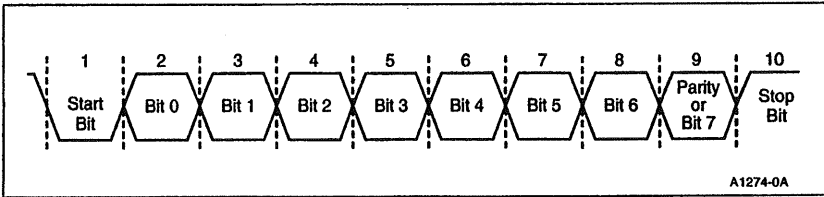


Figure 10-1. Typical 10-Bit Asynchronous Data Frame

When discussing asynchronous communications, it makes sense to talk about the receive machine (RX machine) and the transmit machine (TX machine) separately. Each is completely independent. Transmission and reception can occur simultaneously, making the asynchronous modes full-duplex.

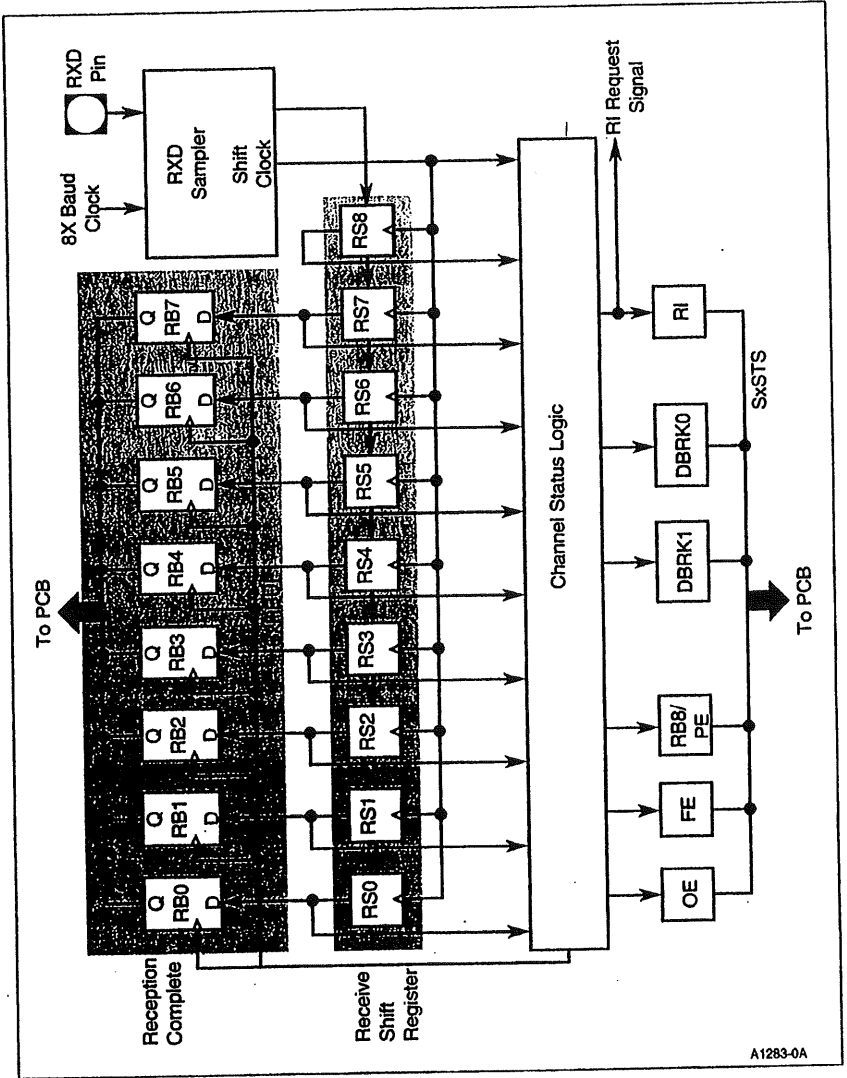
#### 10.1.1.1 RX Machine

The RX machine (Figure 10-2) shifts the received serial data into the receive shift register. When the reception has completed, the data is then moved into the Serial Receive Buffer (SxRBUF) Register. From there, the user can read the received data byte.

The RX machine samples the RXD pin, looking for a logical low (start bit) signifying the beginning of a reception. Once the logical low has been detected, the RX machine begins the receive process. Each expected bit-time is divided into eight samples by the 8X baud clock. The RX machine takes the three middle samples and, based on a two-out-of-three majority, determines the data bit value. This *oversampling* is common for asynchronous serial ports and improves noise immunity. This majority value is then shifted into the receive shift register.

Using this method, the RX machine can tolerate incoming baud rates that differ from its own internal baud rates by 2.5% overspeed and 5.5% underspeed. These limits exceed the CCITT extended signaling rate specifications.

A stop bit is expected by the RX machine after the proper number of data bits. When the stop bit has been validated, the data from the shift register is copied into SxRBUF and the Receive Interrupt (RI) bit is set. Note that the stop bit is actually validated right after its middle three samples are taken. Therefore, the data is moved into SxRBUF and the RI bit is set approximately in the middle of the stop bit time.



A1283-0A

Figure 10-2. RX Machine

**SERIAL COMMUNICATIONS UNIT**

The RX machine can detect several error conditions that may occur during reception:

1. Parity errors — A parity error flag is set when the parity of the received data is incorrect.
2. Framing errors — If a valid stop bit is not received when expected by the RX machine, a framing error flag is set.
3. Overrun errors — If SxRBUF is not read before another reception completes, the old data in SxRBUF is overwritten and an overrun error flag is set. This indicates that data from an earlier reception has been lost.

The RX machine also recognizes two different break characters. The shorter break character is  $M$  bit times, where  $M$  is equal to the total number of bits (start + data + stop) in a frame. The longer break character is  $2M + 3$  bit times. A break character results in at least one null (all zero) character with a framing error being received. Other error flags could be set depending on the length of the break character and the mode of the serial port.

**10.1.1.2 TX Machine**

A block diagram of the TX machine is shown in Figure 10-3. The TX machine logic supports the following features:

- parity generation (even, odd or none)
- Clear-to-Send
- break character transmission
- double-buffered operation

A transmission begins by writing a byte to the Serial Transmit Buffer (SxTBUF) Register. SxTBUF is a holding register for the transmit shift register. The contents of SxTBUF are transferred to the transmit shift register as soon as it is empty. If no transmission is in progress (i.e., the transmit shift register is empty), SxTBUF is copied immediately to the transmit shift register. If parity is enabled, the parity bits are calculated and appended to the transmit shift register during the transfer. The start and stop bits are added when the data is transmitted. The Transmit Interrupt bit (TI) is set at the beginning of the stop bit time.

Double buffering is a useful feature of the TX machine. When the transmit shift register is empty, the user can write two sequential bytes to SxTBUF. The first byte is transmitted immediately and the second byte is held in SxTBUF until the first byte has been transmitted.

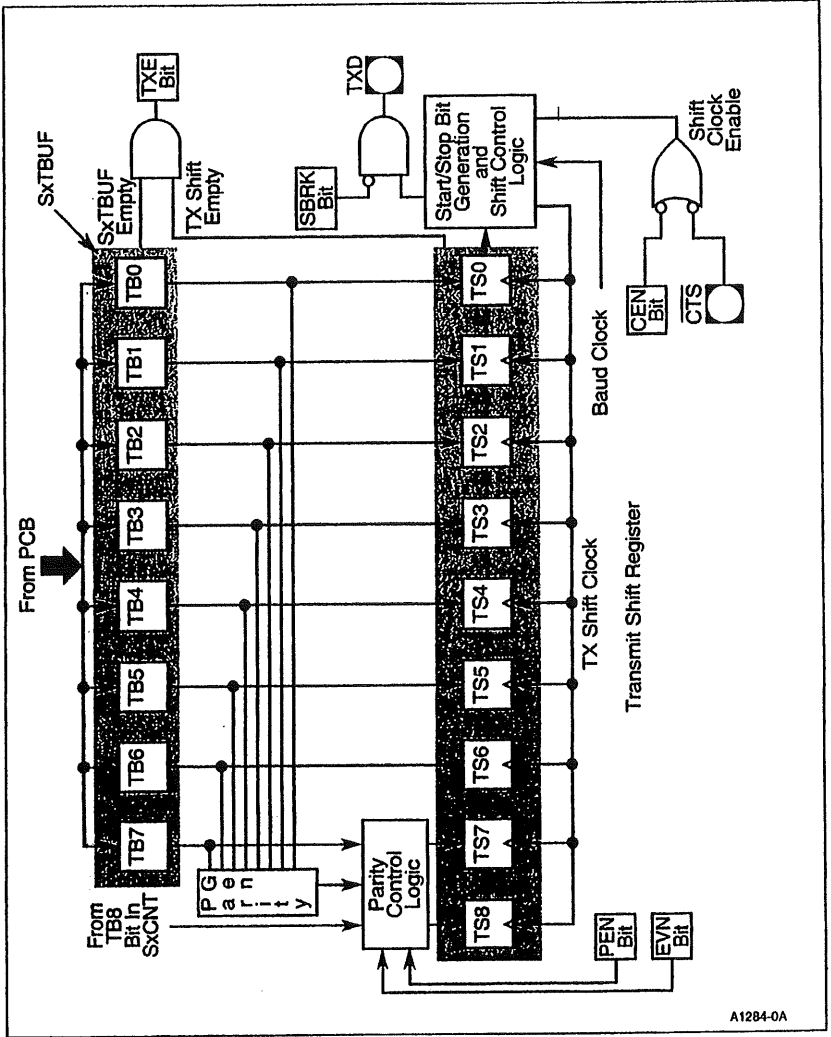


Figure 10-3. TX Machine

The Transmit machine can be disabled by an external source by using the Clear-to-Send feature. When the Clear-to-Send feature is enabled, the TX machine will not transmit until the CTS pin is asserted. The CTS pin is level sensitive. Asserting the CTS pin before a pending transmission for at least 1½ clock cycles ensures that the entire frame will be transmitted. See “CTS Pin Timings” on page 10-18 for details.

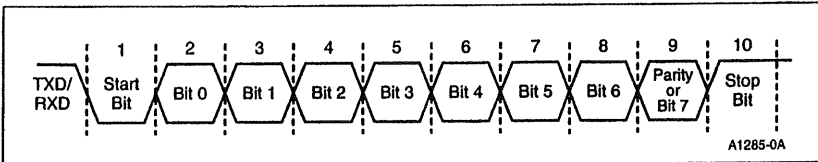
The TX machine can also transmit a break character. Setting the SBRK bit forces the TXD pin immediately low. The TXD pin remains low until the user clears SBRK. The TX machine will continue the transmission sequence even if SBRK is set. Use caution when setting SBRK or characters will be lost.

**10.1.1.3 Modes 1, 3 and 4**

The three asynchronous modes of the serial ports, Modes 1, 3 and 4, operate in approximately the same manner. Mode 1 is the 8-bit asynchronous communications mode. Each frame consists of a start bit, eight data bits and a stop bit, as shown in Figure 10-4. When parity is used, the eighth data bit becomes the parity bit. Both the RX and TX machines use this frame in Mode 1 with no exceptions.

Mode 3 is the 9-bit asynchronous communications mode (see Figure 10-5). Mode 3 is the same as Mode 1 except that a frame contains nine data bits. The ninth data bit becomes the parity bit when the parity feature is enabled. When parity is disabled, the ninth data bit is controlled by the user. (See “Modes 2 and 3 for Multiprocessor Communications” on page 10-14.) Mode 3 can be used with Mode 2 for multiprocessor communications or alone for “8 data bits + parity” frames.

Mode 4 is the 7-bit asynchronous communications mode. Each frame consists of a start bit, seven data bits and a stop bit, as shown in Figure 10-6. Parity is not available in Mode 4. Both the RX and TX machines use this frame in Mode 4 with no exceptions.



**Figure 10-4. Mode 1 Waveform**



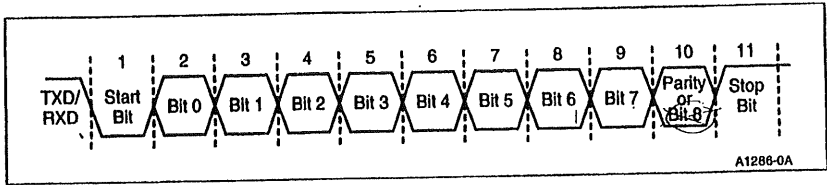


Figure 10-5. Mode 3 Waveform

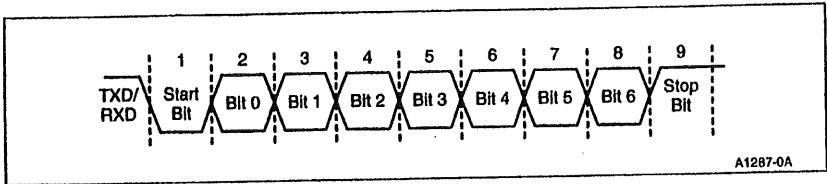


Figure 10-6. Mode 4 Waveform

10.1.1.4 Mode 2

Asynchronous Mode 2 is referred to as the “address recognition mode.” Mode 2 is used together with Mode 3 for multiprocessor communications over a common serial link.

In Mode 2, the RX machine will not complete a reception unless the ninth data bit is a one. Any character received with the ninth bit equal to zero is ignored. No flags are set, no interrupts occur and no data is transferred to SxRBUF. In Mode 3, characters are received regardless of the state of the ninth data bit. The following is brief example of using Modes 2 and 3. See “Master/Slave Example” on page 10-27 for more information.

Assume one master serial port connects to multiple slave serial ports over a serial link. The slaves are initially in Mode 2, and the master is always in Mode 3. The master communicates with one slave at a time. The CPU overhead of the serial communications burdens only the master and the target slave device.

1. The master transmits the “address” of the target slave, with the ninth bit set, over the serial link.
2. All slaves receive the character and check whether that address is theirs.
3. The target slave switches to Mode 3; all other slaves remain in Mode 2.
4. The master and the target slave continue the communication with all ninth data bits equal to zero. The other slave devices ignore the activity on the serial link.



10.2 PROGRAMMING

This section describes how to program the serial port using the appropriate registers. The Serial Receive Buffer Register (SxRBUF) is shown in Figure 10-8 and the Serial Transmit Buffer Register (SxTBUF) is shown in Figure 10-9. These registers have the same functions in any serial port mode.

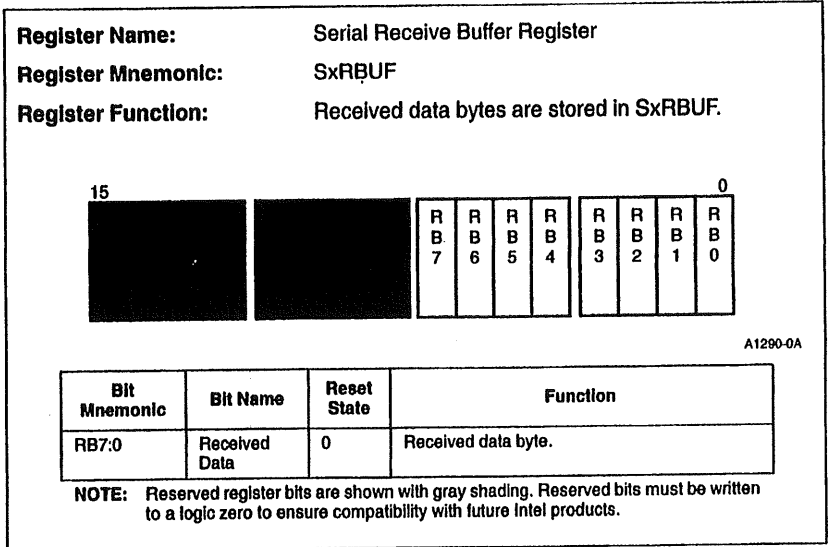
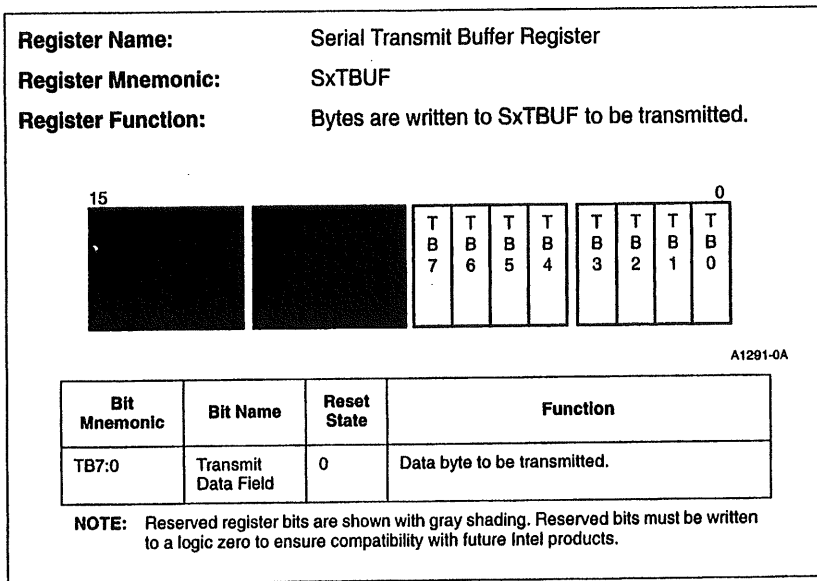


Figure 10-8. Serial Receive Buffer Register (SxRBUF)



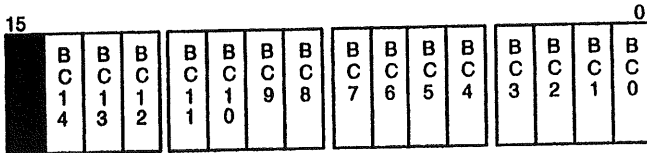
**Figure 10-9. Serial Transmit Buffer Register (SxTBUF)**

### 10.2.1 Baud Rates

The baud rate generator is composed of a 15-bit counter register (BxCNT) and a 15-bit compare register (BxCMP). BxCNT (Figure 10-10) is a free-running counter that is incremented by the baud timebase clock. The baud timebase clock can be either the internal CPU clock or an external clock applied to the BCLK pin. BxCMP (Figure 10-11) is programmed by the user to determine the baud rate. The most-significant bit of BxCMP (ICLK) selects which source is used as the baud timebase clock.

BxCNT is incremented by the baud timebase clock and compared to BxCMP. When BxCNT and BxCMP are equal, the baud rate generator outputs a pulse and resets BxCNT. This pulse train is the actual baud clock used by the RX and TX machines. The baud clock is eight times the baud rate in the asynchronous modes because of the sampling requirements. The baud clock equals the baud rate in the synchronous mode.

**Register Name:** Baud Rate Counter Register  
**Register Mnemonic:** BxCNT  
**Register Function:** 15-bit baud rate counter value.



A1275-0A

Bit Mnemonic	Bit Name	Reset State	Function
BC14:0	Baud rate counter field	0	Reflects current value of the baud rate counter. <b>NOTE:</b> Writing to this register while the serial port is transmitting causes indeterminate operation.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

**Figure 10-10. Baud Rate Counter Register (BxCNT)**

**Register Name:** Baud Rate Compare Register

**Register Mnemonic:** BxCMP

**Register Function:** Determines baud rate for the serial port.

15																	0
	I C L K	B R 1 4	B R 1 3	B R 1 2	B R 1 1	B R 1 0	B R 9	B R 8	B R 7	B R 6	B R 5	B R 4	B R 3	B R 2	B R 1	B R 0	

A1278-0A

Bit Mnemonic	Bit Name	Reset State	Function
ICLK	Internal Clocking	0	Selects the input clock: 0 = BCLK is input to baud clock. 1 = CPU clock is input to baud clock.
BR14:0	Baud Rate Compare Field	0	Sets the compare value for the baud rate clock.

**Figure 10-11. Baud Rate Compare Register (BxCMP)**

The equations in Figure 10-12 show how to calculate the proper BxCMP value for a specific baud rate (where  $F_{CPU}$  = CPU operating frequency =  $\frac{1}{2}$  CLKIN frequency).

	Mode 0	Mode 1-4
If CPU clock is baud timebase clock:	$BxCMP = \frac{CPU}{baudrate} - 1$	$BxCMP = \frac{CPU}{baudrate \times 8} - 1$
If BCLK is baud timebase clock:	$BxCMP = \frac{CPU}{baudrate}$	$BxCMP = \frac{CPU}{baudrate \times 8}$

**Figure 10-12. Calculating the BxCMP Value for a Specific Baud Rate**

Due to internal synchronization requirements, the maximum input frequency to BCLK is one-half the CPU operating frequency. See "BCLK Pin Timings" on page 10-18 for more information. Table 10-1 shows the correct BxCMP values for common baud rates.

**Table 10-1. BxCMP Values for Typical Baud Rates and CPU Frequencies**

Baud Rate	CPU Frequency					
	20 MHz		16 MHz		8 MHz	
	BxCMP Value	% Error	BxCMP Value	% Error	BxCMP Value	% Error
19,200	8082H	0.16	8067H	0.16	8033H	0.16
9,600	8103H	0.16	80CFH	0.16	8067H	0.16
4,800	8208H	-0.03	81A0H	-0.08	80CFH	0.16
2,400	8411H	-0.03	8340H	0.04	81A0H	-0.08
1,200	8822H	0.01	8682H	-0.02	8340H	0.04

**NOTE**

A zero or one value for BxCMP is illegal and results in unpredictable operation. Programming BxCMP during a transmission or reception causes indeterminate operation.

**10.2.2 Asynchronous Mode Programming**

The serial port operation is controlled by two registers. The Serial Port Control (SxCON) Register controls the mode of operation of the serial port (see Figure 10-13). The Serial Port Status (SxSTS) Register acts as the flags register, reporting on errors and the state of the RX and TX machines (see Figure 10-14). Depending on the serial port mode, these registers can have different functionality. This section outlines how to use SxCON and SxSTS to obtain the desired operation from the serial port.

**10.2.2.1 Modes 1, 3 and 4 for Stand-alone Serial Communications**

When using these modes for their respective seven, eight or nine bit data modes, operation is fairly straightforward. The serial port must be initialized correctly (through SxCON), then SxSTS needs to be interpreted.

To configure the serial port, first program the baud rate through the BxCMP register, then program SxCON (Figure 10-13 on page 10-15) as follows.

1. Determine the values for M2:0 for the desired serial port mode.
2. If parity is used, enable it with the PEN bit. Set the sense of parity (even or odd) with the EVN bit. Note that parity is not available in Mode 4 (seven bit data).

3. If the Clear-to-Send feature is used, set the CEN bit to enable it.
4. If receptions are desired, set the REN bit to enable the RX machine. Note the TX machine need **not** be explicitly enabled.

At this point, you will be able to transmit and receive in the mode specified. Now that the serial port is operating, you must correctly interpret its status. This is done by reading the SxSTS register (Figure 10-14 on page 10-16) and interpreting its contents. Reading SxSTS clears all bits except the CTS and TXE bits. SxSTS must first be saved in memory and then each bit can be interpreted individually.

The RI, TI and TXE bits indicate the condition of the transmit and receive buffers. RI and TI are also used with the Interrupt Control Unit for interrupt-based communications. The OE, FE and PE bits indicate any errors when a character is received. Once an error occurs, the appropriate bit remains set until SxSTS is read. For example, assume a character is received with a parity error (PE set) and a subsequent error-free character is received. If the SxSTS register was not read between the two receptions, the PE bit remains set.

#### 10.2.2.2 Modes 2 and 3 for Multiprocessor Communications

Programming for multiprocessor communications is much the same as the stand-alone operation. The only added complexity is that the ninth data bit must be controlled and interpreted correctly.

The ninth data bit is set for transmissions by setting the TB8 bit in SxCON. TB8 is cleared after *every* transmission. TB8 is not double-buffered. This is usually not a problem, as very few bytes are actually transmitted with TB8 equal to one. When writing TB8, make sure that the other bits in SxCON are written with their appropriate value.

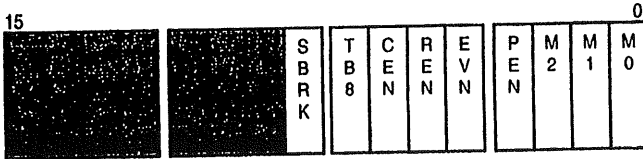
In Modes 2 and 3, the state of the ninth data bit can be determined by the RB8 bit in SxSTS. RB8 reflects the ninth bit for the character currently in SxRBUF. Note that the RB8 bit shares functionality with the PE bit in SxSTS. When parity is enabled, the PE bit has precedence over RB8.

#### 10.2.2.3 Sending and Receiving a Break Character

The serial port can send as well as receive BREAK characters. A BREAK character is a long string of zeros. To send a BREAK character, set the SBRK bit in SxCON. SBRK drives the TXD pin immediately low, regardless of the current serial port mode. The user controls the length of the BREAK character in software by controlling the length of time that SBRK remains set. When writing SBRK, make sure the other bits in SxCON retain their current states.



**Register Name:** Serial Port Control Register  
**Register Mnemonic:** SxCON  
**Register Function:** Controls serial port operating modes.



A1277-0A

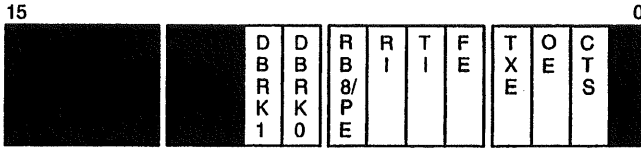
Bit Mnemonic	Bit Name	Reset State	Function																																				
SBRK	Send Break	0	Setting SBRK drives TXD low. TXD remains low until SBRK is cleared.																																				
TB8	Transmitted Bit 8	0	TB8 is the eighth data bit transmitted in modes 2 and 3.																																				
CEN	Clear-to-Send Enable	0	When CEN is set, no transmissions will occur until the CTS pin is asserted.																																				
REN	Receive Enable	0	Set to enable the receive machine.																																				
EVN	Even Parity Select	0	When parity is enabled, EVN selects between even and odd parity. Set for even, clear for odd parity.																																				
PEN	Parity Enable	0	Setting PEN enables the parity generation/checking for all transmissions/receptions.																																				
M2:0	Serial Port Mode Field	0	Operating mode for the serial port channel. <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 5%;">M2</th> <th style="width: 5%;">M1</th> <th style="width: 5%;">M0</th> <th style="width: 85%;">Mode</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>Synchronous Mode0 <span style="float: right;">x</span></td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>10-Bit Asynch Mode1 <span style="float: right;">←</span></td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>11-Bit Asynch Mode2 <span style="float: right;">x</span></td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>11-Bit Asynch Mode3 <span style="float: right;">x</span></td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>9-Bit Asynch Mode4 <span style="float: right;">←</span></td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>Reserved</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>Reserved</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>Reserved</td> </tr> </tbody> </table>	M2	M1	M0	Mode	0	0	0	Synchronous Mode0 <span style="float: right;">x</span>	0	0	1	10-Bit Asynch Mode1 <span style="float: right;">←</span>	0	1	0	11-Bit Asynch Mode2 <span style="float: right;">x</span>	0	1	1	11-Bit Asynch Mode3 <span style="float: right;">x</span>	1	0	0	9-Bit Asynch Mode4 <span style="float: right;">←</span>	1	0	1	Reserved	1	1	0	Reserved	1	1	1	Reserved
M2	M1	M0	Mode																																				
0	0	0	Synchronous Mode0 <span style="float: right;">x</span>																																				
0	0	1	10-Bit Asynch Mode1 <span style="float: right;">←</span>																																				
0	1	0	11-Bit Asynch Mode2 <span style="float: right;">x</span>																																				
0	1	1	11-Bit Asynch Mode3 <span style="float: right;">x</span>																																				
1	0	0	9-Bit Asynch Mode4 <span style="float: right;">←</span>																																				
1	0	1	Reserved																																				
1	1	0	Reserved																																				
1	1	1	Reserved																																				

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 10-13. Serial Port Control Register (SxCON)

The serial port receives BREAK characters of two different lengths. If a BREAK character longer than M bit-times is detected, the DBRK0 bit in SxSTS is set. If the BREAK character is longer than 2M+3 bit-times, DBRK1 in SxSTS is set. M is equal to the total number of bits in a frame. For example, M is equal to 11 (decimal) in Mode 3.

**Register Name:** Serial Status Register  
**Register Mnemonic:** SxSTS  
**Register Function:** Indicates the status of the serial port.



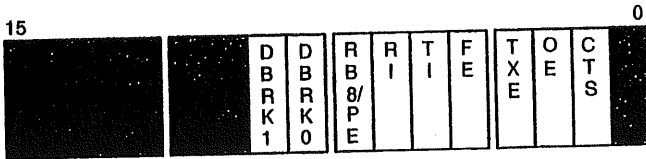
A1278-0A

Bit Mnemonic	Bit Name	Reset State	Function
DBRK1	Detect Break 1	0	Set when a break longer than 2M+3 bits occurs.
DBRK0	Detect Break 0	0	Set when a break longer than M bits occurs.
RB8/PE	Received Bit8/Parity Error	0	Contains the 9th received data bit in modes 2 and 3. PE is set when a parity error occurs. PE is valid only when parity is enabled in Mode 1, 2 or 3.
RI	Receive Interrupt	0	RI is set when a character has been received and placed in SxRBUF. Note that RI need not be explicitly cleared to receive more characters. Writing a one to this bit will not cause an interrupt.
TI	Transmit Interrupt	0	TI is set when a character has finished transmitting. TI determines when one more character can be transmitted. Writing a one to this bit will not cause an interrupt.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 10-14. Serial Port Status Register (SxSTS)

**Register Name:** Serial Status Register  
**Register Mnemonic:** SxSTS  
**Register Function:** Indicates the status of the serial port.



A1278-0A

Bit Mnemonic	Bit Name	Reset State	Function
FE	Framing Error	0	FE is set when a framing error occurs. A framing error occurs when a valid stop bit is not detected.
TXE	Transmitter Empty	1	TXE is set when both SxTBUF and the transmit shift register are empty. TXE determines when two consecutive bytes can be written to SxTBUF for transmission. Accessing SxSTS does not clear TXE.
OE	Overrun Error	0	OE is set when an overrun error occurs. An overrun error occurs when the character in SxRBUF is not read before another complete character is received. SxRBUF always contains the most recent reception.
CTS	Clear To Send	0	CTS is the complement of the value on the CTF pin. Accessing SxSTS does not clear CTS.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

**Figure 10-14. Serial Port Status Register (Continued)**

When either BREAK character is detected, an overrun error occurs (OE is set). SxRBUF will contain at least one null character.

### 10.2.3 Programming in Mode 0

Programming is much easier in Mode 0 than in the asynchronous modes. Configuring SxCON (Figure 10-13 on page 10-15) for Mode 0 requires only two steps:

1. Program M2:0 with the correct combination for Mode 0.
2. If the Clear-to-Send feature is desired, set the CEN bit.

The serial port is now configured for Mode 0. To transmit, write a character to SxTBUF. The TI and TXE bits reflect the status of SxTBUF and the transmit shift register. Note that the SBRK bit is independent of serial port mode functions in Mode 0.

Receptions in Mode 0 are controlled by software. To begin a reception, set the REN bit in SxCON. The RI bit must be zero or the reception will not begin. Data begins shifting in on RXD as soon as REN is set. The asynchronous error flags (OE, FE and PE) and break flags (DBRK0 and DBRK1) are invalid in Mode 0.

## 10.3 HARDWARE CONSIDERATIONS FOR THE SERIAL PORT

There are several interface considerations when using the serial port.

### 10.3.1 $\overline{\text{CTS}}$ Pin Timings

When the Clear-to-Send feature is enabled, transmissions will not begin until the  $\overline{\text{CTS}}$  pin is asserted *while a transmission is pending*. Figure 10-15 shows the recognition of a valid  $\overline{\text{CTS}}$ .

The  $\overline{\text{CTS}}$  pin is sampled by the rising edge of CLKOUT. The CLKOUT high time synchronizes the  $\overline{\text{CTS}}$  signal. On the falling edge of CLKOUT, the synchronized  $\overline{\text{CTS}}$  signal is presented to the serial port.  $\overline{\text{CTS}}$  is an asynchronous signal. The setup and hold times are given only to ensure recognition at a specific clock edge. When  $\overline{\text{CTS}}$  is asynchronously, it should be asserted for at least  $1\frac{1}{2}$  clock cycles to guarantee that the signal is recognized.

$\overline{\text{CTS}}$  is not latched internally. If  $\overline{\text{CTS}}$  is asserted before a transmission starts, the subsequent transmission will not begin. A write to SxTBUF "arms" the  $\overline{\text{CTS}}$  sense circuitry.

### 10.3.2 BCLK Pin Timings

The BCLK pin can be configured as the input to the baud timebase clock. The baud timebase clock increments the BxCNT register. However, the BCLK signal does not run directly into the baud timebase clock. BCLK is first synchronized to the CPU clock (Figure 10-16.) The internal synchronization logic uses a low-to-high level transition on BCLK to generate the baud timebase clock that increments the BxCNT register. The CPU recognizes a low-to-high transition by sampling the BCLK pin low, then high.

The CPU samples BCLK on the rising edge of CLKOUT. The CLKOUT high time synchronizes the BCLK signal. On the falling edge of CLKOUT, the synchronized BCLK signal is presented to the baud timebase clock.

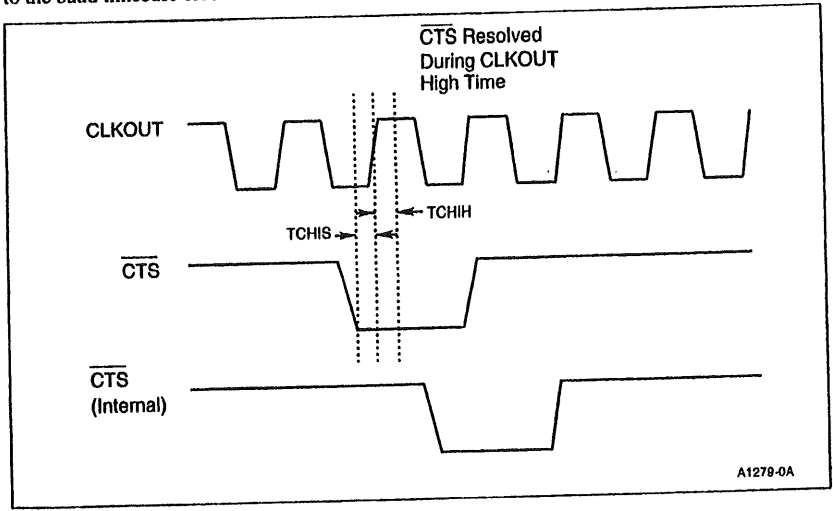


Figure 10-15. CTS Recognition Sequence

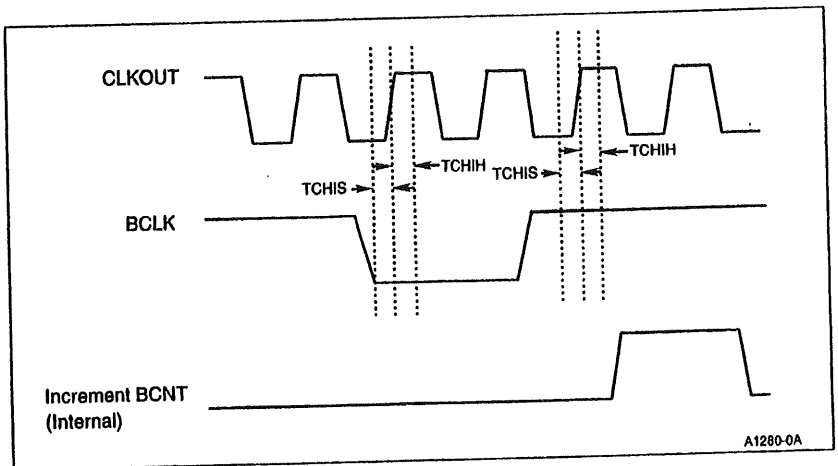


Figure 10-16. BCLK Synchronization

BCLK is an asynchronous input. However, the pin does have setup and hold times, which guarantee recognition at a specific CLKOUT. If the BCLK input signal has high and low times that are both at least  $1\frac{1}{2}$  CLKOUT periods, than synchronization to CLKOUT is not necessary. However, when the BCLK signal has a high or a low time of less than  $1\frac{1}{2}$  CLKOUT periods, meeting the setup and hold times to CLKOUT is necessary to avoid missing BCLK transitions. The maximum input frequency to BCLK is one-half the frequency of CLKOUT (CPU operating frequency).

### 10.3.3 Mode 0 Timings

This section shows the timings of the TXD and RXD pins in Mode 0. In Mode 0, TXD never floats. When not transmitting or receiving, TXD is high. RXD floats except when transmitting a character.

#### 10.3.3.1 CLKOUT as Baud Timebase Clock

The behavior of the transmit/receive clock (on TXD) is governed by the value of BxCMP. When the BxCMP value is greater than or equal to two. The TXD pin is low for two CLKOUT periods and is high for (BxCMP - 1) CLKOUT periods (see Figure 10-17). BxCMP cannot be equal to one, otherwise the serial prot buffer registers (SxRBUF) will not receive the correct data.

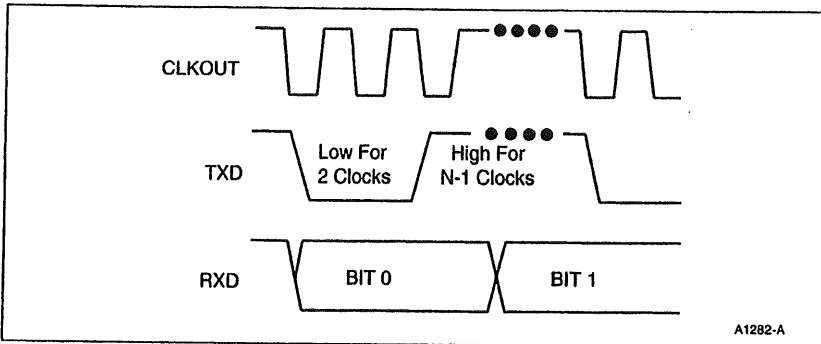


Figure 10-17. Mode 0, BxCMP > 2

For transmissions, the RXD pin changes on the next CLKOUT falling edge following a low-to-high transition on TXD. Therefore, the data on the RXD pin is guaranteed to be valid on the rising edges of TXD. Use the rising edge of TXD to latch the value on RXD. For receptions, the incoming serial data must meet the setup and hold timings with respect to the rising edge of TXD. These timings can be found in the AC timings section of the data sheet.

### 10.3.3.2 BCLK as Baud Timebase Clock

BCLK does not run directly into the baud timebase clock, but is first synchronized to the CPU clock. BCLK causes the baud timebase clock to increment, but transitions on TXD and RXD (for transmissions) still occur relative to CLKOUT.

A low-to-high transition on BCLK increments BxCNT. If BxCNT is equal to BxCMP, TXD goes low approximately  $4\frac{1}{2}$  CLKOUTs later. TXD will always remain low for two CLKOUT periods and then go high. TXD will go low again  $4\frac{1}{2}$  CLKOUTs after BxCNT equals BxCMP. Therefore, the output frequency on TXD is roughly equal to the input frequency on BCLK multiplied by BxCMP. There will be some clock jitter, as the output on TXD will always be some multiple of CLKOUTs. This is due to the internal synchronization.

## 10.4 SERIAL COMMUNICATIONS UNIT INTERRUPTS

Serial communication is usually interrupt-driven. An interrupt needs to occur on each reception and on each transmission of a character. The RI and TI flags in the SxSTS register (Figure 10-14 on page 10-16) provide the interrupt mechanism for the serial ports. The two serial ports, or channels, have different interrupt circuitry. Serial channel 0 is directly supported by the integrated Interrupt Control Unit. Serial channel 1 is supported by the SINT1 output.

### 10.4.1 Channel 0 Interrupts

Figure 10-18 illustrates the channel 0 interrupt circuitry. Channel 0 receptions assert an internal signal, `Receive_Interrupt_Request_0`. This signal is routed both to the Interrupt Control Unit and to the S0STS register, where it sets the RI bit. The RI bit has no effect on the internal interrupt request. Writing to RI does not cause an interrupt, and setting it does not prevent interrupts.

Channel 0 transmissions assert an internal signal, `Transmit_Interrupt_Request_0`. Like the `Receive_Interrupt_Request_0` signal, this signal is routed to the Interrupt Control Unit and to the S0STS register. This signal sets the TI bit in S0STS. Like the RI bit, TI has no effect on the internal interrupt request. Writing to TI does not cause an interrupt, and setting it does not prevent interrupts.

### 10.4.2 Channel 1 Interrupts

Figure 10-18 illustrates the channel 1 interrupt circuitry. Channel 1 receptions assert an internal `Receive_Interrupt_Request_1` signal and transmissions assert an internal `Transmit_Interrupt_Request_1` signal. Serial channel 1 is supported by the SINT1 output. Each internal signal is routed to the S1STS register, where it sets the RI or TI bit. The RI and TI bits are ORed into the SINT1 signal, so setting either bit asserts SINT1. Reading S1STS clears the RI and TI bits and deasserts SINT1. (This is the only method available for deasserting SINT1.)

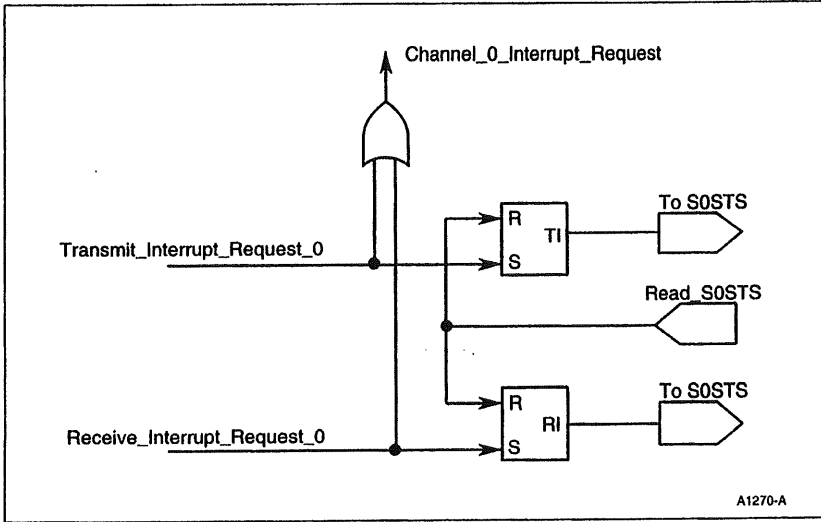


Figure 10-18. Channel 0 Interrupts

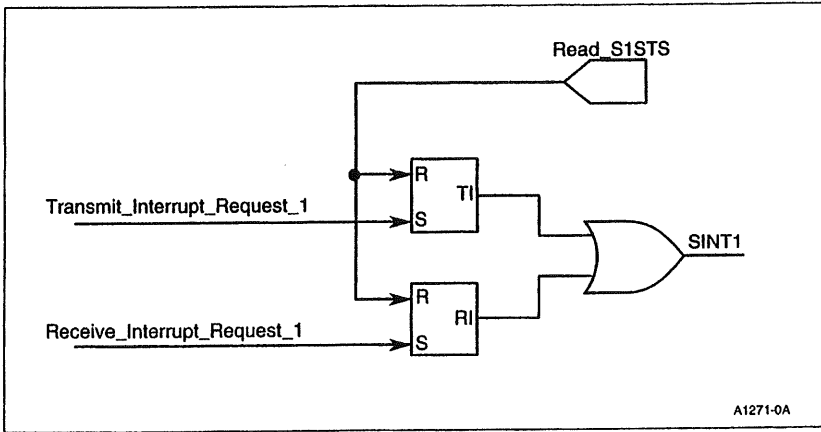


Figure 10-19. Channel 1 Interrupts

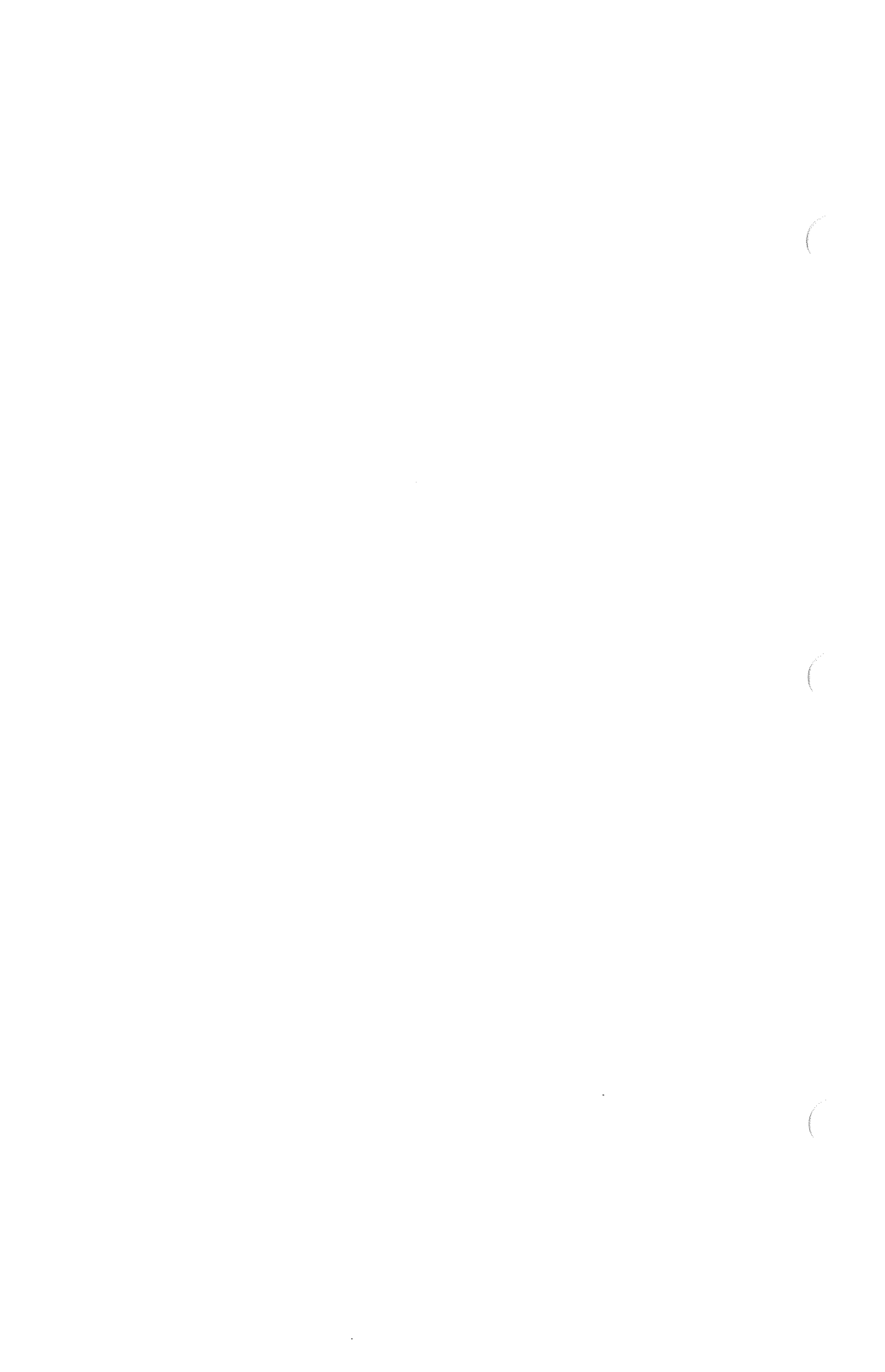


**intel.**<sup>®</sup>

**11**

## **Input/Output Ports**

**I**



## CHAPTER 11 INPUT/OUTPUT PORTS

Many applications do not require full use of all the on-chip peripheral functions. For example, the Chip-Select Unit provides a total of ten chip-select lines; only a large design would require all ten. For smaller designs that require fewer than ten chip-selects, these pins would be wasted.

The input/output ports give system designers the flexibility to replace the functions of unused peripheral pins with general-purpose I/O ports. Many of the on-chip peripheral pin functions are multiplexed with an I/O port. If a particular peripheral pin function is unnecessary in an application, that pin can be used for I/O. The 80C186EB/80C188EB has four types of ports: bidirectional, input-only, output-only, and open-drain bidirectional.

### 11.1 FUNCTIONAL OVERVIEW

All port pin types are derived from a common bidirectional port logic module. Unidirectional and open-drain ports are a subset of the bidirectional module. The following sections describe each port type. The bidirectional port is described in detail, as it is the basis for all of the other port types. The descriptions for the unidirectional and open-drain ports only highlight their specific differences from the common bidirectional module.

#### 11.1.1 Bidirectional Port

Figure 11-1 shows a simplified schematic of a bidirectional port pin. The overall function of a bidirectional port pin is controlled by the state of the Port Control Latch. The output of the Port Control Latch selects the source of output data and the source of the control signal for the three-state output driver. When the port is programmed to act as a peripheral pin, both the data for the pin and the directional control signal for the pin come from the associated integrated peripheral. When a bidirectional port pin is programmed as an I/O port, all port parameters are under software control.

The output of the Port Direction latch enables (or disables) the three-state output driver when the pin is programmed as an I/O port. The three-state output driver is enabled by clearing the Port Direction latch. The data driven on an output port pin is held in the Port Data latch. Setting the Port Direction latch disables the three-state output driver, making the pin an input.

The signal present on the device pin is routed through a synchronizer to a three-state latch that connects to the internal data bus. The state of the pin can be read at any time, regardless of whether the pin is used as an I/O port or for a peripheral function.

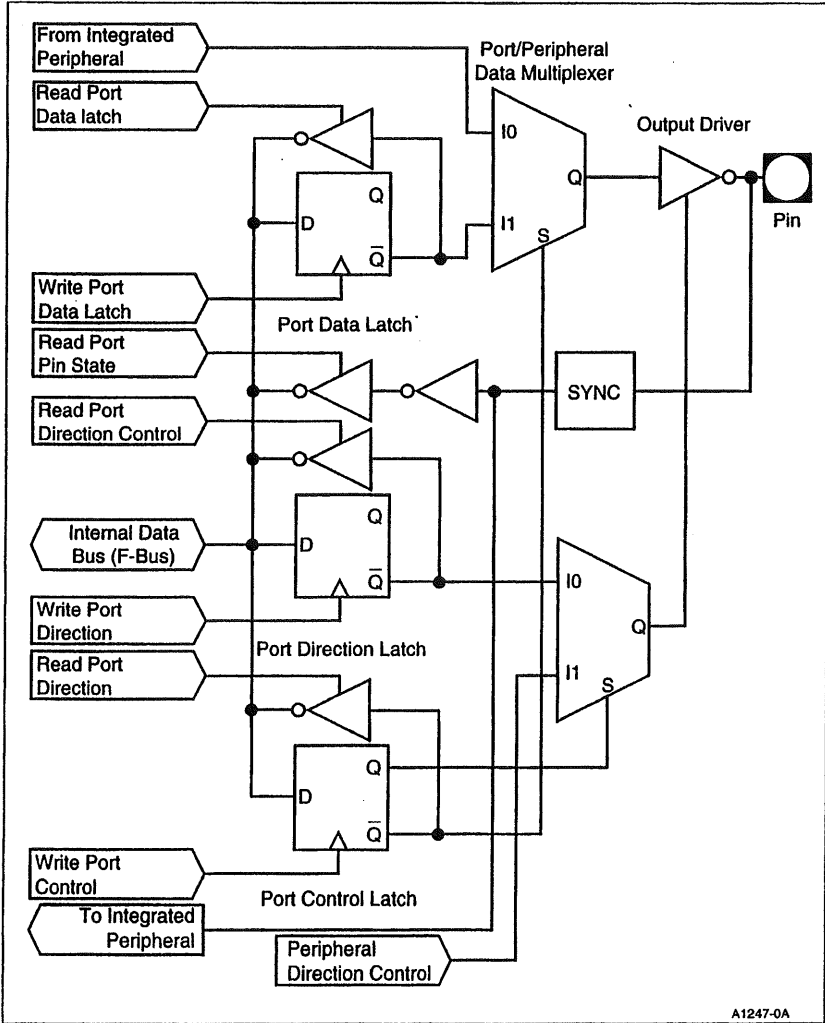


Figure 11-1. Simplified Logic Diagram of a Bidirectional Port Pin

### 11.1.2 Input Port

Figure 11-3 shows the internal construction of an input port pin. An internal connection permanently disables the three-state output driver. The Port Pin register holds the current state (synchronized to the CPU clock) of the input pin. The Port Direction and Port Data bits are not used for an input-only port pin; they can be used for storage.

### 11.1.3 Output Port

Figure 11-3 shows the internal construction of an output port pin. An internal connection permanently enables the three-state output driver. The Port Control latch selects the source of data for the pin, which can be either the on-chip peripheral or the Port Data latch. The Port Direction bit has no effect on an output-only pin; it can be used for storage.

### 11.1.4 Open-Drain Bidirectional Port

Figure 11-4 shows the internal control logic for the open-drain bidirectional port pin. The logic is slightly different from that for the other port types. When the open-drain port pin is configured as an output, clearing the Port Data latch turns on the N-channel driver, resulting in a "hard zero" being present at the pin. A one value in the Port Data Latch shuts off the driver, resulting in a high impedance (input) state at the pin. The open-drain pin can be floated directly by setting its Port Direction bit.

The open-drain ports are not multiplexed with on-board peripherals. The port/peripheral data multiplexer exists for open-drain ports, even though the pins are not shared with peripheral functions. The open-drain port pin floats if the Port Control latch is programmed to select the non-existent peripheral function.

### 11.1.5 Port Pin Organization

The port pins are organized as two functional groups, Port 1 and Port 2. Port 1 consists of eight output-only pins. Port 2 has one bidirectional, two output-only, three input-only, and two open-drain bidirectional pins. Most of the port pins are multiplexed with peripheral functions.

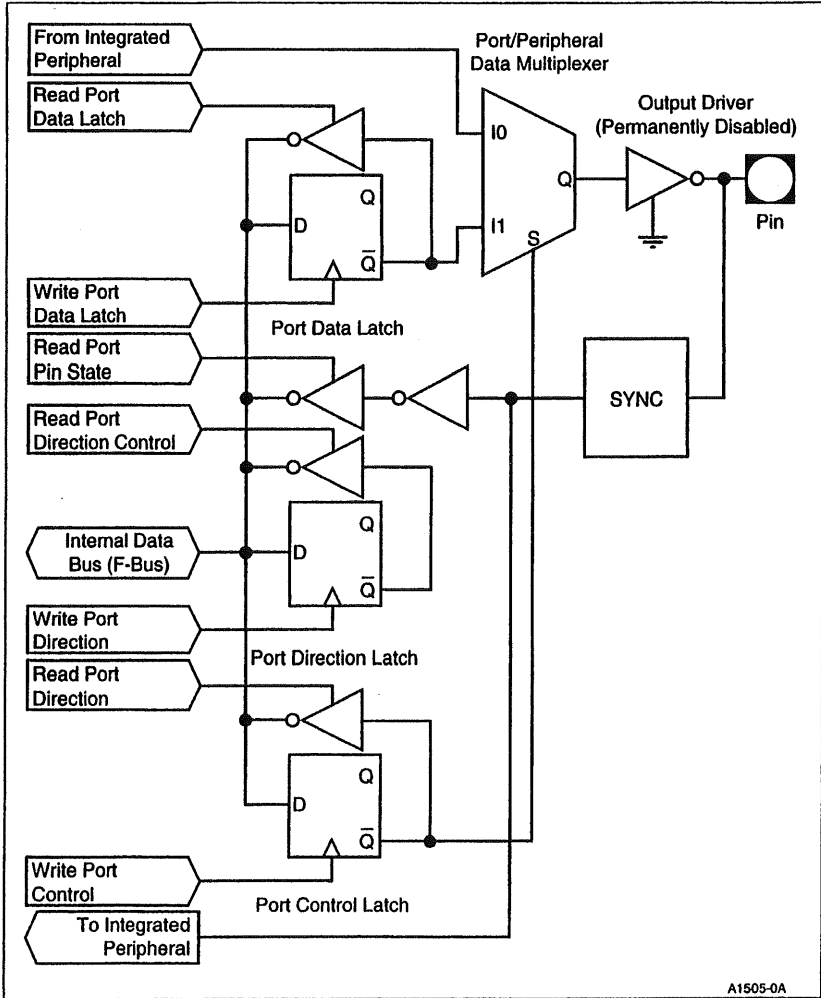


Figure 11-2. Simplified Logic Diagram of an Input Port Pin

A1505-0A







**11.1.5.1 Port 1 Organization**

Port 1 consists of eight **output-only** port pins. The Port 1 pins are multiplexed with the general-purpose chip-selects ( $\overline{\text{GCS7:0}}$ ). Table 11-1 shows the multiplexing options for Port 1.

**Table 11-1. Port 1 Multiplexing Options**

Pin Name	Peripheral Function	Port Function
P1.7/ $\overline{\text{GCS7}}$	GCS7	P1.7
P1.6/ $\overline{\text{GCS6}}$	GCS6	P1.6
P1.5/ $\overline{\text{GCS5}}$	GCS5	P1.5
P1.4/ $\overline{\text{GCS4}}$	GCS4	P1.4
P1.3/ $\overline{\text{GCS3}}$	GCS3	P1.3
P1.2/ $\overline{\text{GCS2}}$	GCS2	P1.2
P1.1/ $\overline{\text{GCS1}}$	GCS1	P1.1
P1.0/ $\overline{\text{GCS0}}$	GCS0	P1.0

**11.1.5.2 Port 2 Organization**

Six of the Port 2 pins are multiplexed with serial channel functions; the other two provide open-drain bidirectional port pin functions. Table 11-2 shows the Port 2 multiplexing options.

**Table 11-2. Port 2 Multiplexing Options**

Pin Name	Peripheral Function	Port Function
P2.7	None	P2.7 (Open-drain)
P2.6	None	P2.6 (Open-drain)
P2.5/BCLK0	BCLK0 (Input)	P2.5
P2.4/CTS1	CTS1 (Input)	P2.4
P2.3/SINT1	SINT1 (Output)	P2.3
P2.2/BCLK1	BCLK1 (Input)	P2.2
P2.1/TXD1	TXD1 (Output)	P2.1
P2.0/RXD1	RXD1 (I/O)	P2.0

**11.2 PROGRAMMING THE I/O PORT UNIT**

Each port is controlled by a set of four Peripheral Control Block registers: the Port Control Register (PxCON), the Port Direction Register (PxDIR), the Port Data Latch Register (PxLTCH) and the Port Pin State Register (PxPIN).

### 11.2.1 Port Control Register

The Port Control Register (Figure 11-5) selects the overall function for each port pin: peripheral or port. For I/O ports, the Port Control Register is used to assign the pin to either the associated on-chip peripheral or to a general-purpose I/O port. For output-only ports, the Port Control Register selects the source of data for the pin: either an on-chip peripheral or the Port Data latch.

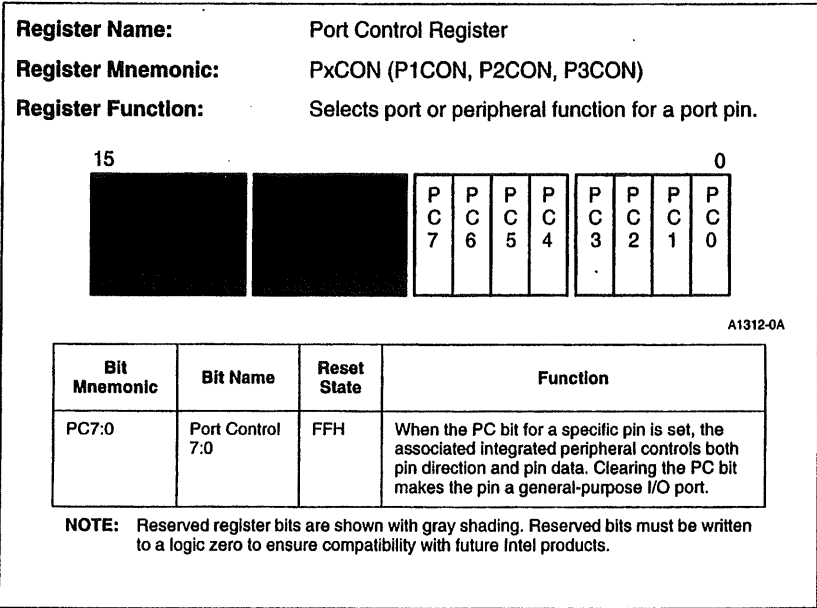


Figure 11-5. Port Control Register (PxCON)

### 11.2.2 Port Direction Register

The Port Direction Register (Figure 11-6) controls the direction (input or output) for each pin programmed as a general-purpose I/O port. The Port Direction bit has no effect on output-only port pins. These unused direction control bits can be used for bit storage.

The Port Direction Register is read/write. When read, the register returns the value written to it previously. Pins with their direction fixed return the value in this register, **not** a value indicating their true direction. The direction of a port pin assigned to a peripheral function is controlled by the peripheral; the Port Direction value is ignored.

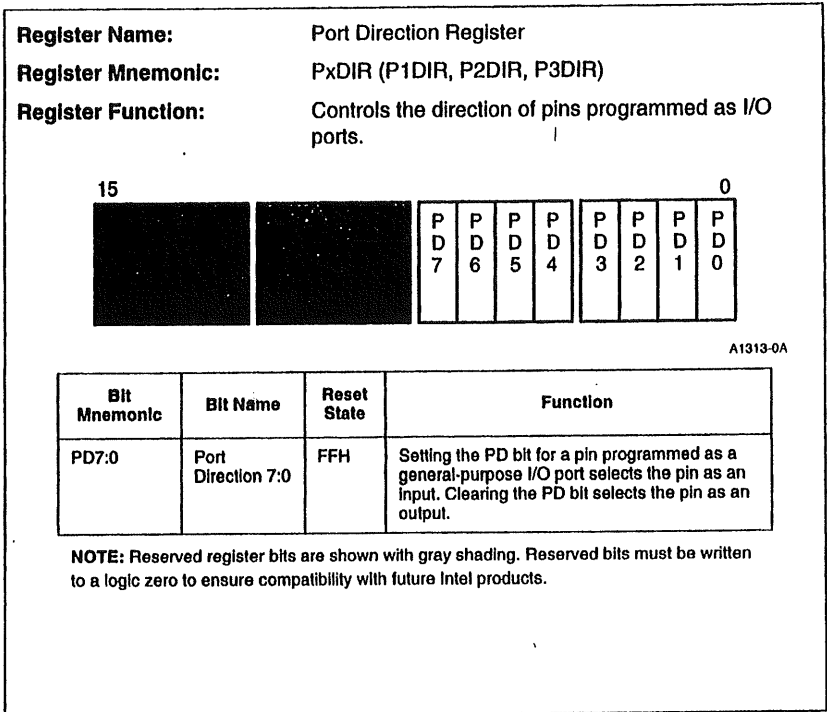


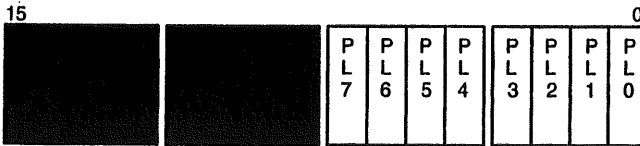
Figure 11-6. Port Direction Register (PxDIR)

### 11.2.3 Port Data Latch Register

The Port Data Latch Register (Figure 11-7) holds the value to be driven on an output or bidirectional pin. This value appears at the pin only if it is programmed as a port.

The Port Data Latch Register is read/write. Reading a Port Data Latch Register returns the value of the latch itself and not that of the associated port pin.

**Register Name:** Port Data Latch Register  
**Register Mnemonic:** P<sub>x</sub>LTCH (P1LTCH, P2LTCH, P3LTCH)  
**Register Function:** Contains the data driven on pins programmed as output ports.



A1314-0A

Bit Mnemonic	Bit Name	Reset State	Function
PL7:0	Port Data Latch 7:0	FFH	The data written to a PL bit appears on pins programmed as general-purpose output ports.

**NOTE:** Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 11-7. Port Data Latch Register (P<sub>x</sub>LTCH)

### 11.2.4 Port Pin State Register

The Port Pin State Register (Figure 11-8) is a read-only register that is used to determine the state of a port pin. When the Port Pin State Register is read, the current state of the port pins is gated to the internal data bus.

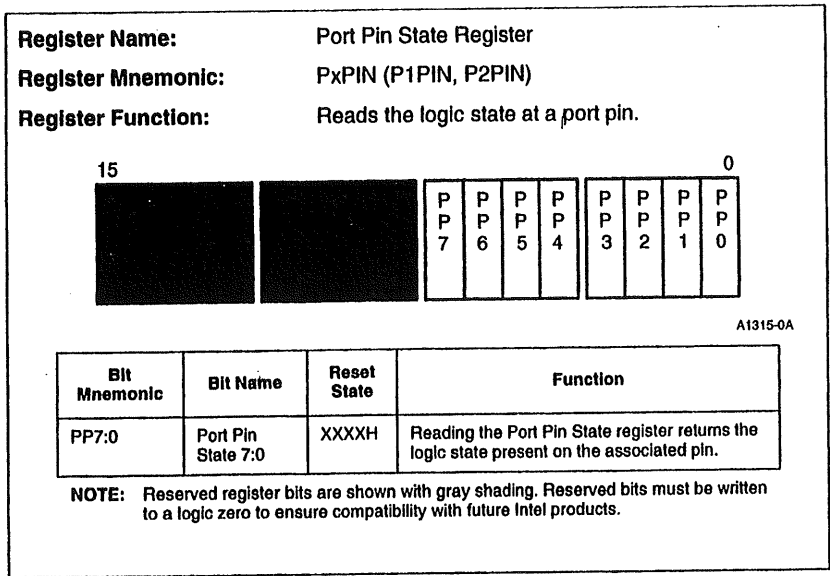


Figure 11-8. Port Pin State Register (PxPIN)

11.2.5 Initializing the I/O Ports

The state of the I/O ports following a reset is as follows:

- Port 1 is configured for peripheral function (general-purpose chip-selects,  $\overline{GCS7:0}$ ).
- Port 2 is configured for peripheral function. The direction of each pin is the default direction for the peripheral function (e.g., P2.1/TXD1 is an output, P2.5/BCLK0 is an input). See Table 11-2 on page 11-7 for details.

There are no set rules for initializing the I/O ports. The Port Data Latch should be programmed before selecting a pin as an output port (to prevent unknown Port Data Latch values from reaching the pins).

## 11.3 PROGRAMMING EXAMPLE

Example 11-1 shows a typical ASM86 routine to configure the I/O ports.  $\overline{GCS7}$  through  $\overline{GCS4}$  are routed to the pins, while P1.0 through P1.4 are used as output ports. The binary value 0101 is written to P1.0 through P1.3. The states of pins P2.6 and P2.7 are read and stored in the AL register.

```

$mod186
name          io_port_unit_example
;
;This file contains sample programming code for the 80C186EB I/O Port Unit.
;
;PCB EQUates in an include file.
#include PCBMAP.inc

code_seg      segment public
               assume cs:code_seg

IO_UNIT_EXAMPL proc near
;write 0101B to data latch for pins P1.3 through P1.0

               mov     dx,P1LATCH
               mov     al,0101b
               out     dx,al

;Gate data latch to output pins. P1.3 to P1.0 are port pins.

               mov     dx,P1CON
               mov     al,0F0h
               out     dx,al

;Read P2.6 and P2.7. We assume they have not been changed to output pins since
;reset.

               mov     dx,P2FIN
               in      al,dx
               and     al,0C0h;           strip unused bits

;AL now holds the states of the P2.6 and P2.7 pins.
IO_UNIT_EXAMPL endp
code_seg      ends
end

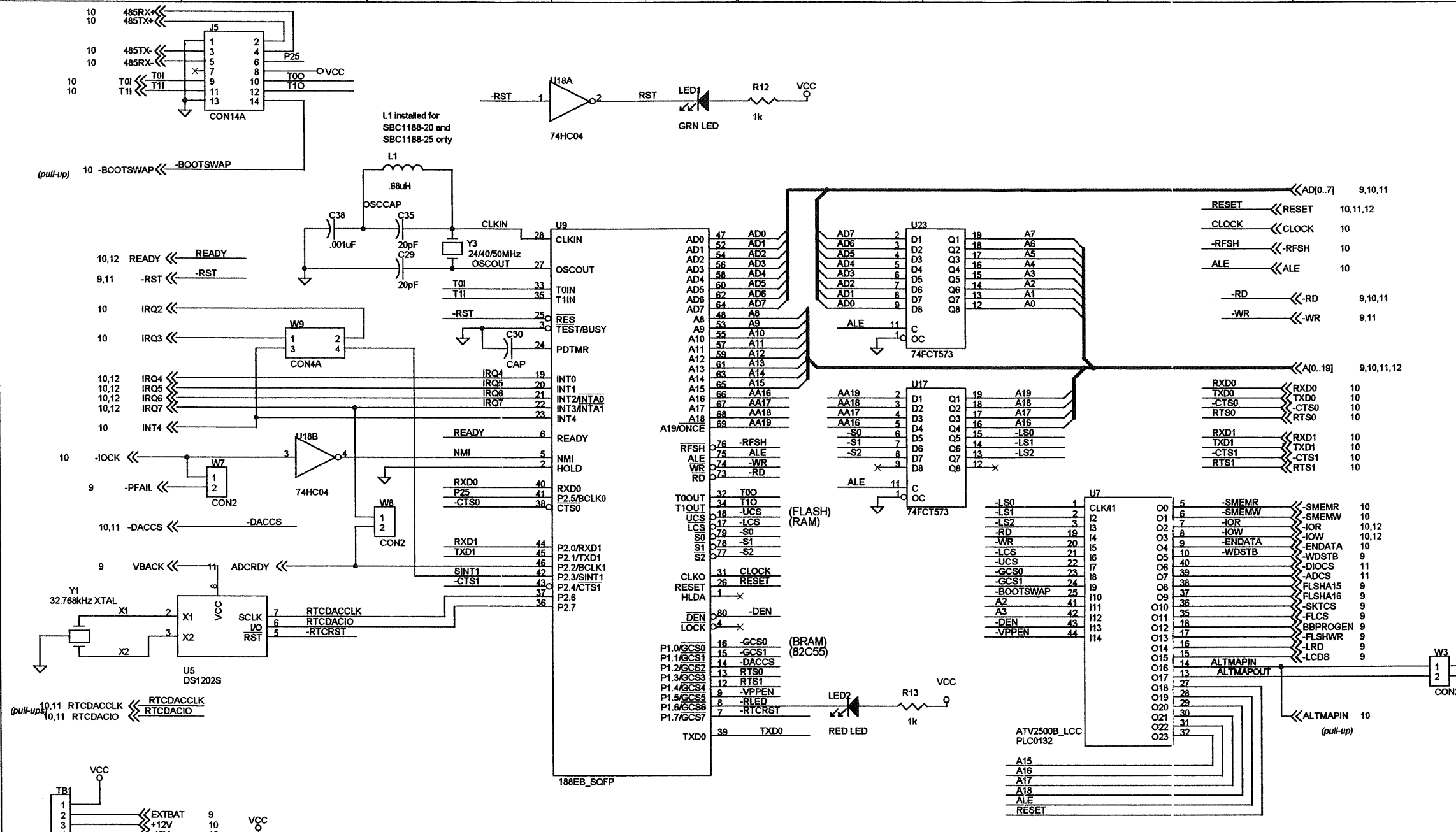
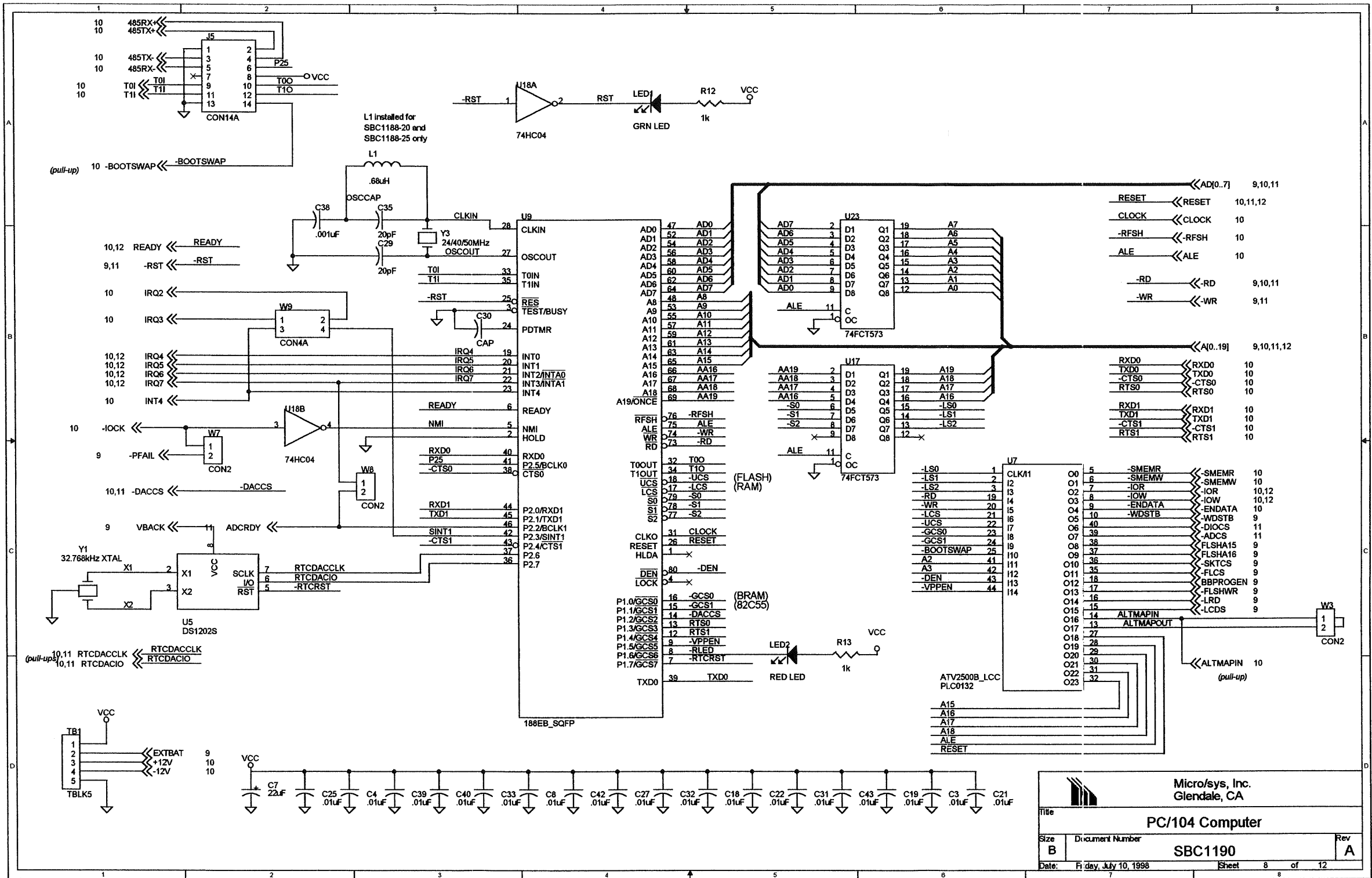
```

**Example 11-1. I/O Port Programming Example**

## **Appendix D - Schematic Drawings**







**Micro/sys, Inc.**  
Glendale, CA

**PC/104 Computer**

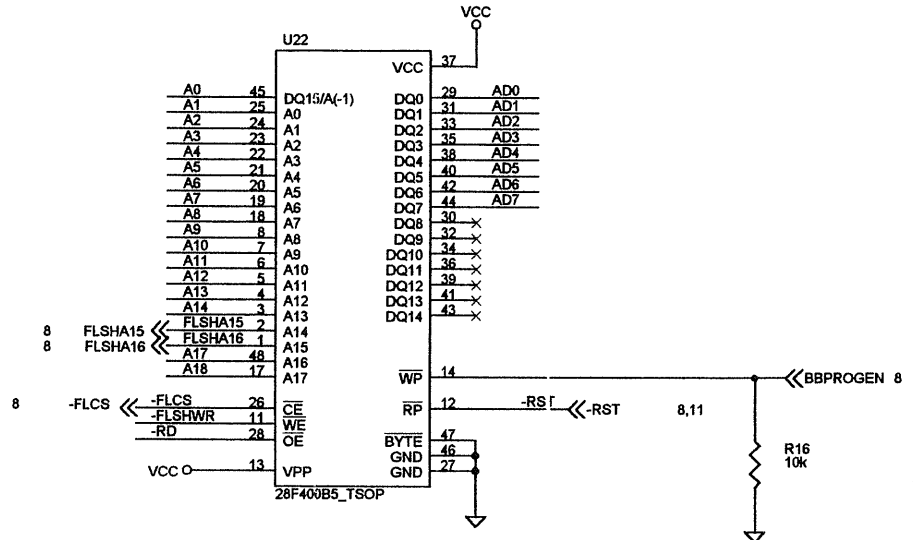
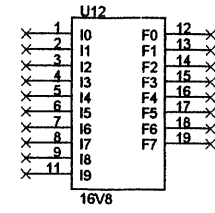
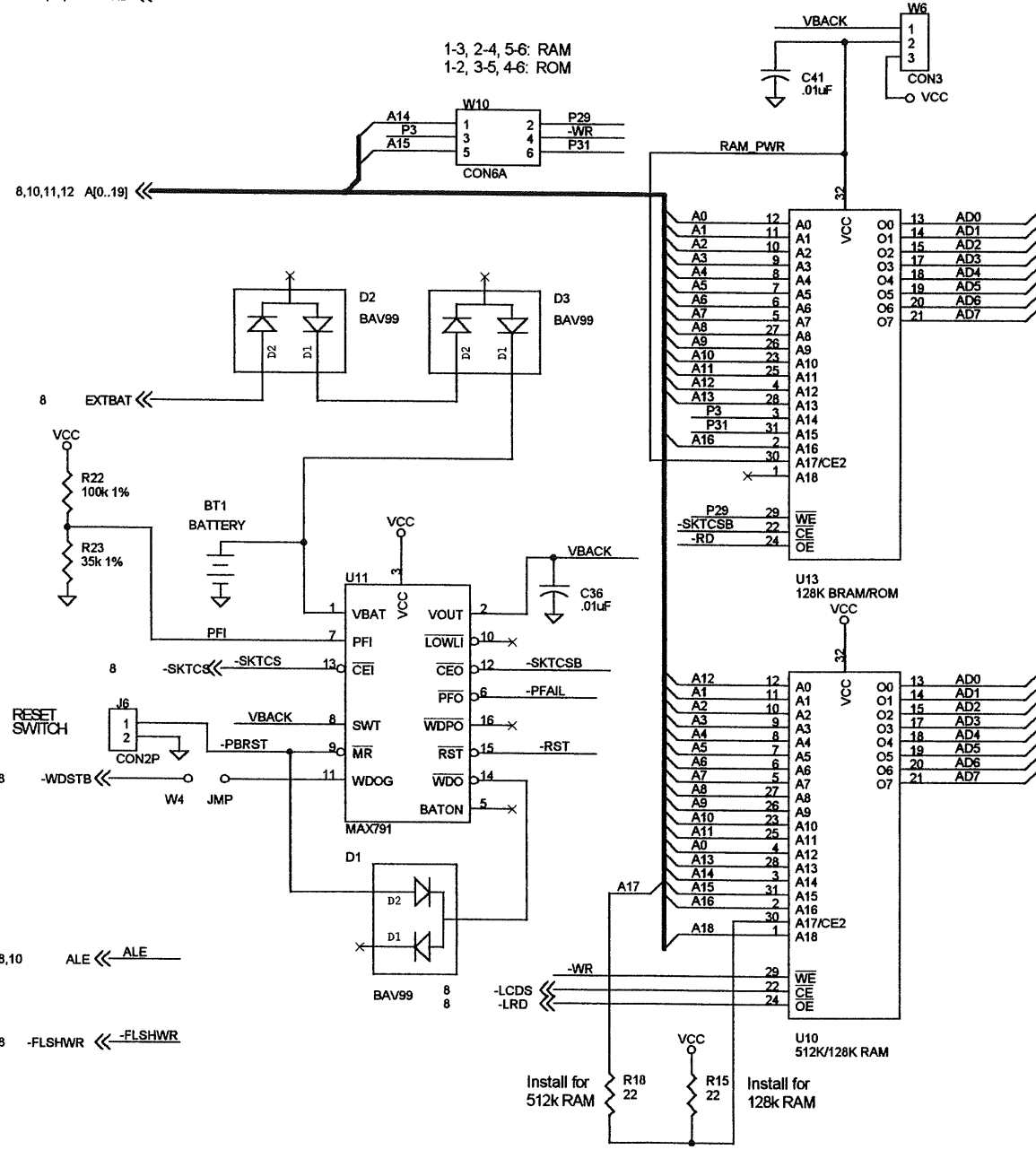
Size: B Document Number: SBC1190 Rev: A

Date: Friday, July 10, 1998 Sheet: 8 of 12



8,10,11 AD[0..7] ←←  
 8,10,11,12 RESET ←← RESET  
 8,11 -WR ←← -WR  
 8,10,11 -RD ←← -RD

1-3, 2-4, 5-6: RAM  
 1-2, 3-5, 4-6: ROM



Install for 512k RAM R18 22

Install for 128k RAM R15 22

VBACK ←← VBACK 8

-PFAIL ←← -PFAIL 8

**Micro/sys, Inc.**  
 Glendale, CA

Title: **80C188EB Computer - Memory**

Size: **B** Document Number: **SBC1190** Rev: **A**

Date: **Friday, July 10, 1998** Sheet: **9** of **12**



8 TOI  
8 T1I  
8,11 -DACCS  
8,11 RTCDACCLK  
8,11 RTCDACIO

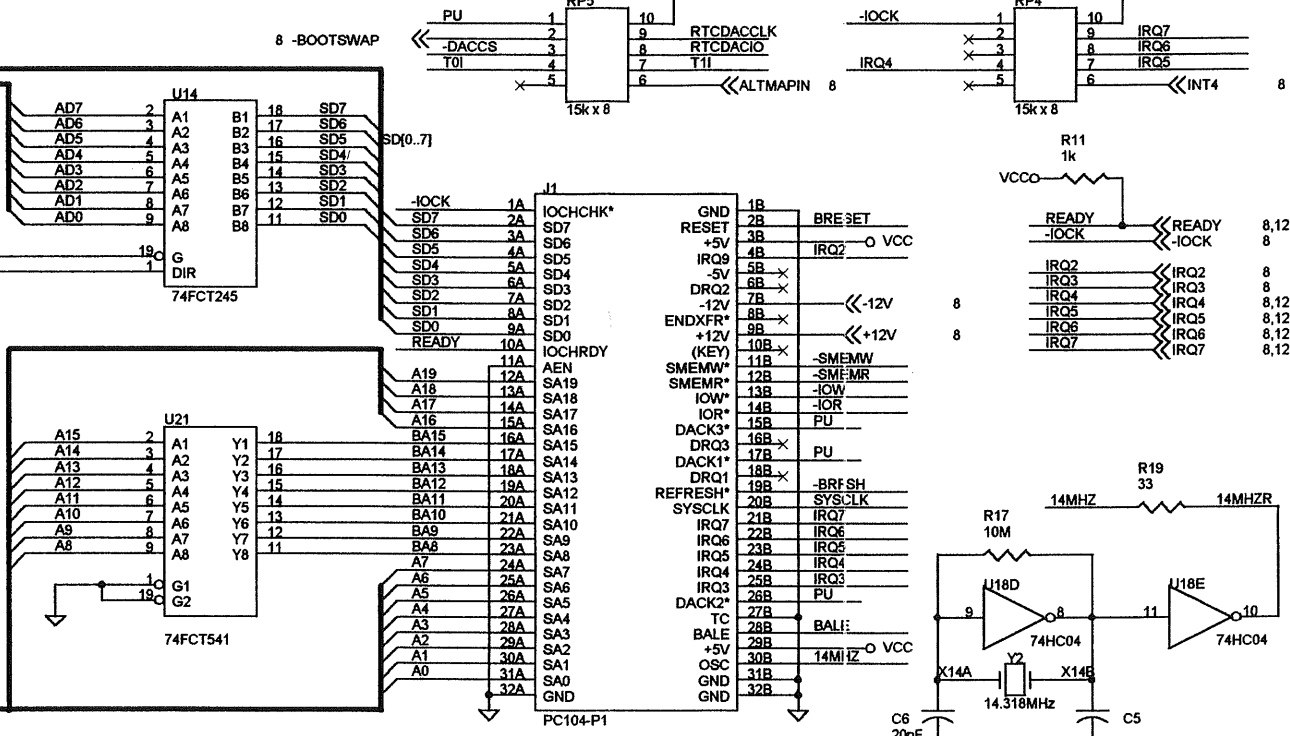
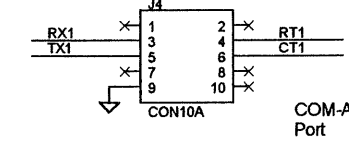
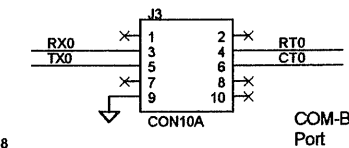
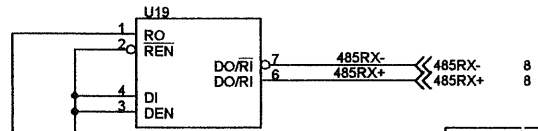
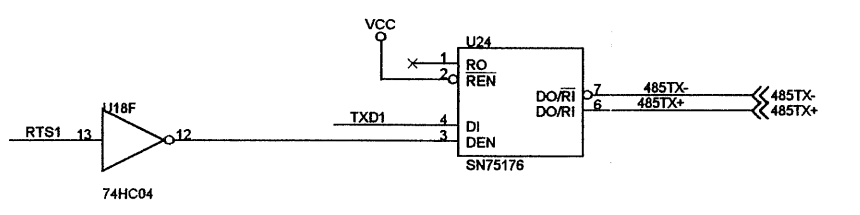
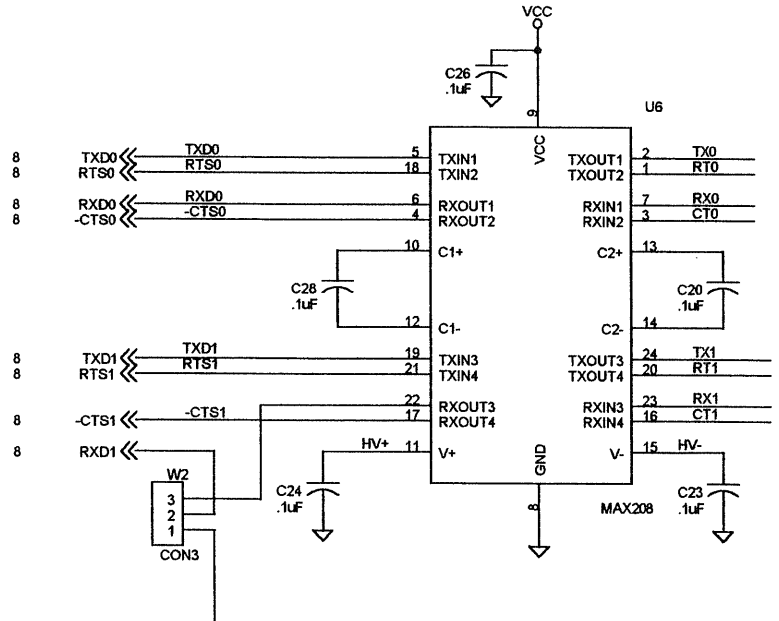
12 SD[0..7]  
8,9,11 AD[0..7]

8 -SMEMR  
8 -SMEMW  
8,12 -IOR  
8,12 -IOW

8 -ENDATA  
8,9,11 -RD

8 ALE  
8 CLOCK  
8 -RFSH  
8,11,12 RESET

8,9,11,12 A[0..19]

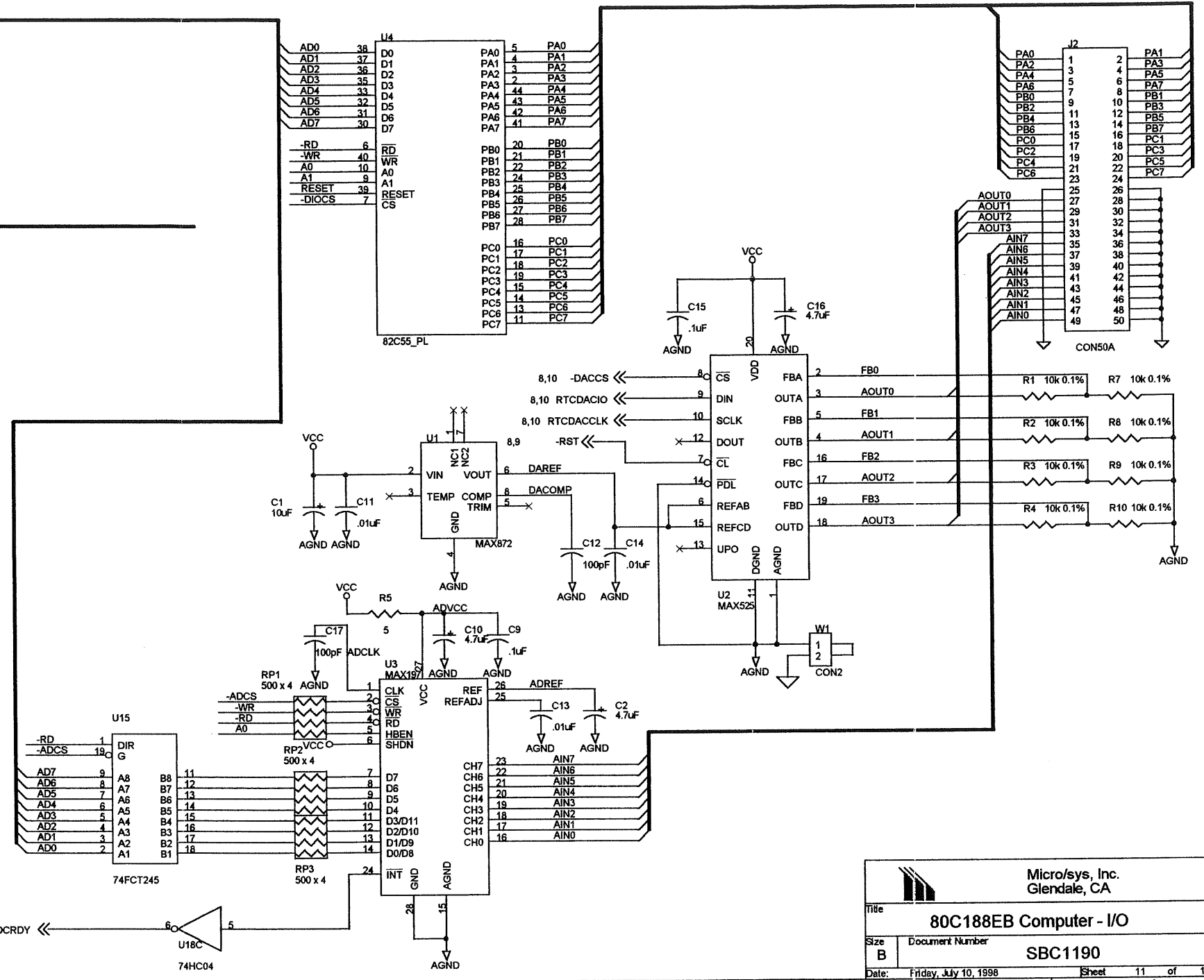
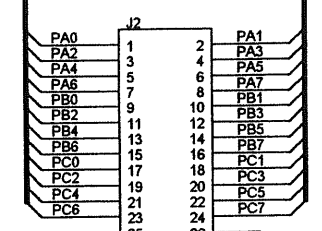
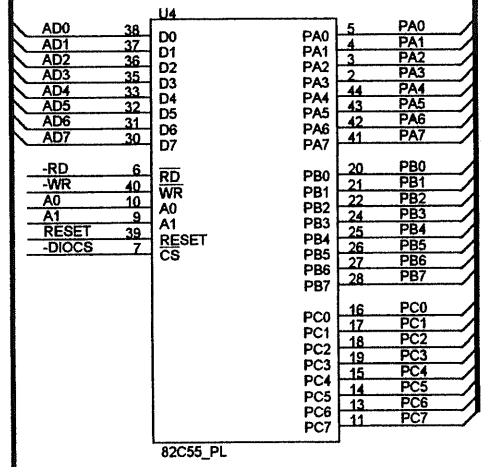




8,9,10 AD[0..7] <<  
 8,10,12 RESET << RESET  
 8,9 -WR << -WR  
 8,9,10 -RD << -RD

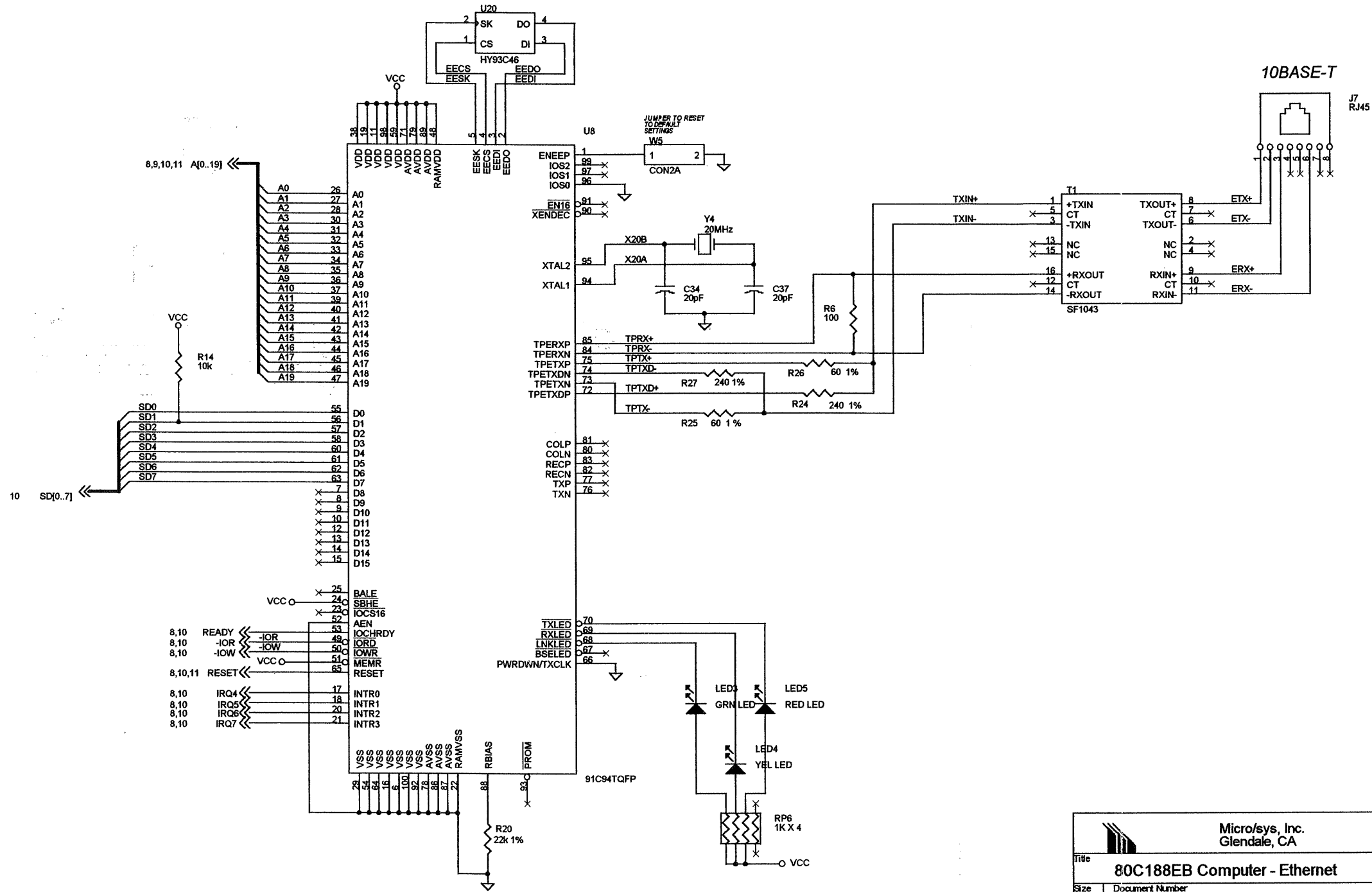
8,9,10,12 A[0..19] <<


8 -DIOCS << -DIOCS  
 8 -ADCS << -ADCS









 <b>Micro/sys, Inc.</b> Glendale, CA	
<b>Title</b> 80C188EB Computer - Ethernet	
<b>Size</b> B	<b>Document Number</b> SBC1190
<b>Date:</b> Friday, July 10, 1998	<b>Rev</b> A
<b>Sheet</b> 12 <b>of</b> 12	

