Software Development Board User' Guide

SPVU002A

TMS34010 Software Development Board User's Guide

Graphics Products

TEXAS INSTRUMENTS

TMS34010 Software Development Board User's Guide

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.

In the absence of written agreement to the contrary, TI assumes no liability for TI applications assistance, customer's product design, or infringement of patents or copyrights of third parties by or arising from use of semiconductor devices described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor devices might be or are used.

WARNING

This equipment is intended for use in a *laboratory* test environment *only*. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits for computing devices persuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications. In which case the user at this own expense will be required to take whatever measures may be required to correct the interference.

Contents

Section *Page*

Illustrations

Figure *Page*

Tables

Table *Page*

1. Introduction

The Software Development Board (SDB) is a high-performance graphics card that facilitates understanding TI graphics products. These include:

- TMS34010 Graphics System Processor
- TMS34070 Video Palette
- TMS4161 Multiport Video RAM

Accompanying this manual and board is a floppy disk containing demonstration, debugging, and diagnostic programs. Installation of the board and the demo software is covered in Section 2 and Section 3.

Notes:

1. It is presumed that persons using the SDB are schooled in the assembly language of the TMS34010 Graphics System Processor. The only way the tutorial program supplied with this board can be an effective learning tool is for the user to understand the instruction set. This set is explained in the Assembler Kit and in the TMS34010 User's Guide (SPVU001).
2. This introduction section covers general information applicable to the SDB. If you wish to proceed directly to SDB installation and demonstration, see Section 2 and Section 3, and cover this introductory information later.

1.1 SDB Features

- Board and demo software factory configured to support a 640 by 480 pixel (horizontal by vertical) resolution (similar to IBM Professional Graphics Display).
- IBM PC card format.
- Maximum resolution of 1024 by 512 pixels with 4 bits per pixel. Display resolution can be altered by changing crystal oscillator and reprogramming timing control registers.
- 256K-byte frame buffer holds display (1024 x 512 x 4 bits per pixel).
- 512K bytes onboard program RAM.
- Program RAM and frame buffer accessed by host from TMS34010's memory-mapped host port.
- Software single step and breakpoint are two of the more than 60 software development commands.
- Debugger software on floppy disk includes software breakpoints, single step, and run-with-count while machine status is displayed on host monitor.
- Reverse assembler and single-line assembler.
- Demonstration and tutorial software on floppy disk.
- Real-time software environment.
- Direct interface to most digital and analog RGB raster-scan monitors.
- TMS34010 32-bit CMOS Graphics System Processor (TMS34010).
- TMS34070 16-of-4096 Color Palette.
- Onboard USART.

1.2 Functional Overview

Figure 1-1 is a typical system consisting of:

- IBM- or TI-compatible PC,
- Graphics monitor with interconnecting cable,
- SDB (Software Development Board) correctly jumpered and installed in PC,
- Applicable software on disk drive.

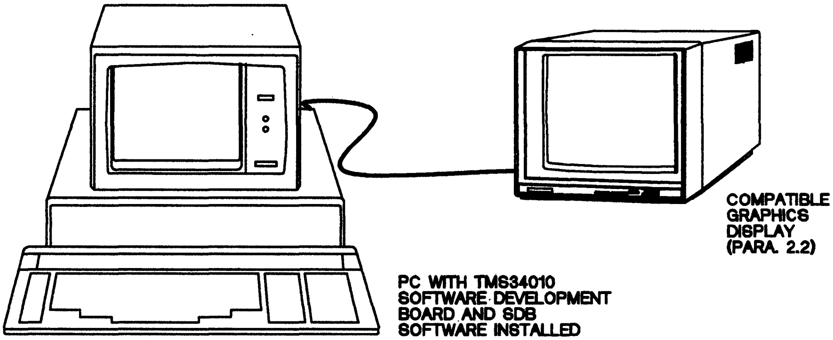


Figure 1-1. Typical Software Development Board System

The graphics SDB is a single card designed around the IBM PC I/O Expansion Bus. The board is a software development tool for programmers writing application software for the TMS34010 Graphics System Processor (TMS34010). This module also demonstrates the simplicity of hardware design using the TMS34010 to develop a high-performance bit-mapped graphics display.

The board comes with interactive debug software on floppy disks. Its features include software breakpoints, software single step, and run with count. At the same time, current machine status is displayed on the top half of the host monitor.

Figure 1-2 is a block diagram of the SDB. The board contains 512K bytes of program RAM for the TMS34010 to execute drawing functions, application programs, and displays. Both program RAM and the frame buffer are accessible to the host by the TMS34010's memory-mapped host port.

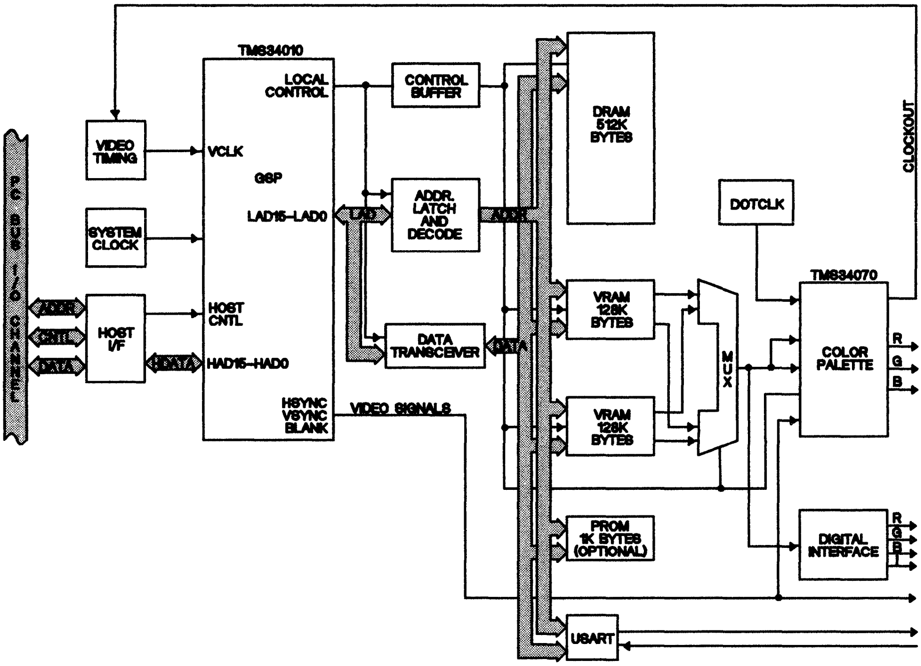


Figure 1-2. SDB Module Block Diagram

The frame buffer consists of eight TMS4161EV4 SIP (single inline package) memory modules organized into four color planes. This allows 16 colors per frame from the digital monitor. The TMS34070 color palette incorporates a 12-bit color lookup table to give the programmer a choice of 16 colors in a frame from a 4096-color palette. Furthermore, the palette incorporates a unique line load feature to allow the color lookup table to be reloaded on every line; meaning 16 of 4096 colors displayed per line.

1.3 Overview of TMS34010 and Development Tools

The TMS34010 Graphics System Processor is a 32-bit microprocessor optimized for graphics systems. It is a member of the TMS340 family of computer graphics products from Texas Instruments.

The TMS34010 is supported by hardware and software development tools, including a C compiler, a full-speed emulator, a software simulator, and an IBM/TI-PC development board. The software development tools that are included with the TMS34010 Assembly Language package include:

- Assembler
- Archiver
- Linker
- Code Conversion Utility
- Simulator[1]

These tools can be installed on the following systems:

- **PCs:**
  - TI-PC with MS-DOS
  - IBM-PC with PC-DOS
- **VAX:**
  - VMS (revision 3.7 and later)
  - DEC Ultrix
  - Unix System V

The TMS34010 assembly language tools create and use object files that are in Common Object File Format, or COFF. COFF object format facilitates modular programming. Object files contain separate blocks (called *sections*) of code and data that can be loaded into different TMS34010 memory spaces.

Figure 1-3 shows the TMS34010 assembly language development flow. The center section of the illustration highlights the most common path; the other portions are optional.

- The **assembler** translates assembly language source files into machine language object files. Source files can contain instructions (discussed in the *TMS34010 User's Guide*), assembler directives, and macro directives. Assembler directives control aspects of the assembly process such as data alignment, placement of source code into sections, and source listings.

[1] The simulator is available in a PC version only.

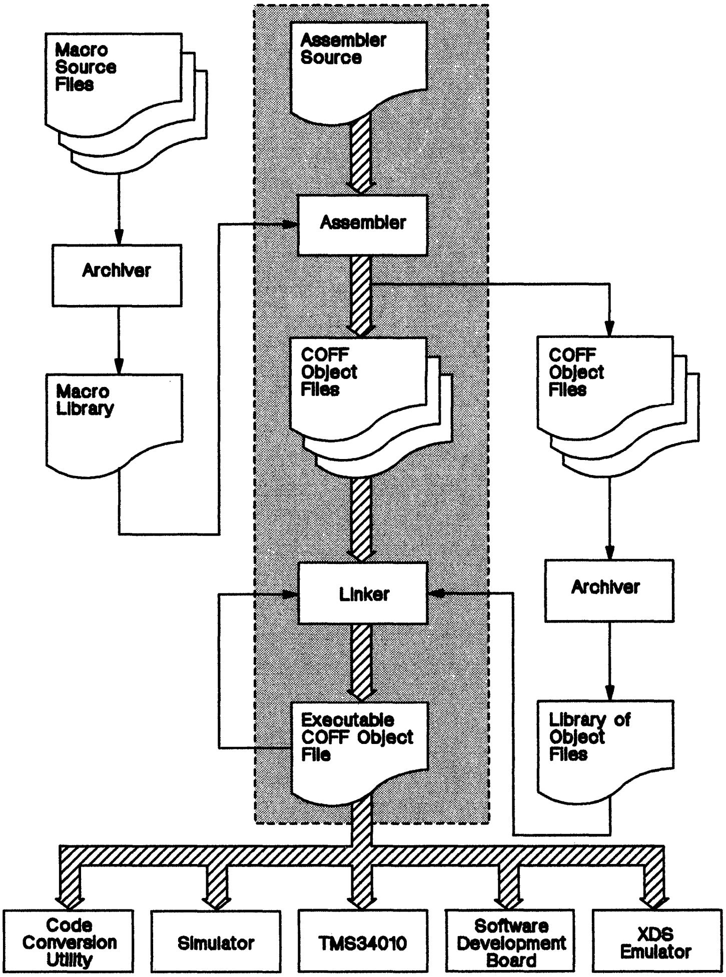


Figure 1-3. Software Development Flow

- The **archiver** allows collecting a group of files into a single archive library (e.g., several macros collected into a macro library). The assembler searches through the library and uses the members called "macros" by the source file. Archivers can also be used to collect object files into an object library. The linker will include the members in the library to resolve external references during the link.

- The **linker** combines object files into a single executable object module, resolving relocation values and external references. As input, the linker accepts relocatable COFF object files (created by the assembler) as well as archived library members and output modules created by a previous linker run. Linker directives also bind sections or symbols to specific addresses or to within specific portions of TMS34010 memory, and define or redefine global symbols.

- The main purpose of this development process is to produce a module that can be executed on the Software Development Board. Other debugging tools available are the Simulator and the XDS/22 Emulator:
  - The **Simulator** is a debugging tool that simulates TMS34010 functions in a configurable graphics environment. The simulator allows you to design, implement, and evaluate both graphics and nongraphics software systems. The simulator command set displays and maintains graphics and machine status information and controls execution of the software system under development. The simulator can execute linked COFF object modules produced by the C compiler, assembler, and linker.
  - The **XDS/22 Emulator** is a realtime, in-circuit emulator.
- Most EPROM programming devices do not accept COFF object files as input. The **code conversion utility** converts a COFF object file into Intel hex or Tektronix hex object format that can be downloaded to an EPROM programmer.

1.4 Manual Organization

Starting with Section 2, this manual is organized as follows:

Section	*Description*
2	Installation of the board including jumper settings and initialization
3	Walkthrough demonstrating some graphics instructions and debugging commands
4	Debugging Command Set in alphabetical order
5	Operation. Includes memory mapping, shadow RAM, Color Palette modes, external interface, interrupts, expansion, specifications
6	Theory of Operation. Includes interfacing between major functional areas and general data flow and functionality of each area
A	Data Sheet for SMC Programmable Communication Interface (PCI) COM-2651
B	Parts List
C	Diagnostics. How to check the SDB should a malfunction be evident. Includes troubleshooting steps.
D	Glossary.

E Hands-On Tutorial. Repeats the first instructions demonstrated in the Tutorial Section (Section 3) but suggests experiments that help explain chip and board functions.

1.5 Applicable Documents

- User's Guides

  TMS34010 User's Guide (SPVU001)
  TMS34010 Software Development Board Schematics (SPVU003)
  TMS34010 Assembler Tools User's Guide (SPDU076)
  TMS34070 Color Palette User's Guide (SPPU016)

- Data Sheets

  TMS34010 Graphics System Processor Data Sheet (SPPS011)
  TMS4161 Video RAM Data Sheet (SMVS003)
  TMS34070-66 Color Palette Data Sheet (SPPS016)
  TM4161EV4 64K x 4 SIP Data Sheet (SMMS614B)
  TM4161EP5 64K x 5 SIP Data Sheet (SMMS615B)

- Technical Papers

  Dual Port Memory with High-Speed Serial Access (technical paper reprint, SMVY001)
  Video Memory Technology & Applications (technical paper reprint, SMVY002)

- Application Reports

  Topological Structure of the TMS4161 Application Report (SMVA003)
  High Performance Memory Access with the TMS4161 Application Report (SMVA005)

- Product Bulletins

  TMS340 Product Bulletin (SPVT001)

  TMS34010 Product Bulletin (SPVT002)

2. Installation

This section describes how to configure the board:

2.1 Items as Shipped

The following items are part of the SDB package:

- Software Development Board (SDB) for TMS34010 Graphics Processor
- Four floppy diskettes (SDB USER INTERFACE)
  - USER INTERFACE, IBM PC (SDB340 Debugger)
  - USER INTERFACE, TI PC (SDB340 Debugger)
  - USER INTERFACE, DEMO & DIAGONOTICS Software
  - USER INTERFACE, LOADER & LIBRARY
- Envelope containing an alternate decode PAL chip for insertion on TI PC systems. Changeout is in socket U3 (shown in Figure 2-3 on page 2-9).
- Software Development Board User's Guide (this book)
- TMS34010 Data Sheet and errata sheet
- Warranty card
- Factory repair authorization and policy

If any item is missing, report this to your distributor.

2.2 Typical System Configuration

The SDB comes configured to be installed as shown in Figure 2-1. Install the board in a PC. The interactive debug display will be on the PC screen. Graphics are shown on an adjacent monitor attached to the DB9 connector on the top (upper) back edge of the SDB. Cabling is with the standard DB9-pin connectors with the monitor. Graphics displays for which the SDB is factory configured include:

- IBM Professional Graphics display
- Princeton Graphics SR-12P
- NEC Multi-Sync monitor (JC-1401P3A)

If your system is an IBM PC with one of the above display monitors, check Table 2-1, Table 2-3, and Table 2-2 for correct factory settings (settings for IBM systems are shown in bold-face type). Then go to Section 2.5 and complete the installation. Otherwise, make the settings as indicated in Table 2-1 through Table 2-2 before installing the board and running the software.

SDB power requirements are listed in Table 5-7 on page 5-21

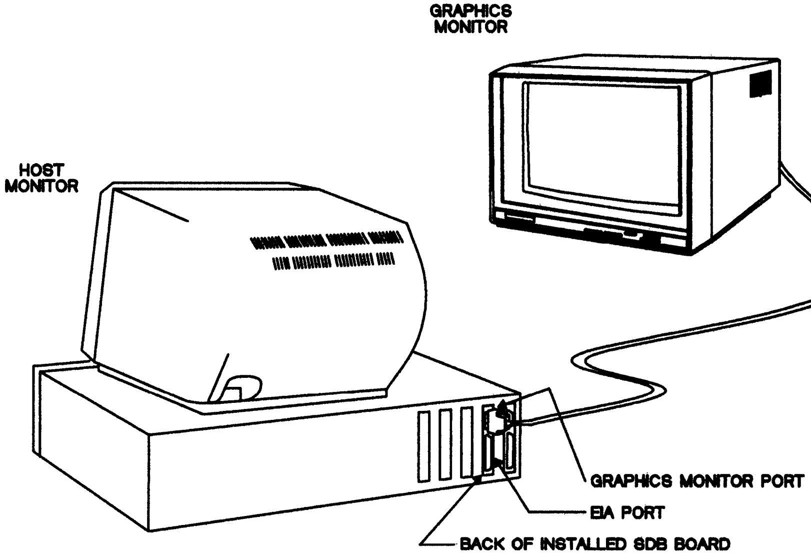


Figure 2-1. Cabling Between PC and Display Monitor

2.3 Jumper Settings

Figure 2-2 and Table 2-1 through Table 2-2 identify jumpers, setting descriptions, and factory settings.

Note that jumper W2 has to be set for a TI PC (set for IBM as shipped) as shown in Table 2-1.

Set jumper W7 according to connector pin 6 from the graphics monitor used:

W7, 1-2	Intensity on pin J4-6
W7, 2-3	Ground on pin J4-6

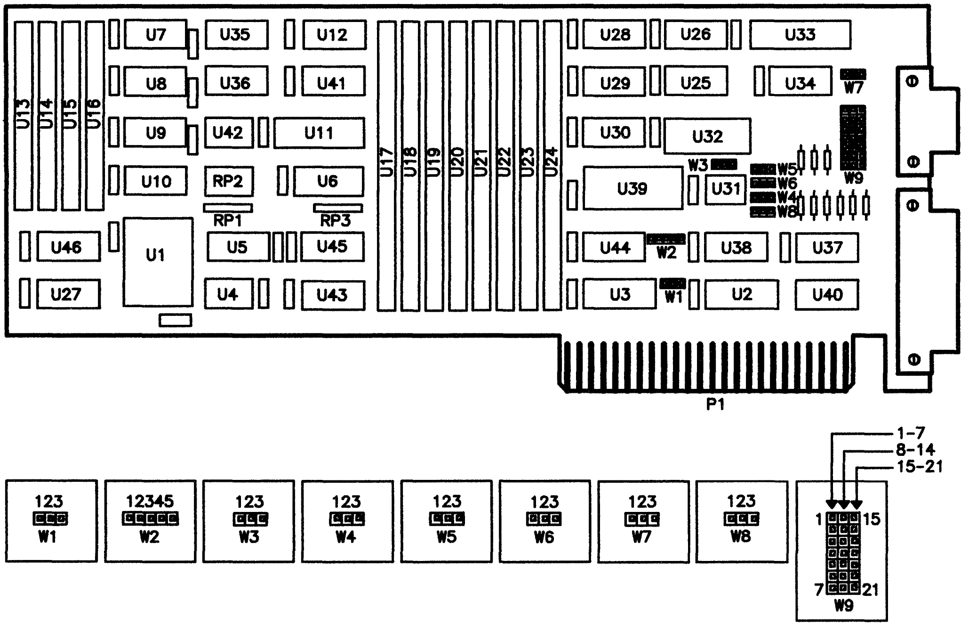


Figure 2-2. Jumper Locations and Configurations

Table 2-1. SDB Jumpers, General

FEATURE ENABLED	JUMPER	POSITION
Host Chip Select ($\overline{\text{HCS}}$ on TMS34010) grounded	W1	2 to 3
Host Chip Select ($\overline{\text{HCS}}$ on TMS34010) to bus decode logic at P1	W1	1 to 2†
Host Interrupt to PC Interrupt Level 3 (IBM PC)	W2	3 to 2†
EMUACKL to PC Interrupt Level 3 (IBM PC)	W2	3 to 4
Host Interrupt to PC Interrupt Level 2 (TI PC)	W2	1 to 2
EMUACKL to PC Interrupt Level 2 (TI PC)	W2	4 to 5
Palette In Line Mode	W3	2 to 3†
Palette In Frame Mode	W3	1 to 2

†As shipped.

Table 2-2. Jumper Settings, Analog Interface

FEATURE ENABLED	JUMPER	POSITION
Analog Interface Enabled	W9	8 to 15†
		9 to 16†
		10 to 17†
		11 to 18†
		12 to 19†
		13 to 20†
		14 to 21†
Negative Vertical Sync on pin J4-5 (connector J4)	W4	1 to 2
	W5	1 to 2
Positive Vertical Sync on pin J4-5 (connector J4)	W4	1 to 2
	W5	2 to 3
Negative Horizontal Sync on pin J4-4 (connector J4)	W6	1 to 2
	W8	1 to 2
Positive Horizontal Sync on pin J4-4 (connector J4)	W6	2 to 3
	W8	1 to 2
Negative Composite Sync on pin J4-4 (connector J4)	W6	2 to 3†
	W8	2 to 3†
Positive Composite Sync on pin J4-4 (connector J4)	W6	1 to 2
	W8	2 to 3
Logic Low on pin J4-5 (connector J4, VIDOUT5)	W4	2 to 3†
	W5	1 to 2
Logic high on pin J4-5 (connector J4, VIDOUT5)	W4	1 to 2
	W5	2 to 3†

†As shipped.

Table 2-3. Jumper Settings, Digital Interface

FEATURE ENABLED	JUMPER	POSITION
Digital Interface Enabled	W9	1 to 8 2 to 9 3 to 10 4 to 11 5 to 12 6 to 13 7 to 14
Negative Vertical Sync on pin J4-9 (connector J4)	W4 W5	1 to 2 1 to 2
Positive Vertical Sync on pin J4-9 (connector J4)	W4 W5	1 to 2 2 to 3
Negative Horizontal Sync on pin J4-8 (connector J4)	W6 W8	1 to 2 1 to 2
Positive Horizontal Sync on pin J4-8 (connector J4)	W6 W8	2 to 3 1 to 2
Negative Composite Sync on pin J4-8 (connector J4)	W6 W8	2 to 3 2 to 3
Positive Composite Sync on pin J4-8 (connector J4)	W6 W8	1 to 2 2 to 3
Ground on pin J4-2 (connector J4)	W7	2 to 3
Intensity on pin J4-2 (connector J4)	W7	1 to 2

2.4 Creating File CONFIG.SYS (IBM PC)

For the Debugger to operate properly on an IBM PC, a CONFIG.SYS file must be resident in the root directory. Use an editor to construct file CONFIG.SYS in the main root directory with the following contents:

```
BUFFERS = 20
FILES = 20
DEVICE = C:/MSDOS/ANSI.SYS    (not for TIPC)
```

In this example, C:/MSDOS/ANSI.SYS is the entire pathname to the system's ANSI.SYS file. If ANSI.SYS is in another directory, use that pathname instead of C:/MSDOS/ANSI.SYS.

2.5 Installation Summary

The following is an installation checkoff list:

1) [_] (Not for TIPC) Verify your system is configured so that device driver ANSI.SYS can be installed at system start (by CONFIG.SYS).

2) [_] Make a copy of the diskettes supplied. Keep the master disks in a safe place for backup purposes.

3) [_] Make certain the host computer:

[_] Has at least 512K bytes RAM, required for software support packages, and

[_] The following addresses are reserved (see memory maps in Figure 2-4):

- For IBM PC: >C7000 to >C7FFF
- For TI PC: >E7000 to >E7FFF

4) [_] The SDB comes ready for use with an IBM PC. If you have a TI PC, first change out the PAL in socket U3 (shown in Figure 2-3) with the PAL marked "TIA0100CC" shipped in its own envelope with the SDB. Figure 2-3 shows chip alignment.

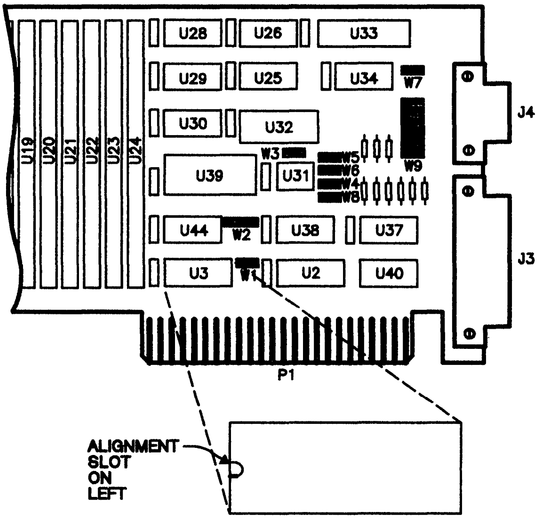


Figure 2-3. Alignment of Alternate PAL in U3

5) [_] Verify that jumper W2 is set for the PC type (IBM or TI) as shown in Table 2-1.

> **Note:**
>
> Sockets U35 and U36 are reserved for future use and are shipped unpopulated.

6) [_] Install board in a PC vacant slot

7) [_] Connect cable between display monitor and the SDB top port (J4, top port at end of board)

8) [_] Boot the MS-DOS system on the host computer. The MS-DOS version must be 2.11 or later, or PC-DOS version must be 2.1 or later. Note that the CONFIG.SYS file must install the device driver ANSI.SYS at boot time (not applicable for TIPC).

9) [_] Insert the Debugger diskette into the host A drive (diskette correctly marked for corresponding host -- "IBM PC" or "TI PC"). Then at the keyboard, enter:

```
A:<CR>              (drive-A designator)
SDB340<CR>          (call Debugger program, IBM)
```

or

```
A:<CR>              (drive-A designator)
SDB340T<CR>         (call Debugger program, TI)
```

10) [_] When the Debugger screen comes up (shown in Figure 2-5), call and execute the Demonstration program from the screen command line. Enter:

```
L TUTOR_E<CR>       (load Demonstration program)

RUN<CR>             (execute program)
```

Note that if the default disk drive is not the floppy drive, the drive designator must precede the file name (e.g., A:TUTOR_E).

If the SDB and display monitor are correctly installed, the Tutorial program will begin display on the display monitor. The program runs as described in Section 3.

Notes:

1. Several steps can be eliminated in the Tutorial program setup by entering an " -f" suffix (note that a space precedes the '-f') when calling the Debugger. This will load the Debugger and call the Tutorial program with one command. For example, with the proper drive designated on an **IBM PC**:

```
SDB340   -f<CR>     (call Debugger & Tutorial)
```

2. To avoid having to maintain a copy of SDB340.GSP and the SDB Help files in each directory in which you are developing software, it is suggested that you create directory \GSPTOOLS and install within it the SDB files (e.g., SDB340.GSP and help files such as GSPESAVE.HLP). Then equate the GSPTOOLS pathname with GSPDIR using the DOS command SET. For example, for files copied onto the C disk and within GSPTOOLS:

```
SET GSPDIR=C:\GSPTOOLS
```

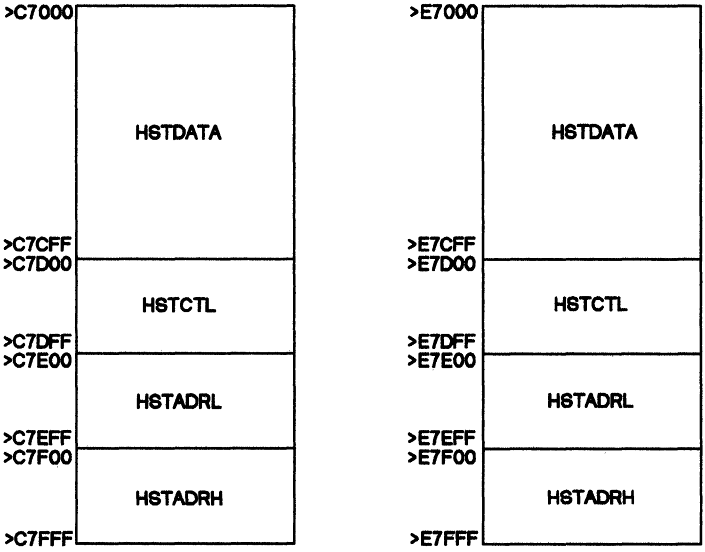
NOTE: Use all upper case and no spaces.

If you wish, execute the Demonstration program which runs four minutes. First remove the Debugger diskette and insert the DEMO & DIAGNOSTIC diskette. Then enter:

```
L MAINDEM<CR>       (load Demonstration program)
```

```
RUN<CR>          (execute Demonstration program)
```

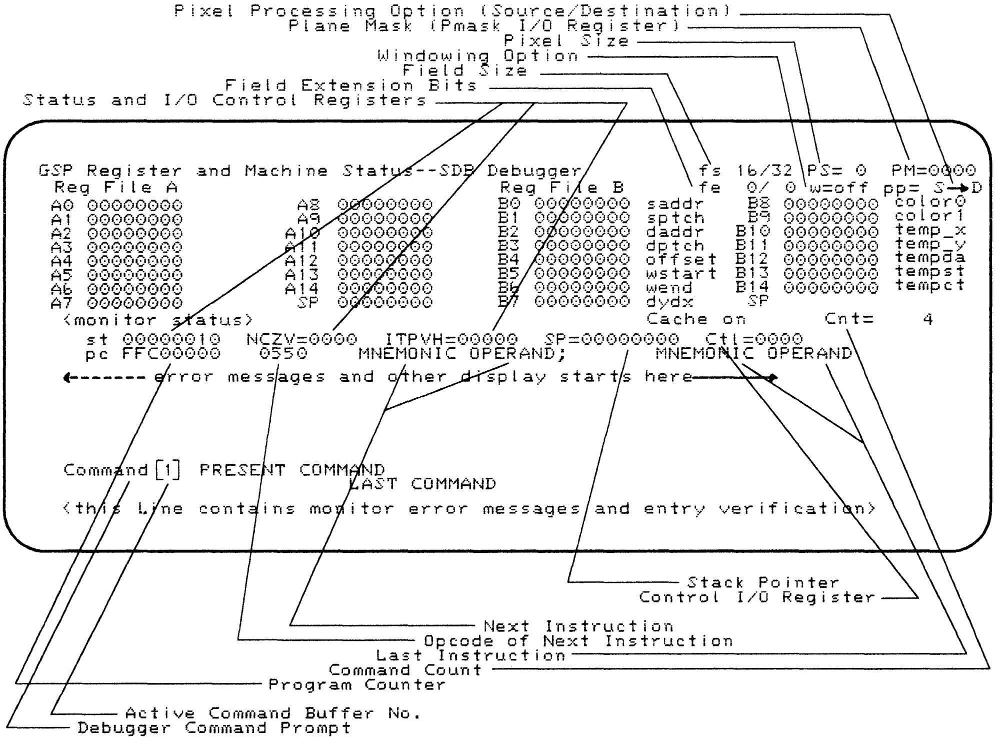
The Demonstration program repeats continuously. While running, you can check several of the Debugger commands such as those demonstrated in Section 3. A full description of the commands is provided in Section 4.



a) FOR IBM-PC

b) FOR TI-PC

Figure 2-4. Register Locations in Reserved Memory



```
GSP Register and Machine Status--SDB Debugger           fs  16/32  PS= 0    PM=0000
 Reg File A                         Reg File B          fe   0/ 0  w=off    pp= S→D
 A0 00000000        A8 00000000     B0 00000000 saddr    B8 00000000 color0
 A1 00000000        A9 00000000     B1 00000000 sptch    B9 00000000 color1
 A2 00000000       A10 00000000     B2 00000000 daddr   B10 00000000 temp_x
 A3 00000000       A11 00000000     B3 00000000 dptch   B11 00000000 temp_y
 A4 00000000       A12 00000000     B4 00000000 offset  B12 00000000 tempda
 A5 00000000       A13 00000000     B5 00000000 wstart  B13 00000000 tempst
 A6 00000000       A14 00000000     B6 00000000 wend    B14 00000000 tempct
 A7 00000000        SP 00000000     B7 00000000 dydx     SP
  <monitor status>                              Cache on        Cnt=     4
    st 00000010   NCZV=0000   ITPVH=00000   SP=00000000    Ctl=0000
    pc FFC00000    0550    MNEMONIC OPERAND;        MNEMONIC OPERAND
 ←------ error messages and other display starts here——→


 Command[1]  PRESENT COMMAND
                  LAST COMMAND
 <this Line contains monitor error messages and entry verification>
```

B Registers

B0	SADDR	Source Address	B6	WEND	Window End Address
B1	SPTCH	Source Pitch	B7	DYDX	Delta Y/Delta X
B2	DADDR	Destination Address	B8	COLOR0	Source Background Color
B3	DPTCH	Destination Pitch	B9	COLOR1	Source Foreground Color
B4	OFFSET	Linear Bit Offset	B10-14		PixBlt Temporary Regs
B5	WSTART	Window Start Address	B15	SP	Stack Pointer

Status and I/O Control Registers

N	Neg.	SR	I	Interrupt Enable	SR
C	Carry	SR	T	Transparency	I/O
Z	Zero	SR	P	PixBlt Interrupted	SR
V	Overflow	SR	V	PixBlt Vert. Direction	I/O
			H	PixBlt Horiz. Direction	I/O

Figure 2-5. Screen Display After Debug Software Installed

3. Tutorial

On each Debugger diskette (one each for IBM and TI PCs) resides a Tutorial program in TMS 34010 object form:

```
TUTOR—E.OUT
```

When executed, this program displays several graphics functions on the display monitor, accenting these by stopping at pre-set software breakpoints. At such points, machine instructions can be inspected on the Debugger panel. Functions include:

- Draw horizontal, vertical, and diagonal lines,
- Perform windowing,
- Write out characters in several fonts and sizes, etc.,
- Demonstrate other aspects of pixel processing.

This section gives a general discussion of the addressing scheme and data used to specify pixels (PIcture ELements -- the smallest controllable element on a screen). It also describes the disk resident Tutorial program and its execution. Also shown is how to assemble and link the three modules of the program to obtain your own source and object, demonstrating one method of constructing a graphics program.

Note:

Appendix E is a repeat of the first instructions demonstrated in this section. The difference is that you are encouraged to make changes to the machine state (e.g., registers, etc.) in order to discover the results of such changes.

Covered in this section:

3.1 Elements of Bit-Mapped Graphics

Note:

This section provides a tutorial approach to comprehending bit-mapped graphics -- including screen format and addressing, pixel addressing and color designation, and the role of several of the B registers. Further information is available in the TMS34010 User's Guide. If you are fully familiar with these subjects and wish to start executing the Tutorial program, go to Section 3.2 on page 3-11.

3.1.1 The Graphics Display

The Tutorial program assumes a 640 by 480 display with a 4-bit-per-pixel, 16-color display. Although the program only displays in a 256 x 128 block in the upper corner of the screen, the display pitch is assumed to be 1024 pixels. This conforms to the memory configuration of the SDB used in conjunction with one of the monitors specified in Figure 2-5 on page 2-12.

3.1.2 Screen Format and Memory Addressing

The Tutorial program uses a display area of 256 x 128 (>100 x >80) pixels, appearing in the upper left corner of the display monitor. When the Tutorial is initialized, the borders of the demonstration area are drawn for viewer reference.

While the display area used by the program is 256 (>100) pixels wide, the program assumes a full screen width of 1024 (>400) pixels. With a pixel size of four-bits-per-pixel, this requires 4096 (>1000) bits per horizontal line. This is **Destination Pitch, stored in register B3**. With a vertical dimension of 512 lines (only 480 are used), total display memory on the SDB is:

512 lines x 1024 pixels/line x 4 bits/pixel = 2,096,152 bits/screen
= 256K bytes/screen

To store this screen in memory, a 256K byte (128K word) memory segment is needed. Thus, the lower 2 M bits from >0000 0000 to >001F FFFF are reserved as the **Frame Buffer** (screen memory). This memory configuration is shown in Figure 3-1 on page 3-4.

MEMORY SPACE (Word Addr.)	OCCUPANTS
>0000 0000 thru 001F FFF0	DISPLAY RAM
>0020 0000 thru >BFFF FFF0	Reserved[†]
>C000 0000 thru >C000 01F0	IOREGS
>FFC0 0000 thru >FFFD FFF0	USER CODE
>FFFE 0000 thru >FFFF FFF0	RESERVED FOR DEBUGGER

[†] For a more detailed description of this area, see Figure 5-4 on page 5-6

Figure 3-1. SDB Memory Map

A key feature of the TMS34010 is XY addressing, which is conveni ent when manipulating information stored in screen memory. The XY mode uses a 32-bit address divided into two parts:

- 16 most-significant bits are the Y (vertical) coordinate
- 16 least-significant bits are the X (horizontal) coordinate

Any one of the general-purpose **A or B registers** can be used to contain this address. This addressing is represented in Figure 3-2.

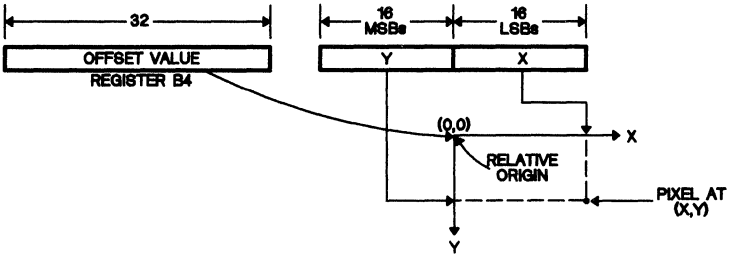


Figure 3-2. Pixel Addressing Using XY Coordinates

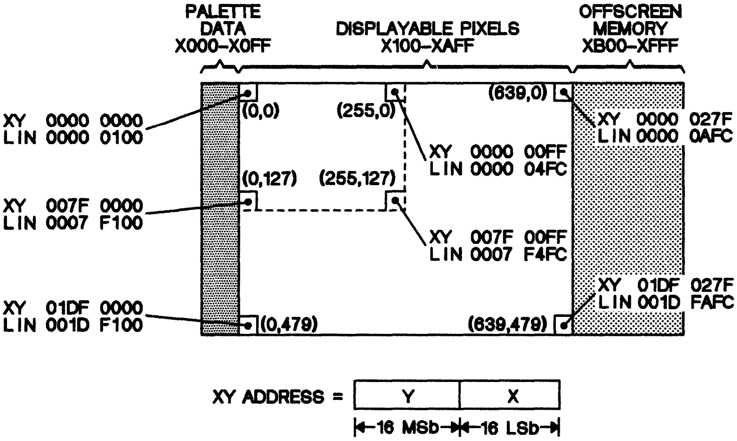


Figure 3-3. Linear and XY Addressing Example for SDB Tutorial Display (Upper Left) and Complete Screen

Figure 3-3 depicts both linear and XY addressing of an SDB screen display. The upper-left corner is the origin of the screen display. The upper leftmost pixel has an XY address of >0000 0000, meaning X =0, Y = 0. The linear address of this pixel is the value in **Register B4, the Offset Register**. If the screen portion of memory starts at >0000 0000, this will be the value loaded into the Offset Register. **Note**: the SDB is shipped in line-load mode and software loads a default of >100 into the Offset Register.

Segments of the *tutorial program's* screen **XY address** may range:

- X segment: >0000 to >00FF, incrementing from left to right
- Y segment: >0000 to >007F, incrementing downwards from the top

Each horizontal line is >1000 bits across (>400 pixels at 4 bits per pixel). For the top line of the display:

- Linear addresses >0000 0000 to >0000 00FF contain palette data for the line.
- Linear addresses >0000 0100 through >0000 0AFF contain displayed pixel data for the line (the Tutorial program uses only >0000 0100 to >0000 04FF).
- Linear addresses >0000 0B00 to >0000 0FFF contain non-displayed offscreen pixel data on the line.
- (For the second line, linear addresses >0000 1000 to >0000 10FF contain palette data for the second line, etc.)

For linear addressing, Registers SPTCH and DPTCH (B1 and B3) must be set to the width, in bits, of the destination array. For XY addressing, I/O Registers CONVSP and CONVDP (offsets >130 and >140) must be set using the following code (width must be a power of 2; e.g., 4096 below):

```
MOVI   4096,A0      ;width of 1K x 4 bits-per-pixel
LMO    A0,A1
MOVE   A1,@CONVSP   ;(use @CONVDP for destination)
```

3.1.3 Storage of Bit-Mapped Images

The Tutorial's display on the monitor is stored as data in off-screen memory. These patterns are stored as a binary data array with one bit representing one pixel. These are brought to the screen using the Pixel Array Operation With Expansion instruction (e.g., PIXBLT B,XY). This operation expands the array, equating each zero bit of the source to the pixel value stored in **Register COLOR0 (B8)**, and each one bit to the value in **Register COLOR1 (B9)**. The amount of memory needed to store an expanded array is equal to the product of the unexpanded array memory size and the pixel size.

3.1.4 Steps in Bit Mapping

To create a graphics pattern (e.g., a font), start with drawing the image on a grid having a width equal to an integer number of data words (16-bits each). Grid height can be any number of rows required by the image.

Next, decide if the image is to be a single foreground color with a single background color. If so, the most efficient method would be to store the image data in compressed binary format and produce the graphics using the pixel array operation with EXPAND. If this is the case, the following procedure can be used to digitize the data into DATA statements.

Step 1, Color.

Since the TMS34010 supports power-of-two pixel widths, the program's pixel size is four bits per pixel, allowing the representation of sixteen colors. The following list (Table 3-1) shows the colors and their corresponding pixel values -- values which are initialized by the Tutorial program at program outset.

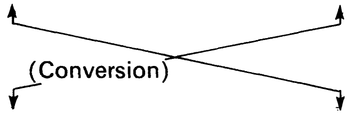
Table 3-1. Numerical Values for Colors

PIXEL VALUE		
(BINARY)	(DECIMAL)	COLOR
0000	0	Black
0001	1	Dark Blue
0010	2	Red
0011	3	Magenta (dark red)
0100	4	Green
0101	5	Cyan (light blue)
0110	6	Yellow
0111	7	White
1xxx	8-15	Various grey scale

Step 2, Digitize Pixels.

Convert the pixel patterns into assembly-language 16-bit data statements in the following format:

```
16-Pixel Pattern:     X..X  .X.X  X...  X.X.
(X = ON, . = OFF)
```



```
data value:           0101  0001  1010  1001  = >51A9
```

Convert each 16-pixel string in the same way, continuing until the total image has been digitized. The resulting hexadecimal .WORD statements are included in the assembly language program in the following example:

```
.WORD >958A,>AAAA,>CC00,>FF11, . . .
```

These would be converted to pixel image patterns (binary 0 = COLOR0 Register colors, binary 1 = COLOR1 Register colors):

```
>51A9,>5555,>0033,>88FF, . . .
```

The following is an example of the data manipulation that takes place during the execution of instruction PIXBLT B,L (pixel array operation with expand from binary to linear).

(1) Register setup:

```
         SADDR (B0) =  0002 4000
         SPTCH (B1) =  0000 0020
         DADDR (B2) =  0000 5300
         DPTCH (B3) =  0000 0400
        OFFSET (B4) =  0000 0000
          DYDX (B7) =  0001 0020
        COLOR0 (B8) =  0000 0000
        COLOR1 (B9) =  2222 2222
```

(2) Pixel Data. The following 16-bit data words are found starting at the source (linear) address:

LINEAR ADDRESS	DATA (HEX)	DATA (BINARY)
>24000	>330F	0011 0011 0000 1111
>24010	>88AA	1000 1000 1010 1010

This is expanded in being written to the destination address with the PIXBLT instruction (COLOR1 color code = 2 = red, COLOR0 code = 0 = black).

DESTINATION ADDRESS	VALUE (HEX)
>5300	>2222
>5310	>0000
>5320	>0022
>5330	>0022
>5340	>2020
>5350	>2020
>5360	>2000
>5370	>2000

Another way to view this is to see the patterns of linear addresses and their expansion to destination addresses side-by-side.

	ADDRESS	VALUE
		(binary)
Source:	>24000	0011 0011 0000 1111
	>24010	1000 1000 1010 1010
		(hexadecimal)
Destination:	>5330-5300	0022 0022 0000 2222
	>5370-5340	2000 2000 2020 2020

(3) Screen Image. This results in a screen image of (G = green, . = black):

```
SCREEN IMAGE:   GGGG....GG..GG...G.G.G.G...G...G
                ↑   --16 pixels--    ↑  --16 pixels--  ↑
```

Compare the pixel patterns, shown in two 16-pixel groups above, with the color patterns in the linear address and the XY address.

(4) Linear-to-XY Conversion. An example of converting linear address to XY address is shown below. When converting from linear to XY screen addressing, the order of significance of each pixel unit within each memory word appears to be reversed. For example:

LINEAR ADDR.	DATA	XY ADDR.	PIXEL VALUE	PIXEL NO.
>0000 0000	>5678	>0000 0000	>8	0
		>0000 0001	>7	1
		>0000 0002	>6	2
		>0000 0003	>5	3
>0000 0010	>9ABC	>0000 0004	>C	4
		>0000 0005	>B	5
		>0000 0006	>A	6
		>0000 0007	>9	7

Note that the least significant pixel in a word is displayed on the screen in front of the next significant pixel. For example, the first displayed 4-bit pixel of a 16-bit word is in the least significant four bits of the word (on right side as shown below in Figure 3-4):

WORD 1	**Pixel 4**	**Pixel 3**	**Pixel 2**	**Pixel 1**
WORD 2	**Pixel 8**	**Pixel 7**	**Pixel 6**	**Pixel 5**
WORD x	**Pixel 4x**	**Pixel 4x-1**	**Pixel 4x-2**	**Pixel 4x-3**

Figure 3-4. Pixel Placement in Memory Words

The pseudo code in Figure 3-5 can convert linear to XY addressing:

```
convtoxy (linear_data)
{
 log_val = (NOT (CONVSP)) AND 0x1F;
 y_part = ( (linear_data - OFFSET) >> log_val);

 /* generate mask for x */
 x_mask = (1l << log_val) - 1l;

 x_part = ((linear_data - OFFSET) AND (x_mask-1)) RIGHT
SHIFT log2_pixsize;

}

long convtolin (yxdata)
{
 log_val = (NOT (CONVSP)) AND 0x1F;

 log_val = (OFFSET +
        ((yxdata.y_part LEFT SHIFT log_val ) OR
         (yxdata.x_part LEFT SHIFT log2_pixsize)) );
}
```

Figure 3-5. Pseudo Code to Convert Linear to XY Addressing

3.2 Calling the Tutorial Program

The program can be called up (1) in a batch along with the Debugger or (2) with a Debugger command. In either case, **the Debugger software must be** either (1) on the **current** disk drive in the **current** directory (2) or in one of the directories in the search path (see MS-DOS PATH command). **In addition,** the file SDB340.GSP must either be on the current disk drive in the current directory or in the drive/directory combination as specified by GSPDIR in the MS-DOS command processor's environment. (See MS-DOS SET command and Section 2.5.)

3.2.1 Batch Call with Debugger

An " -f" parameter (space precedes the '-f') must be added to the Debugger call. When used, the files SDB340 (SDB340T for TI systems) and GSPINPUT.000 must be on the current drive in the current directory.

For an IBM-type PC:

```
SDB340   -f<CR>
```

The same operation for a TI PC:

```
SDB340T   -f<CR>
```

The Debugger will be called and, in turn, execute the Tutorial software. If the " -f" was left off, only the debugger would be called.

3.2.2 Call Tutorial From Debugger

If you are in the Debugger program, call the Tutorial with the Load command. To call the Debugger:

For an IBM-type PC:

```
SDB340<CR>
```

The same operation for a TI PC:

```
SDB340T<CR>
```

The TUTOR_E.OUT program must be on the current disk drive in the current directory and the Debugger display is on the screen as shown in Figure 2-5. Load the Tutorial with the following command:

```
Command[1] L TUTOR_E<CR>
```

3.3 The Tutorial Program

After loading the Tutorial (in Section 3.2), execute it with:

```
Command[1] RU<CR>
```

The Tutorial will execute until the first software breakpoint is encountered. This initial run is required for proper program execution -- it sets up the TMS34010 registers to specify the format of screen memory. The borders of the simulated screen (on the graphics monitor) are drawn for reference. All graphics will be drawn inside this box.

3.3.1 Tutorial Program Flow

After the screen borders are drawn on the graphics monitor, you can choose program flow. The choice is:

- a standard run of the Tutorial Program, or
- a specific demonstration.

A standard run comprises the following specific demonstrations in the order shown in Table 3-2 (descriptions are on the pages shown).

Table 3-2. Order and Location of Demonstrations in Tutorial Program

Order	PC Value	Demonstration	Page
1	>FFC0 0740	Pixel Transfer	3-14
2	>FFC0 07B0	Draw and Advance	3-17
3	>FFC0 0820	Fills	3-22
4	>FFC0 08E0	Pixel Block Transfer	3-26
5	>FFC0 0AE0	Transparency & Pixel Processing	3-29
6	>FFC0 0B20	Windowing	3-31
7	>FFC0 0B80	Text Spacing	3-33

Run Standard Program.
The Tutorial Program can be executed for a standard run by entering the following command:

```
Command[2] RU<CR>
```

The program will be executed in the order shown in Table 3-2. Once a software halt is reached, you can step through the program by pressing the <CR> key at each halt.

Select Specific Demonstration.
You can select any one of the seven routines listed in Table 3-2. To avoid visual confusion, select a specific program only when the simulated graphics screen appears blank except for the screen borders.

To choose the routine, enter:

```
(1)  Command[2] PC FFC00xxx<CR>
```

followed by:

```
(2)  Command[2] RU<CR>
```

where "FFC00xxx" is a program counter value listed in Table 3-2.

When each individual demonstration is completed, the example is cleared and only the screen border remains showing. At this time, you can select and execute another demonstration with the above two steps, or you can execute the next sequential demonstration by another RUn command as in step (2).

The following paragraphs describe the instructions demonstrated in the routines listed in Table 3-2.

3.3.2 Resuming Run Mode

While stopped in a particular demonstration, you can use commands to check aspects of execution (e.g., the DR command to switch between the A-B Registers and the I/O Registers). To resume executing the tutorial sequence, merely reissue the RUn command:

```
Command[2] RU<CR>
```

3.3.3 Clearing the Screen

If the tutorial programs are not executed in the order presented (e.g., such as in Table 3-2), the demonstration area will sometimes not fully erase. Figure 3-12 on page 3-25 in the FILL XY demonstration shows an easy way to clear the demonstration area of the screen, and additional values to be used to blank the entire screen.

3.4 Pixel Transfer (PIXT) PC = >FFC0 0740

Syntax: PIXT <source>,<destination>

Operation: A pixel value specified by the source operand is written to the location indicated by the destination operand. The instruction formats supported by the TMS34010 are:

PIXT	**Rs,*Rd**	Register to indirect linear
PIXT	**Rs,*Rd.xy**	Register to indirect xy
PIXT	***Rs,Rd**	Indirect linear to register
PIXT	***Rs,*Rd**	Indirect linear to indirect linear
PIXT	***Rs.xy,Rd**	Indirect xy to register
PIXT	***Rs.xy,*Rd.xy**	Indirect xy to indirect xy

When the destination is an indirect address of either type (linear or XY), a pixel processing option may be selected via the Control Register to perform an operation on the source pixel value before it is transferred. If the transparency bit is set in the Control Register and the source pixel value is zero, the destination pixel value will not be modified. The size of the pixel must be set in the PSIZE I/O Register and plane masking is in effect as specified in the PMASK I/O Register. If either the source or destination are indirect xy mode, the appropriate conversion factor I/O Register must be loaded.

Demonstration Start: The PIXT demonstration begins at PC = >FFC0 0740

(1) Enter: RU<CR> to begin the PIXT demonstration.

The mnemonic 'PIXT' is drawn in the upper left corner and the registers are set up for five demonstrations of the PIXT instruction. The first example of this instruction is a register-to-register indirect XY move: **PIXT A2,*A1.xy.**

The value of the pixel to be moved is >6 (indicating color yellow as listed in Table 3-1 on page 3-7). It is contained in the four least significant bits (LSbs) of Register A2. This value is written to the XY address contained in Register A1 (>0040 0080), replacing the value which is stored there. Since the destination is in the XY mode, it is necessary to set the CONVDP I/O Register (conversion register, destination pitch) to the appropriate value (>0013 for the demonstration screen size) for conversion to the correct address. This setup writes one pixel colored yellow (>6 as in Table 3-1 on 3-7) to the center of the demonstration screen.

(2) Enter: RU<CR> to execute instruction **PIXT A2,*A1.xy.**

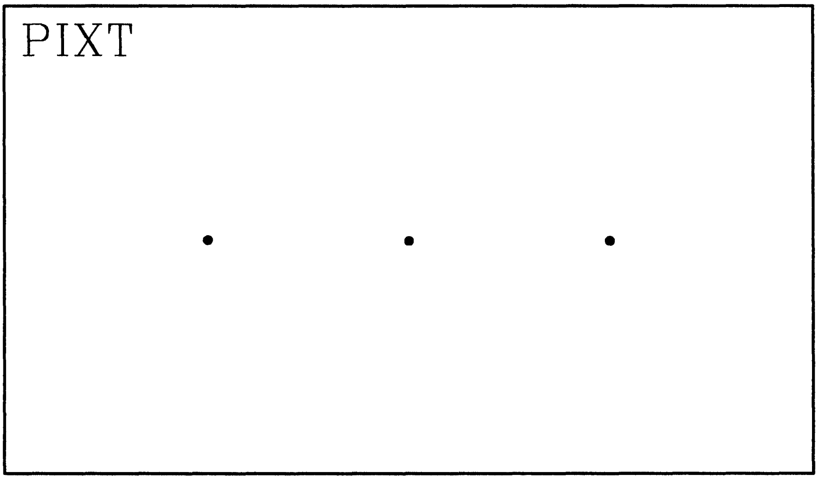


Figure 3-6. PIXT Display

The pixel appears in the center of the demonstration area, and the registers are unchanged.

The instruction **PIXT *A1.xy,A3** employs an XY address stored in Register A1 to point to a pixel value in memory (on the screen) as the source. The CONVSP I/O Register (conversion factor, destination pitch) must be loaded with the appropriate value to convert the XY source address (a program task). The pixel value is then copied into the LSbs of the destination register, A3, with all MSbs set to zero.

(3) Enter: RU<CR> to execute instruction **PIXT *A1.xy,A3.**

The value of the yellow pixel (>6) drawn in the first example is copied into Register A3, replacing >FFFF FFFF with >0000 0006. No other register values change.

The third example of PIXT demonstrates a move from a register to a linear address. Since the move does not use the XY addressing mode, it is not necessary to set either the CONVSP or CONVDP I/O Registers.

(4) Enter: RU<CR> to execute instruction **PIXT A3,*A4**.

The pixel value stored in Register A3 is moved to the linear address stored in A4 (>4 0200). A yellow pixel is drawn to the left of the first pixel.

The fourth PIXT example demonstrates transferring pixels from one XY screen location to another. With both source and destination being XY indirect, both CONVSP and CONVDP must be set up appropriately.

(5) Enter: RU<CR> to execute instruction **PIXT *A1.xy,*A3.xy.**

The pixel value at the XY address in A1 (>0040 0080) is copied to the location at the XY address in A5 (>0040 00C0). The center yellow pixel is copied to the right.

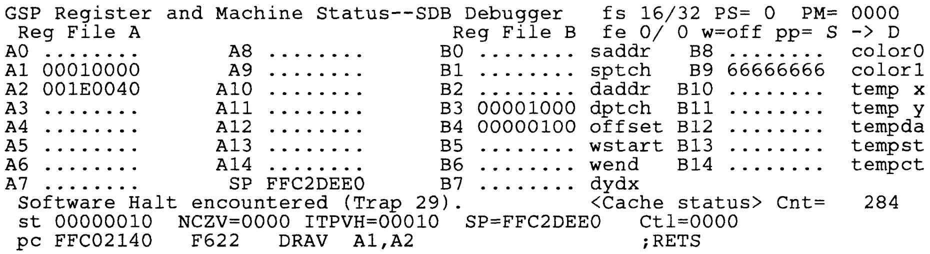
This completes the demonstration of the PIXT pixel transfer instruction.

3.5 Draw and Advance (DRAV) PC = >FFC0 07B0

Syntax: DRAV <Rs(source)>,<Rd(destination)>

Operation: A pixel of COLOR1 Register color is written to the XY location stored in Rd. Immediately afterwards, the value in Rd is incremented by the value in Rs. **NOTE: Rs and Rd must both be in the same register file (either A or B).**

(1) Enter: RU<CR> This writes the mnemonic **DRAV** inside the demonstration box, and the appropriate operand registers are set up for the draw and advance. The display appears as follows (........ = don't care):



```
GSP Register and Machine Status--SDB Debugger   fs 16/32 PS= 0  PM= 0000
 Reg File A                         Reg File B  fe 0/ 0 w=off pp= S -> D
A0 ........       A8 ........     B0 ........ saddr   B8 ........  color0
A1 00010000       A9 ........     B1 ........ sptch   B9 66666666  color1
A2 001E0040      A10 ........     B2 ........ daddr  B10 ........  temp x
A3 ........      A11 ........     B3 00001000 dptch  B11 ........  temp y
A4 ........      A12 ........     B4 00000100 offset B12 ........  tempda
A5 ........      A13 ........     B5 ........ wstart B13 ........  tempst
A6 ........      A14 ........     B6 ........ wend   B14 ........  tempct
A7 ........       SP FFC2DEE0     B7 ........ dydx
 Software Halt encountered (Trap 29).        <Cache status> Cnt=   284
 st 00000010  NCZV=0000 ITPVH=00010  SP=FFC2DEE0    Ctl=0000
 pc FFC02140   F622   DRAV  A1,A2                   ;RETS
```

Figure 3-7. DRAV Screen Display

As shown in the display:

- The instruction **DRAV A1,A2** now appears in the current instruction field of the machine state display.
- Register A2 contains the destination address in XY mode (>001E 0040: Y=001E, X=0040) which is the location to which the pixel will be moved.
- Register B9 is loaded with the >66666666, specifying the color yellow (see table on page 3-6).

When the instruction is executed, a yellow pixel is drawn 64 (>0040) pixels to the right and 30 (>001E) pixels below the origin of the demonstration screen (upper left corner). Then the address value in Register A2 is incremented by the value of Register A1 (>0001 0000: Y=1, X=0). To demonstrate this:

(2) Enter: RU<CR> The DRAV instruction is executed and a software trap follows immediately. One yellow pixel is drawn in the display block.

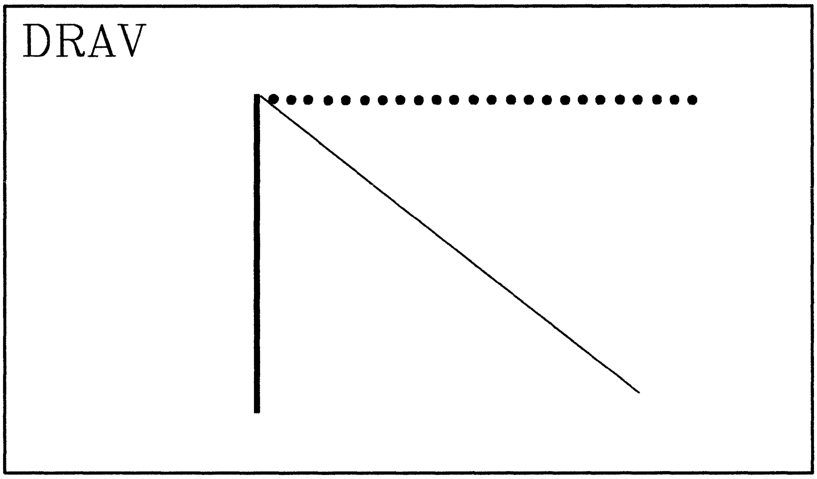


Figure 3-8. DRAV Display

Notice that a pixel has been drawn, and Register A2 is incremented (by >10000). By placing this instruction inside a loop, a line of pixels can be drawn with an X address constant and a Y address repeatedly incremented by one. To see such a loop:

(3) Enter: U<CR> to reverse-assemble the program (shown on the left of the screen); as follows:

```
Lnr Addr  code    Rev Assembly            Comment (not assembled)

xxxxxxxx  xxxx  UNKNOWN
xxxxxxxx  xxxx  UNKNOWN
FFC02160  09C0  MOVI    >004B,A0        Load loop count register
FFC02180  F622  DRAV    A1,A2           Draw and advance one pixel
FFC02190  3C40  DSJS    A0,@FFC02180    Dec Reg, jmp to DRAV if ≠ 0
FFC021A0  091D  TRAP    29              Halt after loop
FFC021B0  09C0  MOVI    >42,A0
```

The reverse-assembled portion contains a loop.

- The yelllow-colored line identifies the instruction just executed ("UNKNOWN" in the example).
- The cyan (light blue) color identifes the instruction *before* the one just executed -- also "UNKNOWN" in the example.
- Green identifies the *next* instruction to be executed. It loads Register A0 with the loop count of >4B (75).
- The next three instructions make a loop to draw a vertical line. (The TRAP 29 is a software breakpoint.)

- AO is decremented. If not zero, a jump to DRAV occurs to complete the loop and execute another draw/advance.

This loop will execute 75 times before the jump is discontinued -- each time drawing another pixel on the screen while incrementing the address in A2 one time in the Y direction. The final result is a vertical line 76 pixels in length.

(4) Enter: <Q> to quit the reverse assembly.

(5) Enter: <SS> (single step command) and watch as the yellow line is slowly draw with each entry of this command. This also allows you to see how the loop counter in Register AO is decremented and how the Y axis value (16 MSbs) in Register A2 is incremented by adding A1 to it.

Enter: RU<CR> to complete the loop and finish the yellow line (6s in Register COLOR1 = yellow) on the screen.

After being incremented by one 75 times, the value in destination Register A2 is now >006A 0040. Note that loop counter AO has been decremented to zero.

Two more examples show some of the flexibility of this instruction. The first employs a bidirectional increment to create a diagonal line.

(6) Enter: RU<CR> to set up the operand registers for a diagonal draw and advance.

The destination register is loaded with the same initial value as in the first example -- steps (1) to (5) above. The incrementing register (A1) contains >0001 0002 (Y=1, X=2), and the loop count in AO has been set to >42 (66). The COLOR1 Register is now >2222 2222, specifying red.

(7) Enter: U<CR> to display the reverse-assembled program.

The loop set up is similar to the first program (step (3)) with the destination address being incremented in both the X and Y directions.

(8) Enter: Q to quit the reverse assembly.

(9) Enter: RU<CR> to draw a diagonal line from the same starting point as in the first example.

Note the destination address is incremented by >42 in the Y direction and by >84 in the X direction. The final A2 value is >0060 00C4.

The final demonstration produces a dotted green horizontal line.

(10) Enter: RU<CR> to set up the register operands.

An identical loop to that above is used in this example.

- Destination Register A2 is the same as used previously.
- Increment value of >0000 0006 is in A1.
- COLOR1 Register contains >4444 4444 (green).

After every pixel is drawn, the X address is incremented by 6, leaving five blank pixels between each green pixel.

(11) Enter: RU<CR> to execute the loop and draw the line.

These are simple examples of the 'draw and advance' employing constant increments. More elaborate schemes of altering the increment can be used to implement various graphical algorithms for figure drawing.

3.6 Fill Array Instructions (FILL XY, FILL L) PC = >FFC0 0820

These instructions perform a pixel processing operation on a memory array using the value in COLOR1 Register as the source pixel value. The destination is defined in either XY or linear addressing mode, depending on which instruction is used.

3.6.1 Fill Array, XY Addressing

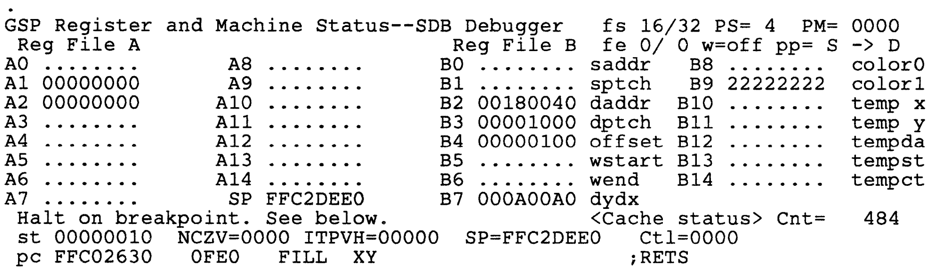
Syntax: FILL XY

Operation: A pixel processing operation is performed between the pixel value stored in the COLOR1 Register and an XY array of memory.

- The XY address in Register DADDR (B2) contains the location of the array's least-significant corner (screen upper left).
- Registers DPTCH, OFFSET, and CONVDP (I/O) must contain values appropriate to the screen-memory format.
- Register DYDX value of >000A 00A0 specifies dimensions of the destination array with the 16 MSbs indicating height and the 16 LSbs indicating width (both in pixels)
- The CONTROL I/O Register specifies the pixel processing option.

(1) Enter: RU<CR> to write the mnemonic **FILL XY** onto the screen and set up the appropriate operand registers to fill a rectangle on the screen.

The screen appears as follows (Figure 3-9):



```
GSP Register and Machine Status--SDB Debugger   fs 16/32 PS= 4  PM= 0000
 Reg File A                             Reg File B  fe 0/ 0 w=off pp= S  -> D
A0 ........        A8 ........        B0 ........ saddr   B8 ........  color0
A1 00000000        A9 ........        B1 ........ sptch   B9 22222222  color1
A2 00000000       A10 ........        B2 00180040 daddr  B10 ........  temp x
A3 ........       A11 ........        B3 00001000 dptch  B11 ........  temp y
A4 ........       A12 ........        B4 00000100 offset B12 ........  tempda
A5 ........       A13 ........        B5 ........ wstart B13 ........  tempst
A6 ........       A14 ........        B6 ........ wend   B14 ........  tempct
A7 ........        SP FFC2DEE0        B7 000A00A0 dydx
 Halt on breakpoint. See below.                  <Cache status> Cnt=   484
 st 00000010  NCZV=0000 ITPVH=00000  SP=FFC2DEE0   Ctl=0000
 pc FFC02630   0FE0   FILL  XY                       ;RETS
```

Figure 3-9. Register Display for Fill Screen, XY Addressing

The instruction **FILL XY** appears in the instruction field of the display, and the necessary registers are loaded to draw a red rectangle to the screen.

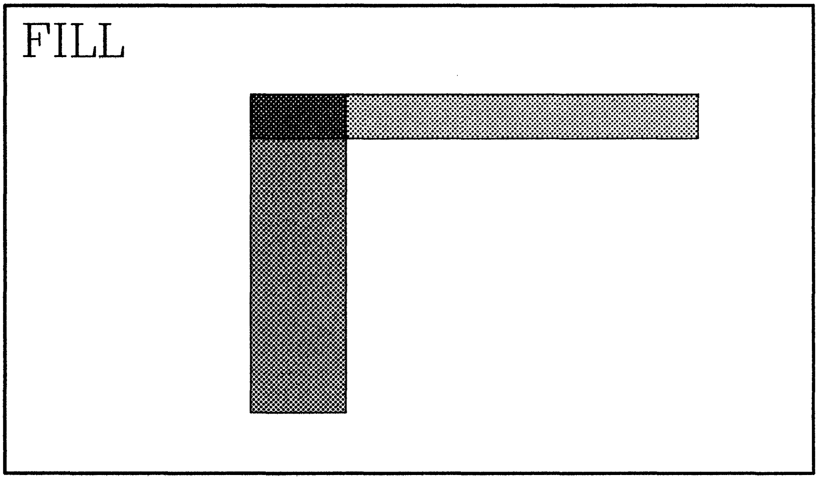


Figure 3-10. FILL Display

- Register DADDR (B2) is loaded to place the upper-left corner of the rectangle at the location >18 pixels below and >40 pixels to the right of screen origin.
- Register DYDX (B7) specifies rectangle height of >000A (10) pixels and width of >00A0 (160) pixels.
- Register COLOR1 (B9) specifies red (>2222 2222).
- Registers DPTCH (B3) and CONVDP (I/O Register -- display with DR command) are loaded with values appropriate for the screen used.

(2) Enter: RU<CR> to draw a red rectangle onto the screen.

Note that the destination address register has become corrupted.

3.6.2 Fill Array, Linear Addressing

Syntax: FILL L

Operation: The **FILL L** instruction is identical to the **FILL XY** except that:

- Register DADDR specifies a linear address to locate the least significant corner of the array.
- Registers OFFSET and CONVDP (I/O) do not have to be loaded since linear addressing mode is used.

(1) Enter: RU<CR> to write the mnemonic **FILL L** onto the screen and set up the appropriate operand registers to fill a rectangle on the screen.

The display appears as follows (Figure 3-11):

```
GSP Register and Machine Status--SDB Debugger   fs 16/32 PS= 4  PM= 0000
 Reg File A                         Reg File B  fe 0/ 0 w=off pp= S -> D
A0 00000020       A8 ........      B0 ........ saddr   B8 ........  color0
A1 ........       A9 ........      B1 ........ sptch   B9 11111111  color1
A2 ........      A10 ........      B2 00018200 daddr  B10 00000000  temp x
A3 ........      A11 ........      B3 00001000 dptch  B11 ........  temp y
A4 ........      A12 ........      B4 ........ offset B12 ........  tempda
A5 ........      A13 ........      B5 ........ wstart B13 00000000  tempst
A6 ........      A14 ........      B6 ........ wend   B14 ........  tempct
A7 ........       SP FFC2DEE0      B7 00460014 dydx
 Halt on breakpoint. See below.                <Cache status> Cnt=   676
 st 00000010  NCZV=0000 ITPVH=00010  SP=FFC2DEE0   Ctl=A200
 pc FFC02790   0FC0   FILL   L                    ;CALLR  FFC072F0
```

Figure 3-11. Register Display for Fill Screen, Linear Addressing

The instruction **FILL L** appears in the instruction field of the display along with register values necessary to draw a blue rectangle.

- Register COLOR1 (B9) specifies all dark blue (>11111111).
- Register DADDR (B2) contains the linear address equal to the XY address used in the **FILL XY** demonstration.
- Register DYDX (B7) specifes the heighth as >46 pixels and width as >14 pixels.
- Pixel processing option chosen is the Boolean OR operation.

The following takes place:

Source (binary)		**Destination (binary)**		**Resulting Destination Value (binary)**
0001 (blue)	ORed	0010 (red)	=	0011 (magenta)
0001 (blue)	ORed	0000 (black)	=	0001 (dark blue)

(2) Enter: RU<CR> to draw a blue rectangle on the screen with the area overlapping the red triangle changing to magenta.

Note that the destination address register has been corrupted.

Figure 3-12 demonstrates how the **FILL XY** demonstration can be used to blank the demonstration area on the screen.

This concludes the fill array demonstration.

Note that this example uses the FILL XY demonstration in the Tutorial program, thus the program must be present (TUTOR_E loaded).

Enter into two Command Buffers (for example purposes, 5 and 6 are used here):

```
Command[5] !PC FFC00820;RU<CR>     Go to FILL XY Demonstration
Command[5] 6<CR>                   Go to buffer 6
Command[6] !B2 20002;B7 7C00FC;B9 0;SS<CR>
```

Reg B2 = Demo Area Upper Left XY Addr.

Reg B7 = Demo Area Lower Right XY Address

Reg B9 = 0 = Black

Single Step to Execute

The following is a summary of the execution sequence:

STEP

```
(1) Command[x] 5<CR>                           Enter 1st Buffer
(2) Command[5] PC FFC00820;RU                  1st Buffer Displayed
(3) Command[5] <CR>                            Execute 1st Buffer
(4) Command[5] 6<CR>                           Enter 2d Buffer
(5) Command[6] B2 20002;B7 7C00FC;B9 0;SS 2d Buffer Displayed
(6) Command[6] <CR>                            Execute 2d Buffer
```

This key sequence -- 5 <CR> <CR> 6 <CR> <CR> -- executes a series of instructions to cause a blanking of the screen. Note that *other* command registers can be substituted for 5 and 6.

To blank *the entire* screen, change the B2 and B7 values to:

B2	0	(Screen upper left)
B7	1CA024A	(Screen lower right)

Figure 3-12. Using Two Command Buffers to Blank Demonstration Area

3.7 Pixel Block Transfers PC = >FFC0 08E0

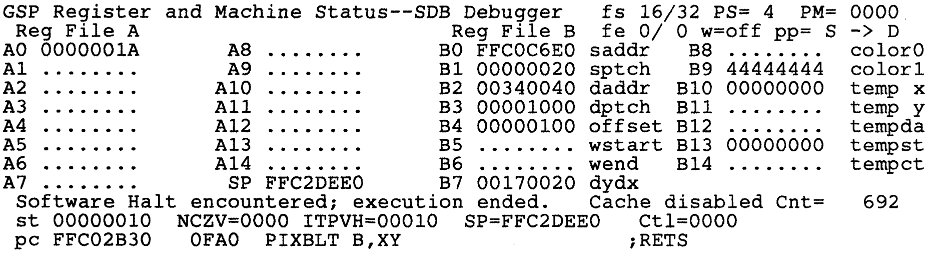
Syntax:

PIXBLT	B,XY	Expand linear to XY
PIXBLT	B,L	Expand linear to linear
PIXBLT	L,L	Linear to linear
PIXBLT	XY,L	XY to linear
PIXBLT	L,XY	Linear to XY
PIXBLT	XY,XY	XY to XY

Operation: The PIXBLT instructions take an array from a location defined by the SADDR (Source ADDRess) Register and use it to operate on an array whose location is defined by the DADDR (Destination ADDRess) Register. This operation is defined by the value of the Pixel Processing bits in the Control Register. The **PIXBLT B,*** (* = destination address mode) instructions expand each bit in the source array by the values in the color registers to the defined pixel size. If the bit is a 1 (one) in the source array, it is expanded using the value in the COLOR1 Register. Otherwise, the value in the COLOR0 applies. The pixel processing operation is performed on the expanded source array and the destination array.

(1) Enter: RU<CR> to write the mnemonic **PIXBLT B,XY** to the screen and set up the appropriate operand registers.

The screen appears as follows (Figure 3-13):



```
GSP Register and Machine Status--SDB Debugger   fs 16/32 PS= 4  PM= 0000
 Reg File A                         Reg File B  fe 0/ 0 w=off pp= S -> D
A0 0000001A       A8 ........      B0 FFC0C6E0 saddr   B8 ........ color0
A1 ........       A9 ........      B1 00000020 sptch   B9 44444444 color1
A2 ........      A10 ........      B2 00340040 daddr  B10 00000000 temp x
A3 ........      A11 ........      B3 00001000 dptch  B11 ........ temp y
A4 ........      A12 ........      B4 00000100 offset B12 ........ tempda
A5 ........      A13 ........      B5 ........ wstart B13 00000000 tempst
A6 ........      A14 ........      B6 ........ wend   B14 ........ tempct
A7 ........       SP FFC2DEE0      B7 00170020 dydx
 Software Halt encountered; execution ended.  Cache disabled Cnt=   692
 st 00000010  NCZV=0000 ITPVH=00010  SP=FFC2DEE0   Ctl=0000
 pc FFC02B30   0FA0  PIXBLT B,XY                  ;RETS
```

Figure 3-13. Register Display for PIXBLT B,XY

The instruction **PIXBLT B,XY** appears in the instruction field and registers are shown loaded with necessary values.

- Register SADDR (B0) contains the linear address of the unexpanded font 'W' whose bit size is in Register DYDX (B7).
- Register SPTCH (B1) contains >20: the width in bits of the source array.
- Register DADDR (B2) contains the destination XY value: address >34 pixels below and >40 pixels to the right of the screen origin.

- I/O Register CONVDP is set to the value appropriate for the 4096-bit (>1000-bit) screen width (use DR command to display I/O Registers).
- COLOR1 Register specifies all green (>44444444).

(2) Enter: RU<CR> to draw a green 'W' on the screen. Note that the destination and source registers have been corrupted.

The next instruction to be demonstrated is **PIXBLT B,L**

(3) Enter: RU<CR> to set the registers to the proper values to demonstrate **PIXBLT B,L.**

The only difference between **PIXBLT B,L** and **PIXBLT B,XY** is that

- The destination address is in linear terms instead of XY. For this example, the destination address is >FFC0 DF00 and the source address is >FFC0 C570, which is the font 'A'.
- The COLOR1 value is >2222 2222, the color red.
- Register DPTCH (B3) contains >1000 (4096), the width in bits of the screen or destination.

Now display the memory by doing the following:

(4) Enter: F FFC0DF00 FFC0E800 FFFF <CR> to fill this memory area with >Fs to later illustrate when the font 'A' is expanded and moved to this location.

Enter DM FFC0DF00<CR> to check for all Fs.

Enter: <CR> again to display further memory beginning at >FFC0 E380. Enter <Q> to quit the memory display.

(5) Enter: RU<CR> to expand font 'A' by the COLOR1 value >2222 2222 (red) and replace the destination array starting at >FFC0 DF00. To verify this, display the memory once more:

(6) Enter: DM FFC0DF00<CR> to verify that 0s and 2s have replaced the >Fs previously seen in memory. These are the values associated with Registers COLOR1 and COLOR0. Two 'As' made by the pixel value of 2s can be vaguely recognized in this memory display.

(7) Enter: <CR> to display the rest of the stored image. Enter <Q> to exit the memory display.

The next instruction demonstration is **PIXBLT L,XY**. To set up:

(8) Enter: RU<CR> to set up the registers to their proper values in order to demonstrate **PIXBLT L,XY**. The proper mnemonic is written to the screen.

When this instruction is executed:

- The expanded 'A' font at >FFC0 DF00 is moved to the XY address value stored in Register DADDR.

- Register SPTCH is set equal to the pitch of the expanded 'A' font in memory.
- I/O Register CONVDP is set to the appropriate value associated with screen pitch.

(9) Enter: RU<CR>. The expanded 'A' font from >FFC0 DF00 replaces the destination array located on the screen >34 lines down and >58 pixels to the right of the screen origin.

(10) Enter: RU<CR> to load the registers for the next instruction, **PIXBLT XY,XY**

The **PIXBLT XY,XY** is demonstrated by copying the letters 'WA' from their location on the screen to a location >34 pixels below and >A0 pixels to the right of the screen origin. This destination address, the source address, and implied operands have already been loaded. I/O Registers CONVSP and CONVDP have been set with appropriate conversion factors.

(11) Enter: RU<CR> to copy the letters 'WA' to the new screen location.

The next instruction demonstration will move the letters 'WA' into memory.

(12) Enter: RU<CR> to write the mnemonic **PIXBLT XY,L** to the screen and load the registers with appropriate values for the demonstration.

PIXBLT XY,L copies the array with XY address >34 0040 to destination address >FFC0 DF00. The expanded green 'W' and red 'A' are stored at address >FFC0 DF00. Therefore, only 4s, 2s, and 0s will be seen if memory is displayed (step (6)).

(13) Enter: RU<CR> to move the green 'W' and red 'A' into memory. To check this move:

(14) Enter: DM FFC0DF00<CR>. This displays memory filled with 4s (green), 2s (red), and 0s (black).

The last instruction to be demonstrated is **PIXBLT L,L** used to copy the 'WA' located at >FFC0 DF00 to the screen at linear address >151CC.

(15) Enter: RU<CR> The mnemonic **PIXBLT L,L** is written on the screen and the registers are loaded with appropriate values for the demonstration.

(16) Enter: RU<CR> to copy the letters 'WA' (at >FFC0 DF00) to the screen.

This concludes the PIXel BLock Transfer instruction demonstration. Press <CR> to clear the screen.

3.8 Transparency and Pixel Processing PC = >FFC0 0AE0

The TMS34010 is capable of performing two powerful operations in conjunction with raster-ops, array fills, and pixel moves:

- Transparency processing
- Pixel processing

3.8.1 Transparency Processing

Transparency is an option enabled by setting the appropriate Control Register bit. When in effect, the destination pixel is not modified if the source pixel is 0 (zero). This allows overlaying an image "on top" of a second image without destroying the features of the underlying image. This is shown in the two demonstrations that follow.

(1) Enter: RU<CR> to set up the transparency/pixel processing demonstration. A blue box is drawn with a yellow 'x' on top of it and both repeated five times. The transparency off occurs in the first box (as labeled).

(2) Enter: RU<CR> to perform a PIXel BLock Transfer with expand, but with transparency off.

The array containing the 'A' is moved inside the first box replacing every pixel previously stored in that location. The array 0 (zero) bits are expanded to black pixels (0 value) and the 1 (one) bits expanded to red (4).

(3) Enter: RU<CR> to set up the second example, with transparency on.

(4) Enter: RU<CR> to perform the PIXel BLock Transfer with expand and with transparency on.

The transfer with expand is the same as the first (in steps (1) and (2)); however, the source pixels of 0 (zero) value do not replace the destination. The 'A' appears to be written on top of the 'X'.

3.8.2 Pixel Processing

Control Register bits 10 through 14 specify the current pixel processing option. When a pixel is moved, the chosen logical or arithmetic function is performed on the source and destination pixels, with the result replacing the previous destination value. The flexibility of these options can achieve many useful results. The following transparency examples demonstrate other options.

(1) Enter: RU<CR> to set up the first example.

This demonstrates the MAX function. The values of each source and corresponding destination pixel are compared, and the greater of the two is written to the destination. The pixel processing option in effect is shown in the screen upper right corner.

(2) Enter: RU<CR> to execute **PIXBLT B,XY** with transparency on and the MAX option.

The **PIXBLT B,XY** is performed similar to that for the transparency demonstration (Section 3.8.1) with the following results:

- Since the value for red (2) is greater than dark blue (1), the portion of the 'A' overlying the blue background replaces the destination.

- Since the value of yellow (6) is greater than red, the yellow 'X' is not replaced.

The next demonstration uses MIN (opposite of MAX).

(3) Enter: RU<CR> to set the second pixel processing example.

(4) Enter: RU<CR> to execute PIXBLT B,XY with transparency on using the MIN option.

Notice that the result of the MIN option is exactly the opposite of MAX. If transparency is disabled, the black background of the 'A' replaces any previous information, 0 (zero) being the minimum value obtainable.

The next example demonstrates the arithmetic SUBS function (subtract with saturation).

(5) Enter: RU<CR> to set up the third pixel processing example.

(6) Enter: RU<CR> to execute the SUBS operation of the 'A' with the fifth block:

DESTINATION		SOURCE		VALUE
1 (dk blue)	minus	2 (blue)	=	0 (black, saturated)
6 (yellow)	minus	2 (blue)	=	4 (green)

The next Boolean demonstration is EXCLUSIVE-OR.

(7) Enter: RU<CR> to set up the next example.

(8) Enter: RU<CR> to XOR the 'A' with the fifth block.

The resulting operation is:

SOURCE		DESTINATION		RESULTING DESTINATION VALUE
0010 (red)	XORed	0001 (dk blue)	=	0011 (magenta)
0010 (red)	XORed	0110 (yellow)	=	0100 (green)

Enter: RU<CR> to clear the demonstration.

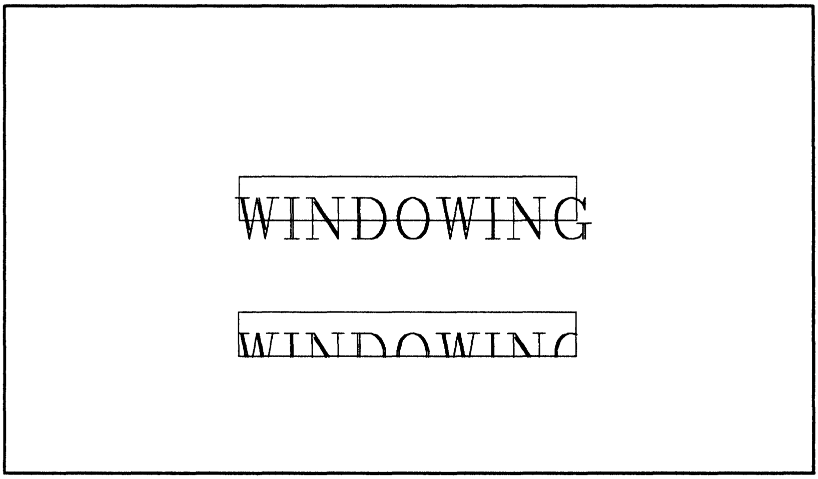


Figure 3-14. Window Display

3.9 Window Demonstration PC = >FFC0 0B20

The windowing option can be used to limit the active region of screen memory that can be modified; thus, it protects the remaining portion of screen memory from corruption. This pixel processing option has three modes available -- determined by the W bits (8 and 9) in the Control I/O Register:

W BIT VALUE (BINARY)	WINDOWING OPTION
00	Pixel writes allowed; no interrupts generated
01	Pixel writes inhibited; interrupts on pixel writes inside window
10	Pixel writes inside window allowed; interrupts on pixel writes outside window
11	Pixel writes outside window inhibited; no interrupts

When the windowing option is enabled, the WSTART and WEND Registers (B5 and B6) contain XY address values defining corners of the window:

- WSTART Register (window start) defines the window's upper left corner, and
- WEND Register (window end) defines the window's lower right corner.

(1) Enter: RU<CR> to draw two yellow borders about the perimeters of the windows. This better illuminates the areas to be used in comparing the two windowing options of clipping (W = 3) and no windowing (W = 0).

The windowing option is off (W = 0) at the beginning of the demonstration.

(2) Enter: RU<CR>. 'WINDOWING' is drawn so that it crosses over the yellow border. Because the windowing option is set w="off", no clipping nor interrupting occurs, and the WSTART and WEND registers are not set since windowing is not used.

(3) Enter: RU<CR>. The windowing option is set to clipping at the border (w="on"). Registers WSTART and WEND are loaded with XY addresses >40 0058 and >4C 00A9.

The values loaded into WSTART and WEND define a window 13 pixels high and 82 pixels wide. The lower yellow box on the screen is the perimeter of this window. Since the windowing option is set to clipping, nothing will be drawn outside this area.

(4) Enter: RU<CR>. 'WINDOWING' will again be drawn on the screen, but this time portions of the letters outside the defined window will be clipped.

If the windowing option is set to interrupt (W = 2) and a pixel operation's destination is found to cross the window boundary, the operation is not executed and program control is defined by an interrupt routine. (A windowing interrupt is not used in this tutorial.)

The TMS34010 provides a **CPW RS,RD** instruction which is useful in using a window. This instruction compares a point to the defined window and returns a code identifying the point's position relative to the window. This is useful for point-plotting algorithms used with a defined window.

This concludes the window demonstration

Enter: RU<CR> to clear the screen.

3.10 Text Spacing Demonstrations PC = >FFC0 0B80

It is important to handle text in graphics systems in various ways, including the spacing of text such as:

- block (Section 3.10.1)
- proportional (Section 3.10.2), and
- kerned (Section 3.10.3).

3.10.1 Block Spacing

This type of spacing displays on the screen a whole block of data in which the font had been defined. This usually takes up much more room than needed for the letters to be written; thus, skinnier letters appear isolated and the larger letters more crowded.

Enter: RU<CR>. The word 'AWAIT' is drawn using block spacing. Each letter is drawn with full 32-pixel blocks adjacent.

3.10.2 Proportional Spacing

This type of spacing puts only the defined character on the screen allowing only a specified amount of space between the two closest points of adjacent letters. This is done by defining the background color (COLOR0) as zero and using the transparency operation when transferring the font to the screen. When the move operation is over, only the defined character (COLOR1) appears on the screen. Tic marks on each letter define the actual width of the letter. For this example, a spacing of three pixels is used between each letter.

Enter: RU<CR> to draw the word 'AWAIT' using proportional spacing. The letters are spaced with three blank pixels between the closest active pixels.

3.10.3 Kerned Spacing

When two adjacent letters can overlap their defined widest regions without touching. these letters are said to kern. For example, the two letters WA can reside side-by-side with the upper right edge of the 'W' over the lower left edge of the 'A'. This allows for more text in a given space.

Enter: RU<00> = to draw the word 'AWAIT' using kerning. The words are spaced with overlapping active areas.

This concludes the text spacing demonstration. A <CR> returns program execution back to the beginning of the demonstration.

4. SDB Commands

Topics in this section include:

4.1 Key Features

Key Features:

- Complete control over machine state
- Efficient system memory use for TMS34010 program and screen memory
- Screen-oriented machine status display
- Versatile command entry with error reporting, file input and multiple command buffers
- Breakpoint and trace features

4.2 SDB Hardware and System Requirements

The SDB is available on 8088-, 8086- and compatible derivatives (running MS-DOS 2.11 and higher).

MS-DOS systems directly supported include:

- The IBM PC, PC/XT, and PC/AT and IBM PC compatible machines with 512K bytes of memory and CGA emulation.
- The Texas Instruments Professional Computer with 3-plane color graphics support and 512K bytes of memory.

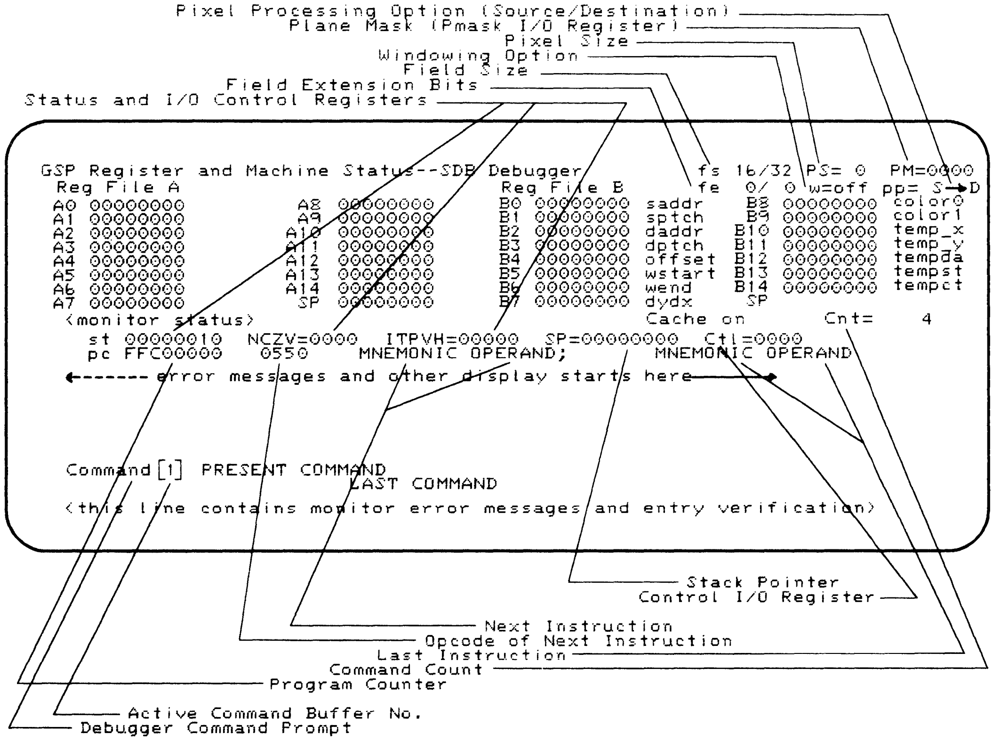
The system requirements for operating SDB for program debug are outlined below.

- A Host Operating and Display system as described above. This includes a graphics monitor to check desired output.
- An editor for manipulating TMS34010 assembly language and C source files.
- The TMS34010 Macroassembler, Linker and, optionally, the TMS34010 C compiler for the creation of input object files.

In addition, the user requires a working knowledge of the TMS34010 instruction set and familiarity with the memory addressing scheme.

4.3 System Description

The SDB monitor display presents the status and effects of these components.



```
GSP Register and Machine Status--SDB Debugger          fs  16/32  PS= 0   PM=0000
 Reg File A                             Reg File B          fe   0/ 0  w=off  pp= S→D
 A0 00000000        A8 00000000      B0 00000000 saddr    B8 00000000 color0
 A1 00000000        A9 00000000      B1 00000000 sptch    B9 00000000 color1
 A2 00000000       A10 00000000      B2 00000000 daddr   B10 00000000 temp_x
 A3 00000000       A11 00000000      B3 00000000 dptch   B11 00000000 temp_y
 A4 00000000       A12 00000000      B4 00000000 offset  B12 00000000 tempda
 A5 00000000       A13 00000000      B5 00000000 wstart  B13 00000000 tempst
 A6 00000000       A14 00000000      B6 00000000 wend    B14 00000000 tempct
 A7 00000000        SP 00000000      B7 00000000 dydx     SP
 <monitor status>                                Cache on          Cnt=     4
   st 00000010   NCZV=0000   ITPVH=00000   SP=00000000   Ctl=0000
   pc FFC00000    0550    MNEMONIC OPERAND;          MNEMONIC OPERAND
 ←------ error messages and other display starts here——→


 Command [1] PRESENT COMMAND
                         LAST COMMAND
 <this line contains monitor error messages and entry verification>
```

B Registers

B0	SADDR	Source Address	B6	WEND	Window End Address
B1	SPTCH	Source Pitch	B7	DYDX	Delta Y/Delta X
B2	DADDR	Destination Address	B8	COLOR0	Source Background Color
B3	DPTCH	Destination Pitch	B9	COLOR1	Source Foreground Color
B4	OFFSET	Linear Bit Offset	B10-14		PixBlt Temporary Regs
B5	WSTART	Window Start Address	B15	SP	Stack Pointer

Status and I/O Control Registers

N	Neg.	SR	I	Interrupt Enable	SR
C	Carry	SR	T	Transparency	I/O
Z	Zero	SR	P	PixBlt Interrupted	SR
V	Overflow	SR	V	PixBlt Vert. Direction	I/O
			H	PixBlt Horiz. Direction	I/O

NOTE: Registers are shown as all zeros for display purposes only.

Figure 4-1. SDB Debugger Screen Display

4.3.1 The SDB Machine State Display

Figure 4-1 shows a typical SDB machine-state display. The SDB is screen oriented; the machine state is displayed as commands are executed. Commands are always entered on the command line, except for commands executed from menus called up from the screen. Commands are displayed in uppercase; even if entered in lowercase (i.e., without pressing the SHIFT key). Figure 4-1 illustrates the default machine state display with the command line.

The following are provided in the Debugger screen display of Figure 4-1:

1) CPU State:
   - Field sizes (FS0 and FS1) - decimal values
   - Field extension bits (FE0 and FE1) - decimal values
   - Pixel size - decimal values
   - Plane mask
   - Windowing option
   - Pixel processing option
2) Internal registers (A and B files), Stack Pointer, and optional names
3) Monitor status messages
4) Status Register
5) Control and status elements
6) Stack Pointer
7) Control register
8) Program Counter and data at that location
9) Next instruction
10) Previous instruction
11) "Scratch" display area - 10 lines
12) Command line
13) Current buffer number
14) Last command entered

The cursor in the following descriptions is represented by an underscore (—). The cursor for the default machine state display is shown in Figure 4-1 immediately following the `Command[1]` prompt at the bottom of the screen. Except for menu-driven commands, information is only entered from the command line in the space following the command prompt. The cursor is generally represented by a full, shaded, upright slow-blinking rectangle.

4.3.1.1 Machine State Display

Beginning on screen line 3 below the headings (display is shown in Figure 4-1), are displayed the TMS34010's 30 general-purpose registers, **A0-A14 and B0-B14**. Also displayed is the Stack Pointer (accesssible as **SP** or register A15 or B15).

The current state of the of the TMS34010 CPU is displayed in the upper right hand corner. This includes the field sizes (**FS0 and FS1**) for fields 0 and 1, respectively, followed by their respective field extension bits (**FE0 and FE1**). All four of these values are extractions from the Status Register and are in decimal.

The field size and field extension used by the current instruction are highlighted in green. If the current instruction does not use either the field size or field extension, then both values will be in yellow.

To the immediate right of the field size is the pixel size, designated **PS**. This is the current value of the PSIZE I/O register, and is the pixel size used by the graphics instructions. The third value on this display line is the value of the plane mask register, designated **PM**.

Just below the PS value is the currently selected windowing option, designated **W**. The windowing option value is contained within the I/O CONTROL register. Any windowing option selected other than *off* is highlighted in cyan (light blue).

To the right of the windowing option is the currently selected pixel-processing option, desgnated **PP**. The pixel-processing option value is given by bits within the I/O CONTROL register. Any pixel-processing option other than source to destination (**S→D**) is highlighted in cyan.

The line immediately below the A and B registers starts with the full Status Register contents, along with several portions of the status register displayed individually, the Stack Pointer register, and the CONTROL I/O register. To the right of the Status Register contents are selected bits within the Status and I/O CONTROL Registers, designated as **NCZV** and **ITPVH**. The names of the bits and their source registers are:

N	Negative	Status Register
C	Carry	Status Register
Z	Zero	Status Register
V	Overflow	Status Register
I	Interrupt Enable	Status Register
T	Transparency	I/O CONTROL Register
P	PixBlt Interrupted	Status Register
V	PixBlt Vertical Direction	I/O CONTROL Register
H	PixBlt Horizontal Direction	I/O CONTROL Register

The last line of the machine state display gives values of:

- Program Counter (**PC**),
- value of the word pointed to by the Program Counter,

- reverse assembly (source statement mnemonic) of that word,
- reverse assembly of the last instruction executed.

If the last instruction is the same as the current instruction, it is in green; otherwise, in cyan (light blue). If the source line is not known, it is marked "UNKNOWN."

Any listed information is given in the 10 blank lines between the reverse assembly line and the command line. This is the scratch display area.

4.3.1.2 Monitor Status Line

Monitor status messages are presented below the A and B registers. Monitor status messages describe what particular function the SDB is performing. In the event that the SDB is halted (due to a breakpoint or an error condition), the monitor status line describes this condition while specific error messages are listed to the information display line and below. Monitor status messages are described individually in their pertinent sections.

4.4 SDB Operation

This section guides you through a typical SDB session to give a feel for the type of operation and debug that is possible with the SDB.

Notes:

1. Through MS-DOS, you can specify a directory in which to find the SDB340.GSP and the Help files. This avoids having to keep multiple files of SDB340.GSP throughout the directory structure. Do this with the DOS SET command to specify the directory:

   ```
   SET GSPDIR=<PATHNAME>
   ```

   In this command, adhere to the following: (1) use only one space in the command -- this is the space after SET, and (2) use only uppercase letters.

   For example purposes, have both SDB340.GSP and the help files in \GSPTOOLS. Thus the following could be used:

   ```
   SET GSPDIR=\GSPTOOLS
   ```

   If this command is either placed in the AUTOEXEC.BAT file or typed as a command, the Debugger will look in the directory GSPTOOLS for SDB340.GSP and the Help files.

2. The files GSPINPUT.000, SDEFIL.xxx, SMSFIL.xxx, and SMIFIL.xxx must always be in the default directory in order to be used by the SDB340. (Commands such as SDE, SMI, SMS, and SWITCH use these files.)

Install the appropriate Debugger diskette into your system (diskette marked IBM DEBUGGER or TI DEBUGGER) before calling it up under MS-DOS with one of the following commands.

4.4.1 Invoking SDB from Disk

The SDB is invoked under MS-DOS by typing one of these commands:

```
SDB340 <CR>          (IBM PC, etc.)

SDB340T <CR>         (TI PC, etc.)
```

You can then load your object code into the SDB using the

```
L <file name> {<offset>} <CR>
```

command giving the object-code file name and the desired program memory start location.

4.4.2 Invoking the SDB Using a File Option

For the IBM PC, the SDB is invoked from the MS-DOS command using the following syntax:

```
SDB340 [-f] [load_file [offset]]  <CR>
```

For the TI PC, enter the following command:

```
SDB340T [-f] [load_file [offset]]  <CR>
```

The one option, -f, initiates the SDB to obtain input from the command input file GSPINPUT.000 -- see the SWITCH command. Incorrect options are ignored.

4.4.3 Initial Display

The SDB initiates by displaying its banner. It then turns on the graphics card, loads any files specified, performs a reset, and then displays the current machine state.

The initial machine state display with no file loaded and with memory initialized to zero appears as shown in Figure 4-1.

4.4.4 SDB Command Line

Since SDB is a both command-driven and screen-oriented, it is controlled by a set of general commands and those tailored to the TMS34010. These provide control over both the device being simulated and the simulated graphics environment. Command examples:

SP	(to access Stack Pointer)
PBX	(to access the PixBlt executing bit PBX)

Note that these command names "suggest themselves" and may be inferred from the TMS34010 descriptions. In most cases, you can change the value of an item in the screen display by merely typing its screen name followed by desired value on the command line.

4.4.4.1 Command Entry

All user-entered commands are initially entered on the command line at the lower lefthand corner of the screen. Commands may be entered in upper or lower case, in any combination (however, lower case are translated to upper case). The command line contains the prompt:

```
Command[0]
```

with the bracketed number 1 being the currently active command buffer (see command buffers in Section 4.4.4.4). The prompt and cursor generally appear as follows:

```
Command[0] _
```

(_ marks the cursor position)

The command line retains the most-recent command entered into the command buffer. It may be written over or edited to enter a new command. The line immediately beneath it shows the most recently stored command and cannot be edited.

The SDB allows editing of the command line with some simple editing keys (BS = BACKSPACE, CTRL = Control key, etc.):

<left arrow> or <BS> or <CTRL-S> or <CTRL-H>	Back up one character
<right arrow> or <CTRL-D>	Forward one character
<TAB> or <CTRL-F>	Forward one word
<SHIFT-TAB> or <CTRL-A>	Backup one word
<DELete> or <CTRL-G>	Delete character
<INSert> or <CTRL-V>	Start inserting characters

Once the command line is edited to your satisfaction, the command can be processed by the SDB by typing either a <CR>, or a <LINEFEED> or <CTRL-J>:

- <CR> truncates the command, executing only that to the *left of the cursor,*
- <LINEFEED> or a <CTRL-J> *executes the entire command line.*

There are ten command buffers (0 to 9), each capable of containing a command or a command sequence. Changing buffers is discussed in Section 4.4.4.4.

A command may be continually executed by pressing <CR> with the cursor in the leftmost position on the command line.

Multiple commands can be entered on one command line by separating them with semicolons.

4.4.4.2 Command Parameters, Numeric Prefixes

Format for numeric parameters in commands can be either in decimal or hexadecimal (hex). **Numeric prefixes** for these:

Prefix	
%	**decimal** value. For example: %124 = >7C.
>	**hexadecimal** value. For example >1000 = %4096.

If you are unsure of the default number base used by a command, use either of the above prefixes (either will be recognized).

Numeric parameter format defaults to the format used most often. For example:

- address parameters default to hex,
- register numbers default to decimal,
- register contents default to hex.

All values are considered positive unless prefixed with a minus sign.

In the examples below, the A command used to modify the contents of the A file registers. The syntax is:

```
A <register number> <register value>
```

All of the following commands will result in the contents of register A12 being >FFFF FFFF.

```
Command[1] A 12 FFFFFFFF
Command[1] A %12 FFFFFFFF
Command[1] A >C FFFFFFFF
Command[1] A 12 -1
Command[1] A 12 -%1
Command[1] A >C ->1
```

> **Note:**
>
> All memory references are given in terms of bit addresses; therefore, word-aligned addresses should have a last ASCII character of 0 (zero). The last four bits of memory addresses given from the command line are forced to zero.

Sixteen-bit hexadecimal values are represented with up to 4 ASCII characters (including leading zeros) while 32-bit hexadecimal values are represented with up to 8 ASCII characters (including leading zeros).

4.4.4.3 Register Value in Commands

Register contents can be used as values in SDB commands. Use the format:

```
R<register designator>           (32-bit value)
```

where the register designator is one of the following:

- any A or B register,
- Stack Pointer (SP),
- Program Counter (PC), or
- Status Register (ST).

Also, a *portion* of the register contents can be designated. Either the *left (16 Y bits) or the right (16 X bits)* can be designated.

Use the format:

```
RY<register designator>          (left 16 bits)
   or
RX<register designator>          (right 16 bits)
```

EXAMPLES:

1) `Command[0] A14 RSP <CR>`

puts the *contents* of the Stack Pointer (SP) into register A14.

2) `Command[0] SP RPC <CR>`

puts the contents of the Program Counter (PC) into the Stack Pointer.

```
3) Command[0] MM  RSP  FFFF <CR>
```

uses the Modify Memory command to change the contents of the single-word *pointed to* by the Stack Pointer (RSP) to the value >FFFF. This example uses the "R" prefix to designate one of the on-chip registers.

```
4) Command[0] A14 -RA14 <CR>
```

puts the *negative* of the contents of A14 into register A14.

```
5) Command[0] MM  RSP  RA1 <CR>
```

uses the Modify Memory command to change the two words *pointed to* by the Stack Pointer (RSP) to the value in A1.

```
6) Command[0] MM  RB2  RYA1 <CR>
```

uses the Modify Memory command to change the single word *pointed to* by Register B2 to the value in the left-most 16 bits of Register A1.

4.4.4.4 Command Buffers

The SDB maintains ten command buffers (0 to 9) for the user to handily store commands. Thus a command or command string can be accessed quickly -- without having to be rekeyed each time. Each buffer has a storage capacity of 64 characters.

The command buffer is chosen by first entering its number (0 to 9), shown inside the square brackets after the **Command** prompt (e.g., 0 for Command Buffer zero below)::

```
Command[0]
```

The (default or present) command buffer can be changed by typing one of the following:

0,1,2,3,4,5,6,7,8,9,+,down arrow,-,up arrow

One of these must be the first entry of the command line. The exited buffer remains unchanged. These entries, followed by <CR>, cause the following:

Entry	
0 to 9	Brings up buffer of the number entered.
+ or down-arrow	Brings up next (higher numbered) buffer.
- or up-arrow	Brings up previous buffer.

Buffer 9 will roll over to 0 in a forward-moving change, and buffer 0 will become 9 in the reverse direction.

Multiple command buffers allow storage of specific commands that can be executed with less key strokes. For example, set up buffers with specific set/change breakpoint commands, or a single RUn command, or a RUn to breakpoint command followed by a display memory command. Figure 3-12 on 3-25 shows the entry of commands in two buffers that will blank the entire screen.

Command buffers may be chained together to provide lengthy command sequences. To do this, give the next buffer number to be executed as the last entry on the command line.

EXAMPLE

```
Command[0] SS; DM 0 200; 1
```

The above command executes a single step, displays memory from address 0 to address 200, and then goes to command buffer 1 to begin execution. A buffer may even reference itself, providing a simple looping mechanism.

```
EXAMPLE                                          COMMENT

     Command[0] A 13 340990BC                    Buffer 0
                     A 13 340990BC

     Command[0] 5 <CR>40990BC                    Enter "5"
                     A 13 340990BC

     Command[5] HELP                             Buffer 5
                     HELP
```

In this case, buffer 0 still contains the command `A 13 340990BC` which may be re-executed by returning to buffer 0.

The +, -, up-arrow, and down-arrow keys move through the buffers in a relative order. See example below:

```
EXAMPLE                                          COMMENT

     Command[0] A 13 340990BC                    Buffer 0
                     A 13 340990BC

     Command[0] + <CR> 13 340990BC               Enter "+"

     Command[1] HELP                             Buffer 1
                     HELP

     Command[1] - <CR>ELP                        Enter "-"

     Command[0] A 13 340990BC                    Bufffer 0
                     A 13 340990BC
```

In other words, buffer 0 retains command `A 13 340990BC` which may be re-executed by returning to buffer 0.

4.4.4.5 Storing a String of Commands in the Buffer

A string of commands may be stored in a buffer without being executed by preceding the string with an exclamation mark (!):

```
Command[1] !A13 45; A14 6000; run  <CR>
            A9 801AC
```

4.4.4.6 Command Error Messages

An error in command or parameter entry causes a monitor error message to appear on the Monitor Message Line beneath the command line as follows:

```
Command[1] && 4@<CR>
            A 9 34010

Command[1] && 4@
Command not recognized; re-enter

Command[1] && 4@
            && 4@
```

Multiple command error messages are queued (i.e., stored up). The next message can be displayed by typing <ESC>.

Note that an <ESC> is needed to clear the error and return to the command line. The command in error is not removed from the buffer so that it may be reviewed. The last message can be reviewed using the LM command.

4.4.5 Error Reporting

Errors are reported back to the user in a number of ways.

- Command-entry errors and monitor-initiated memory-access errors are reported on the line immediately beneath the command line. These error messages are generally self explanatory and describe how to to correct the entry. See Command Error Messages, Section 4.4.4.6.

- Instruction execution errors are displayed in red in the scratch-display area. Execution error messages remain displayed on the screen. They can be reviewed later using the LE command.

An <ESC> clears error messages. The CLS command clears the entire scratch display area (between all the registers at the top and the Command line at the bottom).

4.4.6 Single-Line Assembler

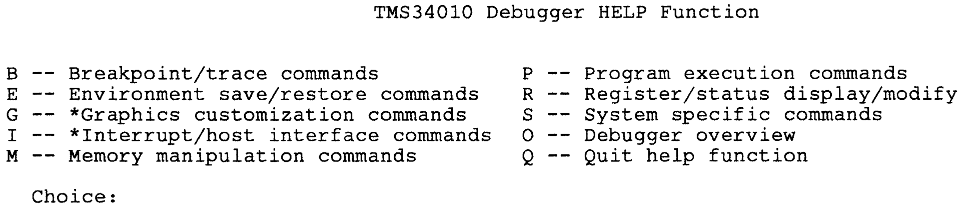
With the MM (Modify Memory) command, the SDB accepts and assembles single statements of TMS34010 assembler code, and places the resulting object value into memory. This assembler accepts only absolute numeric values.

4.4.7 Using the HELP Function

Type HELP or H to enter the Help Utility. The utility displays a menu of choices for each of various classes of instructions. Invoking any of these calls up a file of reference information about the commands.

> **Note:**
>
> The file Help utilities must either be on the current disk drive in the current directory or in the drive/directory combination as specified by GSPDIR in the MS-DOS command processor's environment. (See SET command in "MS-DOS Operating System" handbook and Section 2.5.)

The Help files consist of a command character and a brief summary of each command as shown in Figure 4-2.



```
                                    TMS34010 Debugger HELP Function

B -- Breakpoint/trace commands              P -- Program execution commands
E -- Environment save/restore commands      R -- Register/status display/modify
G -- *Graphics customization commands       S -- System specific commands
I -- *Interrupt/host interface commands     O -- Debugger overview
M -- Memory manipulation commands           Q -- Quit help function

  Choice:
```

*NOTE: The G and I choices are not used with the SDB.

Figure 4-2. SDB Help Utility Menu

4.4.8 Host Interface

The host interface of the SDB board is used by the Debugger exclusively for status communications and I/O functions. Thus the host interface may not be independently accessed while the Debugger is running.

4.4.9 Loading and Running Code

Object code is loaded into the SDB using the

```
L <file name> [offset]
```

command giving the object-code file name and the relocation offset to be added to the addresses in the object file.

An example entry of the load object code command is given below.

```
Command[1] L CODA.OUT 10200
```

This command causes the file CODA.OUT to be opened and its contents read, interpreted and loaded into the simulated TMS34010 with an offset of >00010200. Note that this value offsets any addresses given in the object module. The chart in Figure 4-3 describes protected (allocated) areas of memory. **Beware of offsetting non-relocatable files.**

MEMORY SPACE	OCCUPANTS
>0000 0000 thru 001F FFF0	DISPLAY RAM
>0020 0000 thru >BFFF FFF0	Reserved†
>C000 0000 thru >C000 01F0	IOREGS
>FFC0 0000 thru >FFFD FFF0	USER CODE
>FFFE 0000 thru >FFFF FFF0	RESERVED FOR DEBUGGER

†For a more detailed description of this area, see Figure 5-4 on page 5-6.

Figure 4-3. SDB Memory Map

Note that no checks can be made as to whether screen/program boundaries are violated. Screen and program spaces may be written to and executed out of at will; therefore, be careful how you treat memory. Do not write into the area reserved for the Debugger.

TRAP vectors should be specified at Assembly/Link time. This will enable RESET and any TRAPS to operate correctly.

Note:

Do not modify vectors for **Traps 8, 25, 26, 27, 28, or 29**. Any changes to these traps will cause a system failure.

4.4.10 Saving Machine Status

You can save the current machine state either locally or to a file via a wide range of SDB commands. Local machine state is managed via the following commands:

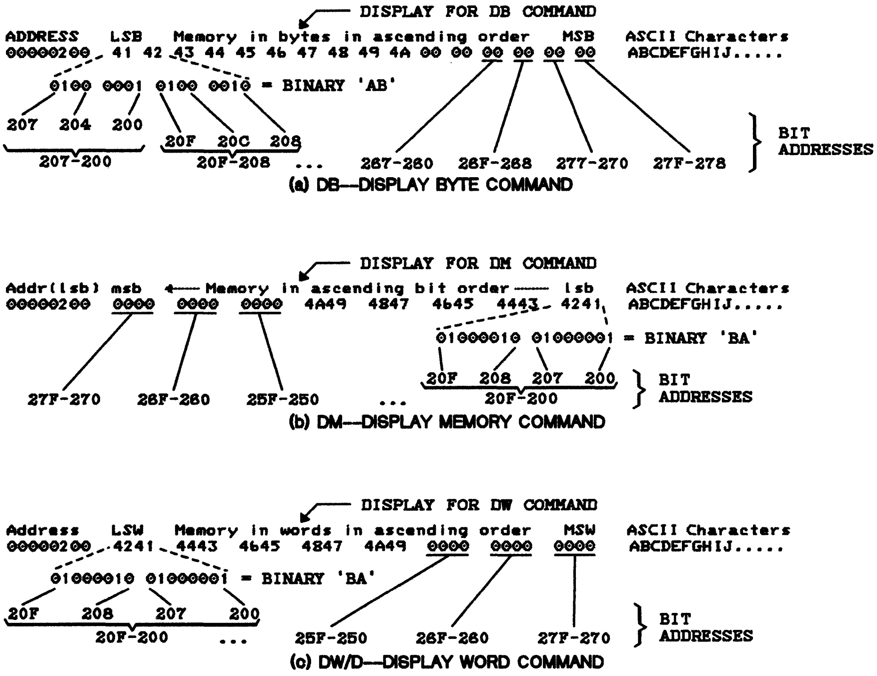
SIO	Save I/O registers
SR	Save registers
SMI val1 val2 {num}	Save memory image from val1 to val2 in file SMIFIL.num
SMS {num}	Save machine status to file SMSFIL.num
RIO	Restore I/O registers locally
RR	Restore registers locally
RMI {num {offset}}	Restore memory image from SMIFIL.num, optional address offset
RMS {num}	Restore machine status from SMSFIL.num
VMI {num {offset}}	Compare memory image with SMIFIL.num, optional address offset

4.5 Comparison of Displays for DB, DM, and DW (D) Commands

A choice of output formats is provided by three display-memory commands:

DB	Display memory in byte format
DM	Display memory
DW/D	Display memory in word format

A complete description of these is in the command section; each with a memory-display example. As shown below in &#differ, each command differs in its display. Note that the example memory used in the figure contains hex values 41 through 4A, the ASCII code for alpha characters A through J (also shown on the display right).



```
ADDRESS    LSB  Memory in bytes in ascending order   MSB   ASCII Characters
00000200  41 42 43 44 45 46 47 48 49 4A 00 00 00 00 00 00  ABCDEFGHIJ.....
```

(a) DB—DISPLAY BYTE COMMAND

```
Addr(lsb) msb  ←— Memory in ascending bit order —— lsb  ASCII Characters
00000200  0000 0000 0000 4A49 4847 4645 4443 4241  ABCDEFGHIJ.....
```

(b) DM—DISPLAY MEMORY COMMAND

```
Address   LSW   Memory in words in ascending order   MSW   ASCII Characters
00000200  4241  4443  4645  4847  4A49  0000  0000  0000   ABCDEFGHIJ.....
```

(c) DW/D—DISPLAY WORD COMMAND

Figure 4-4. Comparison of Memory Displays for DB, DM, DW Commands

Similarities of DB, DM, and DW/D displays:

- A hexadecimal eight-digit (32-bit word) address is on the left
- Memory values are in the center
- Recognizable ASCII characters are on the right (note that each example line in the figure is filled with ASCII values for letters A to J for reference purposes)

Comparison of Displays for DB, DM, and DW (D) Commands

Differences in displays are mainly with the data:

- DB command displays in eight-bit values:
  - Least significant byte (LSB) on the left,
  - MSB on the right.
  - Least significant address bit is the rightmost bit (least significant bit or LSb) of the first byte displayed (on the left).
- DM command displays in 16-bit words:
  - LSB on the right,
  - MSB on the left,
  - Least significant address bit is the rightmost bit in a line.
- DW or D command displays in 16-bit words:
  - LSW (least significant 16-bit word) on the left,
  - MSW on the left,
  - Least significant address bit is the rightmost bit of the LSW.

The descriptions depicted in Figure 4-4 allow you to decide the byte/word display that best suits your needs.

Note:

The memory displays can be cleared with the CLS (clear "scratch" area) command.

4.6 SDB Commands

The SDB commands can be divided into nine functional categories:

- Program Execution Commands
- Register, Machine-State Commands
- Register Field Manipulation Commands
- Status Register Field Manipulation Commands
- Memory Manipulation/Display Commands
- Cache Manipulation Commands
- Breakpoint and Trace Commands
- Debug Environment Control Commands
- Miscellaneous and Special Commands

Table 4-2 lists all the SDB commands, including syntaxes and operation descriptions, according to functional groups. Following Table 4-2, the SDB commands are described in alphabetical order. Table 4-1 describes the abbreviations and symbol definitions used in Table 4-2 and in the individual command descriptions.

Table 4-1. SDB Abbreviations and Symbol Definitions

SYMBOL	DEFINITION
< >	Angle brackets enclose a word or phrase that varies from execution to execution, and must be typed out. For example, <offset> indicates than an offset must be entered. The brackets themselves are not entered.
[ ]	Square brackets indicate one or more optional entries. The brackets themselves are not entered.
{ }	Braces contain a list of items, of which one **must** be chosen.
( )	Parentheses, when used within braces, contain an entire item in the list; they are used to show the logical grouping of a lengthy item.
<CR>	Press the carriage return. Note that some operating systems may require the ENTER key to be pressed instead.
<SP>	Press the SPACE key (bar).
Abbreviations	Command and option abbreviations are indicated by mixed use of uppercase and lowercase letters. For example, RUn means the run command can be entered as RUN or abbreviated RU.
Double-Word Value	32-bit value
MSB	Most significant byte
LSB	Least significant byte
MSb	Most significant bit
LSb	Least significant bit

Table 4-2. SDB Command Summary

PROGRAM EXECUTION COMMANDS	
COMMAND AND SYNTAX	OPERATION DESCRIPTION
CNT [command count]	Display command count
REset	Reset TMS34010
RUn [<instruction count>]	Run for specified no. of instructions
SS[F][U] [<instruction count>]	Single step for specified number of instructions, with or without Fast update and/or Unassembly options
REGISTER COMMANDS	
COMMAND AND SYNTAX	OPERATION DESCRIPTION
A	Display A and B File registers
A{0,..,14} [<double-word value>]	Display/modify an A File register
B	Display A and B File registers
B{0,..,15} [<double-word value>]	Display/modify a B File register
CLA	Clear A File registers
CLB	Clear B File registers
CLIO	Clear I/O registers
CLR	Clear both A and B File registers
CTL [<value>]	Display/modify I/O CONTROL register
DR	Toggle A/B & I/O registers
IO	Display I/O registers
IO{0,..,1F0} [<value>]	Modify specified I/O register
NR<register><name>	Give register a name
PC [<double-word value>]	Display/modify Program Counter
PM [<word value>]	Modify PMASK register
RIO	Restore temporary copy of I/O registers
RR	Restore temporary copy of registers
SIO	Save temporary copy of I/O registers
SP [<double-word value>]	Display/modify Stack Pointer
SR	Save temporary copy of registers
ST [{ ({N, C, Z, V} {0, 1}) , <double-word value>}]	Display/modify the status register or specified status bit

Table 4-2. SDB Command Summary (Continued)

REGISTER FIELD MANIPULATION COMMANDS	
COMMAND AND SYNTAX	**OPERATION DESCRIPTION**
CD [{0, 1}]	Modify cache disable bit
IE [{0, 1}]	Modify interrupt enable bit
PB{H,V} [{0, 1}]	Toggle PBH or PBV bit
PP [<pixel processing option>]	Set specified pixel processing option
PS [<pixel size>]	Set PSIZE register
T [{0, 1}]	Toggle transparency bit
W [{0, 1, 2, 3}]	Set or display specified windowing option
STATUS REGISTER FIELD MANIPULATION COMMANDS	
COMMAND AND SYNTAX	**OPERATION DESCRIPTION**
FE{0,1} {0, 1}	Modify specified field extension bit
FS{0,1} <field size>	Modify specified field size
ITPVH [<5-bit value>]	Display/modify ITPVH bits
NCZV [<4-bit value>]	Display/modify NCZV bits
PBX [{0, 1}]	Toggle PBX bit
ST [{ ({N, C, Z, V} {0, 1}) , <double-word value>}]	Display/modify the status register or specified status bit
MEMORY MANIPULATION/DISPLAY COMMANDS	
COMMAND AND SYNTAX	**OPERATION DESCRIPTION**
CIF	Close input file
CTF	Close trace file
DB <start addr> [<end addr>]	Display bytes
DM <start addr> [<end addr>]	Display memory
D[W] <start addr> [<end addr>]	Display word of memory
F <start addr> <end addr> <word value>	Fill memory with word value
FW <start adr> <end adr> <wrd val>	Find or display memory word
MM <adr>[<word>,<assm st>]	Display or modify memory, word align
MMF <addr> <field value> <field size>	Modify memory field, no word align
RMI [<file no. ext> [<offset>]]	Restore memory image
SMI<s addr> <e addr> [<file no. ext>]	Save memory image
U [<start addr> [<end addr>]]	Unassemble specified range
V <value>	Evaluate data
VMI [<file no. ext> [<offset>]]	Compare memory & disk images
CACHE MANIPULATION COMMAND	
COMMAND AND SYNTAX	**OPERATION DESCRIPTION**
CF [{0, 1}]	Display/modify cache flush bit

Table 4-2. SDB Command Summary (Continued)

BREAKPOINT AND TRACE COMMANDS	
COMMAND AND SYNTAX	**OPERATION DESCRIPTION**
CTF	Close trace file
BP	Display existing breakpoints
BP{0,..,19,X} {Clear, OF, ON, Togl, Quit}	Modify existing breakpoints
BPAI {<address>}	Set breakpoint on address
TR	Display existing traces
TR{0,..,19,X} [{Clear, OFf, ON, Toggle, Quit}]	Modify existing traces
TRAI <address>	Set trace on address
DEBUG ENVIRONMENT CONTROL COMMANDS	
COMMAND AND SYNTAX	**OPERATION DESCRIPTION**
CNT [command count]	Display command count
RDE[file number extension]	Restore debug environment
SDE[file number extension]	Save debug environment
MISCELLANEOUS AND SPECIAL COMMANDS	
COMMAND AND SYNTAX	**OPERATION DESCRIPTION**
CLS	Clear screen scratch area
CTF	Close trace file
HELP	Summary of commands
ID	Display SDB version number
L <filename> [<offset>]	Load COFF file
LE	Display last error messages
LH	Display last halt messages
LM	Display last monitor messages
Q [*] [C] [S]	Quit SDB session
RMS[<file no. extension>]	Restore machine state
SF<filename>	Show system file contents
SMS[<file no. extension>]	Save machine state
SWitch	Switch command input to a file
SY<command string>	Execute system function
U [<start addr> [<end addr>]]	Unassemble specified range

Display A & B File Registers

Syntax **A**

Description The A command displays the A and B file registers. If the A and B file registers are already displayed, then the A command clears and rewrites the display. This works the same as the B command.

Example Display the A and B file registers in the machine state display.

```
Command[1] A  <CR>
```

The registers are displayed (with values) as shown in Figure 4-1 ("SDB Debugger Screen Display") on page 4-5.

Modify/Display an A File Register

Syntax **A{0,..,14} [<double-word value>]**

Description The A# command (the # sign represents A-register number 0–14) allows you to modify or display the contents of the 15 A-file registers. This allows viewing the contents of an A-file register when the text display is off. Register number default is decimal. (To set or inspect the Stack Pointer, see the SP command. To change a register in the B file, see the B# command.)

If the 32-bit *<double-word value>* is specified, it replaces the value of the specified A file register. The default type for <double-word value> is hexadecimal.

Example 1 Modify the contents of register A3:

```
Command[1] A3 FFFFFFFE  <CR>
```

File register A3 now contains the value >FFFF FFFE. Note that you same could obtain the same result using the decimal type override, %-2.

Example 2 Display the contents of register A3:

```
Command[1] A3  <CR>                                  (entry)

Command[1] A3 = FFFFFFFE                             (response)
```

Now the contents of A file register A3 are visible in the command buffer. Note that this form of the command destroys any monitor commands that follow in the same buffer.

Display A & B File Registers

Syntax B

Description The B command causes the default register display to be the A and B file registers. If the A and B file registers are currently displayed, then the B command clears and rewrites the display. The B command works the same as the A command.

Example Display the A and B file registers in the machine state display:

```
Command[1] B  <CR>
```

The registers are displayed (with values) similar to Figure 4-1 ("SDB Debugger Screen Display") on 4-5.

B# Modify/Display a B File Register B#

Syntax **B{0,..,14} [<double-word value>]**

Description The B# command (the # sign represents a B-register number 0–14) allows you to modify or display the contents of any of the 15 B-file registers. This allows you to view the contents of a B-file register when the text display is off. Default for the register number is decimal. (To set or inspect the Stack Pointer, see the SP command.)

If the 32-bit *<double-word value>* is specified, then it replaces the value of the specified B file register. The default type for <double-word value> is hexadecimal.

Example 1 Modify the contents of register B13:

```
Command[1] B13 FFFFFFFF  <CR>
```

B file register B13 now contains the value >FFFF FFFF. Note that you could obtain the same result using the decimal type override, %-1.

Example 2 Display the contents of register B13:

```
Command[1]B13  <CR>  (entry)

Command[1] B13 = FFFFFFFF  (response)
```

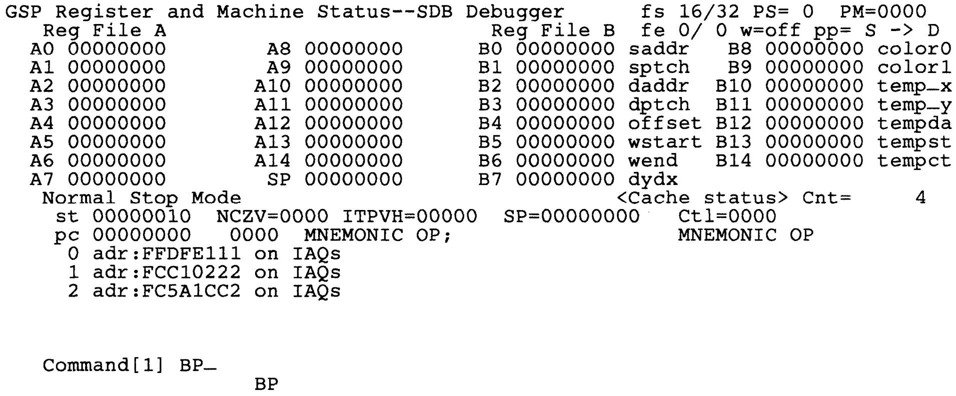
The contents of B file register B13 are now visible in the command buffer. Note that this form of the command destroys any monitor commands that follow in the same buffer.

Note:

The TMS34010 uses the reserved B file registers for temporary storage of intermediate parameters of the PIXBLT and FILL instructions. When the TMS34010 executes one of these instructions, it will not preserve values that you have stored in these registers. Be careful that these instructions do not destroy data that you have stored in the reserved B file registers.

Syntax **BP**

Description The BP command displays all existing breakpoints along with their active/inactive state. Figure 4-5 illustrates a typical display showing breakpoints 0, 1, and 2 (shown below the 'pc' display).



```
GSP Register and Machine Status--SDB Debugger      fs 16/32 PS= 0  PM=0000
   Reg File A                           Reg File B  fe 0/ 0 w=off pp= S -> D
  A0 00000000        A8 00000000      B0 00000000 saddr   B8 00000000 color0
  A1 00000000        A9 00000000      B1 00000000 sptch   B9 00000000 color1
  A2 00000000       A10 00000000      B2 00000000 daddr  B10 00000000 temp_x
  A3 00000000       A11 00000000      B3 00000000 dptch  B11 00000000 temp_y
  A4 00000000       A12 00000000      B4 00000000 offset B12 00000000 tempda
  A5 00000000       A13 00000000      B5 00000000 wstart B13 00000000 tempst
  A6 00000000       A14 00000000      B6 00000000 wend   B14 00000000 tempct
  A7 00000000        SP 00000000      B7 00000000 dydx
   Normal Stop Mode                              <Cache status> Cnt=     4
    st 00000010  NCZV=0000 ITPVH=00000  SP=00000000   Ctl=0000
    pc 00000000   0000  MNEMONIC OP;                  MNEMONIC OP
     0 adr:FFDFE111 on IAQs
     1 adr:FCC10222 on IAQs
     2 adr:FC5A1CC2 on IAQs


   Command[1] BP_
                    BP
```

Figure 4-5. Display Existing Breakpoints Monitor Display Format

Each breakpoint is assigned a reference number (0 to 19). A combined maximum of 20 breakpoints and traces can be defined at one time. The reference numbers shown here are those used in conjunction with the BP# command to manipulate the state of each breakpoint on the list.

BP# Modify Existing Breakpoints BP#

Syntax **BP{0,..,19,X} [{Clear, OFf, ON, Toggle, Quit}]**

Description The BP# command allows you to modify the status of individual breakpoints. The # symbol is a breakpoint reference number from 0–19 (register number as shown by BP command) or the letter X. A combination of up to 20 breakpoints and traces may be defined. If X is specified, then all existing breakpoints are affected. The breakpoint reference number is displayed when the breakpoint is defined, and does not change thoughout the life of the specific breakpoint.

The breakpoint options include:

Clear Destroys the breakpoint.

OFf Deactivates the breakpoint temporarily (but doesn't destroy it).

ON Reactivates a breakpoint that has been turned off.

Toggle Activates a deactivated breakpoint, or deactivates an activated breakpoint. An asterisk (*) next to the number in a breakpoint list indicates *deactivated*.

Quit Terminates the command without changing any breakpoints.

Only the significant letters of each option are processed, as indicated by the uppercase letters in the list (e.g., CLEAR and C are treated the same).

If you do not enter the option as part of the command, the SDB will display the breakpoint and a list of options to select.

Example 1 Toggle breakpoint 3:

```
Command[1] BP3 TOGGLE<CR>
```

or

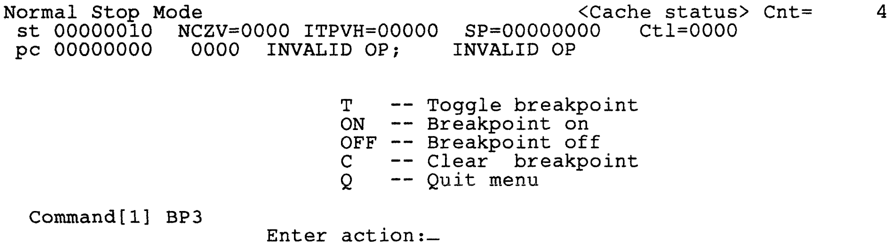
```
Command[1] BP3 T<CR>
```

(shortened version)

Example 2 Enter a breakpoint command without an option:

```
Command[1] BP3  <CR>
```

The SDB will display the breakpoint as follows (Figure 4-6):



```
Normal Stop Mode                              <Cache status> Cnt=      4
 st 00000010  NCZV=0000 ITPVH=00000  SP=00000000   Ctl=0000
 pc 00000000   0000  INVALID OP;    INVALID OP

                           T   -- Toggle breakpoint
                           ON  -- Breakpoint on
                           OFF -- Breakpoint off
                           C   -- Clear  breakpoint
                           Q   -- Quit menu

  Command[1] BP3
                       Enter action:_
```

Figure 4-6. Modify Breakpoints Menu

Now you can enter **T** to toggle the breakpoint (or enter any of the other four options). A **T** toggles the state of breakpoint number 3 to off. Note that breakpoint 3 remains in memory and may be reactivated by the same command sequence or by specifying the **ON** option. Alternatively, it may be deleted with the **CLEAR** option and then overwritten by the BPAI command. You can verify the modification with the BP command.

Example 3 Clear all breakpoints:

```
Command[1] BPX CLEAR
```

BPAI Set Breakpoint on Instruction Address BPAI

Syntax **BPAI <address>**

Description The BPAI command allows you to set breakpoints, causing execution to stop when a specific address is accessed during instruction acquisition.

The <address> is hexadecimal by default. A combination of up to 20 breakpoints or traces may be set.

Example Break execution when SDB attempts to fetch an instruction from location >120F F310.

```
Command[1] BPAI 120FF310<CR>
```

Modify Cache Disable Bit

Syntax CD [{0, 1}]

Description This command allows you to set, reset, or toggle the contents of the CD (cache disable) bit in the I/O Memory Control register (bit 15 or the MSb on the left -- see register figure below). If a 0 or 1 value is not specified, then the CD bit is toggled; otherwise, the CD bit is set to the value. Notice command execution by checking the MSb of the "Ctl=xxxx" field in the SDB status display.

Example Set the CD bit, disabling the cache:

```
Command[1] CD 1  <CR>
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CD	PPOP					PBV	PBH	W		T	RR		RM	reserved	

CD = instruction cache disable
PPOP = pixel proc. operation sel.
PBV = PixBlt vertical direction
PBH = PixBlt horiz. direction

W = window violation
T = pixel transparency enable
RR = DRAM refresh rate
RM = DRAM refresh mode

I/O Memory Control Register

CF Cache Flush/Modify Cache Flush Bit CF

Syntax **CF [{0, 1}]**

Description The CF command allows you to set or reset the I/O HSTCTLH-Register's CF bit (cache flush, bit 14 -- see register figure below) to a 0 or 1:

CF BIT

0 Cache reads are enabled depending upon value of the cache disable (CD) bit.

1 All current data in the cache is invalidated, and accesses to the cache is inhibited until the CF bit is set to 0. The cache fragment present flags are also cleared.

If 0 or 1 is not specified, then the SDB flushes the cache by setting all of the present flags to "not present" and zeroes out the cache contents. The CF bit will not be affected.

Example Clear the CF bit in the HSTCTLH Register, enabling cache access:

```
Command[1] CF 0 <CR>
```

(You can monitor bit change by first issuing the IO command to display the HSTCTLH I/O Register. Observe bit 14.)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HLT	CF	LBL	INCW	INCR	Res	NMIM	NMI	Reserved							

HLT = Halt processing
CF = Cache flush
LBL = Lower byte last

INCR = Increment on read
INCW = Increment on write
NMIM = NMI mode bit
NMI = Nonmaskable interrupt

I/O HSTCTLH Register (Host Interface Control, High Byte)

Syntax **CIF**

Description The CIF command closes the opened input file GSPINPUT.000. Then execution can be restarted with a SWITCH command.

> **Note:** If the CIF command is executed within a command batch stream (e.g., by using the SWITCH command), the input file will be closed and execution automatically begins at the beginnning of the input batch file (i.e., loops continuously in batch stream).

Example Close the input file:

```
Command[1] CIF <CR>
```

Syntax **CLA**

Description The CLA command clears (zeroes) all the A file registers except the Stack Pointer (SP 0 clears Stack Pointer).

Example Clear registers A0–A14 (SP is not changed).

```
Command[1] CLA  <CR>
```

CLB Clear B File Registers CLB

Syntax **CLB**

Description The CLB command clears (zeroes) all the B file registers except the Stack Pointer (SP 0 clears Stack Pointer).

Example Clear registers B0–B14 (SP is not changed).

```
Command[1] CLB  <CR>
```

Syntax **CLIO**

Description The CLIO command clears (zeroes) the on-chip I/O Registers except for those registers with offset values from >00C0 to >0100 and >01C0 to >01F0 (registers and their offsets are listed in Table 4-3 on page 4-58).

Example Clear only the on-chip I/O Registers. (To view this modification, first issue the IO command to display the IO Registers.)

```
Command[1] CLIO  <CR>
```

CLR Clear Both the A and B File Registers CLR

Syntax **CLR**

Description The CLR command clears:

- The A file registers,
- The B file registers, and
- The Stack Pointer.

Example Clear registers A0–A14, B0–B14, and SP:

```
Command[1] CLR <CR>
```

CLS Clear Scratch Area of Display CLS

Syntax CLS

Description The CLS command clears any messages below the standard register display and above the Command line ("scratch" display area).

Example Clear "scratch" area:

```
Command[1] CLS <CR>
```

Syntax CNT

Description The CNT command displays and modifies the value of the command count -- a count of commands executed since being set to zero. This value is also displayed after "Cnt = " in the middle right side of the screen display. Decimal is the default value.

Example 1 Set the command count to 100 (decimal):

```
Command[1] CNT 100<CR>
```

Example 2 Display the command count:

```
Command[1] CNT<CR>
Command[1] CNT =         100
```

CTF Close Trace File CTF

Syntax **CTF**

Description The CTF command closes the opened trace file GSPTRACE.000. This allows inspecting the file with a Show File (SF) command without exiting the SDB.

Example Close the trace file:

```
Command[1] CTF <CR>
```

CTL Display/Modify CONTROL I/O Register CTL

Syntax **CTL [<value>]**

Description The CTL command allows you to modify the contents of the CONTROL (I/O MEMORY CONTROL) register by specifying a 16-bit replacement value, <value>. The default type for <value> is hexadecimal.

CTL also allows you to display the contents of the CONTROL register from the command line by executing the command without a value. This is useful for viewing the contents of the CONTROL register while the text display is off.

Example 1 Modify the contents of the CONTROL register:

```
Command[1] CTL 1046  <CR>
```

Example 2 Display the contents of the CONTROL register:

```
Command[1] CTL<CR>
Command[1] CTL =  1046
```

The contents of the CONTROL register are now visible in the command buffer.

> **Note:**
>
> Using the CTL command as shown in Example 2 **will destroy** any other commands that follow in the same buffer.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CD	PPOP					PBV	PBH	W		T	RR		RM	reserved	

CD = inst. cache disable
PPOP = pixel proc. operation select
PBV = PixBlt vertical direction
PBH = PixBlt horiz. direction

W = window violation
T = pixel transparency enable
RR = DRAM refresh rate
RM = DRAM refresh mode

I/O Memory Control Register

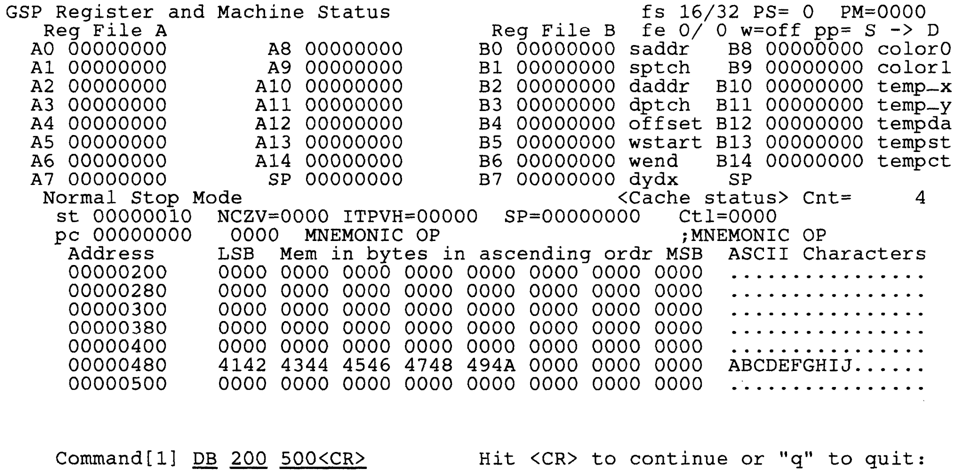
Syntax **DB <start address> [<stop address>]**

Description The DB command displays blocks of TMS34010 memory. The start and stop addresses are expressed in 32-bit hexadecimal mode by using the associated convention (see the beginning of this section).

Example Display a block of memory from address >0200 to >0550:

```
Command[1] DB 00200 550
```

Although the address specified is a bit address, any bit address portion supplied is ignored, and the data is specified in bytes starting on word boundaries. The resulting display is shown in Figure 4-7 in the default display mode. The memory display can be cleared with the CLS command.



```
GSP Register and Machine Status                         fs 16/32 PS= 0  PM=0000
   Reg File A                              Reg File B  fe 0/ 0 w=off pp= S -> D
  A0 00000000        A8 00000000         B0 00000000 saddr    B8 00000000 color0
  A1 00000000        A9 00000000         B1 00000000 sptch    B9 00000000 color1
  A2 00000000       A10 00000000         B2 00000000 daddr   B10 00000000 temp_x
  A3 00000000       A11 00000000         B3 00000000 dptch   B11 00000000 temp_y
  A4 00000000       A12 00000000         B4 00000000 offset B12 00000000 tempda
  A5 00000000       A13 00000000         B5 00000000 wstart B13 00000000 tempst
  A6 00000000       A14 00000000         B6 00000000 wend    B14 00000000 tempct
  A7 00000000        SP 00000000         B7 00000000 dydx        SP
   Normal Stop Mode                                      <Cache status> Cnt=     4
    st 00000010  NCZV=0000 ITPVH=00000  SP=00000000    Ctl=0000
    pc 00000000   0000  MNEMONIC OP                       ;MNEMONIC OP
     Address      LSB  Mem in bytes in ascending ordr MSB  ASCII Characters
     00000200     0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000280     0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000300     0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000380     0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000400     0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000480     4142 4344 4546 4748 494A 0000 0000 0000  ABCDEFGHIJ......
     00000500     0000 0000 0000 0000 0000 0000 0000 0000  ................


    Command[1] DB 200 500<CR>          Hit <CR> to continue or "q" to quit:
```

Figure 4-7. Display Memory Monitor Display Format

If more than nine lines of display are requested, the display halts; a carriage return continues the display. If a carriage return is entered (in lieu of a Q key entry), the display memory portion of the display is cleared, and the new set of information is routed to the top of the scratch-display area. (Note: Enter "Q" **only** if the "Hit <CR> to continue or "q" to quit:" message is displayed in the bottom of the above example. Do not confuse with Q command.)

Note that the start address must be less than the stop address, or the SDB issues an error message.

Note:

The DB, DM, and DW (D) commands display memory in unique ways. Section 4.5 and Figure 4-4 on 4-20 describe the differences.

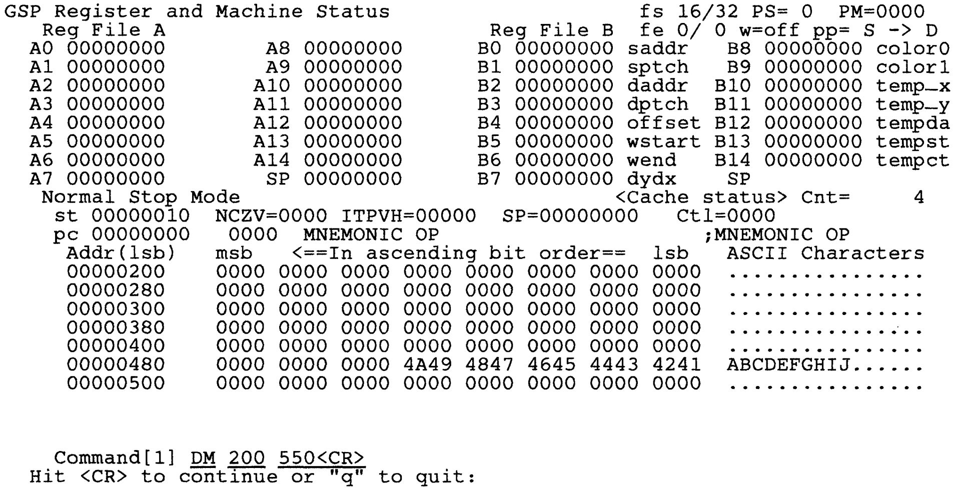
Syntax **DM <start address> [<stop address>]**

Description The DM command displays blocks of TMS34010 memory. The start and stop addresses are expressed in 32-bit hexadecimal.

Example Display a block of memory from address >0200 to address >0550.

```
Command[1] dm 00200 550 <CR>.
```

Although the address specified is a bit address, any bit address portion supplied is ignored, and the data is specified in words. The resulting display is shown in Figure 4-8 in the default display mode. The memory display can be cleared with the CLS command.



```
GSP Register and Machine Status                          fs 16/32 PS= 0  PM=0000
   Reg File A                                   Reg File B  fe 0/ 0 w=off pp= S -> D
  A0 00000000         A8 00000000     B0 00000000 saddr   B8 00000000 color0
  A1 00000000         A9 00000000     B1 00000000 sptch   B9 00000000 color1
  A2 00000000        A10 00000000     B2 00000000 daddr  B10 00000000 temp_x
  A3 00000000        A11 00000000     B3 00000000 dptch  B11 00000000 temp_y
  A4 00000000        A12 00000000     B4 00000000 offset B12 00000000 tempda
  A5 00000000        A13 00000000     B5 00000000 wstart B13 00000000 tempst
  A6 00000000        A14 00000000     B6 00000000 wend   B14 00000000 tempct
  A7 00000000         SP 00000000     B7 00000000 dydx    SP
   Normal Stop Mode                                   <Cache status> Cnt=     4
    st 00000010  NCZV=0000 ITPVH=00000  SP=00000000   Ctl=0000
    pc 00000000   0000  MNEMONIC OP                       ;MNEMONIC OP
     Addr(lsb)   msb   <==In ascending bit order==  lsb   ASCII Characters
     00000200    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000280    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000300    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000380    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000400    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000480    0000 0000 0000 4A49 4847 4645 4443 4241  ABCDEFGHIJ......
     00000500    0000 0000 0000 0000 0000 0000 0000 0000  ................


    Command[1] DM 200 550<CR>
  Hit <CR> to continue or "q" to quit:
```

Figure 4-8. Display Memory Monitor Display Format

If more than nine lines of display are requested, the display is halted waiting for a carriage return entry to continue display. If a carriage return is entered (in lieu of a Q key entry), the display memory portion of the display is cleared, and the new set of information is routed to the top of the scratch-display area. (Note: Enter "Q" **only** if the "Hit <CR> ... "q" to quit:" message is below command line as in above display. Do not confuse with Q command.)

Note that the start address must be less than the stop address for the DM command to operate, or the SDB issues an error message.

> **Note:**
>
> The DB, DM, and DW (D) commands display memory in unique ways. Section 4.5 and Figure 4-4 on page 4-20 describe the differences.

Syntax DR

Description The DR command toggles the display between the I/O registers and the A and B file registers (even if the IO command has not been executed). (Successive <LINE FEED>, CTRL-J, or <CR> entries execute the complete entry at the command line, thus toggling the display.)

Example Display the A and B file registers in the machine state display:

```
Command[1] DR  <CR>
```

Syntax **D[W] <start address> [<stop address>]**

Description This command displays blocks of TMS34010 memory. The start and stop addresses are expressed in 32-bit hexadecimal.

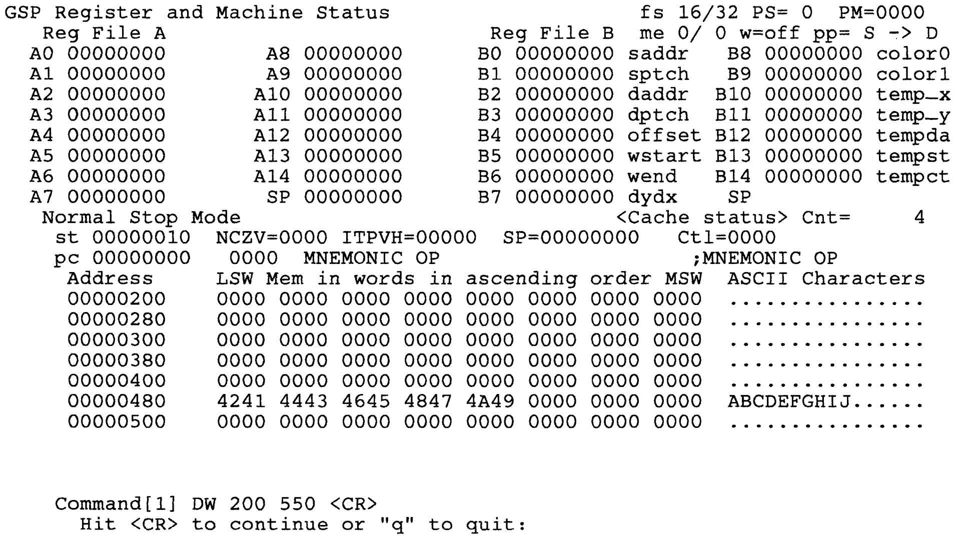
Example Display a block of memory from address >0200 to >0500:

```
Command[1] D 00200 550 <CR>.
```

or

```
Command[1] DW 00200 550 <CR>.
```

Although the addresses are specified as bit addresses, any bit address portion supplied is ignored, and the data is specified in words. The resulting display is shown in Figure 4-9 in the default display mode. The memory display can be cleared with the CLS command.



```
GSP Register and Machine Status                          fs 16/32 PS= 0  PM=0000
   Reg File A                              Reg File B  me 0/  0 w=off pp= S -> D
  A0 00000000          A8 00000000        B0 00000000 saddr   B8 00000000 color0
  A1 00000000          A9 00000000        B1 00000000 sptch   B9 00000000 color1
  A2 00000000         A10 00000000        B2 00000000 daddr  B10 00000000 temp_x
  A3 00000000         A11 00000000        B3 00000000 dptch  B11 00000000 temp_y
  A4 00000000         A12 00000000        B4 00000000 offset B12 00000000 tempda
  A5 00000000         A13 00000000        B5 00000000 wstart B13 00000000 tempst
  A6 00000000         A14 00000000        B6 00000000 wend   B14 00000000 tempct
  A7 00000000          SP 00000000        B7 00000000 dydx    SP
   Normal Stop Mode                                     <Cache status> Cnt=     4
    st 00000010  NCZV=0000 ITPVH=00000  SP=00000000   Ctl=0000
    pc 00000000   0000  MNEMONIC OP                        ;MNEMONIC OP
     Address     LSW Mem in words in ascending order MSW  ASCII Characters
     00000200    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000280    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000300    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000380    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000400    0000 0000 0000 0000 0000 0000 0000 0000  ................
     00000480    4241 4443 4645 4847 4A49 0000 0000 0000  ABCDEFGHIJ......
     00000500    0000 0000 0000 0000 0000 0000 0000 0000  ................


    Command[1] DW 200 550 <CR>
      Hit <CR> to continue or "q" to quit:
```

Figure 4-9. Display Word Monitor Display Format

If more than nine lines of display are requested, the display halts, and waits for you to enter a carriage return to continue the display. If a carriage return is entered (in lieu of a Q key entry), a new set of information is routed to the top of the scratch-display area (after first clearing the memory portion of the display). (Note: Enter "Q" **only** if the "Hit <CR> ... "q" to quit:" message is below command line as in above display. Do not confuse with Q command.)

Note that the start address must be less than the stop address, or the command line returns with no action.

Note:

The DB, DM, and DW (D) commands display memory in unique ways. Section 4.5 and Figure 4-4 on page 4-20 describe the differences.

F Fill Memory with Word F

Syntax **F <start address> <end address> <word value>**

Description The F command fills a block of memory from <start address> to <end address> with <word value>.

- The start address must be less than the stop address.
- The default type for both is hexadecimal.
- The fill value will be treated as a 16-bit value.
- The bit address of both the start and stop addresses are truncated to form a word address (right four bits set to zero).
- The F command may be used to fill screen memory, program memory, or both, as the SDB does not distinguish between the two.

Example Fill memory from >200 to >350, inclusive, with >00AA:

```
Command[1] F 200 350 AA <CR>
              F 200 350 AA
```

You can now check for >AA's in the filled area with the command:

```
Command[1] DM 200 350<CR>
             DM 200 350
```

Syntax **FE {FE bit designator} <SPace> {bit value}**

Description The FE command sets the FE0 or FE1 bits in the Status Register. The first parameter identifes which bit (0 or 1). It is separated from the second parameter with a space (<SP>). The second parameter is the desired bit value:

Bit Value

0	for zero extend, or
1	for sign extend.

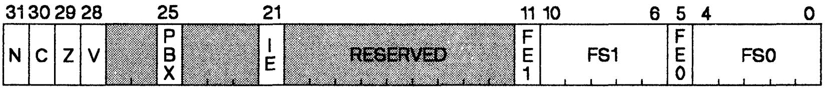
These can be observed in the "fe 0/0" field above the B Register display.

Example 1 Set the FE0 to 1, enabling sign extension:

```
Command[1] FE 0  1  <CR>
```

Example 2 Modify the contents of the FE1 bit to enable zero extension:

```
Command[1] FE1 0  <CR>
```



N = Negative
C = Carry
Z = Zero
V = Overflow
PBX = PixBlt executing

IE = Interrupt Enable
FE1 = Field extend 1
FS1 = Field size 1
FE0 = Field extend 0
FS0 = Field size 0

Status Register

Modify Field Size (FS)

Syntax **FS {FS bit designator} <field size>**

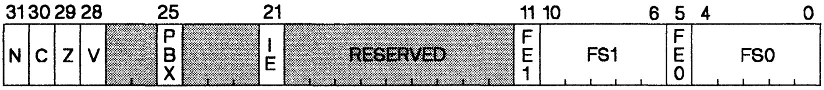
Description The FS0 and FS1 commands allow you to modify the value of the corresponding 5-bit field in the Status Register. A <field size> value (decimal by default) of decimal 1 to 32 can be specified. The current field size can be seen in the "fs aa/bb" field above the display's B Registers.

Example 1 Change the field size of FS0 to 15:

```
Command[1] FS0 15 <CR>
```

Example 2 Change the field size of FS1 to 30:

```
Command[1] FS1 30 <CR>
```



N = Negative
C = Carry
Z = Zero
V = Overflow
PBX = PixBlt executing
IE = Interrupt Enable
FE1 = Field extend 1
FS1 = Field size 1
FE0 = Field extend 0
FS0 = Field size 0

Status Register

Syntax **FW <start address> <end address> <word value>**

Description The FW command locates a specific 16-bit <word value> within a defined two-address boundary. It prints out the value in hexadecimal/decimal format along with the address where found. If no printout occurs, <word value> was not found.

Both <addresses> are 32-bits, in hexadecimal by default. If <address> is not word aligned, it is forced to word alignment by its four lower bits being set to zero.

The <word value> parameter is 16-bit, hexadecimal by default.

Example Use the command to find the value >F0:

```
Command[1] FW F000 F800 F0<CR>
    00F0/240 found at 0000F6A0  Hit <CR> to continue or "q" to quit:
```

Syntax **Help**

Description The HELP command displays a menu of help files.

Example Display the help menu:

```
Command[1] HELP
```

The SDB displays a menu that describes the utility and lists help files for the various classes of commands.

When you select a help file, the SDB displays the file in the same manner as the SF (Show File) command. If the file is not there, then the SDB will inform you.

```
                          TMS34010 Debugger HELP Function

B -- Breakpoint/trace commands            P -- Program execution commands
E -- Environment save/restore commands    R -- Register/status display/modify
G -- *Graphics customization commands     S -- System specific commands
I -- *Interrupt/host interface commands   O -- Debugger overview
M -- Memory manipulation commands         Q -- Quit help function
```

***NOTE: The G and I choices are not used with the SDB.**

Figure 4-10. SDB Help Utility Menu

ID Identify SDB Version Number ID

Syntax ID

Description The ID command prints the version number of the SDB below the command line.

Example Display the SDB version number:

```
Command[1]       ID
   Version 1.20021986
```

Syntax **IE [{0, 1}]**

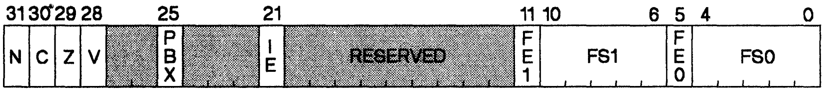
Description The IE command allows you to set, reset, or toggle (set 1 to 0, vice versa) the contents of the IE (interrupt enable) bit in the status register. This level is shown in the"I" bit of the ITPVH field in the center of the display.

If a 0 or 1 value is not specified, the IE bit is toggled; otherwise, the bit is set to the value specified.

Example Set the IE bit:

```
Command[1] IE 1 <CR>
```

The IE bit is set to 1, disabling interrupts.



N = Negative
C = Carry
Z = Zero
V = Overflow
PBX = PixBlt executing

IE = Interrupt Enable
FE1 = Field extend 1
FS1 = Field size 1
FE0 = Field extend 0
FS0 = Field size 0

Status Register

Syntax **IO**

Description The IO command displays the I/O Registers in the top half of the machine state display. Table 4-3 on page 4-58 lists I/O Registers and their offsets.

Example Display the I/O registers:

```
Command[1] IO <CR>
```

Syntax **IO{0,..,1F0}** [**<value>**]

Description The IO# command allows you to inspect or change the contents of any of the memory-mapped I/O Registers. Registers are specified by the register's offset <value> (hexadecimal is default).

Simply specify the offset <value> from the I/O Register base address (>C000 0000) to the desired register. (These registers are listed on the next page in Table 4-3.) Inspect the contents of a particular I/O register by specifying the offset {0 to 1F0} without a replacement value.

> **Note:**
>
> Do not change the contents of the five I/O Registers at offsets from >00C0 to >0100. Doing so can cause the system to fail.

Table 4-3. I/O Register Offsets

Offset†	Register	Description	Offset†	Register	Description
000	HESYNC	Horizontal end sync	100	HSTCTLH	Host control high
010	HEBLNK	Horizontal end blank	110	INTENB	Interrupt enable
020	HSBLNK	Horizontal start blank	120	INTPEND	Interrupt pending
030	HTOTAL	Horizontal end total	130	CONVSP	Source pitch
040	VESYNC	Vertical end sync	140	CONVDP	Destination pitch
050	VEBLNK	Vertical end blank	150	PSIZE	Pixel size
060	VSBLNK	Vertical start blank	160	PMASK	Plane mask
070	VTOTAL	Vertical total	170	–	Reserved
080	DPYCTL	Display control	180	–	Reserved
090	DPYSTRT	Display start	190	–	Reserved
0A0	DPYINT	Display interrupt	1A0	–	Reserved
0B0	CONTROL	Control	1B0	DPYTAP	Display tap address
0C0	HSTDATA	Host data	1C0	HCOUNT	Horizontal count
0D0	HSTADRL	Host address low	1D0	VCOUNT	Vertical count
0E0	HSTADRH	Host address high	1E0	DPYADR	Display address
0F0	HSTCTRL	Host control low	1F0	REFCNT	DRAM refresh count

† The offset is added to the base address of >C000 0000.

Example 1 Set to >F046 the contents of the I/O register located at address >C000 0070 (VTOTAL Register):

```
Command[1] IO 70 F046 <CR>.
```

The I/O VTOTAL register at >C000 0070 now contains >F046.

Example 2 Inspect the contents of the same register:

```
Command[1] IO 70<CR>
```

```
Command[1] IO 70 = F046     (command response)
```

Syntax **ITPVH [<5-bit value>]**

Description The ITPVH command displays the values in bits ITPVH of the Status Register and I/O Control Register (defined in list below). To set one or more of the bits, an entire 5-bit value must be entered (only zeroes and ones accepted). Entering the command without any bit values causes the present contents to be displayed.

Meaning of each bit (ST = Status Register, CR = I/O Control Register):

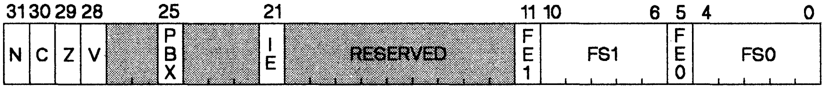
I = Interrupt Enable (ST bit 21)
T = Pixel Transparency Enable (CR bit 5)
P = PixBlt Executing/Interrupt (ST bit 25)
V = PixBlt Vertical Dir. Control (CR bit 9)
H = PixBlt Horizontal Dir. Control (CR bit 8)

Example 1 Reset the value of all ITPVH bits:

```
Command[1] ITPVH 00000<CR>
```

Example 2 List values of all ITPVH bits:

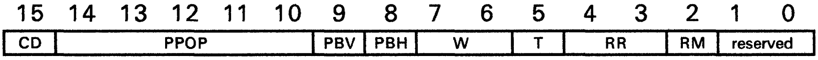
```
Command[1] ITPVH<CR>
Command[1] ITPVH = 00000
```



N = Negative
C = Carry
Z = Zero
V = Overflow
PBX = PixBlt executing

IE = Interrupt Enable
FE1 = Field extend 1
FS1 = Field size 1
FE0 = Field extend 0
FS0 = Field size 0

Status Register



CD = inst. cache disable
PPOP = pixel proc. operation sel.
PBV = PixBlt vertical direction
PBH = PixBlt horiz. direction

W = window violation
T = pixel transparency enable
RR = DRAM refresh rate
RM = DRAM refresh mode

I/O Memory Control Register

L Load COFF File L

Syntax **L <filename> [<offset>]**

Description The L command downloads a COFF module produced by GSPA or the linker into the SDB local memory so that it can be executed. The COFF module may be optionally relocated at load time. Note that the SDB cache is flushed on a successful download so that old code in cache will not be executed.

The <filename> is the COFF module that will be downloaded. It may be specified without an extension; .OUT is the default extension for modules produced by the linker, and .OBJ is the default extension for modules produced by GSPA. If no extension is specified for <filename>, the SDB first attempts to load <filename>. If this file isn't found, the SDB attempts to load <filename>.OUT. Unlinked <filename>.OBJ code may be downloaded, but a warning will be issued and unresolved references will not be resolved.

If the optional <offset> is specified, then the SDB will attempt to relocate the COFF module when downloading by adding <offset> to all relocation entries in the COFF module. The <offset> is treated as a signed 32 bit quantity. If you attempt to download an absolute (non-relocatable) COFF module, the SDB issues a warning and ignores the offset. If the offset is not specified, then all relocation entries are loaded relative to zero.

Example 1 Download COFF module CODE.OUT with offsets of 0 (first example) and >100 0D00 (second example):

```
Command[1] L CODE <CR>.

Command[1] L CODE.OUT 1000D00 <CR>.
```

The second example above causes the file CODE.OUT to be opened and its contents to be read, interpreted, and downloaded into the SDB memory with an offset of >100 0D00.

Example 2 The following is an example of loading a COFF file from drive C by specifying the drive and directory \LASER\OUT:

```
Command[1] L C:\LASER\OUT\CODE <CR>.
```

LE Display Last Errors LE

Syntax LE

Description The LE command lets you view the most recent set of error messages after they have been removed from the screen. Error messages appear in red in the scratch display area of the SDB display.

Example Display previous error messages:

```
Command[1] LE  <CR>
```

Display Last Halts

Syntax **LH**

Description The LH command allows you to view the most recent set of halt messages after they have been removed from the screen. Halt messages are generated by encountering breakpoints. They appear in cyan (light blue) in the scratch display area of the SDB display.

Example Display previous halt messages:

```
Command[1] LH  <CR>
```

Syntax **LM**

Description The LM command allows you to view the most recent set of monitor error messages after they have been removed from the screen. Monitor error messages appear in yellow in the monitor message display area beneath the command entry line.

Example Display previous monitor messages:

```
Command[1] LM  <CR>
```

MM Modify (or Evaluate) Memory -- Word Align MM

Syntax **MM <address> [<single or dbl word value> <assembler stmt>]**

Description The MM command displays and modifies memory.

The <address> parameter is:

- a 32-bit bit address, hexadecimal by default.
- if not word aligned, then <address> is forced to be word aligned by having its lower four bits set to zero.

The area at <address> can be filled with one of the optional values:

- a 16-bit <word value>, hexadecimal by default,
- a 32-bit <double-word value>, hexadecimal by default, or
- a line of TMS34010 <assembler code>.

The default for values specified in <assembler stmt> is hexadecimal except for the TRAP, SETF, and K instructions.

If the optional parameters <word value> or <assembler stmt> are not specified, then the command displays:

- <address> in hexadecimal, decimal, and as an XY and linear address,
- contents at <address> in hexadecimal, decimal, and as disassembled source,
- <address> in hexadecimal and decimal.
- **NOTE** that this form of the command overwrites any trailing commands remaining on the command line; these remaining commands **will not be executed.**

Example 1 Use the command to report on a memory location:

```
Command[1] MM FF8  <CR>
```

This produces the following display, assuming that memory location >FF0 contains >2980, CONVSP is >15, CONVDP is >16, PSIZE = 4, and OFFSET = 0:

```
            in Hex   Decimal           source   destination
  Address:  00000FF8    4088     Y,X:  0003,00FE  0007,007E
                              Linear:  00003FE0   00003FE0
@00000FF0:      2980   10624     ASM:  SRA 20,A0
     Data:      0FF8    4088
```

To check the contents of <address>:

```
Command[1] MM FF8 <CR>                    (enter)

Command[1] MM FF8 = 2980                  (response)
             MM FF8 = 2980
```

This form of the command overwrites any trailing commands on the command line; trailing commands will not be executed. You can use this command to find equivalent linear addresses from XY addresses, although the V command is also provided for this purpose.

Example 2 Use the MM command to modify a memory location:

```
Command[1] MM FF8 FEC4<CR>                (word value)
Command[1] MM FF8 1FEC4<CR>               (double-word value)
Command[1] MM FF8 MOVE A0,B9<CR>          (assembler code)
```

Each of these examples changes the value of the word or words starting at address >FF8 to the type of value on the right. Note that <word value> or <double-word value> is specified indirectly by the number of hexadecimal digits required to hold the result:

- FEC4 is a word value
- 1FEC4 is a double-word value.

A hexadecimal value can be forced to double-word value by including leading zeros. The value 0FEC4 is a double-word value. Values specified with a decimal format override will take up as much space as required to hold the hexadecimal equivalent, but leading zeros are not taken into account. For negative numbers, the space is calculated for the positive equivalent. Thus, -1 is a word value.

Specifying a line of assembler code will modify as many words as it takes for the opcode and its operands to be placed in memory. This can be as many as five words. All values in the assembler code specification must be numeric as opposed to symbolic. For address-relative instructions, the value is specified as the address. The line assembler will calculate the relative offset for you. Except for the requirement that values cannot be symbolic, the syntax of assembly code for the line assembler is the same as described in the assembly language section.

MMF Modify Memory Field -- No Word Align. MMF

Syntax **MMF <address> <field value> <field size>**

Description The MMF command modifies memory -- but not necessarily on word-aligned boundaries -- using a specified field size.

- The <address> is specified by either a 32-bit bit address or an XY address. The default format of <address> is hexadecimal.
- The <field value> is a field of one to 32 bits. Default format is hexadecimal.
- The <field size> is a decimal value (default) from 1 to 32 bits.

Example Modify a memory field:

```
Command[1] MMF FF8 F 4 <CR>
Command[1] MMF FF8 %100 7 <CR>  (decimal format)
Command[1] MMF FF8 1 1 <CR>
```

Each of these examples changes the value of the field starting at address >FF8 to the value following >FF8. If the <field value> is larger than can be contained in <field size>, then the low order bits up to the field size will be inserted. That is, the LSbs (bits) of the value will be placed into the field in memory.

NCZV Display/Modify ST NCZV Bits NCZV

Syntax **ITPVH [<5-bit value>]**

Description The NCZV command displays the values in bits ITPVH of the Status Register (defined in list below). To set one or more of the bits, an entire 4-bit value must be entered (only zeroes and ones accepted). Entering the command without any bit values causes the present contents to be displayed.

The instruction executed dictates meaning of each bit (ST = Status Register):

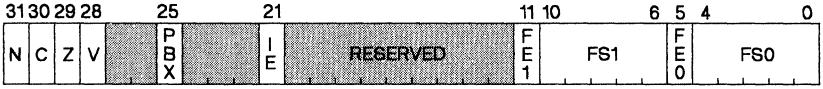
N = Negative (ST bit 31)
C = Carry (ST bit 30)
Z = Zero (ST bit 29)
V = Overflow (ST bit 28)

Example 1 Of the ST NCZV bits, set Negative bit to one, all others zero:

```
Command[1] NCZV 1000<CR>
```

Example 2 List values of all NCZV bits:

```
Command[1] NCZV<CR>
Command[1] NCZV = 1000
```



N = Negative
C = Carry
Z = Zero
V = Overflow
PBX = PixBlt executing

IE = Interrupt Enable
FE1 = Field extend 1
FS1 = Field size 1
FE0 = Field extend 0
FS0 = Field size 0

Status Register

Syntax **NR <register> <name>**

Description The NR command allows you to assign a name to the following:

- Register A0 to A14
- Register B0 to B14
- Stack Pointer

Parameter <register> is one of the registers listed above, and <name> is a substitute name of one to six characters. When assigned, <name> is used in the machine state display and in reverse assemblies. It will appear next to the register in the screen display. It cannot work with the register-value exchange designation (Section 4.4.4.3 on page 4-12).

To delete the name, enter the command with the register, but no name.

Example 1 Designate Register A0 as SUM:

```
Command[1] NR A0 SUM <CR>
```

Thereafter, SUM can be substituted for A0. Also, in reverse assemblies, SUM will be used instead of A0 (e.g., `MOVE A4,A0` will appear as `MOVE A4,SUM`)..

Example 2 To delete SUM as the name for A0:

```
Command[1] NR A0<CR>
```

The name is also erased from the screen.

Syntax **PB{H, V} [bit setting]**

Description These commands let you set the PBH or PBV bits in the I/O Control Register (address >C000 00B0). (Table 4-3 on page 4-58 is a complete list of I/O Registers and their offset values.)

- PixBit horizontal: PBH command
- PixBit vertical: PBV command.

[Bit setting] specifies the action on the bit:

0	Set to zero the specified horizontal or vertical bit
1	Set to one the specified horizontal or vertical bit
No entry	If neither a 1 or zero is entered, the specified bit is toggled.

Note that the value of both bits is readily available in the status-display center as the H and V bits in the ITPVH field.

Example Set the PBV bit:

```
Command[1] PBV 1  <CR>
```

The PBV bit is set to 1, one, causing the PixBlt instructions to decrement in the Y direction. If you then enter:

```
Command[1] PBV  <CR>
```

The PBV bit is toggled back to a 0, causing the PixBlt instructions to increment in the Y direction.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CD	PPOP					PBV	PBH	W		T	RR		RM	reserved	

CD = inst. cache disable
PPOP = pixel proc. operation select
PBV = PixBlt vertical direction
PBH = PixBlt horiz. direction

W = window violation
T = pixel transparency enable
RR = DRAM refresh rate
RM = DRAM refresh mode

I/O Memory Control Register

PBX Set/Toggle PBX Bit in Status Register PBX

Syntax **PBX {0, 1}**

Description The PBX command allows you to set, reset, or toggle the contents of the PBX (PixBlt in progress) bit in the Status Register. If a 0 or 1 value is not specified then the PBX bit is toggled; otherwise, the PBX bit is set to the value specified.

The result of the **{0, 1} operand**:

0	sets PBX bit to zero,
1	sets PBX bit to one.
No entry	If neither a 1 or 0 is entered, the PBX bit is toggled.

Note that the value of this bit is displayed in the center of the status display as the P bit in the ITPVH field.

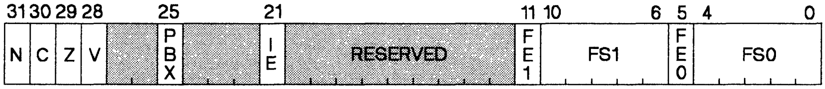
Example Set the PBX bit:

```
Command[1] PBX 1  <CR>
```

The PBX bit is set to 1, causing a RETI instruction to resume execution of a PixBlt instruction. Of course, if a PixBlt instruction was not in progress, then unpredictable results will occur. If you then enter:

```
Command[1] PBX  <CR>
```

The PBX bit is toggled back to 0, causing an interrupted PixBlt instruction not to be resumed upon an RETI instruction.



N = Negative
C = Carry
Z = Zero
V = Overflow
PBX = PixBlt executing

IE = Interrupt Enable
FE1 = Field extend 1
FS1 = Field size 1
FE0 = Field extend 0
FS0 = Field size 0

Status Register

Syntax **PC [<double-word value>]**

Description The PC command allows you to display and modify the contents of the Program Counter.

If no parameter follows PC, then the current PC value will be displayed. (The PC value is also part of the SDB display, seen on the center left of the screen.) **NOTE** that this form of the command **destroys commands that follow in the same command buffer.**

If the optional 32-bit replacement <double-word value> is specified, then the value of the PC is changed to <double-word value>. The default type for <double-word value> is hexadecimal.

Note that the PC always contains a word aligned value (i.e., the lower 4 bits are zero). If <double-word value> is not word aligned, then it is forced to word align by truncating its lower 4 bits to zero before being loaded into the PC.

Example 1 Modify the contents of the PC:

```
Command[1] PC 4302  <CR>
```

The PC now contains the value >00004300. Note the truncation of the lower 4 bits of the value.

Example 2 Display the contents of the PC from the command line by using the command without specifying a value:

```
Command[1] PC  <CR>
Command[1] PC = 00004300
```

This is useful for viewing the contents of the PC while the text display is off. Note that this form of the command **destroys any monitor commands that follow in the same buffer** .

PM Modify PMASK Register PM

Syntax **PM [<word value>]**

Description The PM command allows you to modify or display the contents of the PM (PMASK) I/O register.

- If the optional 16-bit replacement <word value> is not specified, the register contents are displayed. **NOTE** that this form of the command **destroys any remaining commands in the command file.**

- If <word value> is given, then the PM register is changed to <word value>. The default type for <word value> is hexadecimal.

PMASK Register contents are displayed in the "PM = xxxx" field in the upper right of the status display. Other I/O Registers and their addresses are listed in Table 4-3 on page 4-58.

Example 1 Modify the contents of the PM register:

```
Command[1] PM FFFE  <CR>
```

The PM register now contains the value >FFFE. This value allows only the LSb of each word written during graphics instructions to be affected.

Example 2 Display the contents of the PM register from the command line:

```
Command[1] PM  <CR>
Command[1] PM = FFFE
```

This is useful for viewing the contents of the PM register while the text display is off. Note that this form (no <word value> entered) of the command **destroys** any monitor commands that follow in the same buffer.

PP Set Pixel Processing Option PP

Syntax **PP [<PP option value>]**

Description The PP command allows you to modify or display the value of the five PPOP (Pixel Processing Operation Select) bits in the MEMORY CONTROL I/O Register. These bits are defined in Table 4-4. The register is shown on the following page.

- If no parameter follows PP, then the pixel-processing bits values are displayed in decimal.

- If the optional <PP option value> is specified, then the value of the PPOP bits is changed to <PP option value>. The default for <PP option value> is decimal. The number can be from 0 to 21 as listed in Table 4-4.

The contents of these bits are displayed in the "pp=" field of the status display. (The CONTROL and other I/O Registers are listed in Table 4-3 on page 4-58.)

Table 4-4. Pixel Processing Bit Descriptions

PP BITS (DECIMAL)	OPERATION	DESCRIPTION
0	S ---> D	Replace destination with source
1	S AND D ---> D	AND source with destination
2	S AND $\overline{D}$ ---> D	AND source with NOT-destination
3	All-0s ---> D	Replace destination with zeroes
4	S OR $\overline{D}$ ---> D	OR source with NOT-destination
5	S XNOR D ---> D	XNOR source with destination
6	$\overline{D}$ ---> D	Negate destination
7	S NOR D ---> D	NOR source with destination
8	S OR D ---> D	OR source with destination
9	D ---> D	Destination to destination
10	S XOR D ---> D	XOR source with destination
11	$\overline{S}$ AND D ---> D	AND NOT-source with destination
12	All-1s ---> D	Replace destination with ones
13	$\overline{S}$ OR D ---> D	OR NOT-source with destination
14	S NAND D ---> D	NAND source with destination
15	$\overline{S}$ ---> D	Replace destination with NOT-source
16	D + S ---> D	Add source to destination
17	ADD S(D,S) ---> D	Add S to D with saturation
18	D - S ---> D	Subtract source from destination
19	SUB S(D,S) ---> D	Subtract S from D with saturation
20	MAX(D,S) ---> D	Maximum of source and destination
21	MIN(D,S) ---> D	Minimum of source and destination

Example 1 Modify the contents of the PPOP bits:

```
Command[1] PP 14  <CR>
```

The PP bits now contain binary 01110 (decimal 14).

Example 2 Display the contents of the PPOP bits from the command line:

```
Command[1] PP  <CR>

Command[1] PP = 14
```

This is useful for viewing the contents of the PPOP bits while the text display is off. Note that this form of the command destroys any monitor commands that follow in the same buffer.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CD	PPOP					PBV	PBH	W		T	RR		RM	reserved	

CD = inst. cache disable
PPOP = pixel processing operation select
PBV = PixBlt vertical direction
PBH = PixBlt horizontal direction

W = window violation
T = pixel transparency enable
RR = DRAM refresh rate
RM = DRAM refresh mode

I/O Memory Control Register

PS Set PSIZE Register PS

Syntax **PS [<pixel size>]**

Description The PS command allows you to modify or display the contents of the PSIZE (Pixel Size) I/O register.

- If no <pixel size> is given after PS, the contents of the PSIZE Register are displayed. Note that this form of the command **destroys any monitor commands that follow in the same buffer.**
- If the optional replacement <pixel size> value is specified, then the value of the PS register is changed to <pixel size>. The default type for <pixel size> is a decimal of 1, 2, 4, 8, or 16.

Other I/O Registers are listed in Table 4-3 on page 4-58.

Example 1 Modify the contents of the PS register:

```
Command[1] PS 8  <CR>
```

The PS register now contains the value 8. This causes the SDB to process pixels at a size of 8 bits per pixel. Note that the only valid pixel sizes are 1, 2, 4, 8, and 16. Any other value will result in an error.

Example 2 Display the contents of the PS register from the command line:

```
Command[1] PS  <CR>
Command[1] PS = 8
```

This is useful for viewing the contents of the PS register while the text display is off. Note that this form of the command destroys any monitor commands that follow in the same buffer.

Syntax **Q[*][C][S]**

Description The Q command terminates the SDB session.

When you execute the Q command, the SDB will ask you if you are sure that you want to terminate the session. If you answer yes, then all files that the SDB has opened are closed and the SDB terminates execution. You can also enter an option with the command:

Option

*** (asterisk)** You are not asked for verification before termination. This is useful if the SDB is being run as part of a batch stream and you do not wish to have any user input. (See also the -f option in Section 4.4.2 on page 4-9 which invokes the SDB using a command input file.)

C (Clear) The SDB clears the PC screen on exit.

S (Save) The SDB executes the equivalent of the save machine state command with no parameters (i.e., the machine state is saved to the file SMSFILE.000). (See also SMS command.)

All three of the options may be specified, singly or in combination.

Example Terminate the SDB session, clear screen of text and graphics:

```
Command[1] QC<CR>
```

Syntax **RDE [<file number extension>]**

Description The RDE command is used in conjunction with the SDE (Save Debug Environment) command to restore a previously saved debugging environment context. The RDE command restores the saved debugging environment from a file in the default directory. This includes:

- the traces,
- the breakpoints,
- the register names, and
- the command buffers.

If the optional <file number extension> is not specified, then the file is called SDEFIL.000. If <file number extension> is present, it is converted to the ASCII of its decimal representation with zero fill on the left so that a three-character file extension is formed. This limits <file number extension> to 0 to 999, inclusive. This three-character extension is used to form the file name SDEFIL.aaa, where aaa is the three character file extension. The default type for <file number extension> is decimal.

To restore a specific debugging environment via the RDE command, you must specify the same <file number extension> that it was saved with. See the SDE command for saving the debugging environment.

The graphics environment saved via the SDE command includes: the traces, the breakpoints, and register names.

Example 1 Restore the debugging environment from data stored in the file SDEFIL.042:

```
Command[1] RDE 42  <CR>
```

Example 2 Restore the debugging environment from data stored in the default file SDEFIL.000:

```
Command[1] RDE  <CR>
```

Syntax **REset**

Description The RESET command downloads the SDB340.GSP code to re-establish communication with the SDB.

File SDB340x.GSP must be available as described in the **Notes** to Section 4.4 on page 4-9.

Example

```
Command[1] RE  <CR>
```

or

```
Command[1] RESET  <CR>
```

Syntax **RIO**

Description The RIO command restores the contents of the I/O registers from a copy that is kept in SDB340 local memory (as opposed to on disk). The copy of the I/O registers should have previously been saved via the SIO (Save I/O Registers) command. If the registers were not previously been saved, **RIO will set them to zero.**

Example Restore the local copy of the I/O register:

```
Command[1] RIO  <CR>
```

Syntax **RMI [<file number extension> [<offset>]]**

Description The RMI command is used to restore the range of memory that had been saved in a disk file with the SMI (Save Memory Image) command.

The data and the memory addresses will be the same as the area stored with the SMI command. The beginning and ending addresses of the memory image were stored in the file (along with memory data) and so are not needed on the command line.

- The <file number extension> is used to form the filename SMIFIL.aaa, where aaa is the three-character file extension. The default type for <file number extension> is decimal, and is limited to 0 to 999, inclusive. *It must be the same <file number extension> under which it was filed.*
- If the optional <file number extension> is not specified, then the file is called SMIFIL.000.
- The memory image can optionally be offset from its previous location in memory by specifying <offset>. Note: if <offset> is to be specified, you *must also specify* the <file number extension>. The <offset> is hexadecimal by default and is treated as a signed 32-bit value.

To restore a specific memory image via the RMI command, you must specify the same <file number extension> under which it was saved. SDB340 tells you if the file does not exist. If the SDB340 reaches a premature end-of-file condition on the SMIFIL or if the SDB340 encounters a memory write error, the data restored thus far from the file to memory will remain in memory. You will be informed of the incomplete memory restoration in both cases. See the SMI command for saving the memory image.

Example 1 Restore the memory image data stored in the file SMIFIL.100:

```
Command[1] RMI 100 <CR>
```

Example 2 Restore the memory image data stored in the file SMIFIL.047, offsetting the data in memory by a value of >0401 bits:

```
Command[1] RMI 47 0401 <CR>
```

Example 3 Restore the memory image data stored in the default file SMIFIL.000:

```
Command[1] RMI <CR>
```

You could also use the default file with an offset, as shown below (>780 is the offset):

```
Command[1] RMI 0 780 <CR>
```

Syntax **RMS [<file number extension>]**

Description The RMS command restores the machine state of SDB340 from a disk file in the default directory. This machine state is that stored by the SMS (Save Machine State) command.

- To restore a specific machine state via the RMS command, the <file number extension> must be the same as the one underwhich it was saved. You will be told if the restoration file does not exist.
- The <file number extension> is used to form the filename SMSFIL.aaa, where aaa is the three-character file extension. The default type for <file number extension> is decimal, and is limited to 0 to 999, inclusive.
- If the optional <file number extension> is not specified, then the restoration file is called SMSFIL.000.

Machine state elements restored are:

- the A, B, and I/O register files,
- Status Register
- Program Counter, and
- the trap vectors.

See the SMS command for saving the machine state.

Example 1 Restore the machine state data in the file SMSFIL.100:

```
Command[1] RMS 100  <CR>
```

Example 2 Restore the machine state data stored in the default file SMSFIL.000:

```
Command[1] RMS  <CR>
```

RR Restore Temporary Copy of Registers RR

Syntax **RR**

Description The RR command restores the contents of the A and B file registers from a copy that is kept in SDB340 local memory (as opposed to on disk).

- The copy of the registers should have previously been saved via the SR (Save Registers) command.

- If the registers have not previously been saved then they will be set to zero.

Example Restore the local copy of the A and B file registers:

```
Command[1] RR <CR>
```

RUN, RU Run for a number of Instructions RUN, RU

Syntax **RUn [<instruction count>]**

Description The RUN commands executes instructions either continuously or until an optionally specifiable instruction count has been reached. The screen display is not updated until execution stops. You can enter the command as RUN or abbreviate it as RU.

If the optional <instruction count> is specified, then the SDB will execute an <instruction-count> number of instructions and then return to command level. The default type for the <instruction count> is decimal. If the <instruction count> is not specified, then the SDB will execute instructions until one of these halt conditions exits:

- you halt execution with a keystroke,
- an error is encountered,
- a breakpoint is encountered, or
- a TRAP 29 is executed.

Note:

While the TMS34010 is running, pressing ESC will allow it to continue that way while control is given to the keyboard. Thus changes can be made to memory, including the I/O Registers, without halting the TMS34010. Access can be made to such items as the plane mask, cache enable, and pixel processing. Internal elements of the TMS34010 cannot be changed (such as the PC, ST, or the A or B file registers).

To halt the TMS34010, re-enter the RUn command and hit any key other than ESC. You can also use this to exit the SDB while leaving the TMS34010 running.

Example 1 Execute the RUN command with an instruction count of 100:

```
Command[1] RUN 100 <CR>
```

or

```
Command[1] RU 100 <CR>
```

Execution will halt after 100 instructions if none of the halt conditions mentioned above have occurred.

Example 2 Execute the RUN command without an instruction count:

```
Command[1] RUN <CR>
```

or

```
Command[1] RU <CR>
```

Execution will halt only if one of the halt conditions mentioned above has occurred. Also see BP (Execute with BreakPoint).

Syntax **SDE [<file number extension>]**

Description The SDE command is used to preserve the context of a debugging environment. The environment saved includes:

- the traces,
- the breakpoints,
- the register names, and
- the command buffers.

The RDE (Restore Debug Environment) command can be used to restore this environment.

- The <file number extension> is used to form the filename SDEFIL.aaa, where aaa is the three-character file extension. The default type for <file number extension> is decimal, and is limited to 0 to 999, inclusive.
- If the optional <file number extension> is not specified, then the file is called SDEFIL.000.

To restore a specific debug environment via the RDE command, you must specify the same <file number extension> with which it was saved. See the RDE command for restoring procedures.

Example 1 Save the debugging environment in the file SDEFIL.043:

```
Command[1] SDE 43 <CR>
```

Example 2 Save the debugging environment in the default file SDEFIL.000:

```
Command[1] SDE <CR>
```

Syntax **SF <filename>**

Description The SF command displays the contents of the file called <filename>. This allows you access to system files without corrupting or losing the current simulation. The screen is cleared before and after viewing the file.

Example Display the contents of the file EXAMPLE.LST:

```
Command[1] SF EXAMPLE.LST  <CR>
```

Note that the file will be displayed in 23-line blocks, then pause.

- The RETURN key brings up successive 23 line pages of the file.
- The Q key entry halts the display at any time.

This command is useful for displaying assembly listings and linker map files during a debugging session.

Syntax **SIO**

Description The SIO command saves the contents of all of the I/O registers to a copy kept in the SDB local memory (as opposed to on disk). Note that this is temporary memory and is cleared between invocations of the SDB. The I/O registers are restored from this copy via the RIO (Restore I/O registers) command.

Example Save a local copy of the I/O registers:

```
Command[1] SIO  <CR>
```

Syntax

SMI <start address> <end address> [<file number extension>]

Description

The SMI command is used to save a range of memory to disk. The RMI (Restore Memory Image) command returns the block to memory. The SMI command can be used to preserve a specified memory context for debug; it may also be used to store screen data.

The SMI command saves the region of TMS34010 memory from <start address> to <end address>, inclusive, in binary format in a file on disk in the default directory. The default format for both addresses is hexadecimal.

> **Note:**
>
> The addresses specified for the SMI command are inclusive bit addresses and are treated as such. Thus, if you specify the starting and ending addresses as being the same address, then the result will be the save of a single bit of memory. Examples:
>
> - If you wish to save all of the words of memory from 0 up to and including the word starting at >400, then the start and end addresses should be >0 and >40F.
> - If you specified >400 as the ending address, then only the first bit of the word at >400 would be saved.

If <file number extension> is present, it is used to form the file name SMIFIL.aaa, where aaa is the one-to-three-character file extension. This limits <file number extension> to 0 to 999, inclusive. The default type for <file number extension> is decimal.

If the optional <file number extension> is not specified, then the save file is called SMIFIL.000.

If the save file cannot be created or there is an error while writing to the file (i.e. running out of disk space), the saving of memory to the file will terminate and the file will be closed. If you attempt to restore the memory image in the file, then whatever was stored in the file up to the error will be restored. The RMI command will then detect a premature end of file on the restoration file and signal an error.

To restore a specific memory image via the RMI command, you must specify the same <file number extension> under which it was saved. See the RMI command for restoring the memory image.

Example 1

Save the memory image data from address >1A0 to >200F in the file SMIFIL.792. The data is stored in non-compressed, binary-image format.

```
Command[1] SMI 1A0 200F 792  <CR>
```

Example 2

Save a single bit of memory at >1A1 in the file SMIFIL.003.

```
Command[1] SMI 1A1 1A1 3  <CR>
```

Example 3 Save the memory image data from address >440C to >4601 in default file SMIFIL.000

```
Command[1] SMI 440C 4601  <CR>
```

Syntax **SMS [<file number extension>]**

Description The SMS command is used in conjunction with the RMS (Restore Machine State) command to save a machine state of the SDB to a file in the default directory. If the optional <file number extension> is not specified, then the save file is called SMSFIL.000.

If <file number extension> is present, then it is converted to the ASCII of its decimal representation with zero fill on the left so that a three character file extension is formed. (This limits <file number extension> to 0 to 999, inclusive.) This three-character extension is used to form the file name SMSFIL.aaa, where aaa is the three character file extension. The default type for <file number extension> is decimal.

To restore a specific machine state via the RMS command, you must specify the same <file number extension> under which it was saved. See the RMS command for restoring the machine state.

The machine state elements stored are:

- the A, B, and I/O Register files,
- Status Register,
- Program Counter, and
- the trap vectors.

Example 1 Save the machine state data in the file SMSFIL.100:

```
Command[1] SMS 100  <CR>
```

Example 2 Save the machine state data in the default file SMSFIL.000:

```
Command[1] SMS  <CR>
```

Syntax **SP [<double-word value>]**

Description The SP command modifies or displays the contents of the SP Stack Pointer register. If the optional 32-bit replacement value <double-word value> is specified, the contents of the SP register are changed to <double-word value>. **NOTE** that if the <double-word value> parameter is not specified, this form of the command **will destroy** any other commands that would be remaining in a command buffer.

The default type for <double-word value> is hexadecimal. (To modify or display the A and B file register, see the A# and B# commands.)

Example 1 Modify the contents of the SP register:

```
Command[1] SP 4000  <CR>
```

The SP register now contains >0000 4000.

Example 2 Display the SP contents from the command line:

```
Command[1]SP  <CR>
Command[1] SP = 00004000
```

The contents of the SP register are now visible in the command buffer. This is useful for viewing the contents of the SP register while the text display is off. Note that this form of the command destroys any monitor commands that follow in the same buffer.

Syntax **SR**

Description The SR command saves the contents of the A and B file registers in a copy that is kept in the SDB local memory (as opposed to disk). The A and B file registers are restored from this copy via the RR (Restore Registers) command. Note that only one copy of the registers may be saved at a time. A re-invocation of the SR command will overwrite the registers saved with the previous SR command.

Example Save a copy of the A and B registers:

```
Command[1] SR  <CR>
```

SS,SSF,SSFU,SSU Single Step by Count SS,SSF,SSFU,SSU

Syntax **SS[F,FU,U] [instruction count]**

Description The SS command allows you to single step through a program for an [instruction count] number of instructions, with or without Fast update and/or Unassembly.

If none of the optional parameters, including F and U, are specified, then the SDB executes only one assembly language instruction and updates the SDB status display. If the optional [instruction count] is specified, then the SDB will execute [instruction count] instructions with a complete SDB status display update after each instruction execution.

The F and U options allow you to specify whether you want a Fast update – used when stepping for a number of instructions – and whether you want an Unassembly after each instruction:

F The F option inhibits the update of the SDB status display after each instruction except the last. This is functionally equivalent to the RUN command with an [instruction count], but executes slightly slower. Its value lies in using it with the U option, providing a faster single step with unassembly.

U The U option causes the SDB to supply a 5-line reverse assembly after each instruction. The reverse assembly includes the information about the two previous program counter locations, the current program counter location and the two following sequential locations. These are displayed similarly to the display for a U command.

Example 1 Single step for 1 instruction:

```
Command[1] SS <CR>
```

Example 2 Single step for 10 instructions:

```
Command[1] SS 10 <CR>
```

Example 3 Use the F and U options:

```
Command[1] SSF <CR>
Command[1] SSFU 100 <CR>
Command[1] SSU 10 <CR>
```

Note that the F, U, and instruction count options may be used independently or in conjunction with one another.

See also BP (BreakPoint) and RUN (RUN until halted) commands.

ST,STN,STC,STZ,STV Status Register ST,STN,STC,STZ,STV

Syntax **ST [{ ({N, C, V, Z} {0, 1}) , <double-word value>}]**

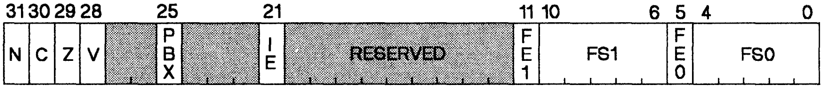
Description The **ST** command allows you to display or modify the contents of the Status Register by specifying either a 32-bit replacement value or a status bit name and a bit replacement value. The default type for <double-word value> is hexadecimal.

Entering **ST** without a value will display the Status Register contents. This is useful for viewing the contents of the Status Register while the text display is off.

You can also selectively set or reset the contents of the four arithmetic status bits with these commands:

Command	*Alters Status Register bit*
STN	N (negative)
STC	C (carry)
STZ	Z (zero)
STV	V (overflow)

Note that the values of these bits are readily available by referencing the NCZV section of the SDB status display.



N = Negative
C = Carry
Z = Zero
V = Overflow
PBX = PixBlt executing

IE = Interrupt Enable
FE1 = Field extend 1
FS1 = Field size 1
FE0 = Field extend 0
FS0 = Field size 0

Status Register

Example 1 Modify the contents of the Status Register using <double-word value>:

```
Command[1] ST F0000046  <CR>
```

Example 2 Turn on the Z bit of the Status Register:

```
Command[1] ST Z 1  <CR>
```

Note that the space after ST is optional when spcifying a particular status bit; it is allowed for clarity.

To turn off the Z bit, enter:

```
Command[1] STZ 0  <CR>
```

Example 3 Display the contents of the Status Register:

```
Command[1]ST  <CR>
```

```
Command[1] ST = F0000046
```

The contents of the Status Register are now visible in the command buffer. Note that this form of the command destroys any monitor commands that follow in the same buffer.

Syntax **SWitch**

Description The SWITCH command modifies the command entry source from the keyboard to the file GSPINPUT.000. Commands are accepted as they occur in the file until a SWITCH command is encountered in the file or an EOF is encountered. At this point, control returns to the keyboard.

Notes:

1. A inadvertent keystroke input **cannot halt** a batch stream's execution. This allows you to leave an executing system unattended.
2. If the SWITCH command is interrupted before completing the command string (e.g., an unexecutable command is encountered) and terminates with an error message, the string can be continued *at the command after the one in error* by issuing another SWITCH command. However, if you instead want to begin with the first command in the file, first issue the CIF (Close Input File) command.

When the SWITCH command is encountered in the file, the SDB returns to accepting input from the keyboard. If you key in another SWITCH command, then the SDB continues accepting input from the file, *continuing at where it had left off* in reading the file. However, if an EOF is encountered or a CIF command executed (see above note), then the input file is closed. Another SWITCH command will then begin reading again from the *top* of the file. You can also cause the SDB to automatically begin reading from the input file by specifying the -f option when the SDB is invoked. This option is covered in Section 4.4.2 on page 4-9.

Example Switch the command input source:

```
Command[1] SWITCH
```

The following is a sample input file.

```
PC  FFC00000
ssu 13
bpai  FFC00000
switch
```

Note that the file contents are automatically converted to uppercase.

Syntax **SY <string>**

Description The SY command executes MS-DOS system functions from the SDB. Edits, assemblies, links, file copies, etc., may be executed via this command while in an SDB session.

Example

```
Command[1] SY COPY \GSP_ASM\HELLO.OBJ HELLO2.OBJ
```

or

```
Command[1] SY GSPASM \GSP_ASM\HELLO.ASM;
```

or

```
Command[1] SY EDIT \GSP_ASM\HELLO.ASM
```

or

```
Command[1] SY CD \GSP_ASM
```

or

```
Command[1] SY DIR B:
```

The SDB status display is cleared and the MS-DOS command is executed. After the command has terminated, the SDB waits for a carriage return before clearing the screen and rebuilding the SDB status display. Normal system control characters will affect the execution in the same manner as if the command had been invoked from the operating system.

T Toggle Transparency Bit T

Syntax **T [{0, 1}]**

Description The T command sets, resets, or toggles the contents of the T (Transparency) bit in the I/O Control register. Note that this is the T bit in the ITPVH section of the SDB status display.

1 A one entry sets the T bit.

0 A zero entry resets the T bit.

No entry If neither a 0 nor a 1 is specified, the T bit is toggled.

Example Set the T bit:

```
Command[1] T 1  <CR>
```

The T bit is set to 1, enabling transparency.

If you then enter:

```
Command[1] T  <CR>
```

the T bit is toggled back to 0, disabling transparency.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CD	PPOP					PBV	PBH	W		T	RR		RM	reserved	

CD = inst. cache disable
PPOP = pixel proc. operation sel.
PBV = PixBlt vertical direction
PBH = PixBlt horiz. direction
W = window violation
T = pixel transparency enable
RR = DRAM refresh rate
RM = DRAM refresh mode

I/O Memory Control Register

Syntax TR

Description The TR command displays all existing traces, along with their active/inactive state.

Note that each trace is assigned a reference number, and that a combined maximum of 20 traces and breakpoints can be defined at one time. The reference numbers specified here are the ones used in conjunction with the TR# command (next page) to manipulate the state of each trace on the list. Traces are defined and modified using the TR# and TRAI commands that immediately follow.

TR# Modify Existing Traces TR#

Syntax **TR{0,..19,X} [{Clear, OFf, ON, Toggle, Quit}]**

Description The TR# command modifies the status of individual traces, specified via the trace reference number (#). This parameter is specified by:

0-19 A decimal integer between 0 and 19 is the trace reference number unique for each trace (allowing a combination of up to 20 traces and breakpoints to be defined).

X If the trace number is the letter "X", then all existing traces are affected.

The trace reference number is displayed upon definition of the trace and can be viewed with the TR command. The trace reference number does not change thoughout the life of the specific trace.

The following options can follow the TR# fields:

Option	*Description*
Clear	The option CLEAR destroys the trace.
OFf	The OFf option deactivates the trace temporarily but does not destroy it.
ON	The ON option is used to reactivate deactivated traces.
Toggle	The TOGGLE option activates deactivated traces and deactivates active traces.
Quit	The Quit option terminates the command **without any changes being made**.

Note that only the significant letters of each option are processed. This allows you to specify a shorthand version of the option. For example, the options CLEAR and C will be treated the same. If you do not enter the option, a menu will be displayed to allow you to select the desired option.

Example 1 Toggle trace 3:

```
Command[1] TR3 TOG
```

or

```
Command[1] TR3  <CR>
```

The second entry causes the SDB to display the trace and the menu shown in Figure 4-11.

```
Normal Stop Mode                                  <Cache status> Cnt=     4
 st 00000010  NCZV=0000 ITPVH=00000  SP=00000000     Ctl=0000
 pc 00000000   0000  MNEMONIC OP;   MNEMONIC OP
 3   TRAI   FFFF1000 - FFFF1005


                          T   -- Toggle trace
                          ON  -- Trace on
                          OFF -- Trace off
                          C   -- Clear  trace
                          Q   -- Quit menu

  Command[1] TR3
                       Enter action:_
```

Figure 4-11. Display Existing Traces Monitor Display Format

At this point, you can enter T to toggle to off the state of trace number 3 (now on). Note that trace 3 remains in memory and may be reactivated by the same command sequence or by specifying the ON option. Alternatively, it may be deleted with the CLEAR option and then overwritten by using the TRAI command that follows. The modification of the trace is verified by the display of the traces the same as it appears after a TR command.

Example 2 CLear all traces:

```
Command[1] TRX CLEAR
```

Set Trace on Address

Syntax **TRAI <address>**

Description The TRAI command writes traces to a trace file called GSPTRACE.000 when a specific address is accessed during instruction acquisition. This file can be closed (for possible viewing with the SF or SY commands) with the CTF (Close Trace File) command.

The <address> parameter is in hexadecimal.

Example Set trace on location >120F F310:

```
Command[1] TRAI 120FF310
```

Syntax **U [<start address> <end address>]**

Description This command unassembles (reverse assembles) blocks of memory dependent upon whether:

- no address is given,
- start address only is given, or
- start and stop addresses are given.

In each case, up to nine instructions can be displayed. If more are needed to complete the command, entering a RETURN displays the next block of instructions.

These options are described in the following examples.

Example 1 No address given.

```
Command[1] U <CR>
```

If you are **single stepping**, the result would be a display of the reverse assemblies of:

- the last two Program Counter locations (last two instructions executed):

  - the previously executed instruction in yellow,
  - the instruction before that (above) in cyan (light blue),

- the current Program Counter location (next instruction) in green, and
- the next two consecutive instructions following the current PC.

If you are in the **run mode**, the last two Program Counter values would be unknown.

Example 2 Unassemble indefinitely from a starting location 880:

```
Command[1] U 880  <CR>
```

Enter the <start address> in hexadecimal format. Continue displaying succeeding locations by entering carriage RETURNs. Terminate the display by entering a Q.

Example 3 Unassemble a range from 880 to 1020:

```
Command[1] U 860 8B0 <CR>
```

The resultant display appears as follows:

```
     Lnr Addr  Opcode     Revassembly
  pc 00000860   09C0      MOVI >004B,A0
  pc 00000880   F622      DRAV A1,A2
  pc 00000890   3C40      DSJS A0,@880
  pc 000008A0   091D      TRAP 29
  pc 000008B0   09C0      MOVI >42,A0


Command[0] U 860 8B0
                U 860 8B0
```

Specify the start and stop addresses in hexadecimal format. After the instruction at <start address> is displayed, you can continue displaying succeeding locations up until the <end address> by entering carriage RETURNs. Terminate the display by entering a Q.

Syntax **V <value>**

Description The V command displays various forms of <value>. The default format for <value> is hexadecimal. The size of <value> can be up to 32 bits.

- The first two lines display <value> as an address in both Y.X and linear format.
- The third line displays *the contents* of the memory word at address <value> as both data and an unassembled source statement. Note that the lower four bits of <value> are ignored since it is treated as a word address.
- The fourth line displays <value> as data in both hexadecimal and decimal form.

Example Use the command to evaluate >FF8.

```
Command[3] V FF8<CR>
```

This produces the following display, assuming that memory location >FF0 contains >2980, CONVSP is >15, CONVDP is >16, PSIZE = 4, and OFFSET = 0:

```
                in Hex   Decimal             source      destination
   Address:   00000FF8     4088        Y,X:  0003,00FE   0007,007E
                                    Linear:  00003FE0    00003FE0
@00000FF0:        2980    10624       ASM:   SRA 20,A0
      Data:       0FF8     4088

           Command[0] V FF8
                      V FF8
```

Syntax **VMI [<file number extension> [<offset>]]**

Description The VMI command compares the data in a disk file to the data starting at a memory address (plus an optional offset value). The memory address is that stored in the disk file with the data during an SMI command. Thus, no memory-address bounds are needed. If the comparison is successful (a "match"), the following message is displayed:

```
        Memory verification succeeded
```

If a comparison is unsuccessful, the following is displayed: 1) address of mismatch, 2) memory value, and 3) file value.

By using the optional <offset>, the values in the disk file can be compared starting at any memory location. The <offset> will be added to the start address stored with the original SMI command to generate the start of memory used in the verification.

- The <file number extension> is used to form the filename SMIFIL.aaa, where aaa is the three-character file extension. The default type for <file number extension> is decimal, and is limited to 0 to 999, inclusive. *It must be the same <file number extension> under which it was saved.*

- If the optional <file number extension> is not specified, then the file is called SMIFIL.000.

- The memory image can optionally be offset from its previous location in memory by specifying <offset>. Note: if <offset> is to be specified, you *must also specify* the <file number extension>. The <offset> is hexadecimal by default and is treated as a signed 32-bit value.

To compare a specific memory image, you must specify the same <file number extension> under which it was saved. SDB340 tells you if the file does not exist. See the SMI command for saving the memory image.

Example 1 Compare the memory image data stored in the file SMIFIL.100:

```
Command[1] VMI 100 <CR>
```

Example 2 Compare the memory image data stored in the file SMIFIL.047, offsetting the data in memory by a value of >C000 bits:

```
Command[1] VMI 47 C000 <CR>
```

Example 3 Compare the memory image data stored in the default file SMIFIL.000:

```
Command[1] VMI <CR>
```

You could also use the default file with an offset, as shown below (>FFE is the offset):

```
Command[1] VMI 0 FFE <CR>
```

Example 4

The following causes a mismatch:

```
Command[1] F 0 200 0 <CR>                      (zero fill)

Command[1] SMI 0 1FF <CR>                      (save image)

Command[1] VMI <CR>                            (compare the two)
     Memory verification succeeded             (comparison OK)

Command[1] MM 10 FF <CR>                       (place FF in 10)

Command[1] VMI <CR>                            (recompare the two)
     00000010: 00FF/0000                       (comparison failed)
```

Syntax **W [<window option>]**

Description The W command modifies the contents of the 2-bit W (windowing) field of the CONTROL I/O Register by specifying a value from 0 to 3. Corresponding values for <window option> are:

0 No windowing; writes to any pixel allowed; no interrupts.

1 Pick function; pixel writes inhibited; interrupt upon write attempt within current window.

2 Pixel writes to window not inhibited; interrupt upon write attempt outside current window.

3 Inhibit pixel write attempt outside window; no interrupts.

Note that the current value of this register field is shown in the "w=" field in the upper-right corner of the status display.

W FIELD IN REGISTER	**"w=" FIELD IN DISPLAY**
00	off
01	pik
10	int
11	on

Example 1 Set for W = 2 (pixel writes to window):

```
Command[1] W 2  <CR>
```

Example 2 Display the contents of the W field from the command line:

```
Command[1] W <CR>
Command[1] W = 00000002
```

(response)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CD	PPOP					PBV	PBH	W		T	RR		RM	reserved	

CD = inst. cache disable
PPOP = pixel proc. operation select
PBV = PixBlt vertical direction
PBH = PixBlt horiz. direction

W = window violation
T = pixel transparency enable
RR = DRAM refresh rate
RM = DRAM refresh mode

I/O Memory Control Register

5. SDB Hardware Operation

This section covers the following:

5.1 Host Port

The TMS34010 has a 16-bit wide host port which allows communication with the local memory and internal control registers. The host port interfaces to four internal programmable 16-bit registers:

- HSTADRL
- HSTADRH
- HSTDATA
- HSTCTL.

These four registers are further defined in Section 5.1.1. They are addressed by the following lines:

- Host Function Select lines HFS0 and HFS1
- Read strobe $\overline{\text{HREAD}}$
- Write strobe $\overline{\text{HWRITE}}$
- Chip select $\overline{\text{HCS}}$

Through this interface, commands, status information, and data are transferred between the TMS34010 and the host processor. Table 5-1 lists the signal logics and resulting operations.

Table 5-1. Signals Controlling Host Port Interface

$\overline{\text{HCS}}$	HFS0	HFS1	$\overline{\text{HREAD}}$	$\overline{\text{HWRITE}}$	OPERATION
1	x	x	x	x	No operation
0	0	0	0	1	HSTADRL read
0	0	0	1	0	HSTADRL write
0	0	1	1	0	HSTADRH read
0	0	1	1	0	HSTADRH write
0	1	0	0	1	HSTDATA read
0	1	0	1	0	HSTDATA write
0	1	1	0	1	HSTCTL read
0	1	1	1	0	HSTCTL write

x = don't care

5.1.1 Definition of Host Port Registers

5.1.1.1 Registers HSTADRL and HSTADRH (Local Memory Pointer)

These registers comprise the 32-bit address pointer to the TMS34010's local memory space.

HSTADRL contains the pointer's 16 least-significant bits (LSbs) mapped into two consecutive bytes since the PC has an 8-bit data bus. The four LSbs of the address are always forced to zero since these bits are only used internally to the TMS34010. HSTADRH contains the pointers 16 MSbs. Figure 5-1 shows these bits.

HIGH BYTE (>C7F01, IBM; >E7E01, TI)								LOW BYTE (>C7F00, IBM; >E7F00, TI)							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16

(a) HSTADRH (POINTER MOST SIGNIFICANT WORD)

HIGH BYTE (>C7E01, IBM; >E7E01, TI)								LOW BYTE (>C7E00, IBM; >E7E00, TI)							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	ALWAYS 0	ALWAYS 0	ALWAYS 0	ALWAYS 0

(b) HSTADRL (POINTER LEAST SIGNIFICANT WORD)

Figure 5-1. Register HSTADRx, Pointer Address to TMS34010 Local Memory

5.1.1.2 Register HSTDATA, Host/TMS34010 Data Transfer

This register contains data to be transferred between (to and from) the TMS34010 local memory bus and the host. Figure 5-2 shows this register.

HIGH BYTE (>C7001, IBM; >E7001, TI)								LOW BYTE (>C7000, IBM; >E7000, TI)							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Figure 5-2. Register HSTDATA, Host/TMS34010 Local Memory Data Transfer

5.1.1.3 Register HSTCTL, TMS34010 Control Register

This register controls various functions of the TMS34010 as shown in Figure 5-3. The HSTCTL Register is the concatenation of the two host interface control registers in the TMS34010 I/O Register file.

HIGH BYTE (>C7D01, IBM; >E7D01, TI)								LOW BYTE (>C7D00, IBM; >E7D00, TI)							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
HALT	CF	LBL	INCR	INCW	RSVD	NMIMODE	NMI	INTOUT	MSGOUT	MSGOUT	MSGOUT	INTIN	MSGIN	MSGIN	MSGIN

Figure 5-3. Register HSTCTL, TMS34010 Control

A description of the HSTCTL bits:

HSTCTL BIT,
HIGH BYTE

D7: HALT When a one, the TMS34010 suspends instruction execution after completion of the instruction in progress.

D6: CF Cache Flush. When a one, the cache is disabled.

D5: LBL Lower Byte Last. Allows you to specify whether indirect access begins when the upper or the lower byte of the register is accessed by the host processor. LBL accommodates 8-bit host processors.

0 When a zero, a local bus cycle is initiated if the host writes to the upper byte of HSTADRH or reads from/writes to the upper byte of HSTDATA.

1 When a one, a local bus cycle is initiated if the host writes to the lower byte of HSADRL or reads from/writes to the lower byte of HSTDATA.

D4: INCR When set to one, the address pointer is incremented prior to a local-memory read cycle.

D3: INCW When set to one, the address pointer is incremented after to a local-memory write cycle.

D1: NMIMODE When NMI bit (below) is a one, this bit determines whether or not to save the Program Counter and Status Register when an NMI occurs:

- NMIMODE a 0 = save machine state on stack before NMI occurs,
- NMIMODE a 1 = don't save machine state if NMI occurs.

D0: NMI When set to one by host, a non-maskable interrupt (NMI) is sent to the TMS34010. This bit is automatically cleared to zero after the interrupt is taken.

HSTCTL BIT, LOW BYTE

D7: INTOUT Interrupt Out, TMS34010 to Host. Allows TMS34010 to interrupt the host processor. When INTOUT is a one, TMS34010 pin $\overline{\text{HINT}}$ is driven active low. When INTOUT is a zero, pin $\overline{\text{HINT}}$ driven high. (TMS34010 interrupts the host by setting INTOUT to one; host clears interrupt by setting INTOUT back to zero.) The following will have no effect:

- attempt to write a zero to this bit by the TMS34010,

or

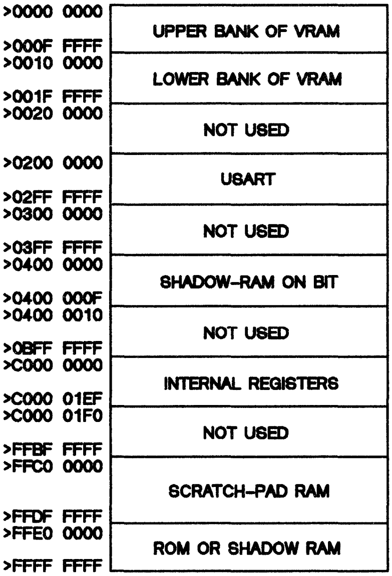
- attempt to write a one to this bit by the host.

D6-D4: MSGOUT The 3-bit buffer of D6, D5, and D4 allows the TMS34010 to send messages to the host. The host can only read this buffer, but the TMS34010 can read and write to it.

D3: INTIN Interrupt TMS34010 by Host. When the host writes a one to this bit, it generates an interrupt request to the TMS34010. The TMS34010 can clear the request by writing a zero to the bit. The value of this bit is shown in the read-only HIP (host interrupt pending) bit of the INTPEND I/O Register (address >C000 0120). Writing to the HIP bit has no effect on the INTIN or HIP bit.

D2-D0: MSGIN The 4-bit buffer allows the host to send messages to the TMS34010. The host can read or write to this buffer, but the TMS34010 can only read it.

The INTPEND Register HI bit holds a read-only representation of the INTIN bit.



NOTE: BECAUSE SOME LEAST-SIGNIFICANT MEMORY-ADDRESS BITS ARE NOT DECODED ON ADDRESS LINES, MEMORY AREAS APPEAR LARGER THAN ACTUAL ON-CHIP MEMORY.

Figure 5-4. Board Memory Map

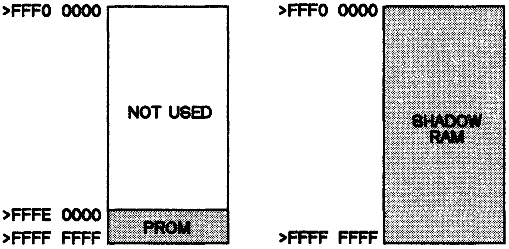
5.2 Local Memory Map

Figure 5-4 is a memory map of the SDB. Included on the board are:

- 512K bytes of onboard scratch-pad DRAM on four TM4256EC4 SIPs (single inline packages),
- 256K bytes of onboard Video RAM (VRAM) for the display buffer on eight TM4161EV4 SIPs.
- 1024 bytes of optional memory in two TBP28S42 bipolar PROMs. This is in the upper part of scratch pad memory -- in the TMS34010's reset and interrupt vector memory area.
- A USART mapped in the TMS34010's local memory space.

The two optional TBP28S42 PROMs provide space for ROM boot code if necessary (see Section 5.2.1). The upper portion of the DRAM actually shadows these ROMs (i.e, the DRAM can be enabled so that it takes over the memory area occupied by the ROM). Once the DRAM is enabled, it remains so until a powerup reset. At that time, the DRAM is disabled and the ROM re-enabled.

The address of the TMS34010 is decoded by a PAL (programmable array logic device). The PAL does not decode all the upper address bits; thus, there are many duplicate images of the memory devices in the memory map. Figure 5-4 contains the location of the mapped devices not including their images. For more information, see Section 6.



NOTE: BECAUSE THE SEVEN LEAST-SIGNIFICANT ADDRESS BITS OF THE PROM ADDRESS ARE NOT DECODED, THE PROM AREA SHOWN WILL APPEAR AS A 1K ON-CHIP MEMORY AREA DUPLICATED 128 (2^7) TIMES.

Figure 5-5. Shadow RAM and PROM in Upper Memory

5.2.1 Enabling Shadow RAM

The SDB has sockets (U35 and U36) for two TBP28S42 bipolar PROMs which reside in the upper part of the TMS34010 memory. When populated, these ROMs are mapped in the TMS34010's reset and interrupt vectors and can be used for bootloading. Shadow RAM is mapped behind these PROMs as shown in Figure 5-5. Enabling/disabling procedures:

- PROM is enabled at powerup reset.
- Shadow RAM is enabled by writing >0001 to location >0400 0000 in the local memory space.
- Shadow RAM can be enabled from the host port by:

  1) loading the HSTDRL Register with >0000, and
  2) loading the HSTADRH Register with >0400, then
  3) writing >0001 to the HSTDATA.

- Once the shadow RAM is enabled, it remains so until a powerup reset, at which time the RAM is disabled and the PROM is re-enabled.

Example code to enable shadow RAM:

```
0000                    code     segment
                                 assume  cs:code
0100                             org     100h
0100 EB 01 90           entpt:   jmp     start

= C000                  HSTSEG   equ     0C000h
= 7000                  HSTDATA  equ     7000h
= 7D00                  HSTCTRL  equ     7D00h
= 7E00                  HSTADRL  equ     7E00h
= 7F00                  HSTADRH  equ     7F00h

0103                    start    proc    near

0103 B8 C000                     mov     ax,HSTSEG
0106 8E C0                       mov     es,ax
0108 26 A1 7D00                  mov     ax,es:(HSTCTRL)
010C 0C 18                       or      al,18h   ;ENABLE AUTO INC ON
                                                   READ AND WRITE
010E 26:A3 7D00                  mov     es:(HSTCTRL),ax
0112 26:C7 06 7E00 0000          mov     word ptr es:(HSTADRL),0000h
0119 26:C7 06 7F00 0400          mov     word ptr es:(HSTADRH),0400h
0120 26:C7 06 7000 0001          mov     word ptr es:(HSTDATA),0001h

0127 CD 20                       int     20h
                        start    endp
0129                    code     ends
                                 end     entpt
```

Figure 5-6. Example Code to Enable Shadow RAM

Note:

The SDB Debugger program uses an all-RAM system, and this code disables the onboard PROM and downloads the reset and interrupt vectors prior to execution.

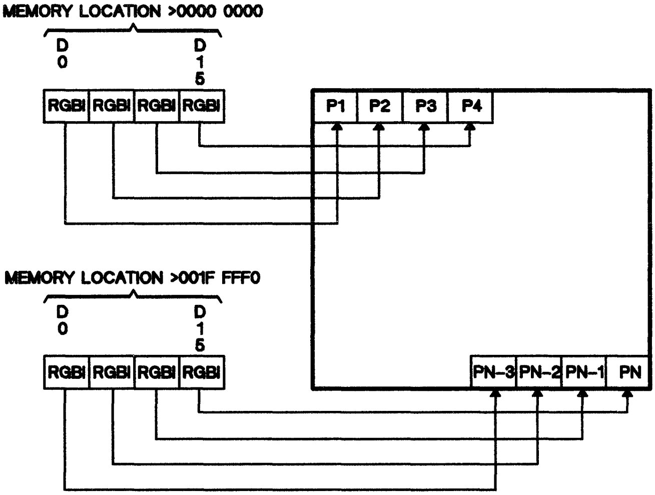


Figure 5-7. Pixel Memory Location and Screen Address

5.2.2 Video Memory Organization

The Frame Buffer (video display memory) is comprised of eight TM4161EV4 SIPs and occupies a contiguous block of TMS34010 local memory -- from >0000 0000 to >001F FFFF. This 256K bytes of multiport memory allows the SDB to contain up to 1024 x 512 pixels with 4 bits per pixel. The memory is organized in packed pixels, with four pixels sharing one 16-bit memory word. Figure 5-7 shows the correlation between the contents of two consecutive memory addresses and their corresponding pixels on the display screen.

The Frame Buffer is divided into two equal sections, a division transparent to the user. This division minimizes the power consumption of the VRAMs.

5.3 Color Palette

The SDB uses the TMS34070 Color Palette, a monolithic IC containing a color lookup table and providing three channels of analog video output for RGB-type CRTs. The Color Palette has sixteen internal registers, of which each contains:

- 12 bits of color information -- blue, red, green (bits D4 to D15)
- 2 bits for attributes (bits D1 & D2):

  - EXT (external pixel attribute to control external circuitry and support applications such as overlaying)

  - REPEAT (supports both rapid filling of screen areas with solid colors and real-time animation)

- 2 "don't care" bits (bits D0 & D3).

Figure 5-8 illustrates the organization of these bits into a data word.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
BLUE MSB	BLUE	BLUE	BLUE LSB	RED MSB	RED	RED	RED LSB	GREEN MSB	GREEN	GREEN	GREEN LSB		EXT	REPEAT	

Figure 5-8. Color-Palette Internal Register Format

The Palette's two modes of operation on the SDB are set using jumper W3 (shown in Figure 2-2). The modes define two different times when the color lookup table should be loaded from video memory:

- W3 set 2-3: TMS34070 in line-load mode (load table prior to start of individual scan line)
- W3 set 2-1: TMS34070 in frame-load mode (load table prior to start of each frame).

5.3.1 Color Palette Frame Load Mode

The TMS34070 Palette is unique in that it loads its 12-bit color lookup table directly from the Frame Buffer. In line-load mode, the first line of information contains the color lookup table data. This line is automatically blanked so that no appear on the screen while the Palette is loading. This loading is accomplished by the TMS34070 monitoring the Blank output of the TMS34010. When Vertical Blanking goes active, the THS34070 loads the first line as color information, while it simultaneously blanks this line. All further horizontal lines are displayed. Figure 5-9 illustrates how the Frame Buffer is loaded into the TMS34070 in frame load mode.

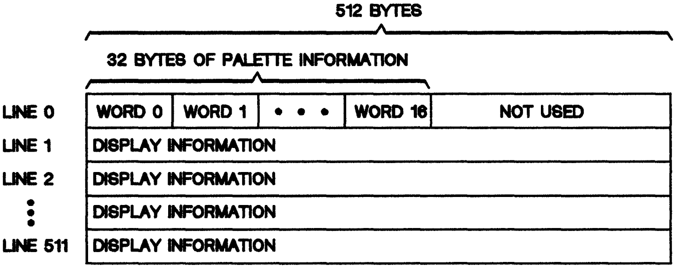
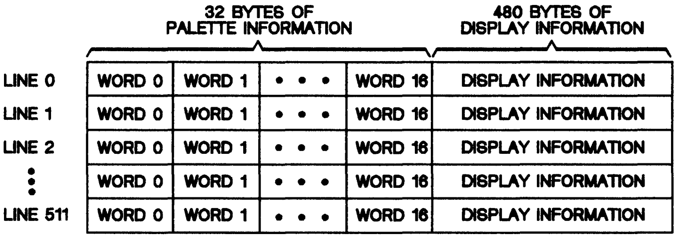


Figure 5-9. Loading Frame Buffer in Frame-Load Mode

5.3.2 Color Palette Line Load Mode

The unique line-load feature of the TMS34070 Palette allows the color lookup table to be reloaded on every horizontal line. This allows the possibility of displaying 16 colors on every horizontal line. The first 32 bytes of the 512-byte horizontal width of the Frame Buffer is used to load the color lookup table in the TMS34070. The remaining 480 bytes can be used for display information, allowing the maximum resolution in this mode to be 960 by 512 pixels with 4 bits per pixel. The TMS34070 automatically blanks the display during the loading of the Palette in the line-load mode. Figure 5-10 illustrates how the Frame Buffer is loaded into the TMS34070 in line-load mode.



32 BYTES OF PALETTE INFORMATION

480 BYTES OF DISPLAY INFORMATION

LINE 0	WORD 0	WORD 1	• • •	WORD 16	DISPLAY INFORMATION
LINE 1	WORD 0	WORD 1	• • •	WORD 16	DISPLAY INFORMATION
LINE 2	WORD 0	WORD 1	• • •	WORD 16	DISPLAY INFORMATION
⋮	WORD 0	WORD 1	• • •	WORD 16	DISPLAY INFORMATION
LINE 511	WORD 0	WORD 1	• • •	WORD 16	DISPLAY INFORMATION

Figure 5-10. Loading Frame Buffer in Line-Load Mode

5.3.3 Color Palette Code

5.3.3.1 Initialize Registers (init_palet function)

This function initializes the 16 color palette registers to their default values. It calls the function to set palette registers which is described in Section 5.3.3.2.

```
/* Declare external functions */
extern void setall_palet();

/* Default color palette values to be loaded at initialization */
short defpalet[16! = {

0x0000, 0xF000, 0x00F0, 0xF0F0, 0x0F00, 0xFF00, 0x0FF0, 0xFFF0,
0x0AF0, 0x0900, 0xFA70, 0xF4A0, 0x17B0, 0x6660, 0x9990, 0xBBB0
};

void init_palet()
{
   setall_palet(defpalet, 0xFFFF, 480, 0);
}
```

Description of arguments being passed (last line above):

- `defpalet = short paletreg[16]` = values for 16 color palette registers.
- `0xFFFF = int regmask` = mask indicating which registers to load.
- `480 = int nlines =` number of lines over which palette is affected.
- `0 = int ystart` = y value at start of affected area of screen.

5.3.3.2 Set Palette Registers (setall_palet function)

This function sets the multiple color palette registers. The following definitions apply:

- Variable nlines gives the number of lines affected by the palette change.
- Variable ystart gives the y value of the first line affected. The y value is specified relative to the xy origin located in the top left corner of the screen.
- Only registers corresponding to one values in the 16 LSBs of the register-load-enable mask are loaded. For example, a mask value of >00000017 would load only registers 0, 1, 2 and 4 from paletreg[] members 0, 1, 2 and 4.

Each 16-bit palette register value is loaded into memory as follows:

MSB	15 12	11 8	7 4	3 0	LSB
	BLU	GRN	RED	ATT	

This code is called by the previous register-initialization routine in Section 5.3.3.1.

```
/*---------------------------------------------------------------
*   This function is designed to be called from a
*   GSP-C program.
*---------------------------------------------------------------
* Usage:  setall_palet(paletreg, regmask, nlines, ystart);
*
* Description of stack arguments:
*   short paletreg[16]; /* values for 16 color palette regs*/
*   int regmask;  /* mask indicating which registers to
*        load */
*   int nlines;  /* number of lines over which palette is
*        affected */
*   int ystart;  /* y value at start of affected area of
*        screen */
*
* Returned in register A8:  Void (undefined).
* Registers altered:   A8
*
*---------------------------------------------------------------
;
  .title   'set all palette'
  .file    'setall_p.asm'
;
; --Declare Global Function Name
;
  .globl    _setall_palet
;
; --Declare Constants
;
CONTROL        .set >C00000B0       ; I/O Register
PMASK          .set >C0000160       ; I/O Register
PPOP_MASK      .set >7C00           ; PPOP field in CONTROL
TRN_MASK       .set >0020           ; T field in CONTROL
STK            .set A14             ; parameter stack
```

```
; --Entry Point
;
_setall_palet:
  SETF   16,0,0                   ;Set FS0=16 and FE0=0
  MMTM   SP,A0,A1,A2,A3           ;Save old A-file regs
  MMTM   SP,B0,B2,B4,B5,B6,B7,B9  ;Save old B-file regs
* Pop four arguments from stack.
  MOVE   *-STK,A0,1          ;Pop palet_array pointer
  MOVE   A0,B0               ;Copy array pointer to B file
  MOVE   *-STK,A0,1          ;Pop reg_mask from stack
  ZEXT   A0,0                ;Use only 16 LSBs of mask
  MOVE   *-STK,A1,1          ;Pop n_lines from stack
  MOVE   *-STK,A8,1          ;Pop ystart from stack
* Convert y values n_lines and ystart to 16 MSBs
* of x-y addresses.
  SLL   16,A1                ;Shift n_lines to 16 MSBs
  ADDK   4,A1                ;Palet reg. width = 4 pixels
  MOVE   A1,B7               ;Load DYDX = (4,n_lines)
  SLL   16,A8                ;Start address = (0,ystart)
* Move X-Y offset to start of frame buffer to allow
* palette access.
  CLR   B4                   ;OFFSET = VRAM base address
* Set window to area containing palette load data.
  CLR   B5                   ;Set WSTART = (0,0)
  MOVI   479*65536+63,B6     ;Set WEND=(63,479)
* Enable all color planes.
  MOVE   @PMASK,A3,1         ;Save old plane mask
  CLR   A1
  MOVE   A1,@PMASK,1         ;Enable all planes
* Set pixel processing = replace.  Turn off transparency.
  MOVE   @CONTROL,A1,0       ;Read CONTROL I/O register
  MOVE   A1,A2               ;Make 2nd copy
  ANDNI   PPOP_MASK+TRN_MASK,A2  ;Set PPOP=0 and T=0.
  MOVE   A2,@CONTROL,0       ;Load new CONTROL register
  MOVE   @CONTROL,A2,0       ;Read it back to be safe!
* Load next palette register over n_lines starting at ystart.
NXTREG:
  SRL    1,A0                ;Examine next mask bit
  JRNC   NOLOAD              ;If mask bit = 0, do not load
  MOVE   *B0+,B9,0           ;Set COLOR1 = next register
  MOVE   A8,B2               ;Set DADDR for next register
  ADDK   4,A8                ;Bump xstart to next register
  FILL   XY                  ;Load register multiple lines
  JRUC   NXTREG              ;Repeat loop
NOLOAD:
  JRZ    DONE                ;Done when mask is all zeros
  ADDK   16,B0               ;Skip over palet_array[n]
  ADDK   4,A8                ;Bump xstart to next register
  JRUC   NXTREG              ;Repeat loop
* Restore previous contents of A- and B-file registers
* where required.
DONE:
  MOVE   A1,@CONTROL,0       ;Restore CONTROL register
  MOVE   A3,@PMASK,1         ;Restore plane mask
  MMFM   SP,B0,B2,B4,B5,B6,B7,B9     ;Restore old B-file regs
  MMFM   SP,A0,A1,A2,A3      ;Restore old A-file registers
  RETS   2                   ;Return
  .end
```

5.4 Serial Port

The Software Devlopment Board is equipped with an RS232C serial output port (J3 shown in Figure 2-2). The USART (universal synchronous/asynchronous receiver transmitter) used on the SDB is the Standard Microsystems Corporation COM 2651.

This port uses a standard male DB25 connector J3 located on the board edge. Its main use is input and output to the SDB for devices such as mice and digitizing tablets. Table 5-2 lists the pinouts for the port.

Table 5-2. Pinout for EIA Port

OUTPUT PORT INTERFACE	
PIN NUMBER	SIGNAL DESCRIPTION
2	Transmit Data (TXD)
3	Receive Data (RXD)
4	Requset to Send (RTS)
5	Clear to Send (CTS)
6	Data Set Ready (DSR)
7	Ground
8	Data Carrier Detect (DCD)
20	Data Terminal Ready (DTR)

The USART is mapped into the TMS34010's local memory space between >0200 0000 and >022F FFFF. Table 5-3 contains the addresses for the internal registers of the COM 2651. Note there are different addresses for reading and writing to the part.

Table 5-3. COM 2651 USART Internal Registers

ADDRESS	FUNCTION
>0200 0000	Read Receive Holding Register
>0200 1000	Read Status Register
>0200 2000	Read Mode Registers 1 and 2
>0200 3000	Read Command Register
>0220 0000	Write to Transmit Holding Register
>0220 1000	Write to SYN1/SYN2/DLE Registers
>0220 2000	Write to Mode Registers 1 and 2
>0220 3000	Write to Command Register

The device used to control this port, the COM 2651, is described in its data sheet in Appendix A. A description of this port is given in Table 6-2 on page 6-16.

5.5 Video Connector (Port J4)

The Software Development Board has a single DB9 video output connector which can be configured for a variety of digital and analog monitors. This is port J4 shown in Figure 2-2 on page 2-4.

As shipped the connector comes configured for the IBM Professional Graphics Display, but can be reconfigured by the onboard jumpers shown in Figure 2-2. The Analog output supports 1-V peak-to-peak with 75-ohm drive capability, while the digital output supports an RGBI interface.

The output connector can be configured to drive both seperate horizontal and vertical syncs in both polarities along with composite syncs of both polarities.

Table 5-4 describes the jumper settings for analog monitors such as the IBM Professional Graphics Display or Princeton Graphics SR-12P. Table 5-5 lists the jumper settings for digital monitors such as the TI Professional Graphics Monitor.

Table 5-4. Analog-Monitor Jumper Settings for Connector J4

Pin Number	Signal Description	Jumper Settings
1	Red	W9: 8 to 15
2	Green	W9: 9 to 16
3	Blue	W9: 10 to 17
4	CSYNC	W9: 11 to 18, W8: 2 to 3, W6: 2 to 3
5	MODE (high)	W9: 12 to 19, W5: 2 to 3, W4: 2 to 3
6	Ground	
7	Ground	
8	Ground	W9: 13 to 20
9	Ground	W9: 14 to 21

Notes:

1. For monitors such as IBM Professional Graphics Display, Princeton Graphics SR-12P, or Equivalent.
2. Resolution: 640 by 480
3. U30 Oscillator frequency: 25 MHz

Table 5-5. Digital-Monitor Jumper Settings for J4

Pin Number	Signal Description	Jumper Settings
1	Ground	W9: 1 to 8
2	Ground	W9: 2 to 9
3	Red	W9: 3 to 10
4	Green	W9: 4 to 11
5	Blue	W9: 5 to 12
6	Ground	W7: 2 to 3
7	Reserved	
8	HSYNC	W9: 6 to 13, W6: 2 to 3, W8: 1 to 2
9	VSYNC	W9: 7 to 14, W4: 1 to 2, W5: 1 to 2

Notes:

1. For monitors such as the TI Professional Graphics Monitor.
2. Resolution: 720 by 300
3. U30 Oscillator frequency: 18.432 Megahertz

5.6 Interrupts, To Host

Jumper W2 (shown in Figure 2-2) can be configured to select the interrupt to the host PC according to which PC is used:

- interrupt level 3 for the IBM PC, or
- interrupt level 2 for the TI PC.

There are two sources for the interrupt:

- the HINTL line for normal operation, and
- the EMUACKL line which is used when the SDB is a target for the TI TMS34010 XDS Emulator.

Figure 5-11 shows how to configure the interrupt jumper for both the IBM and TI PC.

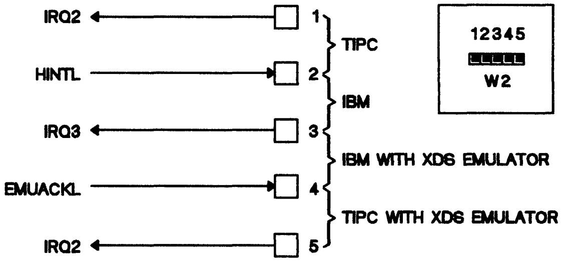


Figure 5-11. IBM/TI PC Interrupt Selection at Jumper W2

The interrupt map for the TMS34010 is shown in Figure 6-15 on page 6-25.

5.7 Expansion Bus

The Software Devlopment Board operates in the TI PC or IBM PC expansion bus (and into the expansion connector of most IBM-compatible machines). The signals that the board uses are shown in Table 5-6. Note that the only difference between the IBM PC and the TI PC for the Software Development Board is the interrupt pins. This difference is handled by a jumper option, described in Section 5.6.

Table 5-6. SDB Pinouts at P3

Signal Name	Pin Number	Signal Name	Pin Number
POWER SIGNALS			
+5 V	B3, B29	-12 V	B7
Ground	B1, B10, B31	+12 V	B9
CONTROL SIGNALS			
$\overline{\text{RDY}}$	A10		
RESET	B2		
$\overline{\text{MEMW}}$	B11	$\overline{\text{IRQ2}}$	B24
$\overline{\text{MEMR}}$	B12	$\overline{\text{IRQ3}}$	B25
DACK0	B19	CLOCK	B10
OSC	B30		
DATA SIGNALS			
D0	A9	D4	A5
D1	A8	D5	A4
D2	A7	D6	A3
D3	A6	D7	A2
ADDRESS SIGNALS			
A0	A31	A10	A21
A1	A30	A11	A20
A2	A29	A12	A19
A3	A28	A13	A18
A4	A27	A14	A17
A5	A26	A15	A16
A6	A25	A16	A15
A7	A24	A17	A14
A8	A23	A18	A13
A9	A22	A19	A12

5.8 Power Consumption

Note that the Software Develpoment Board requires a substainal amount of power. It is recommended that it be installed only in a PC with a minimum supply of 145 watts.

Table 5-7. Power Requirements

SUPPLY VOLTAGE	TYPICAL CURRENT	
+5 Volts	1.40	A
+12 Volts	20	mA
-12 Volts	20	mA

6. Theory of Operation

The following sections explain the theory of operation of the TMS34010 Software Development Board. The theory of operation is divided into six parts:

6.1 PC Bus to TMS34010 (GSP) Interface

The TMS34010 Graphics System Processor (GSP) has a 16-bit wide host port directly onboard which allows the processor to be easily interfaced to any host bus. The host port has four registers HSTDATA, HSTCTL, HSTADRL, and HSTADRH:

- HSTDATA Register is a 16-bit wide register used to transfer data:
  - from the host to the TMS34010's local memory bus, and
  - from the TMS34010's local memory bus to the Host's memory bus.
- HSTCTL register controls various functions within the TMS34010 such as interrupts, cache control, and messages.
- HSTADRH and HSTADRL are two 16-bit registers that comprise the 32-bit address pointer register used for indirect communication to the TMS34010's local bus.

These registers are read from or written to by 16 data lines and 9 control lines which comprise the TMS34010 host interface. The 16 host port data lines (HAD0-HAD15) are connected to the the PC's bus through a 74ALS245 bi-directional buffer. Since the PC's bus is only 8 bits wide and the TMS34010's host port is 16 bits wide, the upper and lower data strobes ($\overline{\text{HUDS}}$ and $\overline{\text{HLDS}}$) of the TMS34010 are connected to the least significant address line of the PC bus. This allows byte access versus word access from the host port. When accessing bytes, the data lines from the bus are connected to both the high and low byte of the TMS34010's host port as shown in Figure 6-1. The direction of the buffer is determined by the bus memory read line ($\overline{\text{MRDL}}$) and enabled on the Host port chip select ($\overline{\text{HCS}}$).

The host ports $\overline{\text{HWRITE}}$ and $\overline{\text{HREAD}}$ signals are connected to the PC bus's memory read ($\overline{\text{MRDL}}$) and memory write ($\overline{\text{MWRL}}$) signals after they are buffered by a 74ALS541 Octal Buffer.

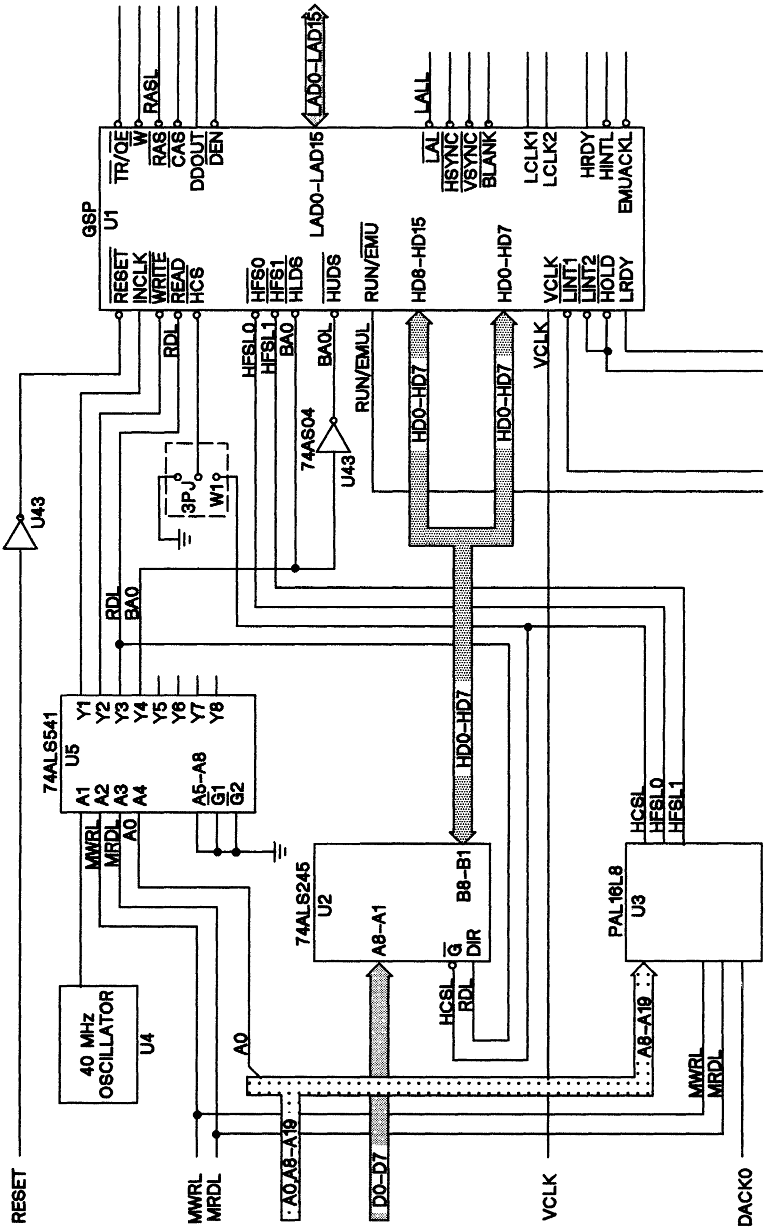


Figure 6-1. Data/Address Bus to TMS34010 Interface

The four Host port registers have been placed into the host PC's memory map. U3 -- a TIBPAL16L8 (a programmable array logic device) -- decodes the Host PC's address lines and generates the host chip select ($\overline{\text{HCS}}$) and the two host function selects HFS0 and HFS1.

Table 6-1 shows the operations performed on the host port when the appropriate control signals are applied. Figure 6-2 shows the memory map for the four host port registers. The HSTDATA register is the largest to allow the autoincrementing feature of the host port to be used on uploads and downloads to and from the host port.

PAL U3 is connected to the address lines A19-A8 of the PC bus and the control signal $\overline{\text{MRDL}}$, $\overline{\text{MWRL}}$, and DACK0. Whenever the Host processor accesses memory from >C7000 to >C7FFF on the bus, a host chip select is generated when $\overline{\text{MRDL}}$ goes active low or $\overline{\text{MWRL}}$ goes active low if DACK0 is active high. DACK0 is used to indicate a memory operation or a DMA operation. By ANDing DACK0 into the equations, the PAL prevents refresh cycles from effecting the Software Development Board. PAL equations are shown:

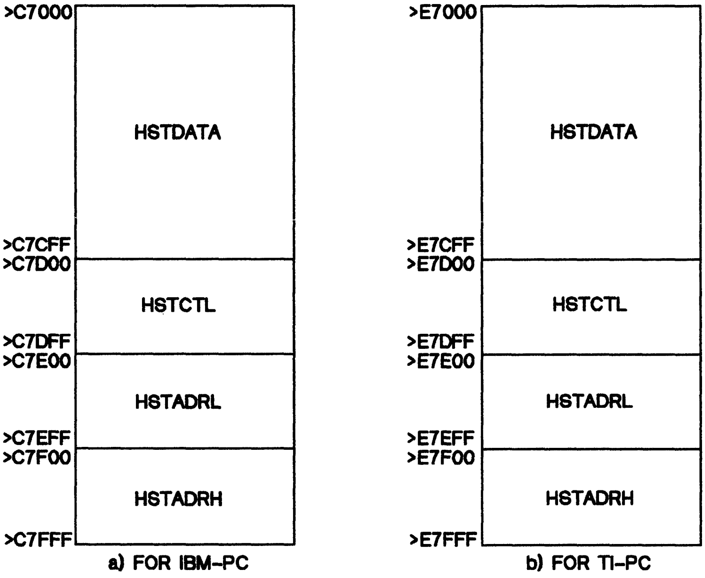
- for the IBM PC and AT host port decode PAL in Figure 6-3,
- for the TI PC in Figure 6-4.

Depending on the memory range accessed between >C7000 to >C7FFF, the appropriate host function selects are generated by the PAL in U3 as shown in Figure 6-2 and Table 6-1.

Table 6-1. Signal Inputs to Select Host Functions

† $\overline{\text{HCS}}$	$\overline{\text{HUDS}}$	$\overline{\text{HLDS}}$	$\overline{\text{HREAD}}$	† $\overline{\text{HWRITE}}$	† $\overline{\text{HFS0}}$	$\overline{\text{HFS1}}$	OPERATION
H	X	X	X	X	X	X	No Operation
L	L	H	L	H	L	L	HSTADRL read high byte (HAD8-HAD15)
L	H	L	L	H	L	L	HSTADRL read high byte (HAD0-HAD7)
L	L	H	H	L	L	L	HSTADRL write high byte (HAD8-HAD15)
L	H	L	H	L	L	L	HSTADRL write high byte (HAD0-HAD7)
L	L	H	L	H	L	H	HSTADRH read high byte (HAD8-HAD15)
L	H	L	L	H	L	H	HSTADRH read high byte (HAD0-HAD7)
L	L	H	H	L	L	H	HSTADRH write high byte (HAD8-HAD15)
L	H	L	H	L	L	H	HSTADRH write high byte (HAD0-HAD7)
L	L	H	L	H	H	L	HSTDATA read high byte (HAD8-HAD15)
L	H	L	L	H	H	L	HSTDATA read high byte (HAD0-HAD7)
L	L	H	H	L	H	L	HSTDATA write high byte (HAD8-HAD15)
L	H	L	H	L	H	L	HSTDATA write high byte (HAD0-HAD7)
L	L	H	L	H	H	H	HSTCTL read high byte (HAD8-HAD15)
L	H	L	L	H	H	H	HSTCTL read high byte (HAD0-HAD7)
L	L	H	H	L	H	H	HSTCTL write high byte (HAD8-HAD15)
L	H	L	H	L	H	H	HSTCTL write high byte (HAD0-HAD7)

†These signals come from the PAL at U3.



a) FOR IBM-PC

b) FOR TI-PC

Figure 6-2. Memory Map for Four Host Port Registers

```
module tmsGSPU3IBM
title 'GSP HOST INTERFACE CONTROL FOR IBM AT >C7000->C7FFF
Designer Ron Peterson Texas Instruments Inc. APRIL 9, 1986'

               U3IBM    device 'P16L8';
 A11,A12,A13,A14,A15,A16,A17,A18,A19           pin 1,2,3,4,5,6,7,8,9;
 A10,A9,A8,W,RD,DACKO                          pin 11,13,14,15,16,17;
 HFS1,HFSO,HCS                                 pin 12,18,19;

 L,H,X =    0,1,.X.;
 INPUTS =   [DACKO,W,RD,A19,A18,A17,A16,A15,A14,A13,A12,A11,A10,A9,A8];
 OUTPUTS =  [HCS,HFS1,HFSO];

equations


 !HCS = ((A19&A18&!A17&!A16&!A15&A14&A13&A12&!W&DACKO)#
         (A19&A18&!A17&!A16&!A15&A14&A13&A12&!RD&DACKO));

 !HFS1 =((A19&A18&!A17&!A16&!A15&A14&A13&A12&A11&A10&A9&!A8)#     "HSTADRL"
         (A19&A18&!A17&!A16&!A15&A14&A13&A12&A11&A10&A9&A8));     "HSTADRH"


 !HFSO =((A19&A18&!A17&!A16&!A15&A14&A13&A12&!A11)#               "HDATA"
         (A19&A18&!A17&!A16&!A15&A14&A13&A12&A11&!A10)#           "HDATA"
         (A19&A18&!A17&!A16&!A15&A14&A13&A12&A11&A10&!A9&!A8)#    "HDATA"
         (A19&A18&!A17&!A16&!A15&A14&A13&A12&A11&A10&A9&!A8));    "HSTADRL"

" HDATA      >C7000->C7CFF"
" HCNTL      >C7D00->C7DFF"
" HSTADRL    >C7E00->C7EFF"
" HSTADRH    >C7F00->C7FFF"

test_vectors (INPUTS -> OUTPUTS)

" D  W  R  A  A  A  A  A  A  A  A  A  A  A  A  A      H  H  H  "
" A     D  1  1  1  1  1  1  1  1  1  1  1  9  8      C  F  F  "
" C        9  8  7  6  5  4  3  2  1  0               S  S  S  "
" K                                                      1  0  "
" 0

[ L, X, X, X, X, X, X, X, X, X, X, X, X, X, X] -> [H, X, X];
[ H, L, H, H, H, L, L, L, H, H, H, L, X, X, X] -> [L, H, L]; "HDATA"
[ H, L, H, H, H, L, L, L, H, H, H, H, L, X, X] -> [L, H, L]; "HDATA"
[ H, L, H, H, H, L, L, L, H, H, H, H, H, L, L] -> [L, H, L]; "HDATA"
[ H, H, L, H, H, L, L, L, H, H, H, L, X, X, X] -> [L, H, L]; "HDATA"
[ H, H, L, H, H, L, L, L, H, H, H, H, L, X, X] -> [L, H, L]; "HDATA"
[ H, H, L, H, H, L, L, L, H, H, H, H, H, L, L] -> [L, H, L]; "HDATA"
[ H, L, H, H, H, L, L, L, H, H, H, H, H, L, H] -> [L, H, H]; "HCNTL"
[ H, H, L, H, H, L, L, L, H, H, H, H, H, L, H] -> [L, H, H]; "HCNTL"
[ H, L, H, H, H, L, L, L, H, H, H, H, H, H, L] -> [L, L, L]; "HSTADRL"
[ H, H, L, H, H, L, L, L, H, H, H, H, H, H, L] -> [L, L, L]; "HSTADRL"
[ H, L, H, H, H, L, L, L, H, H, H, H, H, H, H] -> [L, L, H]; "HSTADRH"
[ H, H, L, H, H, L, L, L, H, H, H, H, H, H, H] -> [L, L, H]; "HSTADRH"

end tmsGSPU3IBM
```

Figure 6-3. Equations for IBM PC and AT Host Port Decode PAL

```
module tmsGSPU3TI
title 'GSP HOST INTERFACE CONTROL FOR TI AT >E7000->E7FFF
Designer Ron Peterson Texas Instruments Inc. APRIL 9, 1986'

               U3TI  device 'P16L8';
 A11,A12,A13,A14,A15,A16,A17,A18,A19           pin 1,2,3,4,5,6,7,8,9;
 A10,A9,A8,W,RD,DACK0                          pin 11,13,14,15,16,17;
 HFS1,HFS0,HCS                                 pin 12,18,19;

L,H,X =   0,1,.X.;
INPUTS =  [DACK0,W,RD,A19,A18,A17,A16,A15,A14,A13,A12,A11,A10,A9,A8];
OUTPUTS = [HCS,HFS1,HFS0];

equations

 !HCS = ((A19&A18&A17&!A16&!A15&A14&A13&A12&!W&DACK0)#
         (A19&A18&A17&!A16&!A15&A14&A13&A12&!RD&DACK0));

 !HFS1 =((A19&A18&A17&!A16&!A15&A14&A13&A12&A11&A10&A9&!A8)# "HSTADRL"
         (A19&A18&A17&!A16&!A15&A14&A13&A12&A11&A10&A9&A8)); "HSTADRH"

 !HFS0 =((A19&A18&A17&!A16&!A15&A14&A13&A12&!A11)#           "HDATA"
         (A19&A18&A17&!A16&!A15&A14&A13&A12&A11&!A10)#       "HDATA"
         (A19&A18&A17&!A16&!A15&A14&A13&A12&A11&A10&!A9&!A8)# "HDATA"
         (A19&A18&A17&!A16&!A15&A14&A13&A12&A11&A10&A9&!A8)); "HSTADRL"

" HDATA   >E7000->E7CFF"
" HCNTL   >E7D00->E7DFF"
" HSTADRL >E7E00->E7EFF"
" HSTADRH >E7F00->E7FFF"

test_vectors (INPUTS -> OUTPUTS)

  "D W  R  A  A  A  A  A  A  A  A  A  A  A  A      H  H  H  "
  "A    D  1  1  1  1  1  1  1  1  1  1  9  8      C  F  F  "
  "C       9  8  7  6  5  4  3  2  1  0            S  S  S  "
  "K                                                  1  0  "
  "0

  [L,X, X, X, X, X, X, X, X, X, X, X, X, X, X] -> [H, X, X];
  [H,L, H, H, H, H, L, L, H, H, H, L, X, X, X] -> [L, H, L]; "HDATA"
  [H,L, H, H, H, H, L, L, H, H, H, H, L, X, X] -> [L, H, L]; "HDATA"
  [H,L, H, H, H, H, L, L, H, H, H, H, H, L, L] -> [L, H, L]; "HDATA"
  [H,H, L, H, H, H, L, L, H, H, H, L, X, X, X] -> [L, H, L]; "HDATA"
  [H,H, L, H, H, H, L, L, H, H, H, H, L, X, X] -> [L, H, L]; "HDATA"
  [H,H, L, H, H, H, L, L, H, H, H, H, H, L, L] -> [L, H, L]; "HDATA"
  [H,L, H, H, H, H, L, L, H, H, H, H, H, L, H] -> [L, H, H]; "HCNTL"
  [H,H, L, H, H, H, L, L, H, H, H, H, H, L, H] -> [L, H, H]; "HCNTL"
  [H,L, H, H, H, H, L, L, H, H, H, H, H, H, L] -> [L, L, L]; "HSTADRL"
  [H,H, L, H, H, H, L, L, H, H, H, H, H, H, L] -> [L, L, L]; "HSTADRL"
  [H,L, H, H, H, H, L, L, H, H, H, H, H, H, H] -> [L, L, H]; "HSTADRH"
  [H,H, L, H, H, H, L, L, H, H, H, H, H, H, H] -> [L, L, H]; "HSTADRH"

 end tmsGSPU3TI
```

Figure 6-4. Equations for TI PC Host Port Decode PAL

6.2 TMS34010 to Memory Interface

The TMS34010 accesses local memory through a 16-bit tri-muxed bus. The TMS34010 supplies the the appropriate signals to interface easily to DRAM's and VRAM's. The device can also interface quite easily to other static devices such as RAMS, ROMS, and peripherals.

The TMS34010 has an onboard instruction cache and memory controller. The CPU runs independently of the memory controller unless it is forced to wait for a memory operation to finish before further execution can continue. When the TMS34010's memory controller does a memory access, the tri-muxed bus (1) first outputs the row address on LAD15-LAD0, then (2) outputs the column address and then (3) reads or writes data. The TMS34010 supplies:

- a row address strobe ($\overline{\text{RAS}}$),
- a column address strobe ($\overline{\text{CAS}}$),
- two local clocks (LCLK1 and LCLK2),
- an address latch enable ($\overline{\text{LAL}}$),
- a write strobe ($\overline{\text{W}}$),
- shift register transfer and output enable ($\overline{\text{TR/QE}}$),
- data direction output (DDOUT), and
- data enable output ($\overline{\text{DEN}}$).

The part also has a local ready (LRDY) to allow wait states when accessing local memory.

The Software Development Board has a variety of memory devices on the TMS34010's local bus. These include:

- 512K bytes of dynamic RAM (DRAM),
- 256K bytes of video RAM (VRAM),
- two optional bipolar programmable read only memories (PROMS),
- a universal synchronous/asynchronous receiver transmitter (USART).

When the TMS34010's memory controller starts a memory cycle (state Q1 in Figure 6-5), the row address is output on the LAD15-LAD0 pins. The LAD7-LAD0 outputs are input into a 74AS573 octal D transparent latch U10. The enable input to U10 is connected to the $\overline{\text{LAL}}$ signal. Since $\overline{\text{LAL}}$ is high during this part of the cycle (states Q1-Q3 in Figure 6-5) the latches are in transparent state causing the outputs to follow the inputs and the row addresses propagate out of the latch to the memory array.

1) Outputs LAD15-LAD13, LAD9, LAD8, LAD1, LAD0, and $\overline{\text{TR/QE}}$ are latched into a 74ALS573 octal D transparent latch U9 at the end of Q2 by the XLATCH signal. XLATCH is created by ANDing the $\overline{\text{RAS}}$ output and the $\overline{\text{LAL}}$ output with a 74AS11 three input AND gate.
2) $\overline{\text{RAS}}$ then falls after state Q2. This strobes the row addresses into the memory devices.
3) The LAD15-LAD0 now outputs the column addresses during state Q4. Address latch enable $\overline{\text{LAL}}$ now goes low during state Q4 latching the column address LAD7-LAD0 in latch U10.
4) The $\overline{\text{CAS}}$ output goes low in state Q5 and this strobes the column addresses into the memory devices.

Figure 6-6 shows the state of the LAD15-LAD0 pins when the row and column addresses are muxed out so that the lower LAD outputs can be buffered and directly driven to the memory array. Figure 6-7 shows how TMS34010 address bits LAD0 to LAD7 are latched at U10 for multiplex operations.

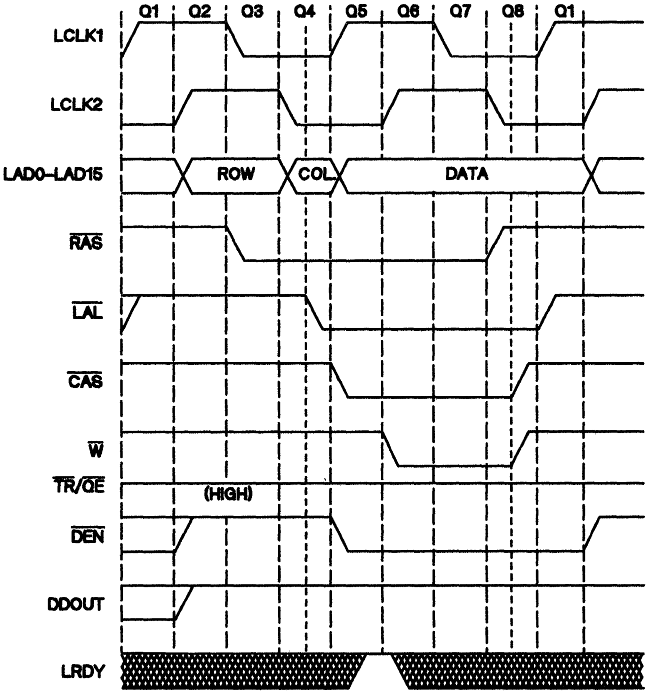
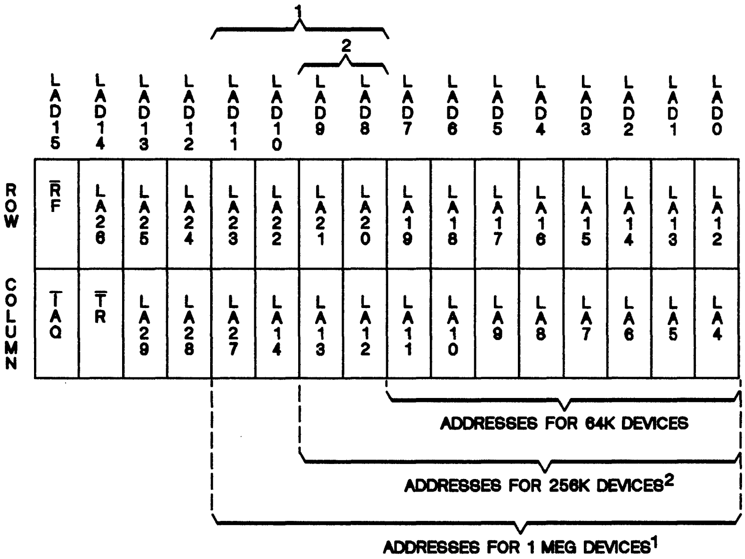


Figure 6-5. TMS34010 Memory Cycle Timing

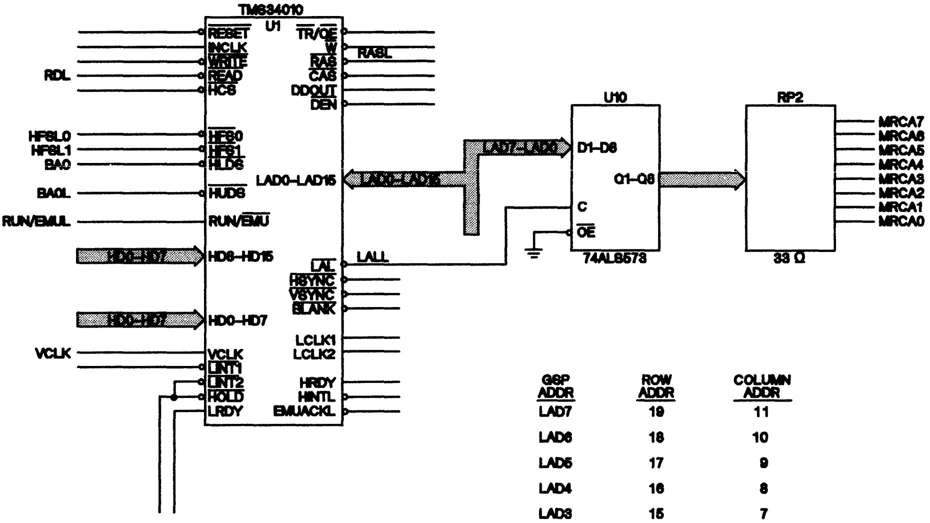


	LAD15	LAD14	LAD13	LAD12	LAD11	LAD10	LAD9	LAD8	LAD7	LAD6	LAD5	LAD4	LAD3	LAD2	LAD1	LAD0
ROW	$\overline{\mathrm{R}}$F	LA26	LA25	LA24	LA23	LA22	LA21	LA20	LA19	LA18	LA17	LA16	LA15	LA14	LA13	LA12
COLUMN	$\overline{\mathrm{T}}$AQ	$\overline{\mathrm{T}}$R	LA29	LA28	LA27	LA14	LA13	LA12	LA11	LA10	LA9	LA8	LA7	LA6	LA5	LA4

1
2
ADDRESSES FOR 64K DEVICES
ADDRESSES FOR 256K DEVICES[2]
ADDRESSES FOR 1 MEG DEVICES[1]

NOTES: 1. REQUIRES 4 BITS EXTERNALLY MUXED FOR LA23-LA20 FOR MEMORY ADDRESSES MRCA8 AND MRCA9.
2. REQUIRES 2 BITS EXTERNALLY MUXED FOR LA20 & LA21 FOR MEMORY ADDRESS MRCA8.

Figure 6-6. LAD15-LAD0 States To Drive Memory Array



GSP ADDR	ROW ADDR	COLUMN ADDR
LAD7	19	11
LAD6	18	10
LAD5	17	9
LAD4	16	8
LAD3	15	7
LAD2	14	6
LAD1	13	5
LAD0	12	4

Figure 6-7. LAD7-LAD0 Multiplexing Latch Circuitry

The bits latched into U9 during state Q2 by the XLATCH signal are used in determining which device should be selected. These outputs are labeled LA26, LA25, LA21, LA20 LA13, LA12, XFRCYCL, and REFCYCL. (The LA bits are the local address bits of the GPS's address bus.) XFRCYCL indicates if the access is a shift register transfer cycle, and REFCYCL dictates if a refresh cycle has been requested. These bits are driven into U11 a TIBPAL20L10 programmable array logic device used for device decoding. Figure 6-8 contains the equations for this PAL.

```
module tmsU11
title 'GSP LOCAL BUS DECODE REV.1
Designer Ron Peterson Texas Instruments Inc. April 21, 1986'

              U11   device 'P20L10';
LCLK1,LCLK2,REFCYC,XFRCYC,RASL,LAL,TRQE             pin 1,2,3,4,5,6,7;
LA26,LA25,LA21,LA20,RESET                           pin 8,9,10,11,13;
RAMOE,RAMEN,RAMOFF,MRCA8,UARTCS,ROMCS               pin 14,15,16,17,18,19;
DMRAS0,DMRAS1,LMRAS,FLGCLK                          pin 20,21,22,23;

equations

!FLGCLK =         (!XFRCYC&!RASL);

!LMRAS =          ((!RASL&LA26&LA25)#
                   (!RASL&!REFCYC));

!DMRAS1 =         ((!RASL&!LA26&!LA25&LA20)#
                   (!RASL&!REFCYC)#
                   (!RASL&!XFRCYC));

!DMRAS0 =         ((!RASL&!LA26&!LA25&!LA20)#
                   (!RASL&!REFCYC)#
                   (!RASL&!XFRCYC));

!ROMCS =          ((LA26&LA25&LA21&LA20&!RAMEN&REFCYC));

!MRCA8 =          ((!LA21&LCLK2&LAL)#
                   (LA20&!LCLK2)#
                   (!MRCA8&!LAL));

!UARTCS =         ((!RASL&!LA26&LA25&REFCYC));

!RAMOFF =         ((RESET)#
                   (RAMEN));

!RAMEN =          ((LA26&!LA25&!LA21&LA20&REFCYC&!RASL)#
                   (RAMOFF));

!RAMOE =          ((LA26&LA25&!RAMEN&!TRQE)#
                   (LA26&LA25&!LA21&!RAMEN&!TRQE)#
                   (LA26&LA25&!LA20&!RAMEN&!TRQE));

 end tmsU11
```

Figure 6-8. Equation for PAL U11

6.2.1 Accessing Local DRAM Memory

If the 512K bytes of system memory comprised of four TM4256EC4 SIPS are accessed, the LMRASL output of PAL U11 goes active. This memory is mapped from >FFC0 0000 to >FFFF FFFF. As discussed in Section 6.2, latch U10 supplies eight of the nine row and column address bits for these memory devices. The ninth address bit (for the row and column address) is supplied from the PAL U11. The TMS34010's two upper address bits are not output, and bits LA29-LA27 are not decoded for $\overline{\text{RAS}}$ selection because they are output at column time. Thus, the upper five bits are actually don't-care bits in selecting external memory devices in this design. When LA26 and LA25 are both high, output $\overline{\text{LMRAS}}$ is driven active low during the $\overline{\text{RAS}}$ interval as shown in Figure 6-9. When the REFCYC output from latch U9 is low and $\overline{\text{RAS}}$ is low, a refresh cycle is taking place and $\overline{\text{LMRAS}}$ again is driven low by PAL U11.

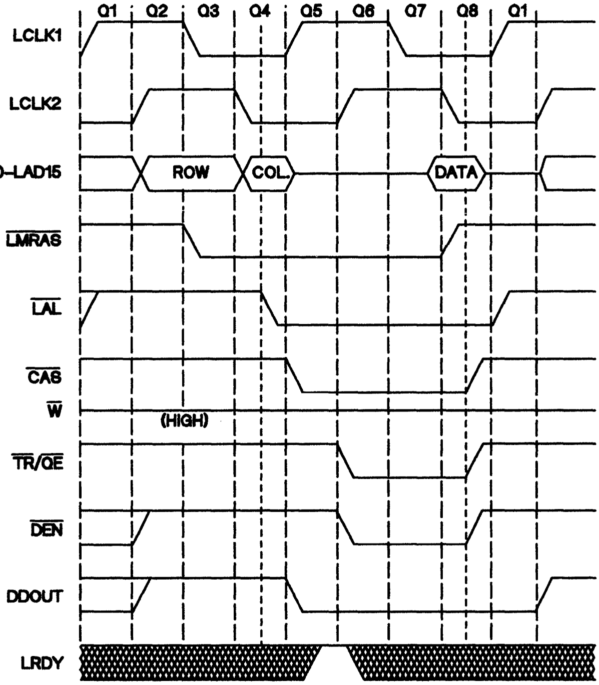


Figure 6-9. Local DRAM Access Timing

Data is enabled on read cycles through two 74ALS244's (U27 and U46) by signal RAMOE from PAL U11. RAMOE is a function of address and TRQE as shown in the PAL equations shown in Figure 6-8. A third term RAMEN has been added to prevent the buffers from enabling on the upper 128K of memory (>FFF0 0000 to >FFFF FFFF) since this section of memory is shadow RAM behind the onboard PROM. Section 6.2.2. discusses this in more detail. The data then is driven through the two 74ALS245 bi-directional buffers U7 and U8. U7 and U8 are enabled by the TMS34010's DEN signal and the direction is controlled by the TMS34010's DDOUT output. On write cycles the data is driven directly into the TM4256EC4 SIPS from the two 74ALS245's U7 and U8. The 74ALS244's are required to isolate the D inputs and Q outputs on the TM4256EC4 SIP memory devices since the TMS34010 does late write cycles (Write falls after CAS).

6.2.2 Accessing PROM and Enabling Shadow RAM

After power-on Reset, the SDB has 1K bytes of optional bipolar PROM enabled. This PROM consists of 2-TBP24S42 devices mapped at the upper portion of the TMS34010's memory space (>FFF0 0000 to >FFFF FFFF). The PROM has DRAM mapped behind it (i.e., they occupy the same area in memory) which can be enabled by writing a >0001 to memory location >0400 0000 in the TMS34010's memory space. Figure 6-10 shows the section of memory that the PROM and shadow RAM occupy in the TMS34010's local memory space. On power-on reset, an R-S latch constructed internally to PAL U11 is reset. The output of this latch is fed back around internally to PAL U11 and is used for decoding whether the PROM is accessed or the shadow RAM is enabled.

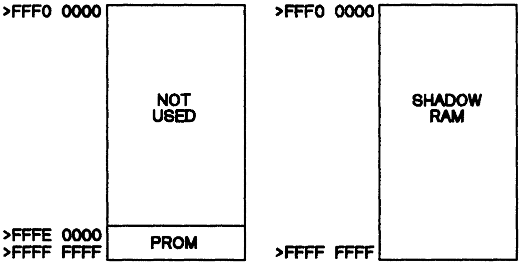


Figure 6-10. PROM Address Space and Shadow RAM

When the TMS34010 accesses memory from >FFF0 0000 to >FFFF FFFF, the decode PAL U11 selects the two PROMs since the R-S latch is still in the reset state. The shadow RAM is enabled by writing to location >0400 0000 with a >0001. This causes the R-S latch to be set, and it remains set until another power-on reset. Setting the R-S latch enables the shadow RAM buffers U27 and U42 and disables the PROM chip select for memory accesses in the range from >FFF0 0000 to >FFFF FFFF.

6.2.3 Accessing the USART

The Software Development Board has an on-board USART for serial communications to mice, digitizing tablets, and other serial devices. The board uses a Standard Microsytems COM2651 for serial communications. (A COM 2651 data sheet is in Appendix A.) The USART is a memory mapped peripheral placed in the TMS34010's local memory space from locations >0200 0000 to >0220 3FFF. The USART has an 8-bit data path and is connected to the lower byte (LAD7-LAD0) on the TMS34010.

When memory is selected between >0200 0000 to >0220 3FFF, decode PAL U11 selects the USART. Besides selecting the USART directly, the UARTCSL signal from PAL U11 is input into U12, a D flip-flop. This flip-flop latchs the USART chip select. The output of this flip-flop is then negatively ANDed with the UARTCSL by the 74AS32 OR-Gate U42 which generates one wait state for every USART access. Since the R-/W line to the USART needs to be set up prior to $\overline{CS}$ the LA21 address line is used to control this operation. Therefore, a different address is used for reading and for writing to the USART's registers. The addresses for the USART registers are shown in Section 6.2.3.

Table 6-2. USART Register Addresses

ADDRESS	FUNCTION
>0200 0000	Read Receive Holding Register
>0200 1000	Read Status Register
>0200 2000	Read Mode Registers 1 and 2
>0200 3000	Read Command Register
>0220 0000	Write to Transmit Holding Register
>0220 1000	Write to SYN1/SYN2/DLE Registers
>0220 2000	Write to Mode Registers 1 and 2
>0220 3000	Write to Command Register

The three outputs of the USART (TxD, $\overline{\text{RTS}}$, and $\overline{\text{DTR}}$) are driven through 75188 transmitters U40 to male DB25 connector J3. The four inputs (RxD, $\overline{\text{DSR}}$ $\overline{\text{CTS}}$, and $\overline{\text{DCD}}$) are driven by 75189 receivers at U37 which are sourced by the same DB25 connector J3.

The USART can generate an interrupt to the TMS34010 from the $\overline{\text{TxRDY}}$ or RxRDY pins on the USART. These pins are the inverted state of status bits in the USART's status register. They indicate that a byte of data has been transmitted or received by the USART and the device is ready for another operation. These outputs are negative ORed together by the 74AS11 and gate U45 to form an interrupt level 1 request to the TMS34010's local bus (Trap 1 at >FFFF FFC0 as shown in Figure 6-15 on page 6-25). This allows the USART to be an interrupt-driven peripheral.

6.3 TMS34010 to Frame Buffer Interface

The Software Development Board has 256K bytes of video memory storage (VRAMS) consisting of eight TM4161EV4 SIPs. This allows a maximum resolution of 1K x 512 pixels with 4 bits per pixel. The frame buffer is mapped from >0000 0000 to >001F FFFF in the TMS34010's local memory space. Accessing the standard port of the memory is similar to accessing the DRAM's.

The VRAM area is divided into two banks of 64K words apiece. This is done to save power by having only half the memories active whenever possible. When the TMS34010 accesses the range from >0000 0000 to >001F FFFF, decode-PAL U11 generates a DMRASL0 or DMRASL1 signal depending if the address is in the upper or lower bank of VRAM. This display-memory $\overline{\mathrm{RAS}}$ strobe will strobe in the eight row addresses which are output through the 74AS573 Octal D Latch U10. The $\overline{\mathrm{CAS}}$ strobe follows shortly thereafter, strobing in the eight column addresses into the video memory. The output buffers on the memory devices are enabled by $\overline{\mathrm{TR/QE}}$. On refresh cycles (indicated by the REFCYCL output of latch U9 low during the $\overline{\mathrm{RAS}}$ interval) both the DMRASL0 and DMRASL1 outputs go low to refresh the memories.

On update cycles, the VRAMs transfer a row of memory to the on-chip 256-bit shift register or the contents of the shift register to the row of memory depending on the state of the read/write line. For normal display operations, the memory is transferred to the shift register. A transfer operation is detected by the XFRCYCL output going low on the $\overline{\mathrm{RAS}}$ interval on the 74ALS573 octal latch U9. The U11 decode PAL creates a DMRASL0 and DMRASL1 signal to both banks of display memory; thus transferring a row of memory to the shift register on the VRAMs.

6.4 Frame Buffer to Video Output Interface

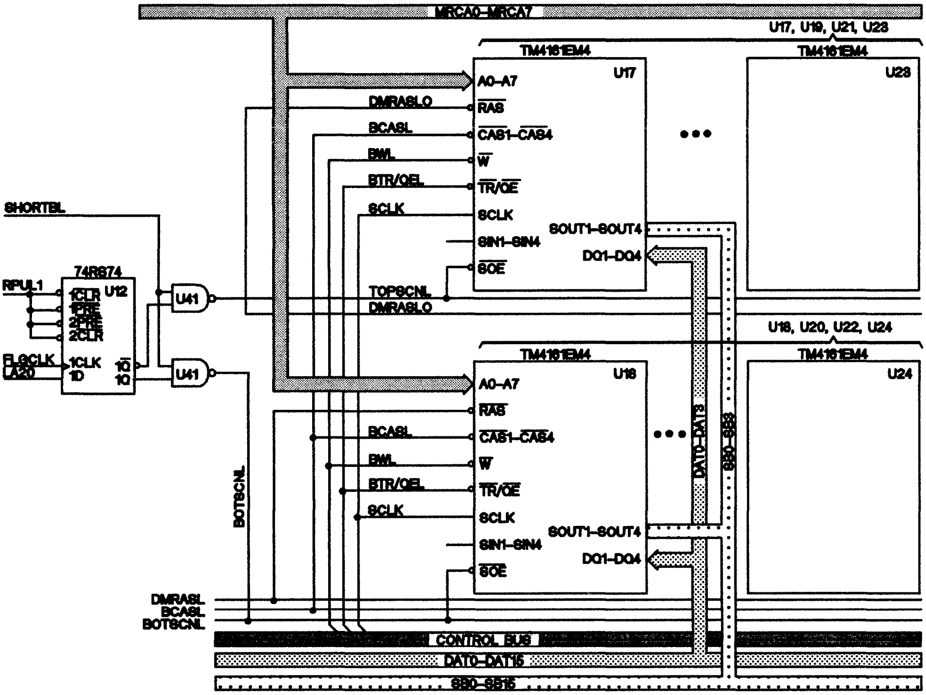


Figure 6-11. Frame Buffer Selection Logic

The frame buffer is divided into two equal parts of 64K words apiece allowing a total video memory space of 256K bytes. Figure 6-11 is a diagram of the organization of the frame buffer in the TMS34010's local memory space. The 32 serial outputs are grouped into two banks of 16 each. These two banks are physically tied together and only one bank is enabled at a time for serial access by using the SOUTS on the VRAM SIPs. The SOUT control logic is accomplished by latching the LA20 address line in flip-flop U12 on a transfer cycle. PAL U11 detects a transfer cycle at $\overline{\text{RAS}}$ time and generates output signal FLGCLK which clocks the state of LA20 into the the flip-flop. LA20 is used because this is the address line which divides the frame buffer in half. The outputs of this flip-flop are ANDed with a blanking signal to create a TOPSCNL and BOTSCNL serial output enable. These two signals allow only either the top or bottom of the frame buffer to be enabled at one time.

The two groups of 16 serial outputs (SB15-SB0) from the frame buffer are driven into two 74ALS257 QUAD 2-to-1 multiplexers U28 and U29. These eight outputs, DA3-DA0 and DB3-DB0, are time muliplexed by signal VCLK which is 1/4 the speed of the actual dotclock frequency as shown in Figure 6-12. The initial clock divide is done by the TMS34070 Color Palette. The palette output (CLKOUT) is input into the 74AS161A at U25 for further division. From here the output has two paths through the TMS34070 Color Palette for an analog output or through some other TTL for the digital output. Signal SCLK to the VRAMs is created by ANDing the blanking signals and the VCLK signal through 74AS11 AND gate U45. SCLK is turned off during the blanking interval so that the shift register transfers to the VRAMs can occur. The blanking signals are aligned to their proper time intervals by the dual D flip-flops in U26.

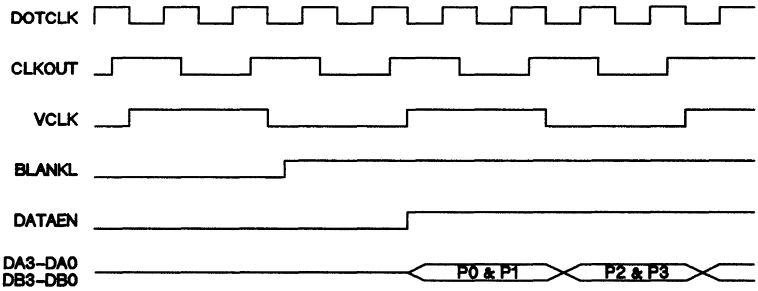


Figure 6-12. Frame Buffer Output Timing

When the Color Palette output is used, the DA3-DA0 and DB3-DB0 outputs from the two 74ALS257 multiplexers are fed directly into the TMS34070. The TMS34070 operates on two pixels at once and time multiplexs them internally onto its RGB output pins. The TMS34070 is unique in that it loads directly from the frame buffer so that no microprocessor interface is necessary. The TMS34070 also has two modes of operation -- line load mode and frame load mode -- which are selected by jumper W3. The RGB outputs are fed into a Wye resistor network for impedance matching and then output to jumper platform W9. W9 is used to select the analog or digital outputs.

When jumper W9 is set for the digital outputs the DA3-DA0 and DB3-DB0 outputs are input into U33 a 74ALS878 Octal Latch with Clear. The Clear is used to blank the digital ouput during the blanking interval. The latch outputs then drive U34, a 74AS257 multiplexer. This multiplexer is time sliced by the CLKOUT output of the TMS34070 Color Palette. The outputs of multiplexer U34 derive a 4-bit RGBI code which is driven into the W9 jumper platform.

The W9 jumper platform is used to select the analog or digital output onto the DB9 connector J3.

6.5 CRT Timing Generation

The CRT timing generation is done internally to the TMS34010. The TMS34010 has eight internal registers which control the timing of the $\overline{\text{HSYNC}}$, $\overline{\text{VSYNC}}$, and BLANK output pins. All timing parameters are fully programmable by these eight 16-bit internal registers.

The horizontal parameters are derived from the VIDCLK input. While the vertical parameters are calculated in horizontal lines, each of the vertical and horizontal parameters has four registers associated with them. These registers are Start Blank, End Blank, End Sync, and Total. Using these two sets of four registers, you can interface to almost any monitor.

The Software Development Board has external logic which allows the user to select any combination of syncs. The user can have positive or negative composite sync. Both negative and positive values of either horizontal or vertical sync is also available. This vast array of syncs is configured by onboard jumpers outlined in Section 2. For more information on CRT timing generation see the TMS34010 User's Guide (SPVU001).

6.6 Software

The SDB340 software debugger consists of two independent but communicating bodies of software:

- PC software: this software resides on the PC side and operates under MS-DOS.
- TMS34010 software: this resides on the TMS34010 and performs commands given to it by the PC software.

The two halves of the SDB340 debugger communicate with each other via the message bits in the HSTCTLL I/O register on the TMS34010.

6.6.1 Program Bootup Sequence

1) When invoked via MS-DOS, the PC software verifies SDB operation by running several minor checks. If these are passed, a second body of software is loaded through the host port to the SDB board. This software resides in the memory labelled "Reserved for Debugger" as shown in the SDB Memory Map (Figure 6-13).
2) Once loaded, the PC software makes more minor checks to determine if the TMS34010 software was loaded correctly. If the checks are passed, the PC software loads the start address of the TMS34010 software into the non-maskable interrupt (NMI) vector (TRAP 8).
3) The PC software then performs an NMI with the NMI mode set to inhibit the pushing of the PC and ST. This keeps the TMS34010 software from being overwritten by the NMI.
4) Then the NMI transfers control to the TMS34010 software which:
   sets up the stack pointer,
   initializes local variables,
   loads the machine state values into a message area, and
   then waits for commands from the PC software.
5) The PC software then uploads the machine state values, displays them, and waits for commands from the user.

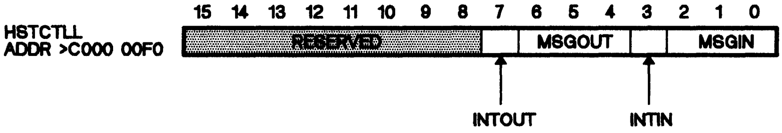
MEMORY SPACE	OCCUPANTS
>0000 0000 thru 001F FFF0	DISPLAY RAM
>0020 0000 thru >BFFF FFF0	Reserved†
>C000 0000 thru >C000 01F0	IOREGS
>FFC0 0000 thru >FFFD FFF0	USER CODE
>FFFE 0000 thru >FFFF FFF0	RESERVED FOR DEBUGGER

†For a more detailed description of this area, see Figure 5-4 on page 5-6

Figure 6-13. SDB Memory Map

6.6.2 SDB340 Communications Protocol

1) When the PC software wishes the TMS34010 software to perform an action, it downloads the machine state values along with a command request value. Handshaking for this transaction is through the HSTCTLL I/O Register (shown in Figure 6-14).
2) The PC software then writes a value of 5 into the MSGIN bits of the HSTCTLL Register.
3) The TMS34010 software detects the 5, and writes a 5 into the MSGOUT bits of the HSTCTLL Register, signaling an acknowledgement to the PC software.
4) Upon completion of the request, the TMS34010 software writes a 2 to the MSGOUT bits of the HSTCTLL register, signaling completion.
5) The PC software uploads the machine state values along with a command response value.
6) Upon completion of the upload, the PC software writes a 2 into the MSGIN bits of the HSTCTLL Register, acknowledging receipt of the data.



BITS	NAME	FUNCTION
8-15	RESERVED	When read, zeroes always returned
7	INTOUT	Output interrupt bit
4-6	MSGOUT	Output message buffer, host read, TMS34010 write)
3	INTIN	Input interrupt bit
0-2	MSGIN	Input message buffer (host write, TMS34010 read)

Figure 6-14. HSTCTLL I/O Register

6.6.3 SDB340 Execution Control

The SDB340 debugger controls execution of target software via software TRAP instructions and the NMI.

The NMI is used to halt the TMS34010 when the target software is allowed to run realtime by the use of a RUN command. The PC software detects a keystroke and then interrupts the target software by issuing an NMI. The NMI trap vector (TRAP 8) must point into the TMS34010 software for the SDB340 to regain control. If the NMI is needed, such as in using the SY command to run another PC program to interface to the TMS34010, the NMI may be changed, *but it must be restored so that the SDB340 can interrupt the TMS34010 target software.*

Single step (SS) and run with count (RUN) are controlled with software traps. Single step is controlled by inserting a TRAP 26 immediately after the instruction that is pointed to by the PC. If the instruction is a branch instruction (JUMP RS, TRAP n, EXGPC, etc.), then a second TRAP 26 is inserted at the possible branch location. The memory at SP and SP(32) are changed to correspond to the PC and ST that are to be used in the instruction, and a RETI is executed.

The RETI returns to the desired program location, performs the instruction, executes the TRAP 26, and returns to the TMS34010 software of the SDB. One requirement is that the following must be in writeable memory:

- the word immediately after the instruction to be executed, or
- the word at the possible branch location, or
- the words preceding the stack.

SDB340 detects these conditions and will not single step when there is a possiblity that the single step will fail. SDB340 also detects and prevents branches into itself on single step. As with the NMI, if the vector associated with TRAP 26 is overwritten and not restored, then the single step function will not operate and the user will lose control of the TMS34010 software. Single stepping or running with count across an instruction that changes the TRAP 26 vector will also cause SDB340 to immediately lose control.

Breakpoint and trace are also controlled with software traps. Trace and breakpoint functions insert a TRAP 29 at the instruction where the breakpoint or trace should take place. You can induce manual user halts by placing a TRAP 29 into your code. Again, the trace and breakpoint functions will not operate if the TRAP 29 vector is overwritten; however, the consequences will not be as severe as overwriting the TRAP 8 and TRAP 26 vectors.

The I/O functions of SDB340 are controlled with TRAP 25, TRAP 27, and TRAP 28. The printf function used with the TMS34010 simulator can also be used with SDB340. This function uses TRAP 27 and TRAP 28. TRAP 25 is reserved for future I/O function expansion. The I/O functions will not operate if these trap vectors are overwritten. The consequences are of the same severity as overwriting the trace and breakpoint trap vectors.

6.6.4 Sharing of the Stack

Implicit in the use of TRAPs to control execution is that the stack is shared between the TMS34010 SDB340 software and the TMS34010 target software. This necessary sharing has side effects that you should be aware of when using the Stack Pointer (SP) in anything other than a strictly conventional manner. For example, the following code will most likely produce unexpected results:

```
move     A0,*SP(-16)        ;Field size 0 = 16
move     *SP(-16), A0
```

The contents of A0 probably will change when this code is stepped. This also applies to RUN with count.

By stepping through the first instruction, the TRAP 26 that is executed will cause the contents of the word above the stack to change to the value of the lower half of the PC. Remember that a trap causes the PC and ST to be pushed onto the hardware stack. In addition, The TMS34010 SDB340 software performs one subroutine call that causes an additional PC to be pushed onto the stack. Therefore, the user can expect the six words immediately above the top of the stack to be corrupted every time that a single step or a run with count is performed.

6.6.5 TMS34010 Interrupt Map

Interrupt vectors for the TMS34010 are shown in Figure 6-15.

Trap Number	Address	32 0	
0	FFFF FFE0	RESET	Reset
1	FFFF FFC0	INT1	External Interrupt 1
2	FFFF FFA0	INT2	External Interrupt 2
3	FFFF FF80	Traps 3-7	
4	FFFF FF60		
5	FFFF FF40		
6	FFFF FF20		
7	FFFF FF00		
8	FFFF FEE0	NMI†	Nonmaskable Interrupt
9	FFFF FEC0	HI	Host Interrupt
10	FFFF FEA0	DI	Display Interrupt
11	FFFF FE80	WV	Window Violation
12	FFFF FE60	Traps 12-24	
13	FFFF FE40		
14	FFFF FE20		
15	FFFF FE00		
16	FFFF FDE0		
17	FFFF FDC0		
18	FFFF FDA0		
19	FFFF FD80		
20	FFFF FD60		
21	FFFF FD40		
22	FFFF FD20		
23	FFFF FD00		
24	FFFF FCE0		
25	FFFF FCC0	Debugger Traps†	
26	FFFF FCA0		
27	FFFF FC80		
28	FFFF FC60		
29	FFFF FC40		
30	FFFF FC20	ILLOP	Illegal opcode
31	FFFF FC00	TRAP 31	

†Do not modify vectors for Traps 8, 25, 26, 27, 28, or 29. Any changes to these traps will cause a system failure.

Figure 6-15. TMS34010 Interrupt Map

A. COM 2651 Programmable Communication Interface

The following is a data sheet on the COM 2651 Programmable Communication interface provided with the permission of the manufacturer, Standard Microsystems Corporation of Hauppauge, N.Y.

STANDARD MICROSYSTEMS CORPORATION

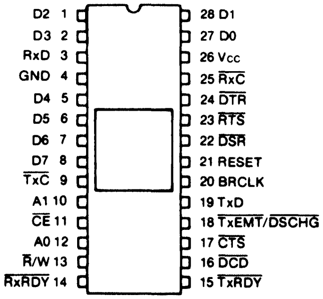
COM 2651
μPC FAMILY

Programmable Communication Interface PCI

FEATURES

- □ Synchronous and Asynchronous Full Duplex or Half Duplex Operations
- □ Re-programmable ROM on-chip baud rate generator
- □ Synchronous Mode Capabilities
  - —Selectable 5 to 8-Bit Characters
  - —Selectable 1 or 2 SYNC Characters
  - —Internal Character Synchronization
  - —Transparent or Non-Transparent Mode
  - —Automatic SYNC or DLE-SYNC Insertion
  - —SYNC or DLE Stripping
  - —Odd, Even, or No Parity
  - —Local or remote maintenance loop back mode
- □ Asynchronous Mode Capabilities
  - —Selectable 5 to 8-Bit Characters
  - —3 Selectable Clock Rates (1X, 16X, 64X the Baud Rate)
  - —Line Break Detection and Generation
  - —1, 1½, or 2-Stop Bit Detection and Generation
  - —False Start Bit Detection
  - —Odd, Even, or No Parity
  - —Parity, Overrun, and framing error detect
  - —Local or remote maintenance loop back mode
  - —Automatic serial echo mode
- □ Baud Rates
  - —DC to 1.0M Baud (Synchronous)
  - —DC to 1.0M Baud (1X, Asynchronous)
  - —DC to 62.5K Baud (16X, Asynchronous)
  - —DC to 15.625K Baud (64X, Asynchronous)
- □ Double Buffering of Data

PIN CONFIGURATION



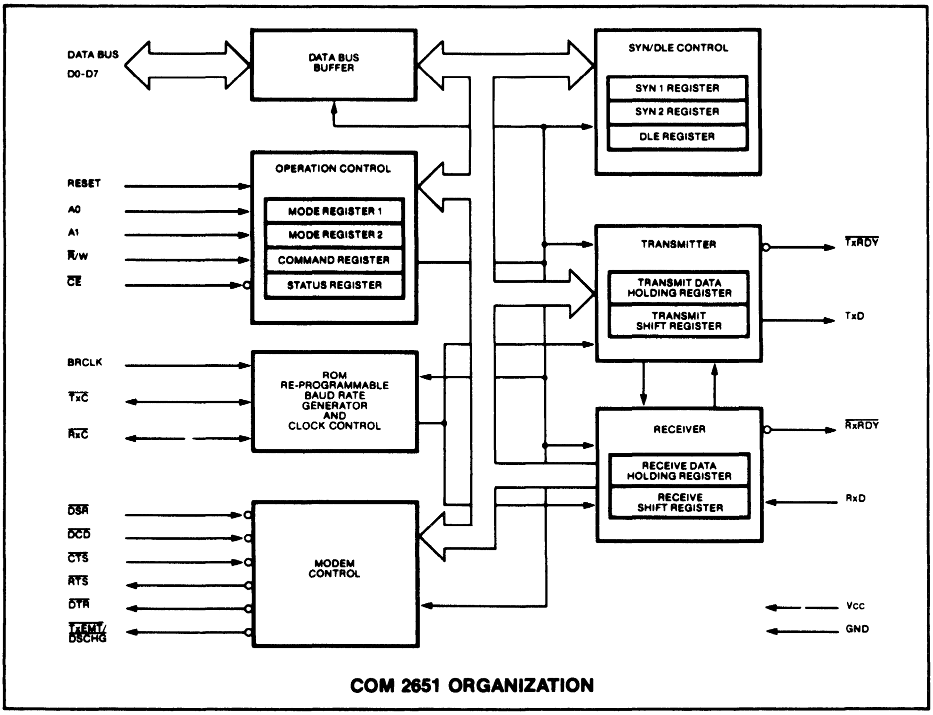
Package: 28-pin D.I.P.

- □ Internal or External Baud Rate Clock
  - —16 Internal Rates:50 to 19,200 Baud
- □ Single +5 volt Power Supply
- □ TTL Compatible
- □ No System Clock Required
- □ Compatible with 2651, INS2651

GENERAL DESCRIPTION

The COM 2651 is an MOS/LSI device fabricated using SMC's patented COPLAMOS® technology that meets the majority of asynchronous and synchronous data communication requirements, by interfacing parallel digital systems to asynchronous and synchronous data communication channels while requiring a minimum of processor overhead. The COM 2651 contains a baud rate generator which can be programmed to either accept an external clock or to generate internal transmit or receive clocks. Sixteen different baud rates can be selected under program control when operating in the internal clock mode. The on-chip baud rate generator can be ROM reprogrammed to accommodate different baud rates and different starting frequencies.

The COM 2651 is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) designed for microcomputer system data communications. The USART is used as a peripheral and is programmed by the processor to communicate in commonly used asynchronous and synchronous serial data transmission techniques including IBM Bi-Sync. The USART receives serial data streams and converts them into parallel data characters for the processor. While receiving serial data, the USART will also accept data characters from the processor in parallel format, convert them to serial format and transmit. The USART will signal the processor when it has completely received or transmitted a character and requires service. Complete USART status including data format errors and control signals is available to the processor at any time.



COM 2651 ORGANIZATION

The COM 2651 is organized into 6 major sections. Communication between each section is achieved via an internal data and control bus. The data bus buffer allows a processor access to all internal registers on the COM 2651.

Operation Control

This functional block stores configuration and operation commands from the processor and generates appropriate signals to various internal sections to control the overall device operation. It contains read and write circuits to permit communications with a processor via the data bus and contains Mode Registers 1 and 2, the Command Register, and the Status Register. Details of register addressing and protocol are presented in the COM 2651 programming section of this specification.

Timing

The COM 2651 contains a Baud Rate Generator (BRG) which is programmable to accept external transmit or receive clocks or to divide an external clock to perform data communications. The unit can generate 16 commonly used baud rates, any one of which can be selected for full duplex operation. Table 6 illustrates all available baud rates.

Receiver

The Receiver accepts serial data on the RxD pin, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and stores the "assembled" character in the receive data holding register until read by the processor.

Transmitter

The Transmitter accepts parallel data from the processor, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin.

Modem Control

The modem control provides three output signals and accepts three input signals used for "handshaking" and status indication between the COM 2651 and a modem.

SYN/DLE Control

This section contains control circuitry and three 8-bit registers storing the SYN1, SYN2, and DLE characters provided by the processor. These registers are used in the synchronous mode of operation to provide the characters required for synchronization, idle fill and data transparency.

Interface Signals

The COM 2651 interface signals can be grouped into two types: the processor-related signals (shown in Table 2) which interface the COM 2651 to the processor, and the device-related signals (shown in Table 3), which are used to interface to the communications equipment.

TABLE 2—PROCESSOR RELATED SIGNALS

PIN NO.	NAME	SYMBOL	FUNCTION
1,2,5,6, 7,8,27,28	Data	D7-D0	Bidirectional; 8 bit, three state data bus used to transfer commands, data and status between the COM 2651 and a processor. D0 is the least significant bit; D7 is the most significant bit.
10,12	Address	A1, A0	Input; Address lines used to select COM 2651 registers.
11	Chip Enable	$\overline{CE}$	Input; when this signal is low, the operation specified by the $\overline{R}$/W, A1 and A0 will be performed. When this input is high, D7-0 are in the high impedance state.
13	Read/Write	$\overline{R}$/W	Input; Processor read/write direction control. This signal defines the direction of the data bus D7-0 when the COM 2651 is selected. D7-0 drives out (read) when this signal is low and accepts data input when this signal is high. The input only has meaning when the chip enable input is active.
14	Receiver Ready	$\overline{RxRDY}$	Output; This signal is the complement of Status Register bit 1 (SR1). When low, it indicates that the Receive Data Holding Register (RHR) has a character ready for input to the processor. It goes high when the RHR is read by the processor, and also when the receiver is disabled. It is an open drain output which can be used as an interrupt to the processor.
15	Transmitter Ready	$\overline{TxRDY}$	Output; This signal is the complement of Status Register bit 0 (SR0). When low, it indicates that the Transmit Data Holding Register (THR) is ready to accept a data character from the processor. It goes high when the data character is loaded. This output is valid only when the transmitter is enabled. It is an open drain output which can be used as an interrupt to the processor.
18	Transmitter empty/data set change	$\overline{TxEMT}$/ $\overline{DSCHG}$	Output; This signal is the complement of Status Register bit 2 (SR2). When low, it indicates that the transmitter has completed serialization of the last character loaded by the processor, or that a change of state of the $\overline{DSR}$ or $\overline{DCD}$ inputs has occurred. This output goes high when the Status Register is read by the processor, if the TxEMT condition does not exist. Otherwise, the THR must be loaded by the processor for this line to go high. It is an open drain output which can be used as an interrupt to the processor.
21	Reset	Reset	Input; A high on this input performs a master reset on the COM 2651. This signal asynchronously terminates any device activity and clears the Mode, Command and Status registers. The device assumes the idle state and remains there until initialized with the appropriate control words.
26	Supply Voltage	V_{CC}	+5 volts supply.
4	Ground	GND	Ground.

TABLE 3—DEVICE RELATED SIGNALS

PIN NO.	NAME	SYMBOL	FUNCTION
3	Receive Data	RxD	Input; Serial data to the receiver. "Mark" is high "space" is low.
9	Transmitter Clock	$\overline{TxC}$	Input or Output; If the external transmitter clock is programmed, this input controls the rate at which the character is transmitted. Its frequency is 1X, 16X or 64X, the Baud rate as programmed by Mode Register 1. The transmitted data changes on the falling edge of the clock. If the internal transmitter clock is programmed, this pin becomes an output at 1X the programmed Baud rate.
16	Data Carrier Detect	$\overline{DCD}$	Input; This signal must be low in order for the receiver to function. The complement appears in the Status Register bit 6 (SR6). When this input changes state a low output on $\overline{TxEMT}$/$\overline{DSCHG}$ occurs.
17	Clear to Send	$\overline{CTS}$	Input; This signal must be low in order for the transmitter to function. If it goes high during transmission, the character in the Transmit Shift Register will be transmitted before termination.
19	Transmit Data	TxD	Output; Serial data from the transmitter. "Mark" is high, "Space" is low. This signal is held in the "Mark" condition when the transmitter is disabled.
20	Baud Rate Clock	BRCLK	Input; The standard device requires a 5.0688MHz clock to the internal Baud rate generator allowing for Baud rate shown in Table 6. The reprogrammable ROM on chip allows for user specificed Baud rates and input frequency. Consult the factory for details. This input is not required if external receive and transmit clocks are used.
22	Data Set Ready	$\overline{DSR}$	Input; This general purpose signal can be used for Data Set Ready or Ring Indicator condition. Its complement appears as Status Register bit 7 (SR7). When this input changes state, a low output on $\overline{TxEMT}$/$\overline{DSCHG}$ occurs.
23	Request to Send	$\overline{RTS}$	Output; This general purpose signal is the complement of the Command Register bit 5 (CR5). It is normally used to indicate Request to Send.

TABLE 3—DEVICE RELATED SIGNALS

PIN NO.	NAME	SYMBOL	FUNCTION
24	Data Terminal	$\overline{\text{DTR}}$	Output; This general purpose signal is the complement of the Command Register bit 1 (CR1). It is normally used to indicate Data Terminal Ready.
25	Receive Clock	$\overline{\text{RxC}}$	Input or Output; If the external receiver clock is programmed, this input controls the rate at which the character is to be received. Its frequency is 1X, 16X, or 64X the Baud rate, as programmed by Mode Register 1. Data are sampled on the rising edge of the clock. If internal receiver clock is programmed, this pin becomes an output at 1X the programmed Baud rate.

COM 2651 OPERATION

The functional operation of the COM 2651 is programmed by a set of control words supplied by the processor. These control words specify items such as synchronous or asynchronous mode, baud rate, number of bits per character, etc. The programming procedure is described in the COM 2651 Programming section of this data sheet.

After programming, the COM 2651 is ready to perform the desired communications functions. The receiver performs serial to parallel conversion of data received from a modem or equivalent device. The transmitter converts parallel data received from the processor to a serial bit stream. These actions are accomplished within the framework specificed by the control words.

Receiver

The COM 2651 is conditioned to receive data when the $\overline{\text{DCD}}$ input is low and the RxEN bit in the command register is true. In the asynchronous mode, the receiver looks for a high to low transition on the RxD input line indicating the start bit. If a transition is detected, the state of the RxD line is sampled again after a delay of one-half of a bit time. If RxD is now high, the search for a valid start bit is begun again. If RxD is still low, a valid start bit is assumed and the receiver continues to sample the input line at one bit time intervals until the proper number of data bits, the parity bit, and the stop bit(s) have been assembled. The data is then transferred to the Receive Data Holding Register, the RxRDY bit in the status register is set, and the $\overline{\text{RxRDY}}$ output is asserted. If the character length is less than 8 bits, the high order unused bits in the Holding Register are set to zero. The Parity Error, Framing Error, and Overrun Error status bits are strobed into the status register on the positive going edge of $\overline{\text{RxC}}$ corresponding to the received character boundary. If a break condition is detected (RxD is low for the entire character as well as the stop bit[s]), only one character consisting of all zeros (with the Framing error status bit set) will be transferred to the Holding Register. The RxD input must return to a high condition before a search for the next start bit begins.

When the COM 2651 is initialized into the synchronous mode, the receiver first enters the hunt mode on a 0 to 1 transition of RxEN (CR2). In this mode, as data is shifted into the Reciver Shift Register a bit at a time, the contents of the register are compared to the contents of the SYN1 register. If the two are not equal, the next bit is shifted in and the comparison is repeated. When the two registers match, the hunt mode is terminated and character assembly begins. If the single SYN operation is programmed, the SYN DETECT status bit is set. If double SYN operation is programmed, the first character assembled after SYN1 must be SYN2 in order for the SYN DETECT bit to be set. Otherwise, the COM 2651 returns to the hunt mode. (Note that the sequence SYN1-SYN1-SYN2 will not achieve synchronization). When synchronization has been achieved, the COM 2651 continues to assemble characters and transfers them to the Holding Register. The RxRDY status bit is set and the $\overline{\text{RxRDY}}$ output is asserted each time a character is assembled and transferred to the Holding Register. The Overrun error (OE) and Parity error (PE) status bits are set as appropriate. Further receipt of the proper SYN sequence sets the SYN DETECT status bit. If the SYN stripping mode is commanded, SYN characters are not transferred to the Holding Register. Note that the SYN characters used to establish initial synchronization are not transferred to the Holding Register in any case.

Transmitter

The COM 2651 is conditioned to transmit data when the $\overline{\text{CTS}}$ input is low and the TxEN command register bit is set. The COM 2651 indicates to the processor that it can accept a character for transmission by setting the TxRDY status bit and asserting the $\overline{\text{TxRDY}}$ output. When the processor writes a character into the Transmit Data Holding Register, the TxRDY status bit is reset and the $\overline{\text{TxRDY}}$ output is returned to a high (false) state. Data is transferred from the Holding Register to the Transmit Shift Register when it is idle or has completed transmission of the previous character. The TxRDY conditions are then asserted again. Thus, one full character time of buffering is provided.

In the asynchronous mode, the transmitter automatically sends a start bit followed by the programmed number of data bits, the least significant bit being sent first. It then appends an optional odd or even parity bit and the programmed number of stop bits. If, following transmission of the data bits, a new character is not available in the Transmit Holding Register, the TxD output remains ir. the marking (high) condition and the $\overline{\text{TxEMT}}/\overline{\text{DSCHG}}$ output and its corresponding status bit are asserted. Transmission resumes when the processor loads a new character into the Holding Register. The transmitter can be forced to output a continuous low (BREAK) condition by setting the Send Break command bit high.

In the synchronous mode, when the COM 2651 is initially conditioned to transmit, the TxD output remains high and the TxRDY condition is asserted until the first character to be transmitted (usually a SYN character) is loaded by the processor. Subsequent to this, a continuous stream of characters is transmitted. No extra bits (other than parity, if commanded) are generated by the COM 2651 unless the processor fails to send a new character to the COM 2651 by the time the transmitter has completed sending the previous character. Since synchronous communication does not allow gaps between characters, the COM 2651 asserts TxEMT and automatically "fills" the gap by ti ansmitting SYN1s, SYN1-SYN2 doublets, or DLE-SYN1 doublets, depending on the state of MR16 and MR17. Normal transmission of the message resumes when a new character is available in the Transmit Data Holding Register. If the SEND DLE bit in the command register is true, the DLE character is automatically transmitted prior to transmission of the message character in the transmit holding register.

COM 2651 PROGRAMMING

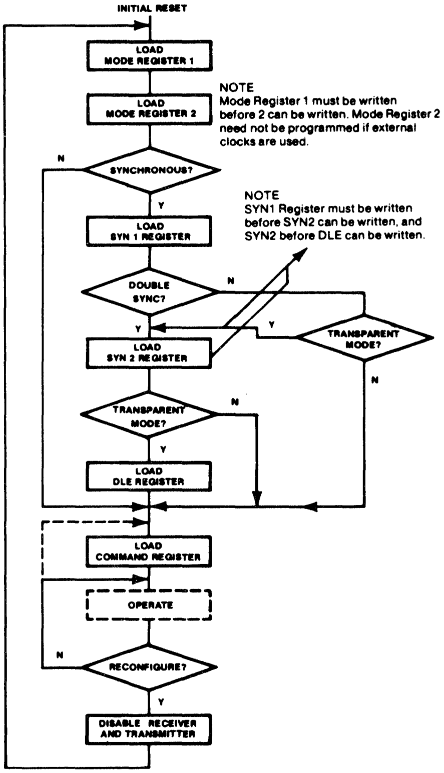
Prior to initiating data communications, the COM 2651 operational mode must be programmed by performing write operations to the mode and command registers. In addition, if synchronous operation is programmed, the appropriate SYN/DLE registers must be loaded. The COM 2651 can be reconfigured at any time during program execution. However, if the change has an effect on the reception of a character the receiver should be disabled. Alternatively if the change is made 1½ $\overline{RxC}$ periods after RxRDY goes active it will affect the next character assembly. A flowchart of the initialization process appears in Figure 1.

The internal registers of the COM 2651 are accessed by applying specific signals to the $\overline{CE}$, $\overline{R}$/W, A1 and A0 inputs. The conditions necessary to address each register are shown in Table 4.

The SYN1, SYN2, and DLE registers are accessed by performing write operations with the conditions A1=0, A0=1, and $\overline{R}$/W=1. The first operation loads the SYN1 register. The next loads the SYN2 register, and the third loads the DLE register. Reading or loading the mode registers is done in a similar manner. The first write (or read) operation addresses Mode Register 1, and a subsequent operation addresses Mode Register 2. If more than the required number of accesses are made, the internal sequencer recycles to point at the first register. The pointers are reset to SYN1 Register and Mode Register 1 by a RESET input or by performing a "Read Command Register" operation, but are unaffected by any other read or write operation.

The COM 2651 register formats are summarized in Tables 5, 6, 7 and 8. Mode Registers 1 and 2 define the general operational characteristics of the COM 2651, while the Command Register controls the operation within this basic framework. The COM 2651 indicates its status in the Status Register. These registers are cleared when a RESET input is applied.

COM 2651 INITIALIZATION FLOW CHART



NOTE
Mode Register 1 must be written before 2 can be written. Mode Register 2 need not be programmed if external clocks are used.

NOTE
SYN1 Register must be written before SYN2 can be written, and SYN2 before DLE can be written.

Figure 1

$\overline{CE}$	A1	A0	$\overline{R}$/W	FUNCTION
1	X	X	X	Tri-state data bus
0	0	0	0	Read receive holding register
0	0	0	1	Write transmit holding register
0	0	1	0	Read status register
0	0	1	1	Write SYN1/SYN2/DLE registers
0	1	0	0	Read mode registers 1 and 2
0	1	0	1	Write mode registers 1 and 2
0	1	1	0	Read command register
0	1	1	1	Write command register

NOTE
See AC Characteristics section for timing requirements.

Table 4—COM 2651 REGISTER ADDRESSING

MODE REGISTER 1 (MR1)

Table 5 illustrates Mode Register 1. Bits MR11 and MR10 select the communication format and Baud rate multiplier. 00 specifies synchronous mode and 1X multiplier. 1X, 16X, and 64X multipliers are programmable for asynchronous format. However, the multiplier in asynchronous format applies only if the external clock input option is selected by MR24 or MR25.

MR13 and MR12 select a character length of 5, 6, 7, or 8 bits. The character length does not include the parity bit, if programmed, and does not include the start and stop bits in asynchronous mode.

MR14 controls parity generation. If enabled, a parity bit is added to the transmitted character and the receiver performs a parity check on incoming data. MR15 selects odd or even parity when parity is enabled by MR14.

In asychronous mode, MR17 and MR16 select character framing of 1, 1.5, or 2 stop bits. (if 1X baud rate is programmed, 1.5, stop bits defaults to 1 stop bits on transmit). In synchronous mode, MR17 controls the number of SYN characters used to establish synchronization and for character fill when the transmitter is idle. SYN1 alone is used if MR17=1, and SYN1-SYN2 is used when MR17=0. If the transparent mode is specified by MR16, DLE-SYN1 is used for character fill and SYN Detect, but the normal synchronization sequence is used. Also DLE stripping and DLE Detect (with MR14=0) are enabled.

MR17	MR16	MR15	MR14	MR13	MR12	MR11	MR10
Sync/Async		Parity Type	Parity Control	Character Length		Mode and Baud Rate Factor	
ASYNCH: STOP BIT LENGTH 00=INVALID 01=1 STOP BIT 10=1½ STOP BITS 11=2 STOP BITS		0=ODD 1=EVEN	0=DISABLED 1=ENABLED	00=5 BITS 01=6 BITS 10=7 BITS 11=8 BITS		00=SYNCHRONOUS 1X RATE 01=ASYNCHRONOUS 1X RATE 10=ASYNCHRONOUS 16X RATE 11=ASYNCHRONOUS 64X RATE	
SYNCH: NUMBER OF SYN CHAR 0=DOUBLE SYN 1=SINGLE SYN	SYNCH: TRANSPARENCY CONTROL 0=NORMAL 1=TRANSPARENT						

NOTE Baud rate factor in asynchronous applies only if external clock is selected Factor is 16X if internal clock is selected Mode must be selected (MR11, MR10) in any case

TABLE 5—MODE REGISTER 1(MR1)

MODE REGISTER 2 (MR2)

Table 6 illustrates Mode Register 2. MR23, MR22, MR21, and MR20 control the frequency of the internal Baud rate generator (BRG). Sixteen rates are selectable. When driven by a 5.0688 MHz input at the BRCLK input (pin 20), the BRG output has zero error except at 134.5, 2000, and 19,200 Baud, which have errors of +0.016% +0.253%, and +3.125% respectively.

MR25 and MR24 select either the BRG or the external inputs $\overline{\text{TxC}}$ and $\overline{\text{RxC}}$ as the clock source for the transmitter and receiver, respectively. If the BRG clock is selected, the Baud rate factor in asynchronous mode is 16X regardless of the factor selected by MR11 and MR10. In addition, the corresponding clock pin provides an output at 1X the Baud rate. Custom Baud rates other than the ones provided by the standard part are available. Contact the factory for details.

MR27	MR26	MR25	MR24	MR23-MR20					
		Transmitter Clock	Receiver Clock	Code	Baud Rate	Theoretical Frequency 16X Clock	Actual Frequency 16X Clock	Percent Error	Divisor
		0=EXTERNAL	0=EXTERNAL	0000	50	0 8 KHz	0 8 KHz	—	6336
NOT USED		1=INTERNAL	1=INTERNAL	0001	75	1 2	1 2	—	4224
				0010	110	1 76	1.76	–	2880
				0011	134 5	2 152	2 1523	0 016	2355
				0100	150	2.4	2.4	—	2112
				0101	300	4 8	4 8	—	1056
				0110	600	9 6	9 6	—	528
				0111	1200	19 2	19 2	—	264
				1000	1800	28 8	28 8	—	176
				1001	2000	32 0	32 081	0 253	158
				1010	2400	38 4	38 4	—	132
				1011	3600	57 6	57 6	—	88
				1100	4800	76 8	76 8	—	66
				1101	7200	115 2	115 2	—	44
				1110	9600	153 6	153 6	—	33
				1111	19200*	307 2	316 8	3 125	16

NOTE *Error at 19200 can be reduced to zero by using crystal frequency 4 9152MHz
16X clock is used in asynchronous mode In synchronous mode, clock multiplier is 1X
Baud rates are valid for crystal frequency = 5 0688MHz

TABLE 6—MODE REGISTER 2 (MR2)

COMMAND REGISTER (CR)

Table 7 illustrates the Command Register. Bits CRO (TxEN) and CR2 (RxEN) enable or disable the transmitter and receiver respectively. A 0 to 1 transition of CR2 forces start bit search (async mode) or hunt mode (sync mode) on the second $\overline{\text{RxC}}$ rising edge. Disabling the receiver causes $\overline{\text{RxRDY}}$ to go high (inactive). If the transmitter is disabled, it will complete the transmission of the character in the Transmit Shift Register (if any) prior to terminating operation. The TxD output will then remain in the marking state (high) while the $\overline{\text{TxRDY}}$ and $\overline{\text{TxEMT}}$ will go high (inactive). If the receiver is disabled, it will terminate operation immediately. Any character being assembled will be neglected.

In asynchronous mode, setting CR3 will force and hold the TxD output low (spacing condition) at the end of the current transmitted character. Normal operation resumes when CR3 is cleared. The TxD line will go high for at least one bit time before beginning transmission of the next character in the Transmit Data Holding Register. In synchronous mode, setting CR3 causes the transmission of the DLE register contents prior to sending the character in the Transmit Data Holding Register. CR3 should be reset in response to the next TxRDY.

Setting CR4 causes the error flags in the Status Register (SR3, SR4, and SR5) to be cleared. This is a one time command. There is no internal latch for this bit.

The COM 2651 can operate in one of four sub-modes within each major mode (synchronous or asynchronous). The operational sub-mode is determined by CR7 and CR6. CR7-CR6=00 is the normal mode, with the transmitter and receiver operating independently in accordance with the Mode and Status Register instructions.

In asynchronous mode, CR7-CR6=01 places the COM 2651 in the Automatic Echo mode. Clocked, regenerated received data is automatically directed to the TxD line while normal receiver operation continues. The receiver must be enabled (CR2=1), but the transmitter need not be enabled. Processor to receiver communications continues normally, but the processor to transmitter link is disabled. Only the first character of a break condition is echoed. The TxD output will go high until the next valid start is detected. The following conditions are true while in Automatic Echo mode:

1. Data assembled by the receiver are automatically placed in the Transmit Holding Register and retransmitted by the transmitter on the TxD output.
2. The transmitter is clocked by the receive clock.
3. $\overline{\text{TxRDY}}$ output=1.
4. The $\overline{\text{TxEMT}}/\overline{\text{DSCHG}}$ pin will reflect only the data set change condition.
5. The TxEN command (CR0) is ignored.

In synchronous mode, CR7-CR6=01 places the COM 2651 in the Automatic SYN/DLE Stripping mode. The exact action taken depends on the setting of bits MR17 and MR16:

1. In the non-transparent, single SYN mode (MR17-MR16=10), characters in the data stream matching SYN1 are not transferred to the Receive Data Holding Register (RHR).
2. In the non-transparent, double SYN mode (MR17-MR16=00), characters in the data stream matching, SYN1, or SYN2 if immediately preceded by SYN1, are not transferred to the RHR. However, only the first SYN1 of an SYN1-SYN1 pair is stripped.
3. In transparent mode (MR16=1), characters in the data stream matching DLE, or SYN1 if immediately preceded by DLE, are not transferred to the RHR. However, only the first DLE of a DLE-DLE pair is stripped.

Note that Automatic Stripping mode does not affect the setting of the DLE Detect and SYN Detect status bits (SR3 and SR5).

Two diagnostic sub-modes can also be configured. In Local Loop Back mode (CR7-CR6=10), the following loops are connected internally:

1. The transmitter output is connected to the receiver input.
2. $\overline{\text{DTR}}$ is connected to $\overline{\text{DCD}}$ and $\overline{\text{RTS}}$ is connected to $\overline{\text{CTS}}$.
3. The receiver is clocked by the transmit clock.
4. The $\overline{\text{DTR}}$, $\overline{\text{RTS}}$ and TxD outputs are held high.
5. The $\overline{\text{CTS}}$, $\overline{\text{DCD}}$, $\overline{\text{DSR}}$ and RxD inputs are ignored.

Additional requirements to operate in the Local Loop Back mode are that CR0 (TxEN), CR1 (DTR), and CR5 (RTS) must be set to 1. CR2 (RxEN) is ignored by the COM 2651.

The second diagnostic mode is the Remote Loop Back mode (CR7-CR6=11). In this mode:

1. Data assembled by the receiver is automatically placed in the Transmit Holding Register and retransmitted by the transmitter on the TxD output.
2. The transmitter is clocked by the receive clock.
3. No data are sent to the local processor, but the error status conditions (PE, OE, FE) are set.
4. The $\overline{\text{RxRDY}}$, $\overline{\text{TxRDY}}$, and $\overline{\text{TxEMT}}/\overline{\text{DSCHG}}$ outputs are held high.
5. CR0 (TxEN) is ignored.
6. All other signals operate normally.

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
Operating Mode		**Request to Send**	**Reset Error**	**Sync/Async**	**Receive Control (RxEN)**	**Data Terminal Ready**	**Transmit Control (TxEN)**
00=NORMAL OPERATION 01=ASYNCH: AUTOMATIC ECHO MODE SYNCH: SYN AND/OR DLE STRIPPING MODE 10=LOCAL LOOP BACK 11=REMOTE LOOP BACK		0=FORCE $\overline{\text{RTS}}$ OUTPUT HIGH 1=FORCE $\overline{\text{RTS}}$ OUTPUT LOW	0=NORMAL 1=RESET ERROR FLAG IN STATUS (FE, OE, PE/DLE DETECT)	**ASYNCH: FORCE BREAK** 0=NORMAL 1=FORCE BREAK **SYNCH: SEND DLE** 0=NORMAL 1=SEND DLE	0=DISABLE 1=ENABLE	0=FORCE $\overline{\text{DTR}}$ OUTPUT HIGH 1=FORCE $\overline{\text{DTR}}$ OUTPUT LOW	0=DISABLE 1=ENABLE

TABLE 7—COMMAND REGISTER (CR)

STATUS REGISTER (SR)

The data contained in the Status Register (as shown in Table 8) indicate receiver and transmitter conditions and modem/data set status.

SR0 is the Transmitter Ready (TxRDY) status bit. It, and its corresponding output, are valid only when the transmitter is enabled. If equal to 0, it indicates that the Transmit Data Holding Register has been loaded by the processor and the data has not been transferred to the Transmit Shift Register. If set equal to 1, it indicates that the Holding Register is ready to accept data from the processor. This bit is initially set when the Transmitter is enabled by CR0, unless a character has previously been loaded into the Holding Register. It is not set when the Automatic Echo or Remote Loop Back modes are programmed. When this bit is set, the $\overline{\text{TxRDY}}$ output pin is low. In the Automatic Echo and Remote Loop Back modes, the output is held high.

SR1, the Receiver Ready (RxRDY) status bit, indicates the condition of the Receive Data Holding Register. If set, it indicates that a character has been loaded into the Holding Register from the Receive Shift Register and is ready to be read by the processor. If equal to zero, there is no new character in the Holding Register. This bit is cleared when the processor reads the Receive Data Holding Register or when the receiver is disabled by CR2. When set, the $\overline{\text{RxRDY}}$ output is low.

The TxEMT/DSCHG bit, SR2, when set, indicates either a change of state of the $\overline{\text{DSR}}$ or $\overline{\text{DCD}}$ inputs or that the Transmit Shift Register has completed transmission of a character and no new character has been loaded into the Transmit Data Holding Register. Note that in synchronous mode this bit will be set even though the appropriate "fill" character is transmitted. TxEMT will not go active until at least one character has been transmitted. It is cleared by loading the Transmit Data Holding Register. The DSCHG condition is enabled when TxEN = 1 or RxEN = 1. It is cleared when the Status Register is read by the processor. When SR2 is set, the TxEMT/DSCHG output is low.

SR3, when set, indicates a received parity error when parity is enabled by MR14. In synchronous transparent mode (MR16=1), with parity disabled, it indicates that a character matching the DLE Register has been received. However, only the first DLE of two successive DLEs will set SR3. This bit is cleared when the receiver is disabled and by the Reset Error command, CR4.

The Overrun Error status bit, SR4, indicates that the previous character loaded into the Receive Holding Register was not read by the processor at the time a new received character was transferred into it. This bit is cleared when the receiver is disabled and by the Reset Error command, CR4.

In asynchronous mode, bit SR5 signifies that the received character was not framed by the programmed number of stop bits. (if 1.5 stop bits are programmed, only the first stop bit is checked.) If the RHR contains all 0's when SR5=1, a break condition is present. In synchronous non-transparent mode (MR16=0), it indicates receipt of the SYN1 character in single SYN mode or the SYN1-SYN2 pair in double SYN mode. In synchronous transparent mode (MR16=1), this bit is set upon detection of the initial synchronizing characters (SYN1 or SYN1-SYN2) and, after synchronization has been achieved, when a DLE-SYN1 pair is received. The bit is reset when the receiver is disabled, when the Reset Error command is given in asynchronous mode, or when the Status Register is read by the processor in the synchronous mode.

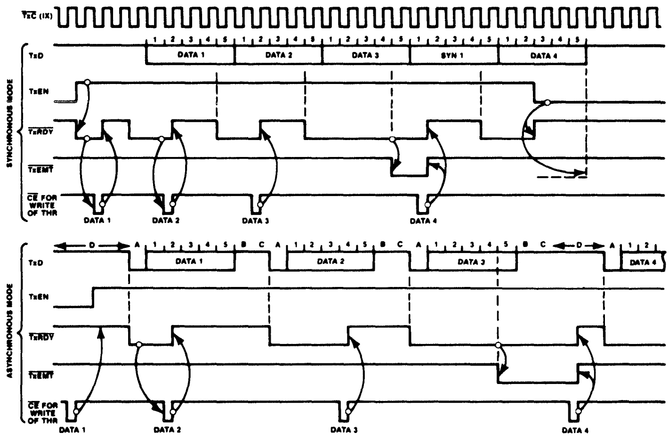
SR6 and SR7 reflect the conditions of the $\overline{\text{DCD}}$ and $\overline{\text{DSR}}$ inputs respectively. A low input sets the corresponding status bit and a high input clears it.

SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
Data Set Ready	**Data Carrier Detect**	**FE/SYN Detect**	**Overrun**	**PE/DLE Detect**	**TxEMT/DSCHG**	**RxRDY**	**TxRDY**
0=$\overline{\text{DSR}}$ INPUT IS HIGH 1=$\overline{\text{DSR}}$ INPUT IS LOW	0=$\overline{\text{DCD}}$ INPUT IS HIGH 1=$\overline{\text{DCD}}$ INPUT IS LOW	**ASYNCH:** 0=NORMAL 1=FRAMING ERROR **SYNCH:** 0=NORMAL 1=SYN CHAR DETECTED	0=NORMAL 1=OVERRUN ERROR	**ASYNCH:** 0=NORMAL 1=PARITY ERROR **SYNCH:** 0=NORMAL 1=PARITY ERROR OR DLE CHAR RECEIVED	0=NORMAL 1=CHANGE IN $\overline{\text{DSR}}$ OR $\overline{\text{DCD}}$, OR TRANSMIT SHIFT REGISTER IS EMPTY	0=RECEIVE HOLDING REG EMPTY 1=RECEIVE HOLDING REG HAS DATA	0=TRANSMIT HOLDING REG BUSY 1=TRANSMIT HOLDING REG EMPTY

TABLE 8—STATUS REGISTER (SR)

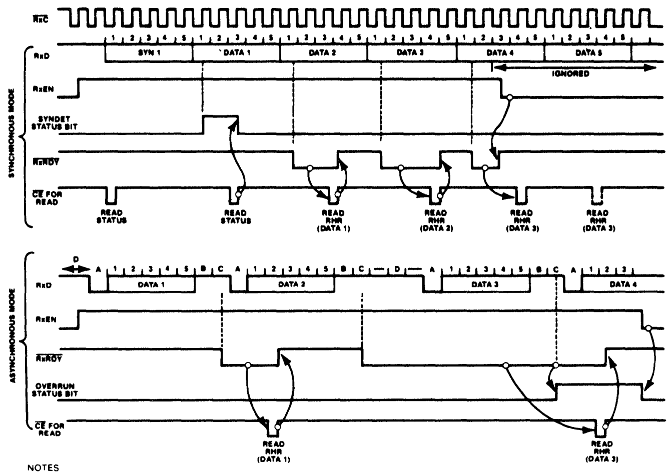
TIMING DIAGRAMS

TxRDY, TxEMT (Shown for 5-bit characters, no parity, 2 stop bits [in asynchronous mode])



NOTES
A = Start bit
B = Stop bit 1
C = Stop bit 2
D = TxD marking condition
TxEMT goes low at the beginning of the last data bit, or if parity is enabled, at the beginning of the parity bit

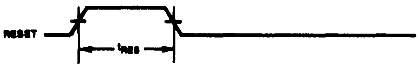
RxRDY (Shown for 5-bit characters, no parity, 2 stop bits [in asynchronous mode])



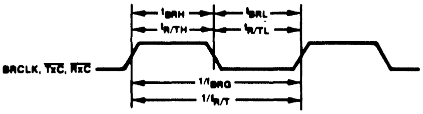
NOTES
A Start bit
B Stop bit 1
C Stop bit 2
D TxD marking condition

TIMING DIAGRAMS (Cont'd)

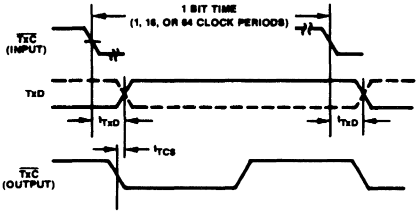
RESET



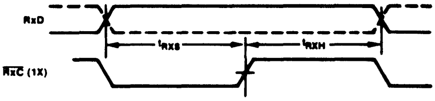
CLOCK



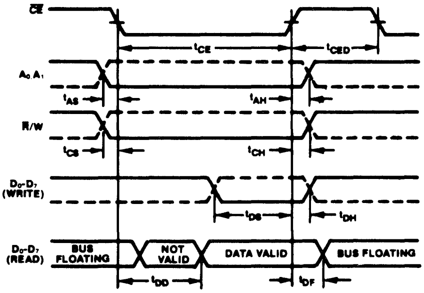
TRANSMIT



RECEIVE



READ AND WRITE



MAXIMUM GUARANTEED RATINGS*

Operating Temperature Range	0°C to + 70°C
Storage Temperature Range	−55°C to +150°C
Lead Temperature (soldering, 10 sec.)	+325°C
Positive Voltage on any Pin, with respect to ground	+18.0V
Negative Voltage on any Pin, with respect to ground	−0.3V

*Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

NOTE: When powering this device from laboratory or system power supplies, it it important that the Absolute Maximum Ratings not be exceeded or device failure can result. Some power supplies exhibit voltage spikes or "glitches" on their outputs when the AC power is switched on and off. In addition, voltage transients on the AC power line may appear on the DC output. For example, the bench power supply programmed to deliver +12 volts may have large voltage transients when the AC power is switched on and off. If this possibility exists it is suggested that a clamp circuit be used.

DC ELECTRICAL CHARACTERISTICS T_A=0°C to +70°C, V_{CC}=5.0V ±5%

PARAMETER		MIN	TYP	MAX	UNIT	TEST CONDITIONS
	Input voltage					
V_{IL}	Low			0.8	V	
V_{IH}	High	2.0				
	Output voltage					
V_{OL}	Low			0.4	V	I_{OL}=1.6mA
V_{OH}	High	2.4				I_{OH}=−100µA
I_{IL}	Input leakage current			10	µA	V_{IN}=0 to 5.25V
	Output leakage current					
I_{LH}	Data bus high			10	µA	V_O=4.0V
I_{LL}	Data bus low			10	µA	V_O=0.45V
I_{CC}	Power supply current			150	mA	
	Capacitance					
C_{IN}	Input			20	pF	
C_{OUT}	Output			20	pF	fc=1MHz Unmeasured pins tied to ground
$C_{I/O}$	Input/Output			20	pF	

AC ELECTRICAL CHARACTERISTICS T_A=0°C to +70°C, V_{CC}=5.0V ±5%

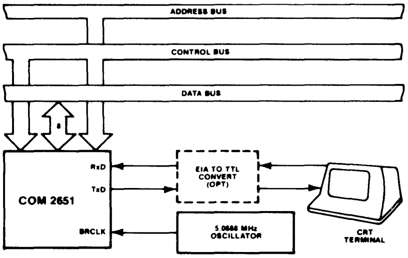
PARAMETER		MIN	TYP	MAX	UNIT	TEST CONDITIONS
	Pulse width					
t_{RES}	Reset	1000			ns	
t_{CE}	Chip enable	300			ns	
	Setup and hold time					
t_{AS}	Address setup	20			ns	
t_{AH}	Address hold	20			ns	
t_{CS}	$\overline{R}$/W control setup	20			ns	
t_{CH}	$\overline{R}$/W control hold	20			ns	
t_{DS}	Data setup for write	225			ns	
t_{DH}	Data hold for write	0			ns	
t_{RXS}	Rx data setup	300			ns	
t_{RXH}	Rx data hold	350			ns	
t_{DD}	Data delay time for read			250	ns	C_L=100pF
t_{DF}	Data bus floating time for read			150	ns	C_L=100pF
t_{CED}	$\overline{CE}$ to $\overline{CE}$ delay	700			ns	
	Input clock frequency					
f_{BRG}	Baud rate generator	1.0	5.0688	5.0738	MHz	
$f_{R/T}$[1]	$\overline{TxC}$ or $\overline{RxC}$	dc		1.0	MHz	
	Clock width					
t_{BRH}	Baud rate high	70			ns	f_{BRG}=5.0688MHz
t_{BRL}	Baud rate low	70			ns	f_{BRG}=5.0688MHz
$t_{R/TH}$[1]	$\overline{TxC}$ or $\overline{RxC}$ high	500			ns	
$t_{R/TL}$[1]	$\overline{TxC}$ or $\overline{RxC}$ low	500			ns	
t_{TXD}	TxD delay from falling edge of $\overline{TxC}$			650	ns	C_L=100pF
t_{TCS}	Skew between TxD changing and falling edge of $\overline{TxC}$ output		0		ns	C_L=100pF

NOTE:

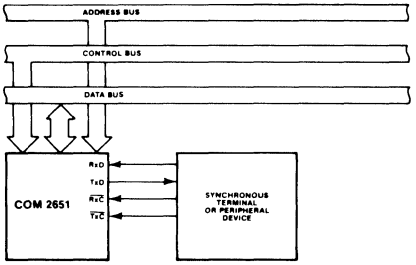
[1] $f_{R/T}$ and $t_{R/TL}$ shown for all modes except Local Loopback. For Local Loopback mode $f_{R/T}$=0.7 MHz and $t_{R/TL}$=700ns min.

TYPICAL APPLICATIONS

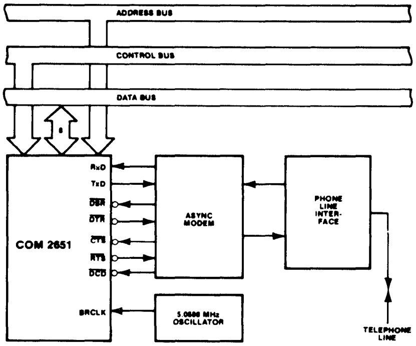
ASYNCHRONOUS INTERFACE TO CRT TERMINAL



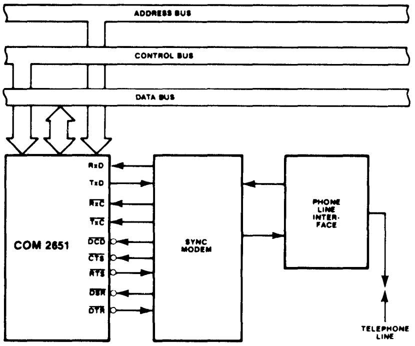
SYNCHRONOUS INTERFACE TO TERMINAL OR PERIPHERAL DEVICE



ASYNCHRONOUS INTERFACE TO TELEPHONE LINES



SYNCHRONOUS INTERFACE TO TELEPHONE LINES



STANDARD MICROSYSTEMS CORPORATION
35 Marcus Blvd Hauppauge N.Y. 11788
(516) 273 3100 TWX 510 227 8898

Circuit diagrams utilizing SMC products are included as a means of illustrating typical semiconductor applications, consequently complete information sufficient for construction purposes is not necessarily given The information has been carefully checked and is believed to be entirely reliable However, no responsibility is assumed for inaccuracies Furthermore, such information does not convey to the purchaser of the semiconductor devices described any license under the patent rights of SMC or others SMC reserves the right to make changes at any time in order to improve design and supply the best product possible

B. List of Materials for TMDS3411804420 Board (SDB)

ITEM	PART DESCRIPTION	LOCATION	QTY/BRD
1	TMDS3411804420 Software Development System PC Board		1
2	Connector DB9 9-Pin Female, AMP 745112-2, or Burndy 107-103-18, or equivalent	J4	1
3	Connector DB25 25-Pin Female, AMP 745106-2, or equivalent	J3	1
4	74AS04 Hex Inverter	U43	1
5	74AS32 Quad OR Gate	U42	1
6	74ALS245 Bidirectional Buffer	U2,U7,U8	3
7	74AS244 Octal Buffer	U6	1
8	74ALS244 Octal Buffer	U27,U46	2
9	74ALS573 Octal Latch	U10	1
10	74AS573 Octal Latch	U9	1
11	74AS00 Quad NAND Gate	U41	1
12	74AS74 Dual D Flip-Flop	U12,U26	2
13	74AS161 Counter	U25	1
14	74ALS257 Quad 2 to 1 Multiplexer	U28,U29	2
15	74AS11 Three-Input AND Gate	U45	1
16	74LS125 Quad Buffer	U44	1
17	74AS257 Quad 2 to 1 Multiplexer	U34	1

ITEM	PART DESCRIPTION	LOCATION	QTY/BRD
18	74ALS541 Octal Buffer	U5	1
19	74ALS878 Octal Latch with Clear	U33	1
20	TBP28S42N Bipolar PROM (optional)	U35,U36	2
21	TIBPAL16L8-25 Programmable Array Logic	U3	1
22	TIBPAL20L10 Programmable Array Logic	U11	1
23	Oscillator, 40 MMHz, Dale XO-33-D-40 or equivalent	U4	1
24	Oscillator, 25 MHz, Dale XO-33-D-25 or Equivalent	U30	1
25	Oscillator, 5.0688 Mz, Dale, XO-33-D-5.07 or Equivalent	U38	1
26	Resistor Pack, 8-Pin SIP, 10K OHM, Bourns 4308R-101-103	RP3	1
27	Resistor Pack, Isolated, 8-Pin SIP, 33 Ohms, Bourns 430Br-1020330	RP1	1
28	Resistor Pack, Isolated, 16-Pin DIP, 33 Ohms, Bourns 4116R-001-330	RP2	1
29	TMS34070NL Color Palette	U32	1
30	TMS34010 Graphics System Processor	U1	1
31	TM4256EC4-12L 256K by 4, Dynamic RAM SIP	U13-U16	4
32	TM4161EV4-15L 64K by 4, Video RAM SIP	U17-U24	8
33	COM2651 SMC USART	U39	1
34	75158 Line Driver	U31	1
35	75188 Line Driver	U40	1

ITEM	PART DESCRIPTION	LOCATION	QTY/BRD
36	75189 Receiver	U37	1
37	Socket, 68-Pin PLCC, AMP 821574-1 or equivalent	XU1	1
38	Socket, 22-Pin DIP, 400-mil, Stamped Contact Tin-Plated, AMP 2-643295-1	XU32	1
39	Socket, 20-Pin DIP, 300-mil, Stamped Contact Tin-Plated, AMP 2-643294-1	XU3,XU35,XU36	3
40	Socket, 24-Pin DIP, 300-mil, Stamped Contact Tin-Plated, AMP 2-641932-1	XU11	1
41	Capacitor Tantalum 22 uF, Panasonic ECS-F1CE226K	C101-C104	4
42	Capacitor Decoupling 0.01 uF, AVX MD015E103ZAA	C1-C12,C25-C39, C41-C46	33
43	Resistor 31.6 Ohm 1/8 W, TRW	R4, R8, R11	3
44	Resistor 68.1 Ohm 1/8 W, TRW	R6, R10, R13	3
45	Resistor 46.4 Ohm 1/8 W, TRW	R5, R9, R12	3
46	Side Edge Bracket		1
47	Mounting Kit for Edge Bracket, AMP 205817-1		2
48	Stake Pins, 0.025 in. Square, BERG 75481-002	W1 to W9	47
49	Rivets for J3 and J4, 0.124 in. Diameter by 0.25 in. length		4

C. Diagnostics for Software Development Board

This appendix covers the following:

C.1 SDB Diagnostic Overview

The 340 Software Development Board (SDB340) comes completely tested from the factory. However, should you ever suspect the board is not performing properly, these diagnostics provide a quick and easy method to verify the unit's integrity.

C.1.1 Diagnostic Disk Contents

The diagnostics come on the diskette marked:

```
Demo & Diagnostic
```

The contents of the diagnostic disk are:

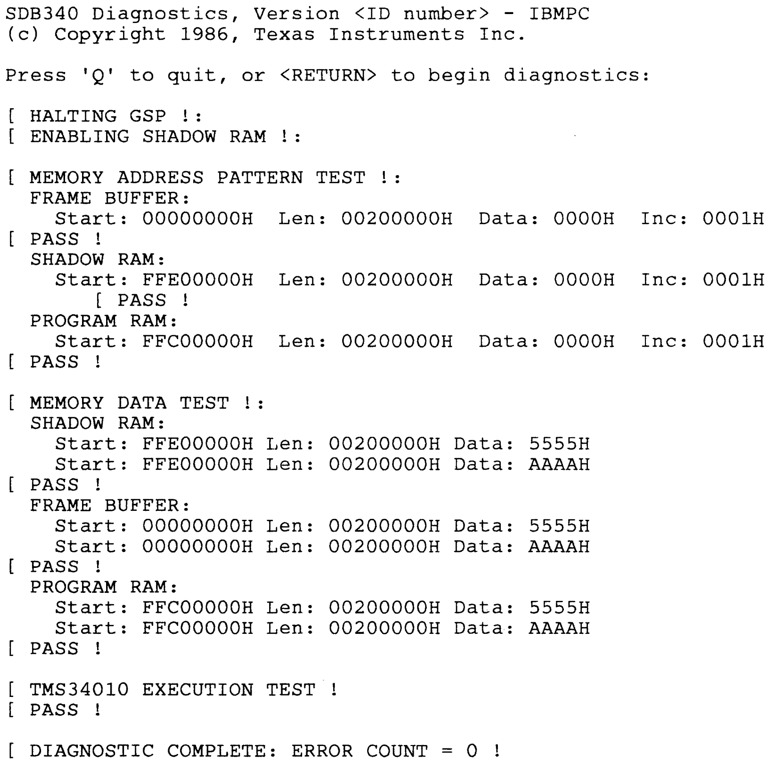
README 1ST	Read first documentation
SDBDIAG EXE	Main diagnostic (IBM-PC Version)
SDBDIAGT EXE	Main diagnostic (TI-PC Version)
SDBDIAG OUT	COFF load file used by diagnostics

C.1.2 Installation and Operation

The diagnostics can either be executed from the floppy disk or from a Winchester disk. In either case, both SDBDIAG.EXE (SDBDIAGT.EXE for TI PC) and SDBDIAG.OUT must be located on the currently selected disk or directory. To execute the diagnostics off the diskette, insert it into drive A, then select the drive by entering A: at the command prompt. Next invoke the diagnostics by typing SDBDIAG. For example:

```
>A:<CR>
>SDBDIAG <CR>
```

A short message will appear on the screen with the version number of the diagnostics and the computer type it was intended to run on. At this point you will be prompted to press 'Q' to quit or to press any other key to begin the diagnostics. If no errors are encountered, the screen display will look as shown in Section 4.5.



```
SDB340 Diagnostics, Version <ID number> - IBMPC
(c) Copyright 1986, Texas Instruments Inc.

Press 'Q' to quit, or <RETURN> to begin diagnostics:

[ HALTING GSP !:
[ ENABLING SHADOW RAM !:

[ MEMORY ADDRESS PATTERN TEST !:
  FRAME BUFFER:
    Start: 00000000H  Len: 00200000H  Data: 0000H  Inc: 0001H
[ PASS !
  SHADOW RAM:
    Start: FFE00000H  Len: 00200000H  Data: 0000H  Inc: 0001H
       [ PASS !
  PROGRAM RAM:
    Start: FFC00000H  Len: 00200000H  Data: 0000H  Inc: 0001H
[ PASS !

[ MEMORY DATA TEST !:
  SHADOW RAM:
    Start: FFE00000H Len: 00200000H Data: 5555H
    Start: FFE00000H Len: 00200000H Data: AAAAH
[ PASS !
  FRAME BUFFER:
    Start: 00000000H Len: 00200000H Data: 5555H
    Start: 00000000H Len: 00200000H Data: AAAAH
[ PASS !
  PROGRAM RAM:
    Start: FFC00000H Len: 00200000H Data: 5555H
    Start: FFC00000H Len: 00200000H Data: AAAAH
[ PASS !

[ TMS34010 EXECUTION TEST !
[ PASS !

[ DIAGNOSTIC COMPLETE: ERROR COUNT = 0 !
```

Figure C-1. Screen Display with No Errors

Output of the SDB340 diagnostics can be redirected to a device other than the screen by specifying the SDBDIAG command followed by a greater-than arrow and the name of the output device.

EXAMPLE:

`SDBDIAG >PRN` (Redirects output to the printer)

Your DOS Users Manual contains terms identifying peripheral devices in the command section. Further information on redirecting output is in these manuals.

C.2 Explanation of Diagnostic Tests

The following briefly explains the different diagnostic tests and the possible status/error messages.

C.2.1 Status Messages

During testing, messages indicate action being taken by the diagnostics. For example:

```
TMS34010 HALTED,
TMS34010 RUNNING,
CACHE FLUSH,
CACHE DISABLE,
CACHE ENABLE,
SHADOW RAM ENABLED.
```

C.2.2 Memory Address Pattern Tests

An incremental pattern is written to SDB memory and then verified. This verifies host port operation and checks the SDB for memory address failures. Each of the three SDB memory segments is tested independently and a PASS or FAIL message is printed after each. The section of memory currently under test is indicated by it's name followed by a starting bit address followed by the length of the segment.

EXAMPLE

```
FRAME BUFFER
    Start: 00000000H Len: 002000000h Data: 0000H Inc: 0001H
```

This indicates the video frame buffer starting at memory address >00000000 is currently under test.

Note:

If the SDB fails the Memory Address Pattern Test, then subsequent tests will produce unreliable results.

C.2.3 Memory Data Pattern Tests

For these tests, an non-incremental pattern is written to SDB memory and then verified to check the SDB340 for memory data failures. Each of the three SDB memory segments is tested independently and a PASS or FAIL message is printed after each. The section of memory currently under test is indicated by it's name followed by a starting bit address followed by the length of the segment:

```
FRAME BUFFER
    Start: 00000000H Len: 002000000h Data: 0000H Inc: 0001H
```

This indicates that the video frame buffer starting at memory address >00000000 is currently under test.

C.2.4 TMS34010 Execution Test

COFF file SDBDIAG.OUT is loaded into SDB memory starting at memory location >00000000. An NMI vector, set up via the host port, points to the beginning of the code. TMS34010 execution then begins by the host initiating an NMI. During the course of the test, the host processor and the TMS34010 send packets of data back and forth. The test fails if

- incorrect data is received by the host, or
- the TMS34010 doesn't respond after an allotted amount of time.

C.3 Troubleshooting

This section describes steps to take if one or more of the diagnostic tests fail.

If a diagnostic test fails, then execute the following steps:

1) Make sure your PC is able to handle the additional power requirements as listed in Table 5-7.

2) Make certain the version of the SDB340 you are using is intended for your computer. The first message appearing after execution shows whether it is an IBM PC or TI PC version.

3) Turn off the power and remove the SDB340 from the host computer. Verify that all components are seated firmly in their sockets.

4) Verify a proper installation by rechecking the installation steps in Section 3.1.4 on page 3-6.

5) Recheck the board jumper settings as listed in the tables in Section 2.3 on page 2-4.

6) Check the edge connector of the SDB340, if it appears dirty, clean it by **gently** rubbing the connector with a pencil eraser.

7) Re-install the SDB340 board, power up the computer and run the diagnostics again. If you continue to get failures, contact the Technical Assistance Hotline at (713) 274-2340.

D. Glossary

absolute address: An address that is permanently assigned by the machine designer to a storage location.

absolute coordinates: The location of a point in terms of X, Y, or Z distance from a predefined origin.

access time: The time interval between the request for information and the instant this information is available.

address: A point into an array of bits, bytes or words of information.

aliasing: A stairstep effect on a raster display of a line or arc segment.

ALU: Arithmetic Logic Unit, a computational element of a digital computer which performs boolean or arithmetic operations.

analog outputs: As opposed to digital output, the amplitude is continuously proportionate to the stimulus, the proportionality being limited by the accuracy of the device.

asynchronous communications: A method of transmitting data in which the timing of character placement of connecting transmitting lines is not critical. The transmitted characters are preceded by a start and followed by a stop bit, thus permitting the interval between characters to vary.

array: 1. A series of related items. 2. An ordered arrangement or pattern of items or numbers, such as a determinant, matrix, vector,or a table of numbers.

ASCII: (American National Standard Code for Information Interchange,1968) The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

aspect ratio: The ratio of width to height. For the rectangular picture transmitted by a television station, the aspect ratio is 4:3.

assemble: To prepare a machine language program from a symbolic language program by substituting absolute operation codes for symbolic operation codes and absolute of relocatable addresses for symbolic addresses.

assembler: A software program that assembles.

assembly language: A programming language which allows a computer user to write a program using mnemonics instead of numeric instructions. It is a low-level symbolic programming language which closely resembles machine code language. The language uses groups of letters; each group represents a single instruction.

attribute: A parameter specifying some characteristic or feature to be applied to subsequent pictorial information.

back porch: The portion of a horizontal blanking pulse that follows the trailing edge of the horizontal synchronizing pulse.

background illumination: The average brightness of a screen.

bandwidth: The difference in frequency between the highest and lowest frequencies involved.

base: 1. A reference value. 2. A number that is multiplied by itself as many times as indicated by an exponent. 3. Same as radix.

base address: A given address from which an absolute address is derived by combination with a relative address.

bidirectional buffer: A buffer capable of acting as an input or as an output but not both at the same time.

bit: A binary digit; usually 0 or 1. (Note: MSb = most-significant bit; LSb = least-significant bit.)

BitBlt: Bit aligned block transfer. Transfer of a rectangular array of pixel information from one location in a bitmap to another with potential of applying 1 of 16 boolean operators during the transfer.

bit map: 1. The digital representation of an image in which bits are mapped to pixels. 2. A block of memory used to hold raster images in a device-specific format.

bit plane: Hardware used as a storage medium for a bit map.

black level: The amplitude of the composite signal at which the beam of the picture tube is extinguished (becomes black) to blank retrace of the beam. This level is established at 75&PCT. of the signal amplitude.

blanking signal: Pulses used to extinguish the scanning beam during horizontal and vertical retrace periods.

border: The area of the physical display that is outside the display area on a CRT display.

branching: A method of selecting, on the basis of results, the next operation to execute while the program is in progress.

breakpoint: A place in a routine specified by an instruction, instruction digit, or other condition, where the routine may be interrupted by external intervention or by a monitor routine.

byte: An 8-bit sequence of adjacent binary digits operated upon as a unit. (Note: MSB = most-significant byte; LSB = least-significant byte.)

central processor unit (CPU): Part of a computer system which contains the main storage, arithmetic unit, and special register groups. It performs arithmetic operations, controls instruction processing, and provides timing signals and other housekeeping operations.

CGI: Computer Graphics Interface. The interface between the device- independent and the device-dependent levels of a graphics system.

CGM: Computer Graphics Metafile. A mechanism for retaining and and transporting graphics data and control information at the level of the Virtual Device Interface.

character: A letter, digit, or other symbol that is used as part of the organization, control, or representation of data.

character field: The rectangular area within which a character is displayed. Also known as image cell. The character field includes intercharacter and interrow spacing.

clipping: Removing parts of display elements that lie outside a given boundary, usually a window or a viewport.

compiler: A translation program that converts a high level language set of instructions into a target machines assembly language.

composite video: The color-picture signal plus all blanking and synchronizing signals. The signal includes luminance and chrominance signals, vertical- and horizontal-sync pulses, vertical- and horizontal-sync pulses, vertical-and horizontal-blanking pulses, and the color-burst signal.

coordinates: A number of X, Y, and Z units that give the location of a point in a coordinate system.

CRT: Cathode Ray Tube. A display tube with a television-like screen.

DAC: Digital-to-analog converter. A device that converts a digital input code to an analog output voltage or current. The analog output level represents the value of the digital input code.

direct access: Pertaining to the process of obtaining data from, or placing data into, storage where the time required for such access is independent of the location of the data most recently obtained or placed in storage.

direct addressing: Method of programming that has the address pointing to the location of data or the instruction that is to be used.

display: A visual representation of data.

display area: The rectangular part of the physical display screen in which information coded in conformance with a video encoding standard is visibly displayed. The display area does not include the border area.

display element: A basic graphic element that can be used to construct a display image.

display memory: The area of memory which is used to hold the graphics image output to the video monitor.

display pitch: The difference in memory addresses between two pixels that appear in vertically adjacent positions (one directly above the other) on the screen.

display unit: A device which provides a visual representation of data.

dot clock: The dot clock controls the rate at which analog video data is output at the analog outputs (RED, GRN and BLU) and the digital output, XAT. All on-chip timing is generated from this clock.

download: To call for and receive a file from another computer storage medium.

dump: To copy the contents of all or part of a storage, usually from an internal storage.

endpoint: The end of a line segment expressed in terms of X, Y, and Z coordinates.

fetch: That portion of a computer cycle during which the next instruction is retrieved from memory.

field: A set of scanning lines that, when interlaced with other such sets, constructs a complete picture on a television or similar raster-scan device.

fill: Solid coloring or shading of a display surface, often achieved as a pattern of horizontal segments.

flag: A binary status indicator whose state indicates whether a particular condition has occurred or is in effect.

frame: 1. The time required to refresh an entire screen. 2. The screen image output during a single vertical sweep.

frame buffer: A portion of memory used to buffer rasterized data to be output to a CRT display monitor. The contents of the frame buffer are often referred to as the bit map of the display and contain the logical pixels corresponding to the points on the monitor screen.

front porch: The portion of a horizontal blanking pulse that precedes the leading edge of the horizontal sync pulse.

GKS: Graphical Kernal System. An application programmer's interface to graphics.

glue logic: The small- and medium-scale-integrated devices necessary to complete the interface between two or more large or very-large-scale integrated devices.

gray scale: A scale of light intensities from black to white.

high impedance: The third state of a three-state output driver, in which the output is driven neither high or low but behaves as an open connection.

hold time: The minimum amount of time that valid data must be present at an input after the device is clocked to ensure proper data acceptance.

horizontal blanking interval: The time during which the display is blanked to cover the horizontal retracing of the electron beam.

horizontal sync: The synchronization signal that enables horizontal retrace of the electron beam of a CRT display. retracing of the electron beam.

icon: A graphic symbol representing a menu item.

indirect addressing: Programming method that has the initial address being the storage location of a word that contains another address. This indirect address is then used to obtain the data to be operated upon.

interlaced scanning: A system of TV-picture scanning. Odd-numbered scanning lines, which make up an odd field, are interlaced with the even-numbered lines of an even field. The two interlaced fields constitute one frame. In effect, the number of transmitted pictures is doubled, thus reducing flicker.

instruction: A statement that specifies an operation and the values or locations of its operands.

instruction cycle: The period of time during which a programmed system obeys a particular instruction.

instruction set: A set of operation codes for a particular computer or family of processors.

interlaced scanning: A system of TV-picture scanning. Odd-numbered scanning lines, which make up an odd field, are interlaced with the even-numbered lines of an even field. The two interlaced fields constitute one frame. In effect, the number of transmitted pictures is doubled, thus reducing flicker.

interrupt: To stop a process in such a way that it can be resumed.

jump: A departure from the normal sequence of executing instructions in a computer.

jump conditions: Conditions defined in a transition table that determine the changes of flip-flops from one state to another state.

label: One or more characters used to identify a statement or an item of data in a computer program.

language: A set of representations, conventions, and rules used to convey information.

linearity: 1. The relationship between two quantities when a change in a second quantity is directly proportional to a change in the first quantity. 2. A constant ratio of cause and effect (as in a straight line representation).

linkage: In programming, coding that connects two separately coded routines.

load: In programming, to enter data into storage or working registers.

location: Any place in which data may be stored.

lookup table: A table used during scan conversion of the digital image that converts color-map addresses into the actual color values displayed.

loop: A sequence of instructions that is executed repeatedly until a terminal condition prevails.

LSb: Least significant bit.

LSB: Least significant byte.

machine code: An operation code that a machine is designed to recognize. Usually expressed in ones and zeros.

machine language: The basic language of a computer. Programs written in machine language require no further interpretation by a computer.

macro: A command that allows a few keystrokes to reproduce a longer string of characters.

macroinstruction: An instruction in a source language that is equivalent to a specified sequence of machine instructions.

macroscopic: 1. Large enough to be observed by the naked eye. 2. Considered in terms of large units or elements.

mapping: An operation that transforms one functional representation of information to another.

mask: A pattern of characters that is used to control the retention or elimination of portions of another pattern of characters.

matrix: An array of X, Y, and Z coefficients for calculating a geometric transformation.

memory: The section of the computer where instructions and data are stored; synonymous with storage.

microprocessor: An IC that can be programmed with stored instructions to perform a wide variety of functions, consisting at least of a controller, some registers, and some sort of ALU.

monitor: A display device used for monitoring a video transmission.

monochrome monitor: A monitor capable of displaying intensities of only a single color.

monolithic integrated circuit: An integrated circuit formed in a single piece, as opposed to a hybrid circuit formed by connecting several pieces.

monotonicity: The quality of proceeding in a uniform manner. For example, the analog level output from a DAC should increase with each increase in the value of the digital input code.

multiplexing: Refers to a process of transmitting more than one set of signals at a time over a single wire or communications link.

MSb: Most significant bit.

MSB: Most significant byte.

NABTS: North American Broadcast Teletext Specification

NAPLPS: North American Presentation Level Protocol Syntax -- a proposed standard for Videotex services.

nonmaskable interrupt: An interrupt request that cannot be disabled.

NTSC: National Television System Committee -- a group representing a wide range of interests in the television broadcast and video industry. The NTSC is instrumental in developing standards.

object code: Output from a compiler or assembler which is itself executable machine code.

object language: The language to which a statement is translated.

object program: The instructions which come out of the compiler or assembler, ready to run on the computer. The object program is the one which can be read by both machines and people.

operand: That which is operated upon. An operand is usually identified by an address part of an instruction.

operation: 1. A defined action, namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result of any permissible combination of operands. 2. The set of such acts specified by such a rule, or the rule itself. 3. The act specified by a single computer instruction. 4. A program step undertaken or executed by a computer, e.g., addition, multiplication, extraction, comparison, shift, transfer. The operation is usually specified by the operator part of an instruction. 5. The specific action performed by a logic element.

origin: The zero intersection of X, Y, and Z axes from which all points are calculated.

overlay: The plane of a graphics display that can be superimposed on another plane.

pack: To compress data in a storage medium by eliminating redundant information in such a way that the original data can later be recovered.

palette: 1. Thin oval or oblong board with a thumb hole, used by artists for mixing and holding colors. 2. A digital lookup table used in a computer graphics display for translating data from the bit map into the pixel values to be shown on the display.

pan: Movement across the X and Y grid.

phase: The time interval for each clock period in a system is divided into two phases. One phase corresponds to the time the clock signal is high, and the other phase corresponds to the time the clock signal is low.

PHIGS: The programmer's Hierarchical Interactive Graphics Standard

pipelining: A design technique for reducing the effective propagation delay per operation by partitioning the operation into a series of stages, each of which performs a portion of the operation. A series of data is typically clocked through the pipeline in sequential fashion, advancing one stage per clock period.

PixBlt (abbreviation of Pixel Block transfer): Operations on arrays of pixels in which each pixel is represented by one or more bits. PixBlt operations are a superset of BitBlt operations, and include not only the commonly-used boolean functions, but also integer arithmetic and other multi-bit operations.

pixel: Picture element. 1. The smallest controllable point of light on a CRT display screen. 2. In a bit-mapped display, the logical data structure that contains the attributes to be shown at the corresponding physical pixel position on the CRT display screen.

primary colors: A set of three colors from which all other colors may be regarded as derived; hence, any of a set of visual stimuli from which all colors may be produced by mixture. Each primary color must be different from the others, and a combination of two primaries must be capable of producing a third. In color television, the three primary colors are red, green and blue.

primitive: The basic display element: point, segment, alphanumeric character, or marker.

primitive attribute: A visual characteristic of an output primitive, such as character size, line style, or blink rate.

priority interrupt: Designation given to method of providing some commands to have precedence over others thus giving one condition of operation priority over another.

program: 1. A series of actions proposed in order to achieve a certain result. 2. Loosely, a routine. 3. To design, write, and test a program as in definition 1 above. 4. Loosely, to write a routine.

prompt: Output to the operator indicating that a specific input device is available.

propagation delay: The time required for a change in logic level at an input to a circuit to be translated into a resulting change at an output.

protocol: A set of rules, formats, and procedures governing the exchange of information between peer processes at the same level.

pulse width: Pulse width, T_W. The time interval between specified reference points on the leading and trailing edges of the pulse waveform.

random access memory (RAM): A memory element which can be written to as well as read.

raster: A rectangular grid of picture elements whose intensity levels are manipulated to represent images. In a bit-mapped display, the bits within a portion of the memory referred to as the frame buffer are mapped to the raster pattern of a CRT monitor.

raster display: A CRT display generated by an electron beam that illuminates the CRT by sweeping the beam horizontally across the phosphor surface in a predetermined pattern, providing substantially uniform coverage of the display area.

raster graphics: Computer graphics in which a display image is composed of an array of pixels arranged in rows and columns.

Raster-Op: The arithmetic or logical combination operation that takes place during the transfer of pixel arrays from one location to another.

raster scan: The grid pattern traced by the electron beam on the face of the CRT in a television or similar raster-scan display device.

read only memory (ROM): A semiconductor storage element containing permanent data preprogrammed at the factory which cannot be changed.

real time: Pertaining to the performance of a computation during the actual time that the related physical process transpires, in order that results of the computation interact with the physical process.

refresh: Method which restores charge on capacitance which deteriorates because of leakage.

register: Temporary storage area for digital data.

relative address: The number that specifies the differnce between the absolute address and the base address.

relative coordinates: Location of a point relative to another data point.

relocate: In computer programming, to move a routine from one portion of storage to another and to adjust the necessary address references so that the routine, in its new location, can be executed.

reset: To restore to normal action.

resolution: The number of visible distinguishable units in the device coordinate space.

retrace: The line traced by the scanning beam or beams of a picture tube as it travels from the end of one horizontal line or field to the beginning of the next line or field.

RGB monitor: Red-Green-Blue Monitor. An RGB monitor is a CRT monitor capable of displaying colors and having separate inputs for the three signals used to drive the red, green and blue guns of the CRT.

rotate: To transform a display or display item by revolving it around a specified axis or center point.

routine: An ordered set of instructions that may have some general or frequent use.

scale: A size change made by multiplying or dividing the coordinate dimensions.

scale factor: The value by which you divide or multiply the display dimensions in a scaling operation.

scaling: Enlarging or reducing all or part of a display image by multiplying the coordinates of display elements by a constant value.

scan: To traverse the surface of the disc with the video displayed.

scan line: A horizontal line traced across a CRT by the electron beam in a television or similar raster-scan device.

scrolling: Moving text strings or graphics vertically or horizontally.

segment: A collection of display elements that can be manipulated as a unit.

sequencing: Control method used to cause a set of steps to occur in a particular order.

setup time: The minimum amount of time that valid data must be present at an input before the device is clocked to ensure proper data acceptance.

shift: A movement of data to the right or left.

shift register: A register in which the stored data can be moved from left to right, or vice versa.

sign position: A position, normally located at one end of a number, that contains an indication of the algebraic sign of the number.

simulator: A device, system, or computer program that represents certain features of the behavior of a physical or abstract system.

software: A set of computer programs, procedures, and possibly associated documentation concerned with the operation of a data processing system, e.g., compilers, library routines, manuals, circuit diagrams.

source language: The language from which a statement is translated.

source program: A computer program written in a source language.

sprite: A graphic object of a specified pattern appearing on its plane in a position determined by a single coordinate pair, specifying the sprite's location on the screen in the horizontal and vertical axis.

stairstepping: Jagged raster representation of diagonals or curves, corrected by antialiasing.

static storage elements: Storage elements which contain storage cells that retain their information as long as power is applied unless the information is altered by external exitation.

stored program: A set of instructions in memory specifying the operations to be performed.

subroutine: A routine that can be part of another routine.

superimposed: Refers to the process that moves data from one location to another, superimposing bits or characters on the contents of specified locations.

symbol: A letter, numeral or mark which represents a numeral, operation or relation. An element of a computer languages's character set.

syntax: The grammatical and structural rules of a language. All higher level programming languages possess a formal syntax.

system diagnostics: Means of self-testing a system under normal operating conditions.

trace: A line of the graphics display.

transformation: Geometric alteration of a graphics display, such as scaling, translation, or rotation.

TTL: Transistor-transistor logic. A kind of bipolar circuit logic that takes its name from the way the basic transistor components are interconnected.

variable: A quantity that can assume any of a given set of values.

vertical blanking interval: The time during which the display is blanked to cover the vertical retracing of the electron beam.

vertical blanking pulse: A positive or negative pulse developed during vertical retrace and appearing at the end of each field. It is used to blank out scanning lines during the verticle retrace interval.

vertical sync: The synchronization signal that enables vertical retrace of the electron beam of a CRT display.

video: That part of a television or similar display device having to do with the reception and generation of the image, as distinguished from audio.

video display processor: A microprocessor device dedicated to the tasks of display memory management (storage, retrieval, and refresh) and generation of all required video, control, and synchronization signals required by a TV display or CRT monitor.

video overlay: The mixing of one video signal with another such that parts of the image carried by the first signal replace the corresponding parts of the image carried by the second signal.

video RAM, VRAM: Video Random-Access Memory. A dual-ported memory device for computer graphics applications, containing two interfaces; one interface to allow a processor to read or write data from an internal memory array; a second interface to provide a serial stream of screen refresh data to a CRT display device.

viewport: The specified window on the display surface that marks the limits of a display.

virtual coordinate system: A coordinate system created by mapping a portion of the world coordinate system to the space available on your device.

virtual space: Space referenced with the coordinates defined by the application.

window: A specified rectangular area of a virtual space shown on the display.

window clipping: Blanking line segments at window boundaries.

wire frame: A three-dimensional image displayed as a series of line segments outlining its surface.

word: A character string or a bit string considered as an entity.

world coordinate system: A device-independent coordinate system used to define display objects.

zoom: To scale a display or display item so it is mangified or reduced on the screen.

zooming: Enlarging or reducing all or part of a display image by multiplying the coordinates of display elements by a constant value.

E. Hands-On Tutorial

This appendix contains several demonstrations from Section 3, with suggestions added to show how you can experiment with the software used in that section. This repeat of Section 3 contains additional text in italics which explain how to modify registers and execute the demo software to discover the effects of such changes. It is recommended that you first run through the exercises in Section 3 (not italicized) before attempting this section.

HANDS-ON DEMONSTRATIONS

The main difference between this section and the tutorial in Section 3 are the suggestions in this appendix to first modify the machine state. In this way, we can visually show how changes in such things as register contents can effect the graphic outcome. It also is meant to encourage experimentation by yourself to further your understanding of the TMS34010. The suggested modifications herein are in italics. You can first run through the regular tutorial demonstrations (not italicized), then run the italicized instructions which start page E-7.

The added italicized suggestions in this section (vs. Section 3) are meant to change the machine state before executing instruction demonstrations. Thus, several aspects of the TMS34010 will be shown such as construction of specific registers. You can first run through the regular tutorial demonstrations (not italicized), then run the italicized instructions which start on page E-7.

In the italicized instructions, keystroke summaries in parentheses are to be used only when values in Table E-4 are preset.

Also note that:

- Not all parts of Section 3 are repeated here, but only the sections on:

  PIXT
  DRAV, and
  FILL.

- This appendix does not contain the introductory material in Section 3.

- However, the installation section is repeated for your convenience.

Table E-1 is provided as a reference when making color changes.

Table E-1. Numerical Values for Colors

PIXEL VALUE		
(BINARY)	(DECIMAL)	COLOR
0000	0	Black
0001	1	Dark Blue
0010	2	Red
0011	3	Magenta (dark red)
0100	4	Green
0101	5	Cyan (light blue)
0110	6	Yellow
0111	7	White
1xxx	8-15	Various grey scale

E.1 Calling the Tutorial Program

The program can be called up (1) in a batch along with the Debugger or (2) with a Debugger command. In either case, the Debugger software must be on the current disk drive.

E.1.1 Batch Call with Debugger

An " -f" parameter (space precedes the '-f') must be added to the Debugger call.

For an IBM-type PC:

```
SDB340  -f<CR>
```

The same operation for a TI PC:

```
SDB340T  -f<CR>
```

The Debugger will be called and, in turn, execute the Tutorial software. If the " -f" was left off, only the debugger would be called.

E.1.2 Call Tutorial From Debugger

If you are in the Debugger program, call the Tutorial with the Load command. To call the Debugger:

For an IBM-type PC:

```
SDB340<CR>
```

The same operation for a TI PC:

```
SDB340T<CR>
```

The TUTOR_E.OUT program must be on the current disk drive and the Debugger display is on the screen. Load the Tutorial with the following command:

```
Command[1] L TUTOR_E<CR>
```

E.2 The Annotated Tutorial Program

After loading the Tutorial (in Section E.1), execute it with:

```
Command[1] RU<CR>
```

The Tutorial will execute until the first software breakpoint is encountered. Continued pressing of the <CR> (RETURN key) will re-execute the RUN command, demonstrating instructions in the order previously done for Section 3 and shown in Table E-2.

Table E-2. Order and Location of Demonstrations in Tutorial Program

Order	PC Value	Demonstration	Page
1	>FFC0 0740	Pixel Transfer	E-7
2	>FFC0 07B0	Draw and Advance	E-12
3	>FFC0 0820	Fills	E-17

E.2.1 Run Standard Program

Execute the program in a standard sequential run by entering the RUN command:

```
Command[2] RU<CR>
```

The program will be executed in the order shown in Table E-2. At each software halt, step through the program by pressing the <CR> key.

E.2.2 Select Specific Demonstration

You can select any one of the routines listed in Table E-2. To avoid visual confusion, select a specific program only when the simulated graphics screen appears blank except for the screen borders.

To choose the routine, enter:

```
(1)  Command[2] PC FFC00xxx<CR>
```

followed by:

```
(2)  Command[2] RU<CR>
```

where "FFC00xxx" is the program counter value listed under "PC Value" in Table E-2.

E.3 Setting Up for Interaction Through the Keyboard

A lot of the exercises change register values prior to an instruction demonstration. To reduce unnecessary keystrokes, the ten command buffers (0-9) can be useful. For example, register B7 (Register dydx) could be modified over and over by placing a "change B7" command in command buffer 7. Then just three keystrokes can cause the register to be updated to the buffer 7 value ("7", <CR>, <CR>).

Note:

To store one or more commands in a buffer:

1) Enter buffer number (0-9) followed by <CR>.
2) Enter an exclamation mark (!) followed by the command (or string of commands separated by semicolons). The ! inhibits execution of the command while being written into the buffer.
3) End with a <CR>.

For example, to place RUN in buffer 0 without executing a RUN:

```
Command[ ] 0<CR>            Enter Command Buffer 0
Command[0] !RUN<CR>         RUN in Command Buffer 0
```

Now go to the next command buffer to enter the next command, etc. (You can also enter other buffers with the up/down cursor controls.)

With the command buffers set up, execute a command by entering:

1) <Command Buffer number><CR> *To go to the desired command buffer*

2) <CR> *To execute the entire command string in the buffer*

Figure E-1 shows how command buffers 5 and 6 can be used to blank the entire screen (Figure 3-12 on page 3-25 shows how to use buffers 5 and 6 to blank only the demonstration area).

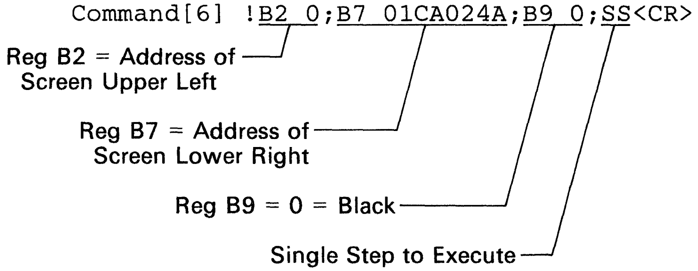
Note that this example uses the FILL XY demonstration in the Tutorial program; thus, the program must be present (TUTOR_E loaded).

Enter into two Command Buffers (for example purposes, 5 and 6 are used here):

```
Command[5]  !PC FFC00820;RU<CR>     PC of the FILL XY Demo

Command[5]  6<CR>                   Go to buffer 6

Command[6]  !B2 0;B7 01CA024A;B9 0;SS<CR>
```



Reg B2 = Address of Screen Upper Left

Reg B7 = Address of Screen Lower Right

Reg B9 = 0 = Black

Single Step to Execute

The following is a summary of the execution sequence:

STEP

```
(1) Command[x]  5<CR>                          Enter 1st Buffer

    Command[5]  PC FFC00820;RU                 1st Buffer Displayed

(2) Command[5]  <CR>                           Execute 1st Buffer

(3) Command[5]  6<CR>                          Enter 2d Buffer

    Command[6]  B2 0;B7 01CA024A;B9 0;SS       2d Buffer Displayed

(4) Command[6]  <CR>                           Execute 2d Buffer
```

This key sequence -- 5 <CR> <CR> 6 <CR> <CR> -- executes a series of instructions causing the screen to be blanked. Note that *other* command registers can be used instead of 5 and 6.

This example blanks the *entire* screen. Figure 3-12 on page 3-25 shows values to blank only the demonstration area in the screen upper left.

Figure E-1. Using Command Buffers 5 & 6 to Blank Entire Screen

E.4 Pixel Transfer (PIXT) PC = >FFC0 0740

E.4.1 PIXT Hands-On Setup

Recommended command-buffer values are listed in Table E-3.

Table E-3. PIXT Demo Suggested Command Buffer Values

CMND BUFFER	ENTER	COMMENT
0	PC FFC00740	PIXT PC start value
1	A1 600080	1st pixel XY address
2	A2 5	Light-blue value in A2
3	RUN	Execute to next trap
4	A4 60200	2d pixel linear address
7	A5 6000C0	3d pixel XY address
8	SS	Single step

NOTE: Buffers 5 and 6 can hold the display blank routine shown in Figure E-1.

HANDS-ON DEMONSTRATIONS

The difference between this section and the tutorial in Section 3 is that interspersed in this section are suggestions to change the machine state before executing instruction demonstrations. These suggestions are in italics and show aspects such as construction of various registers. You can first run through the regular tutorial demonstrations (not italicized), then run the italicized instructions which start on the next page.

In the italicized instructions, keystroke summaries in parentheses are to be used only when values in Table E-4 are preset.

E.4.2 PIXT Demonstration

Syntax: PIXT <source>,<destination>

Operation: A pixel value specified by the source operand is written to the location indicated by the destination operand. The instruction formats supported by the TMS34010 are:

PIXT	**Rs,*Rd**	Register to indirect linear
PIXT	**Rs,*Rd.xy**	Register to indirect xy
PIXT	***Rs,Rd**	Indirect linear to register

PIXT	***Rs,*Rd**	Indirect linear to indirect linear
PIXT	***Rs.xy,Rd**	Indirect xy to register
PIXT	***Rs.xy,*Rd.xy**	Indirect xy to indirect xy

When the destination is an indirect address of either type (linear or XY), a pixel processing option may be selected via the Control Register to perform an operation on the source pixel value before it is transferred. If the following are present:

- the transparency bit is set in the Control Register, and
- the result of the source pixel combined with the destination pixel is zero,

the destination pixel value will not be modified. The size of the pixel must be set in the PSIZE I/O Register and plane masking must be in effect as specified in the PMASK I/O Register. If either the source or destination are indirect xy mode, the appropriate conversion factor I/O Register must be loaded.

Demonstration Start: The PIXT demonstration begins at PC = >FFC0 0740

(a) *To first run the regular demonstration, use continuous <CR>s to write three yellow pixels to the screen (steps (1) through (5)). When the third pixel is written (before the DRAV demonstration begins), restart the PIXT demonstration and follow only the italicized passages. Do this setup by entering PC FFC00740 and RUN commands (execute buffers 0 and 3 in Table E-3. To use values you set per Table E-3, enter the following keystrokes:*

0 <CR> <CR> 3 <CR> <CR>

*Essentially, this executes the first RUN command immediately below, setting up for the PIXT A2,*A1.xy move. The next italicized message makes changes before writing this first pixel. You can use the regular text for reference, but follow only the italicized instructions.*

Now proceed to the next italicized passage.

(1) Enter: RU<CR> to begin the PIXT demonstration.

The mnemonic 'PIXT' is drawn in the upper left corner and the registers are set up for five demonstrations of the PIXT instruction. The first example of this instruction is a register-to-register indirect XY move: **PIXT A2,*A1,xy.**

The value of the pixel to be moved is >6 (color yellow as shown on page E-2 in Table E-1). It is contained in the four least significant bits (LSbs) of Register A2. This value is written to the XY address contained in Register A1 (>0040 0080), replacing the value which is stored there. Since the destination is in the XY mode, it is necessary to set the CONVDP I/O Register (conversion register, destination pitch) to the appropriate value (>0013 for the demonstration screen size) for conversion to the correct address. This setup writes one yellow pixel to the center of the demonstration screen.

The following re-writes the first pixel to the screen but at a lower set of coordinates.

(b) *Change Register A1 to write a new pixel located >20 pixels lower. Place >00600080 in Register A1 (>20 pixels below the previous >0040 Y-axis setting). The X-axis parameter stays at >0080 pixels. (Enter 1 <CR>, <CR> to enter these Table E-3 values.) Note that the Y axis is on the left:*

Register A1 =	*0060*	*0080*
	Y Axis	*X Axis*

(c) *To highlight the new position, change the pixel color to light blue (cyan) by writing "5" to Register A2. (Enter 2 <CR> <CR> to set Table E-3 value.)*

(d) *Enter a RUN command to write the pixel to the bottom center of the display. (Enter 3 <CR> <CR>.)*

Now continue at the next italicized session.

(2) Enter: RU<CR> to execute instruction **PIXT A2,*A1,xy.**

The pixel appears in the center of the demonstration area, and the registers are unchanged.

The instruction **PIXT *A1.xy,A3** employs an XY address stored in Register A1 to point to a pixel value in memory (on the screen) as the source. The CONVSP I/O Register (conversion factor, destination pitch) must be loaded with the appropriate value to convert the XY source address (a program task). The pixel value is then copied into the LSbs of the destination register, A3, with all MSbs set to zero.

(3) Enter: RU<CR> to execute instruction **PIXT *A1.xy,A3.**

The value of the yellow pixel (>6) drawn in the first example is copied into Register A3, replacing >FFFF FFFF with >0000 0006. No other register values change.

The third example of PIXT demonstrates a move from a register to a linear address. Since the move does not use the XY addressing mode, it is not necessary to set either the CONVSP or CONVDP I/O Registers.

(e) *Set up to write the second pixel. First lower the Y axis of the pixel by entering the command-buffer 4 values to change Register A4 from from >40200 to >60200 (affecting the Y axis only). (Enter 4 <CR> <CR>.) NOTE: If you wish, enter your own calculated values.*

(f) *Enter a RUN. (Enter 3 <CR> <CR> <CR>., using Table E-3 buffer values). A second pixel is written in the display bottom left. Go to the next italicized passage.*

(4) Enter: RU<CR> to execute instruction **PIXT A3,*A4**.

The pixel value stored in Register A3 is moved to the linear address stored in A4 (>4 0200). A yellow pixel is drawn to the left of the first pixel.

The fourth PIXT example demonstrates transferring pixels from one XY screen location to another. With both source and destination being XY indirect, both CONVSP and CONVDP I/O Registers must be set up.

(g) *Set up to write the third pixel. First, lower the pixel's Y axis destination by executing command buffer 5 (7, <CR>, <CR>). This changes A5 from >40 00C0 to >60 00C0).*

(h) *Execute RUN (3, <CR> <CR>) to write the third pixel.*

(5) Enter: RU<CR> to execute instruction **PIXT *A1.xy.*A3.xy.**

The pixel value at the XY address in A1 (>0040 0080) is copied to the location at the XY address in A5 (>0040 00C0). The center yellow pixel is copied to the right.

This completes the demonstration of the PIXT pixel transfer instruction.

(i) *For the next instruction, DRAV, set up the command buffers as shown in Table E-4. Then go to the italicized passages that follow.*

E.5 Draw and Advance (DRAV) PC = >FFC0 07B0

E.5.1 DRAV Hands-On Setup

Table E-4. DRAV Demo Suggested Command Buffer Values

CMND BUFFER	ENTER	COMMENT
0	PC FFC007B0	DRAV PC start value
1	A1 1	Increment X axis only
2	A2 900040	Start below 1st rectangle
3	RUN	Execute to next trap
4	B4 120	Change Offset Register
7	A0 %5000	Extend number of loops to 5000
8	SS	Single step
9	B9 5555	Light blue

The display-blank routine can remain in command buffers 5 and 6.

In the italicized instructions, keystroke summaries are shown in parentheses to be used when values in Table E-4 are preset.

E.5.2 DRAV Demonstration

Syntax: DRAV <Rs(source)>,<Rd(destination)>

Operation: A pixel of COLOR1 Register color is written to the XY location stored in Rd. Immediately following, the value in Rd is incremented by the value in Rs. **NOTE: Rs and Rd must both be in the same register file (either A or B).**

(1) Enter: RU<CR> This writes the mnemonic **DRAV** inside the demonstration box, and the appropriate operand registers are set up for the draw and advance. The display appears as shown in Figure E-2 (........ = don't care):

(a) *Enter successive <CR>s to draw the three lines in the regular demonstration: yellow vertical, red diagonal, and green horizontal. After this, set up to redraw the lines with different parameters.*

(b) *Start again at the beginning of this demonstration by entering the PC start value (PC value and two <CR>s, or using Table E-4 values, enter 0 <CR> <CR>).*

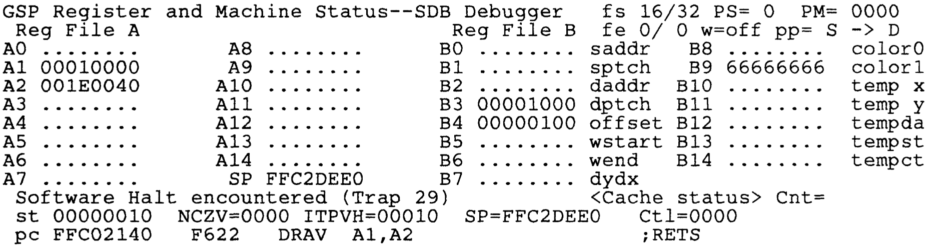
(c) *Enter a RUN (3 <CR> <CR> or enter <CR> until Register A2 contains 001E0040.*

(d) *Lower the Y axis value to a point below the display area by entering >900040 in A2 (2 <CR> <CR>)*

(e) *Line color can be changed by entering a different value in B9 (buffer 9); single hexadecimal values can be used (9 <CR> <CR>).*

(f) *Enter a RUN to write one pixel below the display area, now ready to begin a single step draw of the vertical line (3 <CR> <CR>).*

Go to the next italicized entries.



```
GSP Register and Machine Status--SDB Debugger   fs 16/32 PS= 0  PM= 0000
 Reg File A                            Reg File B  fe 0/ 0 w=off pp= S -> D
A0 ........         A8 ........      B0 ........ saddr   B8 ........  color0
A1 00010000         A9 ........      B1 ........ sptch   B9 66666666  color1
A2 001E0040        A10 ........      B2 ........ daddr  B10 ........  temp x
A3 ........        A11 ........      B3 00001000 dptch  B11 ........  temp y
A4 ........        A12 ........      B4 00000100 offset B12 ........  tempda
A5 ........        A13 ........      B5 ........ wstart B13 ........  tempst
A6 ........        A14 ........      B6 ........ wend   B14 ........  tempct
A7 ........         SP FFC2DEE0      B7 ........ dydx
 Software Halt encountered (Trap 29)             <Cache status> Cnt=
 st 00000010  NCZV=0000 ITPVH=00010  SP=FFC2DEE0   Ctl=0000
 pc FFC02140   F622   DRAV  A1,A2                  ;RETS
```

Figure E-2. DRAV Screen Display

As shown in the display:

- The instruction **DRAV A1,A2** now appears in the current instruction field of the RMS display.
- Register A2 contains the destination address in XY mode (>001E 0040: Y=001E, X=0040) which is the location to which the pixel will be moved.
- Register B9 is loaded with the >66666666, specifying the color yellow (see table on page E-2).

When the instruction is executed, a yellow pixel is drawn 64 (>0040) pixels to the right and 30 (>001E) pixels below the origin of the demonstration screen (upper left corner). Then the address value in Register A2 is incremented by the value of Register A1 (>0001 0000: Y=1, X=0). To demonstrate this:

(2) Enter: RU<CR> The DRAV instruction is executed and a software trap follows immediately. One yellow pixel is drawn in the display block.

(g) *With one pixel drawn to the screen below the display area, you can now watch a vertical line slowly drawn by single stepping (SS command in buffer 8 followed by continuous <CRs>) through the loop shown in the reverse assembly code that follows.*

(h) *Line color can be changed by entering a different value in B9 (buffer 9); single hexadecimal values can be used.*

(i) *Single step routines stop when A0 reaches zero. To increase A0, enter a higher value using buffer 7 (7 <CR> <CR>).*

(j) *To draw to the right, change the Y axis increment in A1 to zero and set a positive value on the X axis (right* *four hex values). A negative value causes a line to the left. Control the X and Y advance with the Register A1 increment that is added to the XY destination in A2 (buffer 1).*

When ready, you can quit this session and go to the next italicized entry by placing a loop counter value of one (1) in Register A0 and entering RUN.

Notice that a pixel has been drawn, and Register A2 is incremented (by >10000). By placing this instruction inside a loop, a line of pixels can be drawn with an X address constant and a Y address repeatedly incremented by one. To see such a loop:

(3) Enter: U<CR> to reverse-assemble the program (shown on the left of the screen); as follows:

```
          Op-
Lnr Addr  code    Rev Assembly           Comment (not assembled)

xxxxxxxx  xxxx  UNKNOWN
xxxxxxxx  xxxx  UNKNOWN
FFC02160  09C0  MOVI    >004B,A0         Load loop count register
FFC02180  F622  DRAV    A1,A2            Draw and advance one pixel
FFC02190  3C40  DSJS    A0,@FFC02180     Dec Reg, jmp to DRAV if ≠ 0
FFC021A0  091D  TRAP    29               Halt after loop
FFC021B0  09C0  MOVI    >42,A0
```

The reverse-assembled portion contains a loop.

- The yelllow-colored line identifies the instruction just executed ("UNKNOWN" in the example).
- The cyan (light blue) color identifes the instruction *before* the one just executed -- also "UNKNOWN" in the example.
- Green identifies the *next* instruction to be executed. It loads Register A0 with the loop count of >4B (75).
- The next three instructions make a loop to draw a vertical line. (The TRAP 29 is a software breakpoint.)
- A0 is decremented. If not zero, a jump to DRAV occurs to complete the loop and execute another draw/advance.

This loop will execute 75 times before the jump is discontinued -- each time drawing another pixel on the screen while incrementing the address in A2 one time in the Y direction. The final result is a vertical line 76 pixels in length.

(4) Enter: <Q> to quit the reverse assembly.

(5) Enter: RU<CR> to execute the loop and draw a yellow line (6s in Register COLOR1) on the screen.

After being incremented by one 75 times, the value in destination Register A2 is now >006A 0040. Note that loop counter A0 has been decremented to zero.

Two more examples show some of the flexibility of this instruction. The first employs a bidirectional increment to create a diagonal line.

(6) Enter: RU<CR> to set up the operand registers for a diagonal draw and advance.

- **(k)** *Set up for drawing a diagonal line by entering the same XY start coordinates (A2) used for the vertical line (enter value stored in buffer 2 (2 <CR> <CR>).*
- **(l)** *By single stepping (buffer 7), a red diagonal line is slowly drawn.*
- **(m)** *Color can be changed in Register B9 (buffer 9).*
- **(n)** *Notice the Y and X increments in Register A1 (Y in 16 bits on left, X on right) to set the angle. These can be varied to change the angle of the line (Y value: positive = down, negative = up; X value: positive = left, negative = right; combinations cause diagonals).*
- **(o)** *With the line going horizontal, vary the values in Register COLOR1 (buffer 9) to 4444, 444, 44, and 4, noting changes in the line. You can also mix colors (2424, 2224, etc.).*
- **(p)** *Register A2 contains the destination point of the pixel (buffer 2). Changing A2 relocates the pixel to another part of the screen.*

  Entering a loop count of one (1) in Register A0 followed by a RUN sends the program to the next italicized entries at the end of step (10).

The destination register is loaded with the same initial value as in the first example -- steps (1) to (5) above. The incrementing register (A1) contains >0001 0002 (Y=1, X=2), and the loop count in A0 has been set to >42 (66). The COLOR1 Register is now >2222 2222, specifying red.

(7) Enter: U<CR> to display the reverse-assembled program.

The loop set up is similar to the first program (step (3)) with the destination address being incremented in both the X and Y directions.

(8) Enter: Q to quit the reverse assembly.

(9) Enter: RU<CR> to draw a diagonal line from the same starting point as in the first example.

Note the destination address is incremented by >42 in the Y direction and by >84 in the X direction. The final A2 value is >0060 00C4.

The final demonstration produces a dotted green horizontal line.

(10) Enter: RU<CR> to set up the register operands.

An identical loop to that above is used in this example.

- Destination Register A2 is the same as used previously.
- Increment value of >0000 0006 is in A1.
- COLOR1 Register contains >4444 4444 (green).

After every pixel is drawn, the X address is incremented by 6, leaving five blank pixels between each green pixel.

(q) *Instead of a <CR> (in 11 below), single step (SS in buffer 7). Because of the X increment value of 6 in Register A1, six pixels are skipped (appear black) between one pixel of the color specified in the COLOR1 Register (buffer 9). You can vary the increment value (buffer 1).*

(r) *You can also change the color characteristics of the line by varying Register COLOR1 (buffer 9).*

Enter a RUN command (buffer 3) to go to the **FILL** *demonstration.*

(11) Enter: RU<CR> to execute the loop and draw the line.

These are simple examples of the 'draw and advance' employing constant increments. More elaborate schemes of altering the increment can be used to implement various graphical algorithms for figure drawing.

E.6 Fill Array Instructions (FILL XY, FILL L) PC = >FFC0 0820

These instructions perform a pixel processing operation on a memory array using the value in COLOR1 Register as the source pixel value. The destination is defined in either XY or linear addressing mode, depending on which instruction is used.

E.6.1 Fill Array Hands-On Setup

Table E-5. FILL-Demo Command Buffer Suggested Values

CMND BUFFER	ENTER	COMMENT
0	PC FFC00820	FILL PC start value
2	B2 00290040	Lower the upper-left corner (DADDR)
3	RUN	Execute to next trap
4	SS	Single Step
7	B7 00280050	2x height, ½x width dimension in DYDX
9	B9 2255	Red/cyan combination

The screen-clear routine can remain in command buffers 5 and 6.

In the italicized instructions, keystroke summaries are shown in parentheses to be used only when values in Table E-4 are preset.

E.6.2 Fill Array Demonstration

Syntax: FILL XY

Operation: A pixel processing operation is performed between the pixel value stored in the COLOR1 Register and an XY array of memory.

- The XY address in Register DADDR (B2) contains the location of the array's least-significant corner (screen upper left).
- Registers DPTCH, OFFSET, AND CONVDP (I/O) must contain values appropriate to the screen-memory format.
- Register DYDX value of >000A 00A0 specifies dimensions of the destination array with the 16 MSbs indicating heighth and the 16 LSbs indicating width (both in pixels)
- The CONTROL I/O Register specifies the pixel processing option.

(a) *It is recommended to first run through the complete FILL demonstration; then re-run it following the italicized hands-on suggestions. Return to FILL beginning by entering the PC value and a <RUN> command (0 <CR> <CR> 3 <CR> <CR>).*

Go to the next italicized instructions.

(1) Enter: RU<CR> to write the mnemonic **FILL XY** onto the screen and set up the appropriate operand registers to fill a rectangle on the screen.

The screen appears as follows (Figure E-3):

```
GSP Register and Machine Status--SDB Debugger    fs 16/32 PS= 4  PM= 0000
 Reg File A                         Reg File B   fe 0/ 0 w=off pp= S -> D
A0 ........        A8 ........      B0 ........ saddr   B8 ........  color0
A1 00000000        A9 ........      B1 ........ sptch   B9 22222222  color1
A2 00000000       A10 ........      B2 00180040 daddr  B10 ........  temp x
A3 ........       A11 ........      B3 00001000 dptch  B11 ........  temp y
A4 ........       A12 ........      B4 00000100 offset B12 ........  tempda
A5 ........       A13 ........      B5 ........ wstart B13 ........  tempst
A6 ........       A14 ........      B6 ........ wend   B14 ........  tempct
A7 ........        SP FFC2DEE0      B7 000A00A0 dydx
 Halt on breakpoint. See below.                 <Cache status> Cnt=   484
 st 00000010  NCZV=0000 ITPVH=00000  SP=FFC2DEE0   Ctl=0000
 pc FFC02630   0FE0   FILL  XY                   ;RETS
```

Figure E-3. Register Display for Fill Screen, XY Addressing

The instruction **FILL XY** appears in the instruction field of the display, and the necessary registers are loaded to draw a red rectangle to the screen.

- Register DADDR (B2) is loaded to place the upper-left corner of the rectangle at the location >18 pixels below and >40 pixels to the right of screen origin.
- Register DYDX (B7) specifies rectangle height of >000A (10) pixels and width of >00A0 (160) pixels.
- Register COLOR1 (B9) specifies red (>2222 2222).
- Registers DPTCH (B3) and CONVDP (I/O Register -- display with DR command) are loaded with values appropriate for the screen used.

(2) Enter: RU<CR> to draw a red rectangle onto the screen.

Note that the destination address register has become corrupted.

The following make changes before re-executing the first **FILL** *instruction to draw a rectangle in the vertical axis.*

(b) *Change the XY address in B2 to lower the starting point of the rectangle (2 <CR> <CR>) from >0018 to >0029 in the Y axis. The X axis remains the same.*

(c) *Change the COLOR1 Register to >2255 for a striped effect with two colors -- red and cyan (9 <CR> <CR>).*

(d) *Execute single step to draw (4 <CR> <CR>).*

(e) *Return to FILL beginning by entering the PC value and a <RUN> command (0 <CR> <CR> 3 <CR> <CR>).*

(f) *Change the upper left corner start address in buffer 2 to an arbitrary value. (E.g., you could double both the X and Y pixel addresses.)*

(g) *Change the rectangle length (Y axis or 16 MSbs of B7/DYDX) and width (X axis or 16 LSbs of B7/DYDX). E.g., B7 = >00280050 will double the height and halve the width (7 <CR> <CR>).*

(h) *Change the COLOR1 Register (buffer 9) to a different value.*

(i) *Execute a single step (SS in buffer 4 or 4 <CR> <CR>).*

This concludes the examples using the tutorial software for experimentation. Please feel free to proceed and run through these exercises again, using the values given or checking out your own.

Index

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

TI Worldwide Sales Offices

ALABAMA: Huntsville: 500 Wynn Drive, Suite 514, Huntsville, AL 35805, (205) 837-7530.

ARIZONA: Phoenix: 8825 N. 23rd Ave., Phoenix, AZ 85021, (602) 995-1007.

CALIFORNIA: Irvine: 17891 Cartwright Rd., Irvine, CA 92714, (714) 660-8187; **Sacramento:** 1900 Point West Way, Suite 171, Sacramento, CA 95815, (916) 929-1521; **San Diego:** 4333 View Ridge Ave., Suite B., San Diego, CA 92123, (619) 278-9601; **Santa Clara:** 5353 Betsy Ross Dr., Santa Clara, CA 95054, (408) 980-9000; **Torrance:** 690 Knox St., Torrance, CA 90502, (213) 217-7010; **Woodland Hills:** 21220 Erwin St., Woodland Hills, CA 91367, (818) 704-7759.

COLORADO: Aurora: 1400 S. Potomac Ave., Suite 101, Aurora, CO 80012, (303) 368-8000.

CONNECTICUT: Wallingford: 9 Barnes Industrial Park Rd., Barnes Industrial Park, Wallingford, CT 06492, (203) 269-0074.

FLORIDA: Ft. Lauderdale: 2765 N.W. 62nd St., Ft. Lauderdale, FL 33309, (305) 973-8502; **Maitland:** 2601 Maitland Center Parkway, Maitland, FL 32751, (305) 660-4600; **Tampa:** 5010 W. Kennedy Blvd., Suite 101, Tampa, FL 33609, (813) 870-6420.

GEORGIA: Norcross: 5515 Spalding Drive, Norcross, GA 30092, (404) 662-7900

ILLINOIS: Arlington Heights: 515 W. Algonquin, Arlington Heights, IL 60005, (312) 640-2925.

INDIANA: Ft. Wayne: 2020 Inwood Dr., Ft. Wayne, IN 46815, (219) 424-5174; **Indianapolis:** 2346 S. Lynhurst, Suite J-400, Indianapolis, IN 46241, (317) 248-8555.

IOWA: Cedar Rapids: 373 Collins Rd. NE, Suite 200, Cedar Rapids, IA 52402, (319) 395-9550.

MARYLAND: Baltimore: 1 Rutherford Pl., 7133 Rutherford Rd., Baltimore, MD 21207, (301) 944-8600.

MASSACHUSETTS: Waltham: 504 Totten Pond Rd., Waltham, MA 02154, (617) 895-9100.

MICHIGAN: Farmington Hills: 33737 W. 12 Mile Rd., Farmington Hills, MI 48018, (313) 553-1500.

MINNESOTA: Eden Prairie: 11000 W. 78th St., Eden Prairie, MN 55344 (612) 828-9300.

MISSOURI: Kansas City: 8080 Ward Pkwy., Kansas City, MO 64114, (816) 523-2500; **St. Louis:** 11816 Borman Drive, St. Louis, MO 63146, (314) 569-7600.

NEW JERSEY: Iselin: 485E U.S. Route 1 South, Parkway Towers, Iselin, NJ 08830 (201) 750-1050

NEW MEXICO: Albuquerque: 2820-D Broadbent Pkwy NE, Albuquerque, NM 87107, (505) 345-2555.

NEW YORK: East Syracuse: 6365 Collamer Dr., East Syracuse, NY 13057, (315) 463-9291; **Endicott:** 112 Nanticoke Ave., P.O. Box 618, Endicott, NY 13760, (607) 754-3900; **Melville:** 1 Huntington Quadrangle, Suite 3C10, P.O. Box 2936, Melville, NY 11747, (516) 454-6600; **Pittsford:** 2851 Clover St., Pittsford, NY 14534, (716) 385-6770; **Poughkeepsie:** 385 South Rd., Poughkeepsie, NY 12601, (914) 473-2900.

NORTH CAROLINA: Charlotte: 8 Woodlawn Green, Woodlawn Rd., Charlotte, NC 28210, (704) 527-0930; **Raleigh:** 2809 Highwoods Blvd., Suite 100, Raleigh, NC 27625, (919) 876-2725.

OHIO: Beachwood: 23408 Commerce Park Rd., Beachwood, OH 44122, (216) 464-6100; **Dayton:** Kingsley Bldg., 4124 Linden Ave., Dayton, OH 45432, (513) 258-3877.

OREGON: Beaverton: 6700 SW 105th St., Suite 110, Beaverton, OR 97005, (503) 643-6758.

PENNSYLVANIA: Ft. Washington: 260 New York Dr., Ft. Washington, PA 19034, (215) 643-6450; **Coraopolis:** 420 Rouser Rd., 3 Airport Office Park, Coraopolis, PA 15108, (412) 771-8550.

PUERTO RICO: Hato Rey: Mercantil Plaza Bldg., Suite 505, Hato Rey, PR 00919, (809) 753-8700.

TEXAS: Austin: P.O. Box 2909, Austin, TX 78769, (512) 250-7655; **Richardson:** 1001 E. Campbell Rd., Richardson, TX 75080, (214) 680-5082; **Houston:** 9100 Southwest Frwy., Suite 237, Houston, TX 77036, (713) 778-6592; **San Antonio:** 1000 Central Parkway South, San Antonio, TX 78232, (512) 496-1779.

UTAH: Murray: 5201 South Green SE, Suite 200, Murray, UT 84107, (801) 266-8972.

VIRGINIA: Fairfax: 2750 Prosperity, Fairfax, VA 22031, (703) 849-1400.

WASHINGTON: Redmond: 5010 148th NE, Bldg B, Suite 107, Redmond, WA 98052, (206) 881-3080.

WISCONSIN: Brookfield: 450 N. Sunny Slope, Suite 150, Brookfield, WI 53005, (414) 785-7140.

CANADA: Nepean: 301 Moodie Drive, Mallorn Center, Nepean, Ontario, Canada, K2H9C4, (613) 726-1970. **Richmond Hill:** 280 Centre St. E., Richmond Hill L4C1B1, Ontario, Canada (416) 884-9181; **St. Laurent:** Ville St. Laurent Quebec, 9460 Trans Canada Hwy., St. Laurent, Quebec, Canada H4S1R7, (514) 335-8392.

ARGENTINA: Texas Instruments Argentina S.A.I.C.F.: Esmeralda 130, 15th Floor, 1035 Buenos Aires, Argentina, 1 + 394-3008.

AUSTRALIA (& NEW ZEALAND): Texas Instruments Australia Ltd.: 6-10 Talavera Rd., North Ryde (Sydney), New South Wales, Australia 2113, 2 + 887-1122; 5th Floor, 418 St. Kilda Road, Melbourne, Victoria, Australia 3004, 3 + 267-4677; 171 Philip Highway, Elizabeth, South Australia 5112, 8 + 255-2066.

AUSTRIA: Texas Instruments Ges.m.b.H.: Industriestrabe B/16, A-2345 Brunn/Gebirge, 2236-846210.

BELGIUM: Texas Instruments N.V. Belgium S.A.: Mercure Centre, Raketstraat 100, Rue de la Fusee, 1130 Brussels, Belgium, 2/720.80.00.

BRAZIL: Texas Instruments Electronicos do Brasil Ltda.: Rua Paes Leme, 524-7 Andar Pinheiros, 05424 Sao Paulo, Brazil, 0815-6166.

DENMARK: Texas Instruments A/S, Mairelundvej 46E, DK-2730 Herlev, Denmark, 2 - 91 74 00.

FINLAND: Texas Instruments Finland OY: Teollisuuskatu 19D 00511 Helsinki 51, Finland, (90) 701-3133.

FRANCE: Texas Instruments France: Headquarters and Prod. Plant, BP 05, 06270 Villeneuve-Loubet, (93) 20-01-01; Paris Office, BP 67 8-10 Avenue Morane-Saulnier, 78141 Velizy-Villacoublay, (3) 946-97-12; Lyon Sales Office, L'Oree D'Ecully, Batiment B, Chemin de la Forestiere, 69130 Ecully, (7) 833-04-40; Strasbourg Sales Office, Le Sebastopol 3, Quai Kleber, 67055 Strasbourg Cedex, (88) 22-12-66; Rennes, 23-25 Rue du Puits Mauger, 35100 Rennes, (99) 31-54-86; Toulouse Sales Office, Le Peripole—2, Chemin du Pigeonnier de la Cepiere, 31100 Toulouse, (61) 44-18-19; Marseille Sales Office, Noilly Paradis—146 Rue Paradis, 13006 Marseille, (91) 37-25-30.

TEXAS INSTRUMENTS

GERMANY (Fed. Republic of Germany): Texas Instruments Deutschland GmbH: Haggertystrasse 1, D-8050 Freising, 8161 + 80-4591; Kurfuerstendamm 195/196, D-1000 Berlin 15, 30 + 882-7365; III, Hagen 43/Kibbelstrasse, .19, D-4300 Essen, 201-24250; Frankfurter Allee 6-8, D-6236 Eschborm 1, 06196 + 8070; Hamburgerstrasse 11, D-2000 Hamburg 76, 040 + 220-1154, Kirchhorsterstrasse 2, D-3000 Hannover 51, 511 + 648021; Maybachstrabe 11, D-7302 Ostfildern 2-Nelingen, 711 + 547001; Mixikoring 19, D-2000 Hamburg 60, 40 + 637 + 0061; Postfach 1309, Roonstrasse 16, D-5400 Koblenz, 261 + 35044.

HONG KONG (+ PEOPLES REPUBLIC OF CHINA): Texas Instruments Asia Ltd., 8th Floor, World Shipping Ctr., Harbour City, 7 Canton Rd., Kowloon, Hong Kong, 3 + 722-1223.

IRELAND: Texas Instruments (Ireland) Limited· Brewery Rd., Stillorgan, County Dublin, Eire, 1 831311.

ITALY: Texas Instruments Semiconduttori Italia Spa: Viale Delle Scienze, 1, 02015 Cittaducale (Rieti), Italy, 746 694.1; Via Salaria KM 24 (Palazzo Cosma), Monterotondo Scalo (Rome), Italy, 6 + 9003241; Viale Europa, 38-44, 20093 Cologno Monzese (Milano), 2 2532541; Corso Svizzera, 185, 10100 Torino, Italy, 11 774545; Via J. Barozzi 6, 40100 Bologna, Italy, 51 355851.

JAPAN: Texas Instruments Asia Ltd.: 4F Aoyama Fuji Bldg., 6-12, Kita Aoyama 3-Chome, Minato-ku, Tokyo, Japan 107, 3-498-2111; Osaka Branch, 5F, Nissho Iwai Bldg., 30 Imabashi 3- Chome, Higashi-ku, Osaka, Japan 541, 06-204-1881; Nagoya Branch, 7F Daini Toyota West Bldg., 10-27, Meieki 4-Chome, Nakamura-ku Nagoya, Japan 450, 52-583-8691.

KOREA: Texas Instruments Supply Co.: 3rd Floor, Samon Bldg., Yuksam-Dong, Gangnam-ku, 135 Seoul, Korea, 2 + 462-8001.

MEXICO: Texas Instruments de Mexico S.A.: Mexico City, AV Reforma No. 450 — 10th Floor, Mexico, D.F., 06600, 5 + 514-3003.

MIDDLE EAST: Texas Instruments: No. 13, 1st Floor Mannai Bldg., Diplomatic Area, P.O. Box 26335, Manama Bahrain, Arabian Gulf, 973 + 274681.

NETHERLANDS: Texas Instruments Holland B.V., P.O. Box 12995, (Bullewijk) 1100 CB Amsterdam, Zuid-Oost, Holland 20 + 5602911.

NORWAY: Texas Instruments Norway A/S: PB106, Refstad 131, Oslo 1, Norway, (2) 155090.

PHILIPPINES: Texas Instruments Asia Ltd.: 14th Floor, Ba- Lepanto Bldg., 8747 Paseo de Roxas, Makati, Metro Manila, Philippines, 2 + 8188987.

PORTUGAL: Texas Instruments Equipamento Electronico (Portugal), Lda.: Rua Eng. Frederico Ulrich, 2650 Moreira Da Maia, 4470 Maia, Portugal, 2-948-1003.

SINGAPORE (+ INDIA, INDONESIA, MALAYSIA, THAILAND): Texas Instruments Asia Ltd.: 12 Lorong Bakar Batu, Unit 01-02, Kolam Ayer Industrial Estate, Republic of Singapore, 747-2255.

SPAIN: Texas Instruments Espana, S.A.: C/Jose Lazaro Galdiano No. 6, Madrid 16, 1/458.14.58.

SWEDEN: Texas Instruments International Trade Corporation (Sverigefilialen): Box 39103, 10054 Stockholm, Sweden, 8 - 235480.

SWITZERLAND: Texas Instruments, Inc., Reidstrasse 6, CH-8953 Dietikon (Zuerich) Switzerland, 1-740 2220.

TAIWAN: Texas Instruments Supply Co.: Room 903, 205 Tun Hwan Rd., 71 Sung-Kiang Road, Taipei, Taiwan, Republic of China, 2 + 521-9321.

UNITED KINGDOM: Texas Instruments Limited: Manton Lane, Bedford, MK41 7PA, England, 0234 67466; St. James House, Wellington Road North, Stockport, SK4 2RT, England, 61 + 442-7162.

BM

TI Sales Offices

ALABAMA: Huntsville (205) 837-7530.

ARIZONA: Phoenix (602) 995-1007;
Tucson (602) 624-3276.

CALIFORNIA: Irvine (714) 660-8187;
Sacramento (916) 929-0192;
San Diego (619) 278-9601;
Santa Clara (408) 980-9000;
Torrance (213) 217-7010;
Woodland Hills (818) 704-7759.

COLORADO: Aurora (303) 368-8000.

CONNECTICUT: Wallingford (203) 269-0074.

FLORIDA: Ft. Lauderdale (305) 973-8502;
Altamonte Springs (305) 260-2116;
Tampa (813) 870-6420.

GEORGIA: Norcross (404) 662-7900.

ILLINOIS: Arlington Heights (312) 640-2925.

INDIANA: Ft. Wayne (219) 424-5174;
Indianapolis (317) 248-8555.

IOWA: Cedar Rapids (319) 395-9550.

MARYLAND: Baltimore (301) 944-8600.

MASSACHUSETTS: Waltham (617) 895-9100.

MICHIGAN: Farmington Hills (313) 553-1500;
Grand Rapids (616) 957-4200.

MINNESOTA: Eden Prairie (612) 828-9300.

MISSOURI: Kansas City (816) 523-2500;
St. Louis (314) 569-7600.

NEW JERSEY: Iselin (201) 750-1050.

NEW MEXICO: Albuquerque (505) 345-2555.

NEW YORK: East Syracuse (315) 463-9291;
Melville (516) 454-6600; **Pittsford** (716) 385-6770;
Poughkeepsie (914) 473-2900.

NORTH CAROLINA: Charlotte (704) 527-0930;
Raleigh (919) 876-2725.

OHIO: Beachwood (216) 464-6100;
Dayton (513) 258-3877.

OREGON: Beaverton (503) 643-6758.

PENNSYLVANIA: Blue Bell (215) 825-9500.

PUERTO RICO: Hato Rey (809) 753-8700

TEXAS: Austin (512) 250-7655;
Houston (713) 778-6592; **Richardson** (214) 680-5082;
San Antonio (512) 496-1779.

UTAH: Murray (801) 266-8972.

VIRGINIA: Fairfax (703) 849-1400.

WASHINGTON: Redmond (206) 881-3080.

WISCONSIN: Brookfield (414) 785-7140.

CANADA: Nepean, Ontario (613) 726-1970;
Richmond Hill, Ontario (416) 884-9181;
St. Laurent, Quebec (514) 335-8392.

TI Regional Technology Centers

CALIFORNIA: Irvine (714) 660-8140,
Santa Clara (408) 748-2220.

GEORGIA: Norcross (404) 662-7945.

ILLINOIS: Arlington Heights (312) 640-2909.

MASSACHUSETTS: Waltham (617) 895-9197.

TEXAS: Richardson (214) 680-5066.

CANADA: Nepean, Ontario (613) 726-1970

Customer Response Center

TOLL FREE: (800) 232-3200

OUTSIDE USA: (214) 995-6611
(8:00 a.m. — 5:00 p.m. CST)

TI Distributors

TI AUTHORIZED DISTRIBUTORS IN USA

Arrow Electronics
General Radio Supply Company
Graham Electronics
Hall-Mark Electronics
Kierulff Electronics
Marshall Industries
Newark Electronics
Schweber Electronics
Time Electronics
Wyle Laboratories
Zeus Component, Inc. (Military Only)

TI AUTHORIZED DISTRIBUTORS IN CANADA

Arrow Electronics Canada
Future Electronics

TI AUTHORIZED DISTRIBUTORS IN USA

—OBSOLETE PRODUCT ONLY—
Rochester Electronics, Inc.
Newburyport, Massachusetts
(617) 462-9332

ALABAMA: Arrow (205) 837-6955;
Hall-Mark (205) 837-8700; Kierulff (205) 883-6070;
Marshall (205) 881-9235; Schweber (205) 895-0480.

ARIZONA: Arrow (602) 968-4800;
Hall-Mark (602) 437-1200; Kierulff (602) 437-0750;
Marshall (602) 968-6181; Schweber (602) 997-4874;
Wyle (602) 866-2888.

CALIFORNIA: Los Angeles/Orange County:
Arrow (818) 701-7500, (714) 838-5422;
Hall-Mark (818) 716-7300, (714) 669-4700,
(213) 217-8400; Kierulff (213) 725-0325, (714) 731-5711,
(714) 220-6300; (818) 407-2500;
Marshall (818) 407-0101, (818) 459-5500,
(714) 458-5395; Schweber (818) 999-4702;
(714) 863-0200; (213) 327-8409; Wyle (213) 322-8100,
(818) 880-9001, (714) 863-9953; Zeus (714) 921-9000;
Sacramento: Hall-Mark (916) 722-8600;
Marshall (916) 635-9700; Schweber (916) 929-9732;
Wyle (916) 638-5282;
San Diego: Arrow (619) 565-4800;
Hall-Mark (619) 268-1201; Kierulff (619) 278-2112;
Marshall (619) 578-9600; Schweber (619) 450-0454;
Wyle (619) 565-9171;
San Francisco Bay Area: Arrow (408) 745-6600;
(415) 487-4600; Hall-Mark (408) 946-0900;
Kierulff (408) 971-2600; Marshall (408) 942-4600;
Schweber (408) 946-7171; Wyle (408) 727-2500;
Zeus (408) 998-5121.

COLORADO: Arrow (303) 696-1111;
Hall-Mark (303) 790-1662; Kierulff (303) 790-4444;
Marshall (303) 451-8444; Schweber (303) 799-0258;
Wyle (303) 457-9953.

CONNECTICUT: Arrow (203) 265-7741;
Hall-Mark (203) 269-0100; Kierulff (203) 265-1115;
Marshall (203) 265-3822; Schweber (203) 748-7080.

FLORIDA: Ft. Lauderdale: Arrow (305) 429-8200;
Hall-Mark (305) 971-9280; Kierulff (305) 486-4004;
Marshall (305) 977-4880; Schweber (305) 977-7511;
Orlando: Arrow (305) 725-1480;
Hall-Mark (305) 855-4020; Kierulff (305) 682-6923;
Marshall (305) 841-1878; Schweber (305) 331-7555;
Zeus (305) 365-3000;
Tampa: Hall-Mark (813) 530-4543;
Marshall (813) 576-1399.

GEORGIA: Arrow (404) 449-8252;
Hall-Mark (404) 447-8000; Kierulff (404) 447-5252;
Marshall (404) 923-5750; Schweber (404) 449-9170.

ILLINOIS: Arrow (312) 397-3440;
Hall-Mark (312) 860-3800; Kierulff (312) 250-0500;
Marshall (312) 490-0155; Newark (312) 784-5100;
Schweber (312) 364-3750.

TEXAS
INSTRUMENTS

INDIANA: Indianapolis: Arrow (317) 243-9353;
Graham (317) 634-8202; Hall-Mark (317) 872-8875;
Marshall (317) 297-0483;
Ft. Wayne: Graham (219) 423-3422.

IOWA: Arrow (319) 395-7230;
Schweber (319) 373-1417.

KANSAS: Kansas City: Arrow (913) 541-9542;
Hall-Mark (913) 888-4747; Marshall (913) 492-3121;
Schweber (913) 492-2921.

MARYLAND: Arrow (301) 995-0003;
Hall-Mark (301) 988-9800; Kierulff (301) 840-1155;
Marshall (301) 840-9450; Schweber (301) 840-5900;
Zeus (301) 997-1118.

MASSACHUSETTS: Arrow (617) 933-8130;
Hall-Mark (617) 667-0902; Kierulff (617) 667-8331;
Marshall (617) 658-0810; Schweber (617) 275-5100,
(617) 657-0760; Time (617) 532-6200;
Zeus (617) 863-8800.

MICHIGAN: Detroit: Arrow (313) 971-8220;
Marshall (313) 525-5850; Newark (313) 967-0600;
Schweber (313) 525-8100;
Grand Rapids: Arrow (616) 243-0912.

MINNESOTA: Arrow (612) 830-1800;
Hall-Mark (612) 941-2600; Kierulff (612) 941-7500;
Marshall (612) 559-2211; Schweber (612) 941-5280.

MISSOURI: St. Louis: Arrow (314) 567-6888;
Hall-Mark (314) 291-5350; Kierulff (314) 997-4956;
Schweber (314) 739-0526.

NEW HAMPSHIRE: Arrow (603) 668-6968;
Schweber (603) 625-2250.

NEW JERSEY: Arrow (201) 575-5300,
(609) 596-8000; General Radio (609) 964-8560;
Hall-Mark (201) 575-4415, (609) 235-1900;
Kierulff (201) 575-6750, (609) 235-1444;
Marshall (201) 882-0320, (609) 234-9100;
Schweber (201) 227-7880.

NEW MEXICO: Arrow (505) 243-4566.

NEW YORK: Long Island: Arrow (516) 231-1000;
Hall-Mark (516) 737-0600; Marshall (516) 273-2053;
Schweber (516) 334-7555; Zeus (914) 937-7400.
Rochester: Arrow (716) 427-0300;
Marshall (716) 235-7620; Schweber (716) 424-2222.
Syracuse: Marshall (607) 798-1611.

NORTH CAROLINA: Arrow (919) 876-3132,
(919) 725-8711; Hall-Mark (919) 872-0712;
Kierulff (919) 872-8410; Marshall (919) 878-9882;
Schweber (919) 876-0000.

OHIO: Cleveland: Arrow (216) 248-3990;
Hall-Mark (216) 349-4632; Kierulff (216) 831-5222;
Marshall (216) 248-1788; Schweber (216) 464-2970.
Columbus: Arrow (614) 885-8362;
Hall-Mark (614) 888-3313;
Dayton: Arrow (513) 435-5563;
Kierulff (513) 439-0045; Marshall (513) 236-8088;
Schweber (513) 439-1800.

OKLAHOMA: Arrow (918) 665-7700;
Kierulff (918) 252-7537; Schweber (918) 622-8000.

OREGON: Arrow (503) 684-1690;
Kierulff (503) 641-9153; Wyle (503) 640-6000;
Marshall (503) 644-5050.

PENNSYLVANIA: Arrow (412) 856-7000,
(215) 928-1800; General Radio (215) 922-7037;
Schweber (215) 441-0600, (412) 782-1600.

TEXAS: Austin: Arrow (512) 835-4180;
Hall-Mark (512) 258-8848; Kierulff (512) 835-2090;
Marshall (512) 837-1991; Schweber (512) 458-8253;
Wyle (512) 834-9957;
Dallas: Arrow (214) 380-6464;
Hall-Mark (214) 553-4300; Kierulff (214) 840-0110;
Marshall (214) 233-5200; Schweber (214) 661-5010;
Wyle (214) 235-9953; Zeus (214) 783-7010;
Houston: Arrow (713) 530-4700;
Hall-Mark (713) 781-6100; Kierulff (713) 530-7030;
Marshall (713) 895-9200; Schweber (713) 784-3600;
Wyle (713) 879-9953.

UTAH: Arrow (801) 972-0404;
Hall-Mark (801) 972-1008; Kierulff (801) 973-6913;
Marshall (801) 485-1551; Wyle (801) 974-9953;

WASHINGTON: Arrow (206) 643-4800;
Kierulff (206) 575-4420; Wyle (206) 453-8300;
Marshall (206) 747-9100.

WISCONSIN: Arrow (414) 792-0150;
Hall-Mark (414) 797-7844; Kierulff (414) 784-8160;
Marshall (414) 797-8400; Schweber (414) 784-9020.

CANADA: Calgary: Future (403) 235-5325;
Edmonton: Future (403) 438-2858;
Montreal: Arrow Canada (514) 735-5511;
Future (514) 694-7710;
Ottawa: Arrow Canada (613) 226-6903;
Future (613) 820-8313;
Quebec City: Arrow Canada (418) 687-4231;
Toronto: Arrow Canada (416) 661-0220;
Future (416) 638-4771;
Vancouver: Future (604) 294-1166
Winnipeg: Future (204) 339-0554

BS

May 1987
Revision A
2539567-9701
Printed in U.S.A.

Texas
Instruments