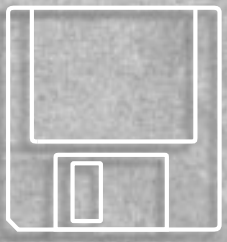
80296SA Evaluation Board Manual

KEEPING YOU ONE DESIGN AHEAD



intel

October, 1996
Order Number: 272947-001

80296SA Evaluation Board Manual

October 1996

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

CONTENTS

FIGURES

TABLES

1

Guide to Manual

CHAPTER 1
GUIDE TO THIS MANUAL

This manual contains information for design engineers who are familiar with the principles of microcontrollers. It describes using the 80296SA Evaluation Board kit for developing and evaluating an embedded application design based on the 80296SA MCS® 96 microcontroller. The 80296SA evaluation board kit contains hardware and software that enables you to write, execute, monitor, and debug application software.

1.1 MANUAL CONTENTS

This manual has six chapters and an appendix.

This chapter provides an overview of the manual. It summarizes the contents of the remaining chapters and the appendix. It also describes notational conventions and terminology; lists related documents, products, data sheets, and user manual supplements; and gives important numbers for obtaining application support.

Chapter 2, “Getting Started with the 80296SA Evaluation Board” — includes a list of the kit contents and instructions on initializing the evaluation board and installing the software.

Chapter 3, “80296SA Evaluation Board Functional Overview” — describes the 80296SA evaluation board; it includes a component-level diagram and describes the installation of memory devices.

Chapter 4, “Introduction to the Embedded Controller Monitor (ECM)” — introduces the user interface software, which comprises ECM96SA and RISMSA.

Chapter 5, “ECM96SA Commands” — describes the part of the Embedded Controller Monitor (ECM) that executes on the host PC.

Chapter 6, “RISM Registers and Commands” — describes the commands for the 80296SA reduced instruction set monitor (RISMSA), the part of the Embedded Controller Monitor (ECM) that executes on the evaluation board microcontroller.

Appendix A, “Components, Jumpers, and Connectors” — provides figures and tables to help you configure the 80296SA evaluation board. It also provides other information for you to consider as you develop your hardware design.

1.2 NOTATIONAL CONVENTIONS AND TERMINOLOGY

The following notations and terminology are used throughout this manual.

— The pound symbol (#) has two meanings, depending on the context. When used with a signal name, it indicates that the signal is active low. When used in an instruction, the symbol prefixes an immediate value in immediate addressing mode.

italics — Italics identify variables and introduce new terminology. The context in which italics are used distinguishes between the two possible meanings.

Variables in registers and signal names are commonly represented by *x* and *y*, where *x* represents the first variable and *y* represents the second variable. For example, in register P*x*_MODE.*y*, *x* represents the variable that identifies the specific port, and *y* represents the register bit variable [7:0 or 15:0].

X — Uppercase X (no italics) represents an unknown value or a "don't care" state or condition. The value may be either binary or hexadecimal, depending upon the context. For example, the hexadecimal value FF2XAFH indicates that bits 11:8 are unknown; 10XX in binary context indicates that the two least significant bytes (LSBs) are unknown.

Board Components — The following abbreviations are used to represent discrete and active components.

C*x*	capacitor
D*x*	diode
DP*x*	LED bank
E*x*	jumper
JP*x*	connector
L*x*	inductor
P*x*	port
R*x*	resistor
RP*x*	resistor pack
S*x*	switch
U*x*	device socket (e.g., latch, buffer, memory, controller)

Assert and Deassert The terms *assert* and *deassert* refer to the act of making a signal active (enabled) and inactive (disabled), respectively. The active polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low (equal to or less than V_{OL}); to assert ALE is to drive it high (equal to or greater than V_{OH}); to deassert RD# is to drive it high; to deassert ALE is to drive it low.

Instructions Instruction mnemonics are shown in upper case; however, you may use either upper case or lower case in your source code.

NC The term "NC" is an abbreviation for "no connection." It indicates that no connection is required.

Numbers Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character H. Decimal and binary numbers are represented by their customary notations. (That is, 255 is a decimal number and 11111111 is a binary number. In some cases, the letter B is appended to binary numbers for clarity.)

Units of Measure The following abbreviations are used to represent units of measure:

A	amps, amperes
mA	milliamps, milliamperes
Kbyte	kilobytes
KHz	kilohertz
KΩ	kilo-ohms
Mbyte	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
V	voltage, volts
VDC	voltage, direct current
VAC	voltage, alternating current
W	watts
µA	microamps, microamperes
µF	microfarads
µs	microseconds

Register Bits Bit locations are indexed by 7:0 (or 15:0), where bit 0 is the least-significant bit and 7 (or 15) is the most-significant bit. An individual bit is represented by the register name, followed by a period and the bit number. For example, WSR.7 is bit 7 of the window select register. In some discussions, bit names are used. For example, the name of WSR.7 is HLDEN.

Register Names Register names are shown in upper case. For example, TIMER2 is the timer 2 register; timer 2 is the timer. If a register name contains a lowercase character, it represents more than one register. For example, P*x*_REG represents four registers: P1_REG, P2_REG, P3_REG, and P4_REG.

Reserved Bits Certain bits are described as *reserved* bits. In illustrations, reserved bits are indicated with a dash (—). These bits are not used in this device and may be used in future implementations. To help ensure that a current software design is compatible with future implementations, reserved bits should be cleared (given a value of "0"), unless otherwise noted.

Set and Clear The terms *set* and *clear* refer to the value of a bit or the act of giving it a value. When a bit is *set*, its value is "1"; *setting* a bit gives it a "1" value. When a bit is *clear*, its value is "0"; *clearing* a bit gives it a "0" value.

Signal Names Signal names are shown in upper case. When several signals share a common name, an individual signal is represented by the signal name followed by a number. For example, the EPA signals are named EPA0, EPA1, EPA2, etc. Port pins are represented by the port abbreviation, a period, and the pin number (e.g., P1.0, P1.1). A pound symbol (#) appended to a signal name identifies an active-low signal.

Command Lines For command line input to software, such as MS-DOS* and ECM96SA, this manual uses notation described in Section 5.2, "Command Line Notation".

1.3 RELATED DOCUMENTS

Table 1-1 lists the names of documents that are useful in designing systems using an 80296SA embedded microcontroller. The documents are available through Intel Literature (1-800-548-4725 in the U.S. and Canada) on the Intel World Wide Web site (http://www.intel.com).

Table 1-1. Related Documents

Document Name	Order Number
80296SA Commercial CHMOS 16-Bit Microcontroller	272748
80296SA Microcontroller User's Manual	272803
AP-125, Designing Microcontroller Systems for Electrically Noisy Environments	210313
AP-715, Interfacing an 1^2C Serial EEPROM to an MCS® 96 Microcontroller	272680
AP-717, Migration from the 8XC196Nx to the 80296SA	272730

1.4 APPLICATION SUPPORT SERVICES

You can get up-to-date technical information from a variety of electronic support systems: the World Wide Web, the FaxBack* service, and Intel's Brand Products and Applications Support bulletin board service (BBS). These systems are available 24 hours a day, 7 days a week, providing technical information whenever you need it.

In the U.S. and Canada, technical support representatives are available between 5 a.m. and 5 p.m. Pacific Stand Time (PST). Outside the U.S. and Canada, please contact your local distributor. You can order product literature from Intel literature centers and sales offices.

Table 1-4 lists the information you need to access these services.

Table 1-2. Intel Application Support Services

Service	U.S. and Canada	Asia-Pacific and Japan	Europe
World Wide Web	http://www.intel.com/	http://www.intel.com/	http://www.intel.com/
FaxBack*	800-525-3019	503-264-6835 916-356-3105	+44(0)1793-496646
BBS	503-264-7999 916-356-3600	503-264-7999 916-356-3600	+44(0)1793-432955
Help Desk	800-628-8686 916-356-7999	Please contact your local distributor.	Please contact your local distributor.
Literature	800-548-4725	708-296-9333 +81(0)120 47 88 32	+44(0)1793-431155 England +44(0)1793-421777 France +44(0)1793-421333 Germany

1.4.1 World Wide Web

We offer a variety of information on the World Wide Web (http://www.intel.com/design/mcs96). Also visit Intel's Web site for financial information, history, and news.

1.4.1.1. FaxBack* Service

FaxBack is an on-demand publishing system that sends documents to your fax machine. You can get product announcements, change notifications, product literature, device characteristics, design recommendations, and quality and reliability information from FaxBack 24 hours a day, 7 days a week.

Think of the FaxBack service as a library of technical documents you can access with your phone. Just dial the telephone number listed in Table 1-2 on page 1-5 and respond to the system prompts. After you select a document, the system sends a copy to your fax machine.

Each document has an order number and is listed in a subject catalog. The first time you use Fax-Back, you should order the appropriate subject catalogs to get a complete listing of document order numbers. Catalogs are updated regularly, so call for the latest information.

1.4.2 Bulletin Board Service (BBS)

The bulletin board system (BBS) lets you download files to your computer. The application BBS has the latest *Ap*BUILDER software, hypertext manuals and datasheets, software drivers, firmware upgrades, application notes and utilities, and quality and reliability data.

Any customer with a modem and computer can access the BBS. The system provides automatic configuration support for 1200- through 19200-baud modems. Use these modem settings: no parity, 8 data bits, and 1 stop bit.

To access the BBS, just dial the telephone number (see Table 1-2 on page 1-5) and respond to the system prompts. During your first session, the system asks you to register with the system operator by entering your name and location. The system operator will set up your access account within 24 hours. At that time, you can access the files on the BBS.

NOTE

In the U.S. and Canada, you can get a BBS user's guide, a master list of BBS files, and a list of FaxBack documents by calling 1-800-525-3019. Use these modem settings: no parity, 8 data bits, and 1 stop bit.

2

Getting Started with the 80296SA Evaluation Board

CHAPTER 2
GETTING STARTED WITH THE 80296SA EVALUATION BOARD

The 80296SA evaluation board kit contains hardware and software that enables you to write, execute, monitor, and debug application software. This chapter includes a list of the 80296SA evaluation board kit contents. It previews the hardware and software design tools, and it steps you through the procedure for initializing and running the evaluation board.

2.1 EVALUATION BOARD KIT CONTENTS

The 80296SA Evaluation Board kit includes the following items.

- 80296SA Evaluation Board, which includes:
  - An 80296SA embedded microcontroller in a shrink quad flatpack (SQFP) package
  - Two external memory devices (installed on the board): 256-Kword flash (pre-programmed with the RISMSA monitor) and 256Kbyte-SRAM for downloaded user code (see Figure 2-1 on page 2-2)
- A DB-9S RS-232 9-pin straight-through cable.
- 3.5-inch MS-DOS* Diskette. This diskette contains Embedded Controller Monitor (ECM) software, which is detailed in Chapters 4, 5, and 6. The software consists of the following files that run and debug 80296SA programs from a host PC:
  - ecm96sa.exe
  - sar_main.asm
  - sar_main.lst
  - sar_main.obj
  - sar_main.hex
  - 80296sa.inc
- Third-party vendor software.
- *Ap*BUILDER Interactive programming software.
- Various documentation on the device and development tools.
- Technical Documentation: The evaluation board kit includes this manual, the *80296SA Evaluation Board User's Manual*. For available related documentation, see "Related Documents" on page 1-5. A set of evaluation board schematics is also provided in an envelope.

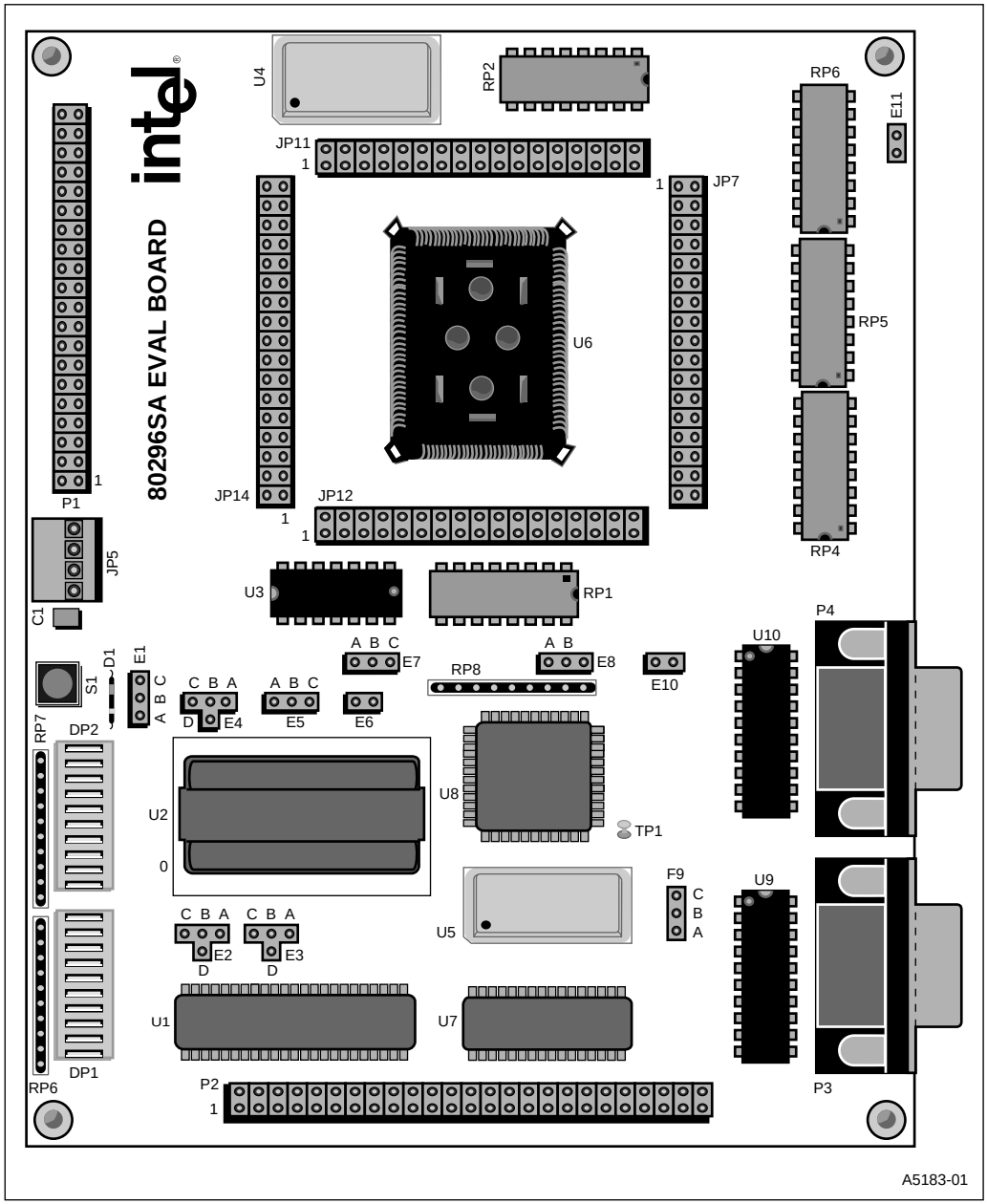


Figure 2-1. 80296SA Evaluation Board Layout

2.2 CONNECTING THE EVALUATION BOARD TO THE HOST SYSTEM

Complete the following procedure to connect the 80296SA to the host system and power up the evaluation board (Figure 2-1 on page 2-2 shows the location of the evaluation board's power, ground, and serial port connections.):

1. Turn off power to the PC and the power supply.
2. Connect the serial port cable (DB9) from the board's P4 connector to either the com1 or com2 serial port on your PC.

   (You will use the board-to-PC connection **after** invoking the ECM, but connect the cables now. If you need details about the individual components of the DB9 cable, see Table A-3 on page A-7.)
3. Connect the power cable from the power supply to the JP5 connector on the 80296SA evaluation board.

   Use a regulated +5 VDC power supply. Lower voltage might not operate the evaluation board. Higher voltage might damage the evaluation board. An unregulated power supply may cause unpredictable failure conditions. A regulated +12VDC is optional. Even so, you will need it if you program or erase flash.

   CAUTION

   The power-supply plug is keyed, so it should easily attach to the board; if you forced it on, you may have attached it backward. Powering up the board through a backward plug may damage board components.

4. Turn on the PC and power supply. The LED (light-emitting diode) banks at DP1 and DP2 on the 80296SA evaluation board should flash through a power-up sequence. At power-on, LEDs 1 through 8, in both banks, sequentially blink. LED 9 on DP1 comes on during the power-up sequence, while LED 9 on DP2 remains off during the entire power-up sequence. LED 10 remains on to indicate that +5 VDC is applied to the board.
   - — If the LED bank flashed, the board is working correctly and you are ready to skip to the next section.
   - — If the LED bank did not flash, continue to step 5.
5. If the LED bank did not flash as described, check the following items:
   - — Be sure that power is supplied to the board. Check the connection between the power supply cable and the board's power connector.
   - — Confirm that the jumper settings are correct for the memory devices shipped with the board (or for a memory device that you have installed). See Table A-2 on page A-4.
   - — Press the reset button (S1) on the 80296SA evaluation board. If the board still does not respond, see Chapter 1, "Application Support Services" on page 1-5 for assistance.

2.3 INVOKING THE EMBEDDED CONTROLLER MONITOR SOFTWARE

After the 80296SA evaluation board is initialized and executing RISMSA from the flash, you can start the ECM and run the demonstration program by completing the following procedure:

1. Insert the 3.5-inch diskette in the drive of your PC.
2. Create a directory for the ECM software and copy the contents of the diskette to the directory.
3. To invoke ECM, complete the appropriate bulleted item:
   - To invoke ECM from the directory you created in step 2 of this procedure, type the following command from a DOS prompt: ecm96sa <Enter>
   - To invoke ECM from the diskette, type the following command from a DOS prompt: [drivename] :\ecm96sa <Enter>
     (For example, if the diskette is in drive A, type a:\ecm96sa <Enter>)
4. Observe the ECM96SA monitor screen on your host PC.

   When you invoke the ECM96SA program, it communicates with the board and interrupts the RISMSA monitor. The continuous LED sequencing terminates, and a steady pattern displays. The ECM96SA program displays the baud rate followed by an asterisk (*); this is the input prompt. At this point, you can use the ECM96SA commands described in Chapter 5.

3

80296SA Evaluation Board Functional Overview

CHAPTER 3
80296SA EVALUATION BOARD FUNCTIONAL OVERVIEW

This chapter describes functional units of the 80296SA evaluation board. The board is designed as a basic demonstration system for evaluating hardware and software performance. This chapter also includes a block diagram of the board and a diagram of the major components of the board with a brief description of each functional section.

3.1 BLOCK AND COMPONENT DIAGRAMS OF THE BOARD

Figure 3-1 is a block diagram of the 80296SA evaluation board. The diagram illustrates the four main parts of the board: the 80296SA microcontroller, digital I/O, memory, and the interface between the 80296SA and the host PC. As shipped, the board has a both a 128-Kbyte and a 64-Kword SRAM. It also has a 256-Kword flash.

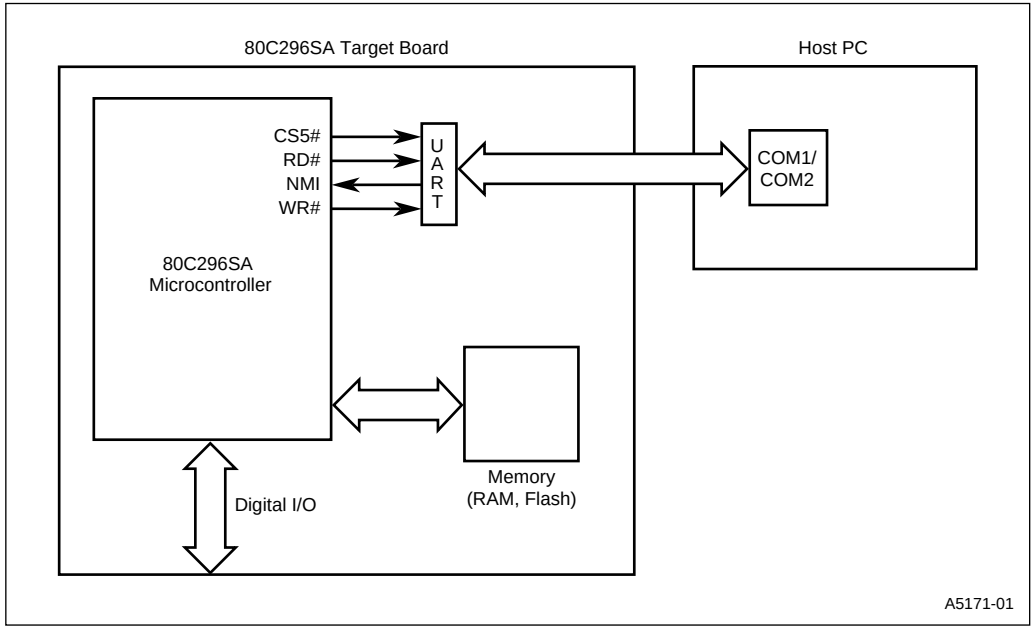


Figure 3-1. 80296SA Evaluation Board Block Diagram

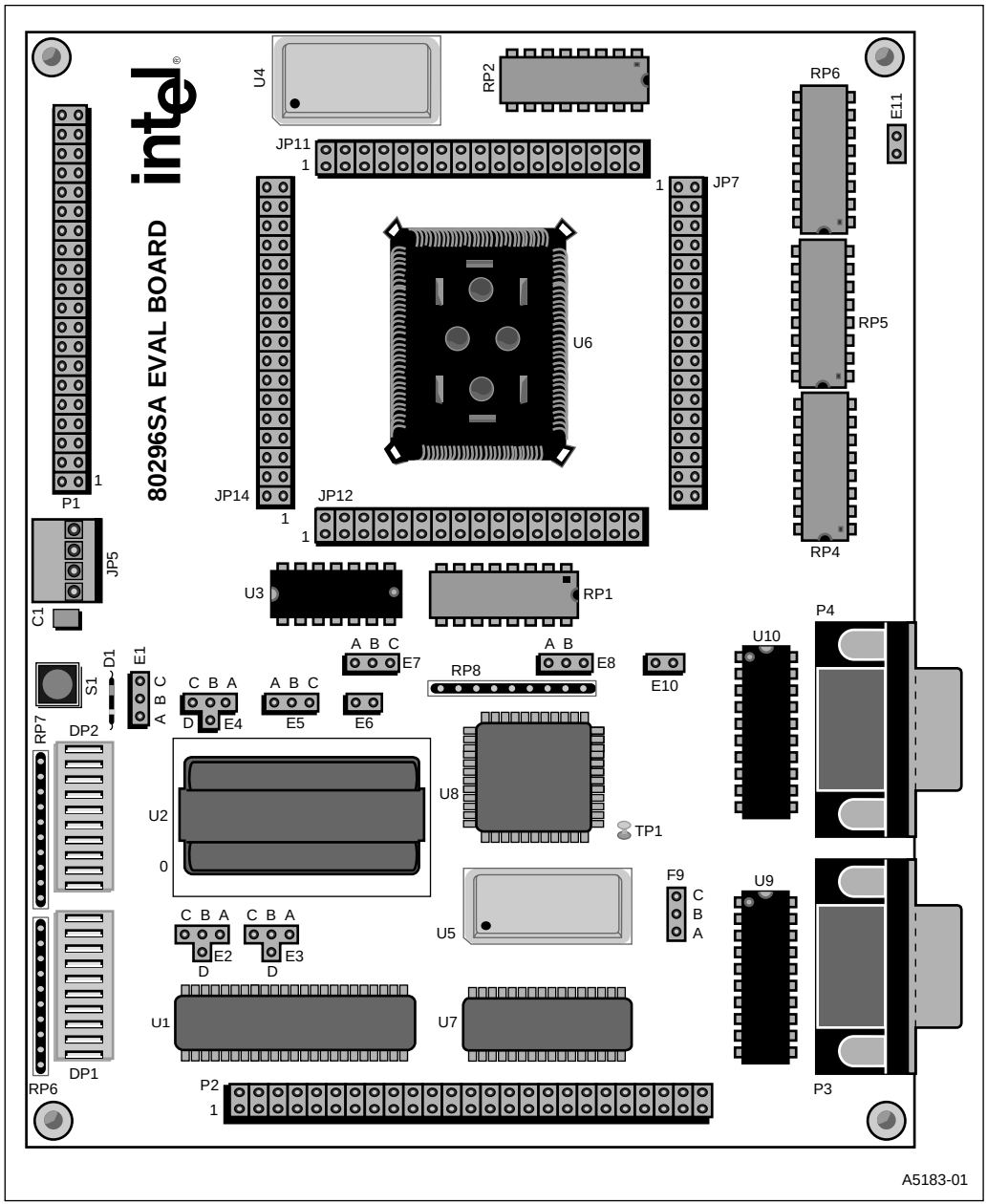


Figure 3-2. Component-level Diagram of the 80296SA Evaluation Board

Figure 3-2 is a component-level diagram of the evaluation board. Physically, the 80296SA evaluation board measures 4.75 inches × 7 inches. Component details are in both Appendix A, “Components, Jumpers, and Connectors”.

3.2 THE 80296SA MICROCONTROLLER

The 80296SA is a highly integrated 16-bit CHMOS microcontroller. Like all MCS® 96 microcontrollers, the 80296SA is optimized for control applications. It is pin-compatible with the 8XC196NP and 8XC196NU microcontrollers; however, some electrical and timing differences exist. Please consult the Intel web site at http://www.intel.com for a current datasheet.

A key feature of the 80296SA is its extremely fast execution engine; it is capable of high program and data throughput. The internal address/data bus maps into a single, linear 16-Mbyte address space for both code and data storage. Other important features include a 32-bit accumulator to increase math performance, an 8-byte prefetch queue, and clock doubling/quadrupling circuitry with phase-locked loop circuitry to support 50 MHz operation at + 5 VDC. Refer to the *80296SA Microcontroller User’s Manual* (order number 272803) for additional feature information.

3.3 HOST INTERFACE

The host interface is a connection between the host PC serial port (com1 or com2) and the 80296SA serial I/O port (Figures 3-1 and 3-2). The com1 or com2 port connects to a 9-pin connector (P4) on the board and then to the on-board serial I/O (SIO) port via an RS-232 interface. Four 80296SA signals are used for the host interface. The RS-232 interface uses the nonmaskable interrupt (NMI) to signal the 80296SA that a character from the host is ready for reception.

3.4 DIGITAL I/O

You can use all digital I/O functions of ports 1–4 for general purposes, except for the following chip selects:

- P3.0/CS0#
- P3.1/CS1#
- P3.2/CS2#
- P3.5/CS5#

3.5 80296SA MEMORY SYSTEM

The 80296SA evaluation board is configureable for extended and nonextended external memory transfer. A key to using the 80296SA memory interface is understanding the relationship between internal memory addresses and external memory addresses. The 80296SA has 24 internal address bits (A23:0). These bits support an internal 16-Mbyte linear address space (see Figure 3-3 on page 3-5). For convenience in discussions, the internal address space (000000H–FFFFFFH) can be viewed as comprising 16 memory pages; each page is 1 Mbyte in size (00000H–FFFFFH).

On the external system bus, the lower 20 of the 24 internal address bits are implemented by external pins: A19:0 in demultiplexed mode, or A19:16 and AD15:0 in multiplexed mode. The lower 16 address/data pins (AD15:0) are the same as those in all other MCS 96 microcontrollers. EPORT provides the four extended address pins (A19:16). Therefore, each 20-bit external address can originate from any one of 16 internal 24-bit addresses (see Figure 3-3 on page 3-5). If the internal 24-bit address issued to the external bus during a fetch cycle is FF2018H, the 20 external-address pins output that address minus the upper four address bits, or F2018H. Further, the address seen by an external device depends on how many of the 20 address pins are connected to the external device. For more information, see the *80296SA Microcontroller User's Manual*.

Some specific internal addresses are reserved for in-circuit emulation or are not available externally. Table 3-1 describes the availability of these internal addresses.

Table 3-1. Reserved or Non-Available Addresses

Address	Description	Notes
FF0000-FF00FFH	Reserved for ICE	Always reserved
001F00-001FFFH	Peripheral SFRs	Always internal
000000-0001FFH	Upper and lower register file (register RAM, stack pointer, and CPU SFRs)	Always internal

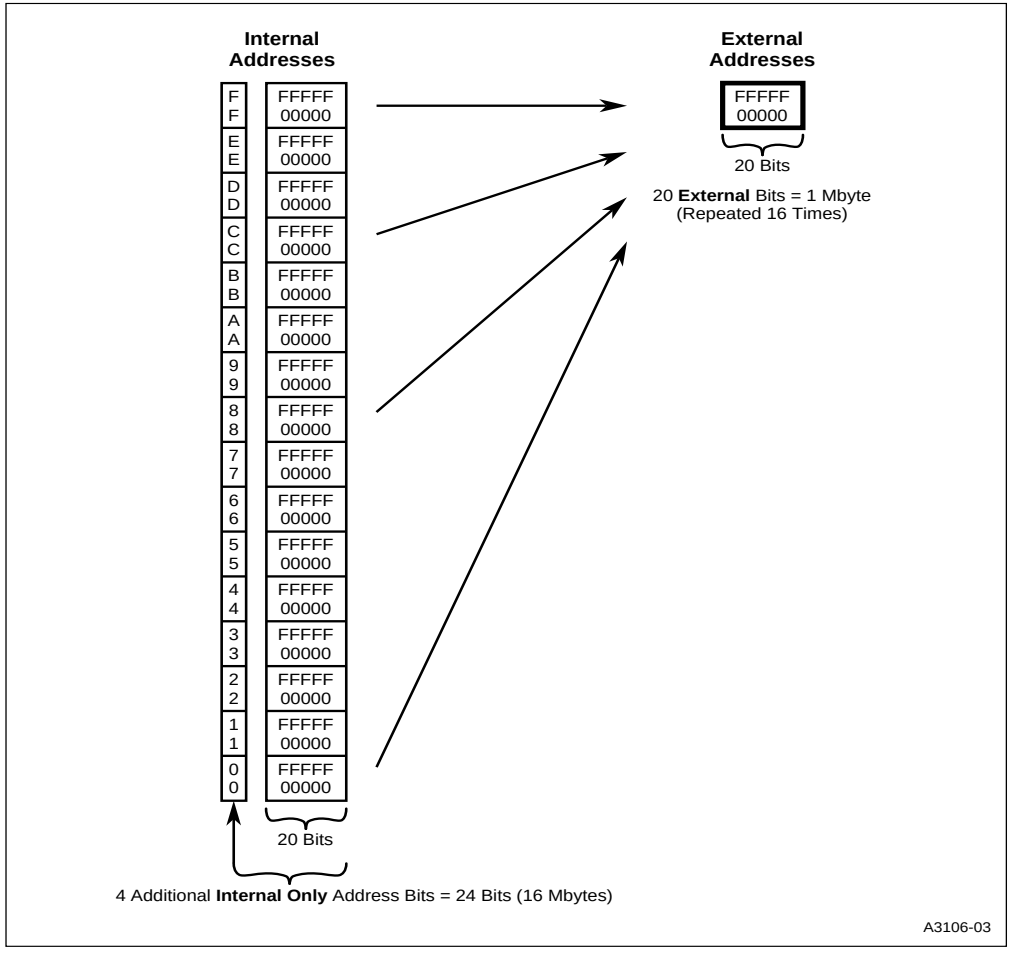


Figure 3-3. 80296SA Memory Map

3.5.1 Memory Modes

The MODE64 bit (CCB1.1) selects one of two memory modes: 1-Mbyte mode (CCB1.1=0) and 64-Kbyte mode (CCB1.1=1). In 1-Mbyte mode, code can execute from any page in the linear 1-Mbyte space. In 64-Kbyte mode, code can execute only from the 64-Kbyte area FF0000–FFFFFFH. Extended instructions (e.g., JUMP, BRANCH, and CALL) do **not** function in the 64-Kbyte mode. However, this mode provides compatibility with existing software written for MCS 96 64-Kbyte memory maps. The RISMSA software configures the 80296SA in 1-Mbyte mode.

NOTE

Data accesses are the same in the 1-Mbyte and 64-Kbyte modes. The device can access data in any page. Data accesses to page 00H are nonextended. Data accesses to any other page are extended.

3.5.2 Using SRAM, EPROM, and Flash

The 80296SA evaluation board supports SRAM (static operation) in 8-bit and 16-bit modes, and flash devices. The 80296SA ships with the following (to see a visual representation of the items listed below, see Figure 3-2 on page 3-2):

- 128-Kbyte SRAM in U7
- 64-Kword SRAM in U1
- 128-Kword flash with the RISMSA software installed in U2

At power-up, the 80296SA boots from this flash in U2, which is selected by chip-select 0 (CS0#). You can then download your application code to the 128-Kbyte SRAM in U7, which is selected by CS1#. (See “Connecting the Evaluation Board to the Host System” on page 2-3 and “Invoking the Embedded Controller Monitor Software” on page 2-4.)

The memory can also be used in other ways. For example, you can program a flash device and execute that software without using the Embedded Controller Monitor (ECM).

4

Introduction to the Embedded Controller Monitor (ECM)

CHAPTER 4
INTRODUCTION TO THE EMBEDDED CONTROLLER MONITOR (ECM)

This chapter introduces the Embedded Controller Monitor (ECM) user interface. This is the interface between the PC-resident software and the evaluation board firmware. The ECM software consists of two programs: ecm96sa.exe and sar_main.hex. The commands for these programs are described in Chapter 5, “ECM96SA Command Notation” and Chapter 6, “RISM Registers and Commands”.

4.1 EMBEDDED CONTROLLER MONITOR

ECM is the software interface between the host system and the user code running on the evaluation board. It provides basic debug capabilities, including loading object files into system RAM, examining and modifying variables, and executing and stepping through code.

The 80296SA evaluation board uses a version of the ECM written for the MCS® 96 microcontrollers with extended addressing capability. The ECM environment comprises two independent programs: sar_main.hex and ecm96sa.exe. The sar_main.hex program (referred to as RISMSA) resides in the evaluation board flash; 80296SA executes it. The ecm96sa.exe software (known as ECM96SA) resides and executes in DOS*- and Windows*-based PCs and BIOS-compatible computers.

RISMSA is a reduced instruction set monitor for the 80296SA. It executes rudimentary operations issued by ecm96sa.exe, which operates in the host PC. RISMSA consists of approximately 700 bytes of 80296SA code: a short section of initialization code and an interrupt service routine (ISR) that processes interrupts from the host system. The RISMSA ISR consists of a short prologue and then a case-jump to one of several handlers.

ECM96SA, executing in the host PC, provides commands for loading and running code on the 80296SA. It also has features that facilitate test and debug tasks. For example, it can use include, list, and log files to record on-line ECM sessions and construct batch ECM sessions.

Partitioning the ECM into two separate programs supports a number of goals in developing this system:

- The RISMSA code in the evaluation board is simple and small. This maximizes the space available for user code.
- The ECM96SA user interface's features expand beyond the resources of the evaluation board because ECM96SA runs in the host PC.
- RISMSA and ECM96SA run concurrently. They allow you to interrogate and modify the state of the evaluation board system while it is running.

4.2 RESTRICTIONS

The ECM operates under several restrictions:

- Several user stack words are reserved for RISMSA software use when the evaluation board processes a host interrupt (see the CAUTION on page 5-17). Internal register locations 0001E0H–0001FFH are reserved for RISMSA code use. Users must ensure that no registers in this partition are used by code operating with the RISMSA.
- A 9600-baud asynchronous serial port must be available on the host PC.
- The TRAP instruction is reserved.
- The breakpoint and step commands operate only with user code located in RAM. (User code located in EPROM or flash memory cannot use the breakpoint and step commands.)

5

ECM96SA Commands

CHAPTER 5
ECM96SA COMMANDS

This chapter describes the ECM96SA commands. To begin using ECM96SA, see the procedures for powering up the board (“Connecting the Evaluation Board to the Host System” on page 2-3) and invoking ECM96SA for the first time (“Invoking the Embedded Controller Monitor Software” on page 2-4).

5.1 ECM DEFINED

ECM96SA is the portion of the Embedded Controller Monitor (ECM) that runs on the host PC. It provides several tools with RISMSA for testing and debugging code on the evaluation board. ECM96SA commands support tasks such as displaying and modifying program variables, initializing and operating program breakpoints, and single-stepping program execution.

5.2 COMMAND LINE NOTATION

This subsection explains command line notation. Even though the commands are listed in lowercase, both ECM96SA and DOS are case insensitive.

5.2.1 ECM96SA Command Notation

When entering ECM96SA commands, use the basic rules below (Table 5-1 includes examples of the rules):

- Use parameters and keywords when using commands that affect specific addresses and files.
- Use a comma as a Boolean OR. For example [this,that] is interpreted as [this] OR [that].
- Insert a hyphen immediately before the command when invoking ECM96SA.

Table 5-1. ECM96SA Command Notation

Rules	Example Command Line Notation and Descriptions†
Parameter	**Example**: string *byte_address* <Enter> **Parameter**: *byte_address* (used to specify a specific address)
Keyword	**Example**: go [from *code_address1* till *code_address2*] <Enter> **Keyword**: till (used to indicate a range. In this example, it indicates the range between the two parameters *codeaddress1* and *codeaddress2*.)

† The square brackets [ ] indicate an optional argument.

Table 5-1. ECM96SA Command Notation (Continued)

Rules	Example Command Line Notation and Descriptions†
Comma	**Example**: dasm [*code_address*], [*count*] <Enter> **Comma**: used to separate distinct parameters. In this example, it separates the parameters [*code_address*] and [*count*].
Hyphenation	**Example**: ecm96sa [-com1(default), com2] <Enter> **Mandatory item**: A hyphen must precede the first command. **Description**: only used to invoke ECM96SA.

† The square brackets [ ] indicate an optional argument.

5.2.2 DOS Command Rules

When entering DOS commands, follow these basic rules (Table 5-2 includes examples of the rules):

- Use parameters and keywords when using commands that affect specific addresses and files.
- Use commas to separate parameters.

Table 5-2. DOS Command Notation

Rules	Example Command Line Notation and Descriptions†
Parameter	**Example**: string *byte_address* <Enter> **Parameter**: *byte_address* (used to specify a specific address)
Keyword	**Example**: go [from *code_address1* till *code_address2*] <Enter> **Keyword**: till (used to indicate a range. In this example, it indicates the range between the two parameters *code_address1* and *code_address2*.)
Comma	**Example**: dasm [*code_address*], [*count*] <Enter> **Comma**: used to separate distinct parameters. In this example, it separates the parameters [*code_address*] and [*count*].

† The square brackets [ ] indicate an optional argument.

5.3 INITIALIZING AND TERMINATING ECM

The commands discussed in Table 5-3 invoke and terminate ECM96SA from DOS, specify numerical bases (octal, decimal, or hexadecimal), and temporarily exit to DOS.

Table 5-3. Commands for Invoking and Terminating ECM96SA

Command Names	Command Notations and Descriptions[1,2]
ecm96sa	**Notation**: ecm96sa [-*option1, option2, optionN*] <Enter> **Description**: Loads and executes the ECM96SA software. Command options are described below. You can enter string options in any order; if the options are contradictory, the system accepts the last option entered. If ECM96SA detects valid CTS (Clear to Send) and DSR (Data Set Ready) signals from the appropriate COM port, it signs on and displays one of the following command prompts: • When the board is executing code, it displays a greater-than sign (>). • When the board is **not** executing code, it displays an asterisk (*). • When CTS or DSR is not present, ECM96SA notifies you and asks if you want to proceed or exit. If you proceed, ECM96SA may operate properly, but your serial port or cabling may have a problem that will prevent proper operation.
com	**Notation**: [-com1 (default), com2] **Description**: Specifies the serial communication port to be used for host interface. The default is COM1.
baud	**Notation**:[-baud 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200] **Description**: Specifies the host-evaluation board communication rate. Baud rates higher than 9600 baud may not be supported on 8088-based PCs. A baud rate of 9600 baud can load 8 Kbytes of data in about 20 seconds. A baud rate of 57600 can load 8 Kbytes of data in about 4 seconds.
notypes	**Notation**: [-notypes] **Description**: Causes the object file loader to ignore type definition records in the object module. If it is invoked, the I/O routines recognize only basic data types, such as BYTEs, WORDs, and LONGs. More complex data types, such as PLM arrays and structures, are not recognized.

Notes:

1. All commands used to invoke ECM96SA begin with a hyphen.
2. The square brackets [ ] indicate an optional argument.

5.4 GENERAL ECM96SA COMMANDS

Issue the general commands discussed in Table 5-4 after you invoke ECM96SA.

Table 5-4. General ECM96SA Commands

Command Names	Command Notations and Descriptions†
dos	**Notation**: dos <Enter> **Description**: Lets you temporarily leave ECM96SA and return to DOS to run other application software. To return to ECM96SA, type: exit <Enter>
exit	**Notation**: exit <Enter> **Description**: This command has two functions: • Returns the user to ECM96SA from DOS. When it returns, ECM96SA has the conditions that were in effect when it was temporarily suspended. • Closes any file that ECM96SA has opened and exits to DOS. You can use this command even if the evaluation board is running a program (execution continues). ECM96SA sets the selected COM port to 9600 baud, 8 data bits, no parity, and one stop bit, and then returns to DOS. The quit command also performs this duty.
base	**Notations**: • base <Enter> • base = *10t* • base [= 10o, 10t (breakpoint default), 10h (address default)] <Enter> **Description**: Displays the default arithmetic base. The default base is used to display variables and to enter numbers into the command parser. However, you can override the default base during input by adding an override character to the end of the number. The override characters are: o (octal), t (decimal), and h (hexadecimal). You must add the override character immediately after the last digit of the number. **Do not include a space**. Program addresses are always displayed in hexadecimal; and, breakpoint numbers are always displayed in decimal.
quit	**Notation**: quit <Enter> **Description**: Closes any file that ECM96SA opened and exits to DOS. You can use this command even if the evaluation board is running a program (execution continues). ECM96SA sets the selected COM port to 9600 baud, 8 data bits, no parity, and one stop bit, and then returns to DOS. The quit command also performs this duty.

† The square brackets [ ] indicate an optional argument.

5.5 FILE OPERATIONS

This section describes the commands that ECM96SA uses to load and save object code, enter predefined strings of commands, log commands, and record entire debug sessions including user entries and the response generated by ECM96SA on the host screen.

5.5.1 Loading and Saving Object Code

ECM96SA accepts object files generated by Tasking (formerly BSO) development tools in the OMF96 version 3.0 format. ECM96SA does not accept files containing unresolved externals or files containing relocateable records. Pass these files through the RL196 linker to resolve the externals and/or absolutely locate the relocateable segments.

To load new code from the PC into the 80296SA evaluation board, use the load and program operations. The load command downloads code that will reside in RAM. Code that will reside in flash is downloaded to the evaluation board using the "program/erase" commands.

Table 5-5 discusses the ECM96SA commands that currently operate on object files.

Table 5-5. ECM96SA Commands that Operate on Object Files

Command Names	Command Notations and Descriptions
load	**Notation**: load *filename* <Enter> **Description**: Loads the content records of the object *filename* into the evaluation board's code RAM or external RAM. The LOAD instruction cannot be used on flash.
save	**Notations**: • save *filename* <Enter> • save *code_address1* to *code_address2* in *filename* <Enter> **Description**: Saves a region of memory as an object file that can be reloaded into the evaluation board's memory.
ramrism	**Notation**: ramrism **Description**: Copies RISM to external RAM and switches RISM execution to RAM. This command overwrites the contents of external RAM FF2080h-FF2700h.
flashrism	**Notation**: flashrism **Description**: Switches RISM execution back to the flash. This is the default.
erase	**Notation**: erase *addr* <Enter> **Description**: Erases the flash block associated with the given address. The command then erases the flash block associated with the *addr* argument. The user must issue an erase command for each block of flash that the user wants to erase. This command overwrites the contents of external RAM at FF2080-FF2700H.

Table 5-5. ECM96SA Commands that Operate on Object Files (Continued)

Command Names	Command Notations and Descriptions
program	**Notation**: program *filename* <Enter> **Description**: Programs the flash with the object *filename*. It does not do an erase before programming. Use the erase command to erase each block that will be reprogrammed before programming. This command overwrites the contents of external RAM at FF2080-FF2700H.

5.5.2 Flash Memory Program/Erase

This subsection discusses commands used for programming and erasing flash memory.

WARNING

As a rule, do not erase or program over flash locations 0F2000-0F2700H because the RISM is in this location. Erasing or programming over the RISM renders the board useless. The only exception to this rule is you can use the procedures when updating the RISM. Each update will include specific procedures that must be followed exactly. Altering the update instructions may render the board useless.

You can easily program the flash regions that do not contain RISM by typing the following commands:

`erase` *`baseaddress`* `<Enter>.`

`program` *`filename`* `<Enter>.`

Where:

`baseaddress` = the base address of the memory block you want to erase

`filename` = the name of the file that contains your absolute code

5.5.3 Include, Log, and List Files

Include files contain commands that ECM96SA executes. You can prepare a command sequence off-line and later have ECM96SA execute the commands just as if they were entered from the keyboard. An include file can be tedious to generate with a text editor. However, ECM96SA can use a log file to store characters that you enter during an ECM96SA session. Later, you can use the log file as an include file to recreate a command sequence. List files keep a running record of commands you enter and the response ECM96SA generates.

You can insert comments in list and log files to make them easier to understand. A comment begins with a semicolon (;) and ends with an <Enter> or <Esc> character. The semicolon is part of the comment. The <Enter> or <Esc> character is not part of the comment.

When creating a log file, keep in mind you can place characters in the file to help you transform the file into a list file. You can use the list file to re-create command sequences. List files keep a running record of both the commands you enter and the responses ECM96SA generates.

With the list file and log file commands, you can either overwrite existing data in the file or append data to the file. By using default filenames, you can gather list and log data in the default files and avoid having to create and manage a large number of separate files. ECM96SA appends the date and time to log files and list files whenever they are opened. This information makes it easier for you to use a text editor to sort the data from the debug sessions.

The commands involved in include, log, and list operations are discussed in Table 5-6.

Table 5-6. Include, Log, and List Commands

Command Names	Command Notations and Descriptions
include	**Notation**: include *filename* <Enter> **Description**: Attempts to open *filename* as a read-only file. If the file can be opened, the command parser takes commands from that file. These commands must contain the exact sequence of ASCII characters you would type to execute them from the keyboard. Once the command parser reaches the end of the file, the file closes. Only one include file can be opened at a time.
pause	**Notation**: pause *filename* <Enter> **Description: (Use within include files**.) Pause is not a file-oriented command. When the command parser reads this command, it stops parsing and waits for you to press <Space> from the keyboard. The <Space> character cannot come from the include file. The pause command provides a way to pause in the middle of an include file operation. When you press <Space>, the parser continues parsing commands within the include file.
list	**Notations**: • list *filename* <Enter> • list <Enter> **Description**: Attempts to open *filename* as a writable file. If a file with *filename* already exists, ECM96SA asks if the file is to be overwritten or if the new data should be appended to the end of the existing file. It then opens the file and stamps it with the current date and time from the system clock. After this, the file records the commands you enter and the responses ECM96SA generates. If you do not enter a *filename*, the list command uses the last *filename* entered as part of a list *filename* command. If you have not entered any list *filename* commands, it uses the default filename “LIST.ECM”

Table 5-6. Include, Log, and List Commands (Continued)

Command Names	Command Notations and Descriptions
listoff	**Notation**: listoff <Enter> **Description**: Closes the last list file specified by the list command. If no filename is specified, it uses the default filename “LIST.ECM”. ECM96SA then stops recording new commands and responses.
liston	**Notation**: liston <Enter> **Description**: Re-opens the list file in the append mode, so recording can start again. It also stamps the list file with the current date and time from the system clock. This stops new list information from being recorded.
log	**Notations**: • log *filename* <Enter> • log <Enter> **Description**: Attempts to open *filename* as a writable file. If a file with *filename* already exists, ECM96SA asks if the file is to be overwritten or if the new data should be appended to the end of the file. It then opens the file and stamps it with the current data and time. After this, the file records the commands you enter. This file may contain nonprintable characters (e.g., <Esc>). If you do not enter a filename, the log command uses the last *filename* entered as part of a log *filename* command. If you have not entered any log *filename* commands, it uses the default filename “LOG.ECM”
logoff	**Notation**: logoff <Enter> **Description**: Closes a log file that has been specified by the log command. ECM96SA then stops recording new commands.
logon	**Notation**: logon <Enter> **Description**: Re-opens the list file in the append mode, so recording can begin again. LOGON also stamps the list file with the current date and time from the system clock.

5.6 PROGRAM CONTROL

Commands in this group control program execution and allow you to reset the microcontroller, set execution breakpoints, start execution, stop execution, step, and super-step.

5.6.1 80296SA Reset

The following command resets the 80296SA without resetting the entire evaluation board.

reset **Notation**: reset chip <Enter>

Description: Physically resets the microcontroller by writing 0XXXX0001B to the RISM_DATA register. It then issues a “monitor_escape rism” command, which causes the evaluation board to execute a reset (RST) instruction.

5.6.2 Breakpoint Features

You can use breakpoints to stop execution at specified addresses. You may also use breakpoints to examine and/or modify registers and memory before resuming execution.

NOTE

When breakpoints are used to halt application code, microcontroller timers and peripherals (such as EPA, serial ports, and PWM) may remain active.

5.6.2.1 Breakpoint Operation

ECM96SA provides 16 program execution breakpoints, BR0 to BR15, and a set of commands to set or clear the breakpoints. A command activates a breakpoint by assigning a specific address of an instruction where execution is to stop. For example, if "br2 = 0ff209dh <enter>", execution halts at address FF209DH. You must set the breakpoint to the address of the first (least significant) byte of the instruction. If a breakpoint is set to an address that is not the first byte of an instruction, execution is unpredictable.

To clear a breakpoint (make it "inactive") assign a zero to the breakpoint (e.g., "br2 = 0 <Enter>"). When execution begins, ECM96SA saves the application code byte at any active breakpoint and substitutes a TRAP instruction for the saved byte. When the TRAP instruction executes, ECM96SA restores the application code byte to its original address and decrements the application program counter to point at the restored instruction. The application code stops executing immediately **before** the instruction with a breakpoint. Two things happen on the screen when a break occurs:

- The prompt changes from a greater-than symbol (>) to an asterisk (*), indicating a halt condition has occurred.
- The target status (shown in the control panel at the top of the console screen) changes from "TARGET STATUS...RUNNING" to "TARGET STATUS...STOPPED".

Many monitor programs similar to ECM96SA display a message on the console when a break occurs (e.g., "program break at 001234H"). However, ECM96SA does not output a special break message. Because the system supports concurrent interrogation of the evaluation board on which the application code is running, a break can occur while you are displaying or modifying the state of the evaluation board. Special break messages interrupt command execution.

5.6.2.2 Breakpoint Commands

Breakpoint commands can display breakpoints while the application code is running or stopped. The commands can activate breakpoints only while the application code is stopped. Table 5-7 lists the breakpoint commands' notations and descriptions.

NOTE

When possible, avoid using BR0 and BR1 with the breakpoint command. The GO command with the TILL option can implicitly set BR0 and BR1 and thereby overwrite the addresses entered with the breakpoint command.

Table 5-7. Breakpoint Command Notations and Descriptions

Command Notations†	Command Descriptions
br <Enter>	Displays all active breakpoints (i.e., ≠ 0) or informs you that no breakpoints are active.
br [*bp_number* = *code_address*] <Enter>	Sets the breakpoint specified by *bp_number* to the value *code_address*. For example, to set breakpoint 3 to the address FF21A0H, type "br3=0ff21a0 <Enter>". (The BR command echoes this address as "21a0"; you can also enter the address FF21A0 as "21a0".) In this example, to clear the breakpoint, you would type "br3 = 0 <Enter>".
br [*bp_number*] <Enter>	Displays a breakpoint value and optionally changes the setting. ECM96SA displays the setting of the selected breakpoint and waits for input. After typing (or not typing) a new value, you can press <Enter> or <Esc>: • <Enter> — Terminates the command. • <Esc> — Displays the next sequential breakpoint. Enter an address value to set the breakpoint or press <Esc> again to display the next breakpoint; the command wraps around from the last breakpoint (15) to the first breakpoint (0).

† The square brackets [ ] indicate an optional argument.

5.6.3 Program Execution Commands

The GO command and its options allow you to start and stop execution at specified addresses. You can execute this command only if the application code is stopped. In addition, a HALT command allows you to stop execution (when the application code is running).

The GO commands that set breakpoints use BP0 and BP1. Any break value already in one of these breakpoints is overwritten by the GO commands. As discussed in "Breakpoint Operation" on page 5-9, program execution stops just **before** execution of the instruction at the breakpoint address. ECM96SA then temporarily deactivates that breakpoint when execution resumes (so the instruction can be executed) and finally reactivates the breakpoint. However, if execution stops at a breakpoint and no other breakpoint is set, the breakpoint is permanently deactivated, and you must use the HALT command to stop the application program.

Table 5-8 lists the GO and HALT commands' notations and descriptions.

Table 5-8. Go and Halt Command Notations and Descriptions

Command Names	Command Notations and Descriptions[1,2]
go	**Notation**: go <Enter> **Description**: Starts application code execution with the current value of the application's program counter (PC) and the current breakpoint array.
	Notation: go [forever] <Enter> **Description**: Clears the breakpoint array and starts execution at the current value of the application's PC.
	Notation: go [from *code_address*] <Enter> **Description**: Loads the application's PC with *code_address* and starts program code execution with the current breakpoint assignments.
	Notation: go [from *code_address* forever] <Enter> **Description**: Loads the application's PC with *code_address*, clears the breakpoint array, and begins program code execution.
	Notation: go [from *code_address1* till *code_address2*] <Enter> **Description**: Loads the application's PC with *code_address1*, sets the first default breakpoint (BP0) to the value of *code_address2*, and then begins program code execution.
	Notation: go [from *code_address1* till *code_address2* or *code_address3*] <Enter> **Description**: Functions like the previous command except that it also sets the second default breakpoint (BP1) to the value of *code_address3*.
	Notation: go [till *code_address*] <Enter> **Description**: Sets the first default breakpoint (BP0) to *code_address* and then begins the program code execution with the current setting of the application's PC and the breakpoint array.
	Notation: go [till *code_address1* or *code_address2*] <Enter> **Description**: Functions like the previous command except that it also sets the second default breakpoint (BP1) to the value of *code_address2*.

Notes:

1. Enter all hexadecimal addresses with a leading zero and no spaces (e.g., "0ff1209h").
2. The square brackets [ ] indicate an optional argument.

Table 5-8. Go and Halt Command Notations and Descriptions

Command Names	Command Notations and Descriptions[1,2]
halt	**Notation**: halt <Enter> **Description**: Stops program code execution by forcing the microcontroller to execute a jump-to-self instruction in a reserved location.

Notes:

1. Enter all hexadecimal addresses with a leading zero and no spaces (e.g., “0ff1209h”).
2. The square brackets [ ] indicate an optional argument.

5.6.4 Program Sequence Control

The ECM96SA interface supports the instruction sequence commands necessary to single-step your application code. These commands are useful for testing and debugging short sections of code. This section defines the commands and certain limitations presented by this type of program flow control.

5.6.4.1 STEP/SUPER-STEP Operation

ECM96SA provides STEP commands for executing code one instruction at a time. SUPER-STEP commands are similar, except they treat subroutines and interrupt service routines (ISRs) as single instructions. Between instructions, you can use ECM96SA commands to check the states of the variables changed by the instruction to ensure that the program is operating properly. STEP commands allow a far more detailed view of program behavior. The disadvantage is that STEP commands do not occur in real time. This restriction makes it difficult or even impossible to use STEP commands with code that is dependent upon real-time events.

In some situations, STEP operations with enabled interrupt systems are confusing because interrupt service routines are also sequenced one instruction at a time. To avoid this problem, ECM96SA artificially locks out interrupts with the basic STEP command operation.

SUPER-STEP is similar to STEP; however, SUPER-STEP interrupts are not artificially suppressed. An interrupt service routine or a subroutine call (and the body of the subroutine it calls) is treated as one indivisible instruction by the SUPER-STEP command. This allows you to ignore the details of subroutines and interrupt service routines while you view code operation. When an instruction uses SUPER-STEP, all service routines associated with enabled pending interrupts are executed. This allows limited stepping through code while operation continues in a concurrent environment; however, the system does not operate in real time. A better approach is to use the GO command to execute to a specified breakpoint and then STEP through the code.

ECM96SA implements the STEP operation by using the TRAP instruction. To STEP over a given instruction, ECM96SA determines the subsequent instruction (or all possible subsequent instructions for a conditional branch) and places a TRAP instruction at these locations. A TRAP is also set at location FF2080H in case the evaluation board is reset during the STEP. ECM96SA allows the application program to execute until the program encounters TRAP locations. ECM96SA then restores all overwritten application code bytes.

A SUPER-STEP operation is similar to a STEP; however, ECM96SA treats the CALL instruction as a special case. During a STEP, ECM96SA puts the TRAP at the evaluation board call address; during a SUPER-STEP, ECM96SA places the TRAP at the instruction following the CALL. When the application's EI bit is saved, it suppresses interrupts during STEP (but not SUPER-STEP); then, ECM96SA restores the interrupt. To ensure that the executed instruction does not modify the EI bit, ECM96SA simulates several instructions (PUSHF, POPF, PUSHA, POPA, DI, EI) as opposed to the microcontroller executing the instructions. ECM96SA also simulates the IDLPD instruction during a STEP to prevent the evaluation board from locking up. The simulation treats the IDLPD as a two-byte NOP. Instruction simulation occurs only with STEP operations. During a GO or a SUPER-STEP operation, the evaluation board executes all instructions.

5.6.4.2 STEP and SUPER-STEP Commands

ECM96SA has four STEP commands and four corresponding SUPER-STEP commands. Aside from the interrupt operation differences discussed earlier, the STEP and SUPER-STEP commands behave the same way, so they are described here together. The command definition uses the phrase "single-step" instead of STEP or SUPER-STEP.

Table 5-9 lists the STEP and SUPER-STEP command notations and descriptions.

Table 5-9. STEP and SUPER-STEP Command Notation and Description

Command Notations†	Command Descriptions
[step | super-step] [-*option1, option2*] <Enter>	Single-steps your application code one instruction at a time.
[step | super-step] [*count*] <Enter>	Single-steps *count* times.
[step | super-step] [from *code_address*] <Enter>	Loads the application's PC with the value of *code_address* and then single-steps one time.
[step | super-step] [from *code_address, count*] <Enter>	Loads the application's PC with the value of *code_address* and then single-steps *count* times.

† The square brackets [ ] indicate an optional argument.

5.7 SUPPORTED DATA TYPES

ECM96SA provides commands to display and modify program variables, including the following data types: BYTE, CHAR, WORD, DWORD, REAL, STACK, and STRING. ECM96SA commands allow you to display variables or to initialize them either individually or as regions of memory that contain variables of the given type. ECM96SA also supports microcontroller variables. You can examine the window select register (WSR); and you can examine and modify the program counter (PC), the program status word (PSW), and the stack pointer (SP).

NOTE

Memory locations 0001E0H–0001FFH are reserved for use by RISMSA. ECM96SA gives a warning if you attempt to modify these memory locations.

Table 5-10 contains definitions for supported data types.

Table 5-10. Supported Data Types

Data Types	Data Type Definitions
byte	A BYTE is an 8-bit variable. No alignment rules are enforced for BYTE variables.
char	A CHAR is a special case of a BYTE. CHAR variables are displayed as ASCII characters.
word	A WORD is a 16-bit variable. The address of a WORD is the address of its least significant byte. A WORD must start at an even byte address.
dword	A DWORD is a 32-bit variable. The address of a DWORD is the address of its least significant byte. A DWORD must start on an address that is evenly divisible by four. This more restrictive alignment rule applies only to ECM96SA commands when the single line assembler is used (see “Single Line Assembler (SLA) Commands” on page 5-18).
real	A REAL is a 32-bit binary floating-point number that conforms to the FPAL-96 definition. The 32 bits contain a sign bit, an 8-bit exponent field, and a 23-bit fraction field. ECM96SA commands use standard scientific notation to represent REAL numbers. Note that FPAL-96 has special representations for +infinity and for NaNs (Not a Number, used to signal error conditions). If ECM96SA detects one of these special values, it outputs an appropriate text string instead of trying to display the value in scientific notation.
stack	A STACK is a 16-bit variable that resides in the system stack. The address of a stack variable (*stack_address*) is relative to the current stack pointer and must be even word aligned.
string	A STRING is a sequence of ASCII characters terminated by the NUL character, which has the binary value of zero.

5.7.1 BYTE, WORD, DWORD, and REAL Commands

ECM96SA has four basic commands to examine and modify BYTE, WORD, DWORD, and REAL variables. There is an additional command for WORD variables only.

Table 5-11 lists the BYTE, WORD, DWORD, and REAL commands' notations and descriptions.

Table 5-11. BYTE, WORD, DWORD, and REAL Command Notations

Command Notations[1,2]	Descriptions
variable [*variable_address*] <Enter>	Examine and possibly modify one or more variables at sequential addresses. ECM96SA displays the hexadecimal address and the value of the variable in the default base. You can then terminate the command, modify the variable, or examine the variable at the next address: • <Enter> — Allows you to terminate the command. • *variable_value* — Assign this value to the variable. Allows you to terminate the command with <Enter> or examine the next variable by pressing <Esc>. • <Esc> — Allows you to examine the next variable. You can then terminate the command (<Enter>), assign a value (variable_value), or examine the next variable (<Esc>).
variable [*variable_address* = *variable_value*] <Enter>	Modify the value of a single variable.
variable [*variable_address* to *variable_address*] <Enter>	Examine the values of the variables in a range of addresses. In numerical form, ECM96SA displays an address followed by up to 16 bytes of memory as BYTE, WORD, DWORD, or REAL values. To stop the output, press <Space>. To resume the output, press <Space> again. To terminate the command press <Enter>.
variable [*variable_address1* to *variable_address2* = *variable_value*] <Enter>	Initialize a region of memory to a given value. At 9600 baud, setting each value takes a little over one millisecond. To terminate the command press <Enter>; this leaves only a part of the memory region initialized.
word [*word_address1* to *word_address2* = *word_address3* to *word_address4*] <Enter>	Copy a block of memory from the second address range to the first address range. This command applies to WORD variables only. To terminate the command, press <Enter>; this leaves only a part of the memory region copied.

1. Replace the variable with BYTE, WORD, DWORD, or REAL (e.g., "word 0ff0080h = 0 <Enter>").
2. The square brackets [ ] indicate an optional argument.

5.7.2 STACK Commands

There are two commands for examining the stack. Both commands can be used whether the application program is running or stopped.

Table 5-12 lists the STACK command's notations and descriptions.

Table 5-12. Stack Command Notations and Descriptions

Command Notations†	Command Descriptions
stack *stack_address* <Enter>	Examine the 16-bit variable at a given offset from the stack pointer. ECM96SA executes a "word *word_address*" command where *word_address* takes the value of the system stack pointer *stack_address*.
stack [*stack_address1* to *stack_address2*] <Enter>	Examine a sequence of 16-bit variables at a fixed offset in the system stack. ECM96SA executes a "word *word_address1* to *word_address2*" command where both *word_address* fields are formed by adding the corresponding *stack_address* to the current value of the system stack pointer. Press <Space> to stop the output for a long display. Press <Space> again to resume output, or press <Enter> to terminate the command.

† The square brackets [ ] indicate an optional argument.

5.7.3 STRING Commands

There is only one form of the STRING command:

string — **Notation**: string *byte_address*

Description: The line begins with a hexadecimal display of *byte_address* followed by the NUL-terminated ASCII string starting at that address. For long strings, only the first 60 characters display. When trailing characters are stripped, decimal points (.) are substituted for the first three characters stripped.

5.7.4 Register Command Variables

You can read microcontroller variables at any time, but you can modify them only while the evaluation board program is stopped. With these commands you can display and load the program counter (PC), program status word (PSW), window select register (WSR), and stack pointer (SP). Display is in the default base.

Use the commands in Table 5-13 to access register variables associated with the microcontroller rather than with the program..

Table 5-13. Register Variable Notations and Descriptions

Register Names	Register Command Notations †
program counter	**Notations**: • pc <Enter> • pc [= *byte_address*] <Enter>
program status word	**Notations**: • psw <Enter> • psw [= *word_value*] <Enter>
window select register	**Notations**: • wsr <Enter> • wsr [= *word_value*] <Enter>
stack pointer	**Notations**: • sp <Enter> • sp [= *word_address*] <Enter>

† The square brackets [ ] indicate an optional argument.

5.7.5 Displaying and Modifying the Stack Pointer (SP)

RISMSA stores two words in the stack pointer area to retain the program counter (PC) and the program status word (PSW) during an ECM96SA host interface interrupt. For this reason, when you display the stack pointer with the SP command or the STACK command, the displayed value is always offset by a value that compensates for the host interrupt overhead. This makes storing the host-interrupt related PC and PSW transparent at the evaluation board command level. However, you must allow for the extra stack space used when calculating total stack space requirements. This transparency is convenient but potentially confusing if you display the stack pointer with the SP command and then either view or directly modify location 18H (the internal register address of the stack pointer). It is recommended that you do not directly modify the stack pointer with internal register address 18H.

CAUTION

To avoid conflict with the evaluation board's stack operations, modify the stack pointer only with the SP command or by executing application code. Do not attempt to directly modify the stack pointer via register address 18H. (Specific implementations of the RISMSA may prevent you from overwriting register 18H and thereby force the use of the SP command.) Always use the SP or STACK command to manipulate the stack pointer.

5.8 ASSEMBLY AND DISASSEMBLY

ECM96SA supports examining and modifying code memory using the standard mnemonics for the MCS® 96 assembler (ASM96). Although standard mnemonics are used, ECM96SA does not build a symbol table of user symbols as assembly mnemonics are entered. This limits the software to operate as a single line assembler (SLA). References are never made to information entered on other lines. The SLA does not generate labels. The ECM96SA SLA accepts mnemonics for all standard instructions that can be executed by the microcontroller. It does not accept "generic" instructions, such as BE or CALL, processed by ASM96 into standard instructions for MCS 96 microcontrollers. Neither does it accept JE, SCALL, or LCALL, which are the specific instructions understood by an MCS 96 microcontroller.

5.8.1 Single Line Assembler (SLA) Commands

The SLA is useful for assembling short code sequences to patch application code as it is tested. These on-line software routines are useful for testing or patching programs, but the tool is not intended as a replacement for a full-featured assembler (such as ASM96) working with an in-circuit emulator. You can invoke the SLA whether application code is being executed or not. It is not recommended that you dynamically modify code executed during your modification session.

Table 5-14 lists the SLA command's notations and descriptions.

Table 5-14. SLA Command Notations and Descriptions

Command Notations†	Command Descriptions
asm [*code_address*] <Enter>	Causes ECM96SA to enter the SLA mode. The assembly program counter (APC) is set to *code_address*. Assembly language code, entered by the user, is converted to object code and loaded into the evaluation board's memory. ECM96SA flags erroneous inputs but remains in the SLA mode. To terminate this mode, type "end <Enter>" (the only directive understood by the SLA).
asm <Enter>	Functions like the "asm *code_address* <Enter>" command except that the APC is not initialized. The first time the SLA is used, APC is set to FF2080H. Otherwise, APC points to the byte following the last instruction generated by the SLA.

† The square brackets [ ] indicate an optional argument.

5.8.2 Disassembly Commands

The disassembler converts binary object code in the evaluation board memory to ASM96 mnemonics. Use these commands for checking program patches or examining a portion of a program for which a listing is not available. You can use these commands whether application code is running or stopped.

Table 5-15 lists the disassembler command's notations and descriptions.

Table 5-15. Disassembler Command Notations and Descriptions

Command Notation†	Command Description
dasm <Enter>	Disassembles the instruction currently pointed to by the application's program counter (APC).
dasm [*count*] <Enter>	Reads the current value of the application's program counter (APC) and disassembles *count* instructions beginning at that location. The parameter *count* must be less than 256T (100H) so the command parser can distinguish this command from the command "dasm *code_address* <Enter>". (This restriction does not apply to the "dasm *code_address*, *count* <Enter>*"* instruction.) During lengthy displays, you can stop the output to the console by pressing <Space> and resume output by pressing <Space> again. Press <Enter> to terminate the command.
dasm [*code_address*] <Enter>	Disassembles the instruction at *code_address*. The parameter *code_address* must be greater than or equal to 256T (100H) so that the command parser can distinguish it from the "dasm *count* <Enter>*"* instruction.
dasm [-*code_address, count*] <Enter>	Disassembles *count* instructions starting with the instruction at *code_address*. During lengthy displays, you can stop the output to the console by pressing <Space> and resume output by pressing <Space> again. Press <Enter> to terminate the command.
dasm [*code_address* to *code_address*] <Enter>	Disassembles the region of memory specified. If an instruction crosses the ending address of the region, it is completely disassembled before the command terminates. During lengthy displays, you can stop the output to the console by pressing <Space> and resume output by pressing <Space> again. Press <Enter> to terminate the command.

† The square brackets [ ] indicate an optional argument.

6

RISM Registers and Commands

CHAPTER 6
RISM REGISTERS AND COMMANDS

This chapter describes the reduced instruction set monitor (RISM). The full RISM command set described in this chapter exists in the external flash on the 80296SA target board. The target board runs this software under normal 80296SA operation.

6.1 RISM REGISTERS

Table 6-1 discusses RISM registers.

Table 6-1. RISM Registers

Registers	Definitions
RISM_DATA	A 32-bit register that acts as the primary data interface between software running in the host (PC) and the RISM running in the target (80296SA).
RISM_ADDR	A 24-bit register that contains the address to be used for reading and writing target memory.
RISM_STAT	An 8-bit register used to store RISM status and state information. This register contains the following Boolean flags: • DLE_FLAG: Indicates the next character received by the RISM should be treated as a data byte even if its value corresponds to an implemented command. • RUN_FLAG: Indicates that the target is running user code. • TRAP_FLAG: Indicates a software TRAP has occurred while running user code suspending its execution.
USER_PC	Saves the user's program counter while the user's code is not executing. Note that program execution must be stopped to use this command.
USER_PSW	Saves the user's program status word while the user's code is not executing.

6.2 RISM STRUCTURE

The RISM resides in the target system. It provides the interface between the target system and the user interface that resides in the host system. It is also compact and simple. This serves two purposes:

- The RISM can reside in a user's system with minimal impact on available memory.
- The RISM is easy to port into the target's environment.

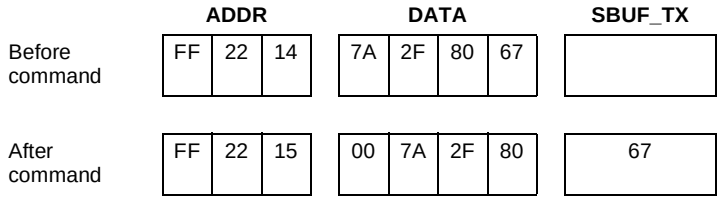
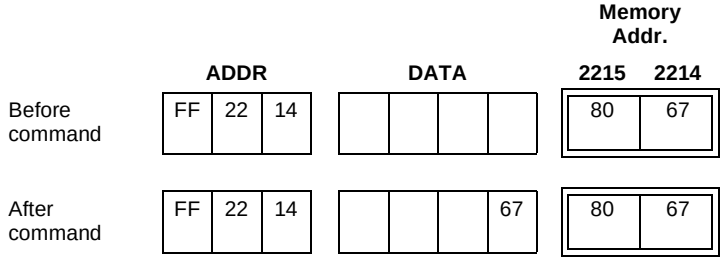
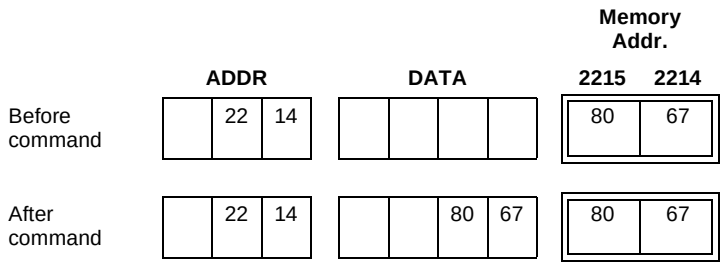
The RISM internal state structure is simple: only three internal flags can change the way RISM deals with a character sent by the host.

- **DLE_FLAG:** When this flag is set, the next received character is assumed to be a data byte as opposed to a command byte.
- **RUN_FLAG:** This flag is set when the target is running user code. It can modify the operation of some RISM commands.
- **TRAP_FLAG:** This flag is set when the user code has been halted because the 80296SA executed a TRAP instruction. The TRAP_FLAG is cleared when the RISM starts the execution of user code.

6.3 RISM COMMAND DESCRIPTIONS

Table 6-2 on page 6-3 details the operation of each command sent to the RISM.

Table 6-2. RISM Command Descriptions

Value	Command	Description
00H	SET_DLE_FLAG	Sets the DLE flag in bit 0 of the MODE register to tell the RISM the next byte on the serial port is data that should be loaded into the DATA register. The flag is cleared as soon as the byte is read.
02H	TRANSMIT	Transmits the low byte of the DATA register to the serial port through the CHAR register, shifts the DATA register right (long) by eight bits, and increments ADDR by one. 
04H	READ_BYTE	Puts the contents of the (byte) memory address pointed to by the ADDR register into the low byte of the DATA register. 
05H	READ_WORD	Puts the contents of the (word) memory address pointed to by the ADDR register into the low byte of the DATA register. 
06H	READ_DOUBLE	Reads the double-word of memory pointed to the address register and places the results in the RISM_DATA register.

<table>
<tr><th>Value</th><th>Command</th><th>Description</th></tr>
<tr><td>07H</td><td>WRITE_BYTE</td><td>Puts the low byte of the DATA register into the memory address pointed to by the ADDR register and increments ADDR by one.

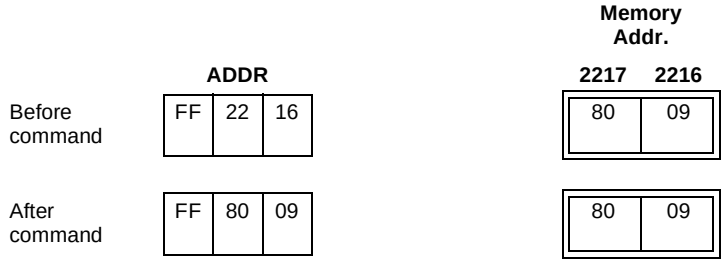
<table>
<tr><th></th><th colspan="3">ADDR</th><th colspan="4">DATA</th><th colspan="2">Memory Addr.
2217 2216</th></tr>
<tr><td>Before command</td><td>FF</td><td>22</td><td>16</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>FF</td><td>FF</td></tr>
<tr><td>After command</td><td>FF</td><td>22</td><td>17</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>FF</td><td>09</td></tr>
</table>

NOTE: To write to an OTPROM location, V_{PP} must be at +12.5 VDC. To write to an internal RAM location, V_{PP} can be at either +5.0 or +12.5 VDC.</td></tr>
<tr><td>08H</td><td>WRITE_WORD</td><td>Puts the low word of the DATA register into the memory address pointed to by the ADDR register and increments ADDR by two.

<table>
<tr><th></th><th colspan="3">ADDR</th><th colspan="4">DATA</th><th colspan="2">Memory Addr.
2217 2216</th></tr>
<tr><td>Before command</td><td>FF</td><td>22</td><td>16</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>FF</td><td>FF</td></tr>
<tr><td>After command</td><td>FF</td><td>22</td><td>18</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>80</td><td>09</td></tr>
</table>

NOTE
To write to an OTPROM location, V_{PP} must be at +12.5 VDC. To write to an internal RAM location, V_{PP} can be at either +5.0 or +12.5 VDC.</td></tr>
<tr><td>09H</td><td>WRITE_DOUBLE</td><td>Stores the RISM_DATA register in the double-word of memory pointed to by the RISM_ADDR register and increments the RISM_ADDR register (by four) to point at the next memory double-word.</td></tr>
<tr><td>0AH</td><td>LOAD_ADDRESS</td><td>Puts the low word of the DATA register into the ADDR register.

<table>
<tr><th></th><th colspan="3">ADDR</th><th colspan="4">DATA</th></tr>
<tr><td>Before command</td><td>FF</td><td></td><td></td><td>F1</td><td>05</td><td>22</td><td>16</td></tr>
<tr><td>After command</td><td>FF</td><td>22</td><td>16</td><td>F1</td><td>05</td><td>22</td><td>16</td></tr>
</table></td></tr>
</table>

Value	Command	Description
0BH	INDIRECT _ADDRESS	Puts the word from the memory address pointed to by the ADDR register into the ADDR register. 
0CH	READ_PSW	Loads the RISM_DATA register with the PSW (Program Status Word) associated with the user's code. Most RISM implementations must check RUN_FLAG to determine how to access the user's PSW.
0DH	WRITE_PSW	Loads the PSW (Program Status Word) associated with the user's code from the RISM_DATA register. The host software will only invoke this command while user code is not running.
0EH	READ_SP	Loads the RISM_DATA register with the SP (Stack Pointer) associated with the user's code.
0FH	WRITE_SP	Loads the SP (Stack Pointer) from the RISM_DATA register. This command also pushes two values into the newly created stack area. These values are the PC (first) and PSW (second) associated with the idle loop which executes while user code is not running. The host software will only invoke this command while user code is not running.
10H	READ_PC	Loads the RISM_DATA register with the PC (Program Counter) associated with the user's code. Most RISM implementations will have to check RUN_FLAG to determine how to access the user's PC.
11H	WRITE_PC	Loads the PC (Program Counter) associated with the user's code from the RISM_DATA register. The host software will only invoke this command while user code is not running.
12H	START_USER	PUSHes the user PC, PSW, and WSR onto the stack and starts the application program from the location contained in the user PC. The RISM PC, PSW, and WSR will also be in the stack, so allow enough room on the stack for all six words. You can interrogate memory locations while your program is running. The RISM interrupts your program to process the command and then returns execution to your program.
13H	STOP_USER	Halts execution of the application program, POPs the user PC, PSW, and WSR from the stack, and PUSHes the RISM PC, PSW, and WSR back onto the stack. The RISM PC contains the location of the Monitor_Pause routine, so the RISM returns to Monitor_Pause.
	TRAP_ISR	A pseudo command that cannot be issued directly by the host software. It is executed when a TRAP instruction is executed. The TRAP instruction is used by ECM to implement software breakpoints and single stepping. On the 80296SA target board, these functions are supported for code execution from on-chip code RAM or the external RAM (cannot insert TRAP into flash).

Value	Command	Description
14H	REPORT _STATUS	Loads a value into the DATA register. This value indicates the status of the application program: **Value** **Status** 00 halted 01 running 02 trapped
15H	MONITOR _ESCAPE	Provides for the addition of RISM commands for special purposes; it uses the RISM_DATA register to extend the command set of the RISM. If the value of the lower 16 bits of the RISM_DATA register is one (RISM_DATA = 0XXXX0001H) then the evaluation board microcontroller should execute either a reset (RST) instruction or a software initialization routine. The basic RISM requires only one of these “extended” commands.

A

Components, Jumpers, and Connectors

APPENDIX A
COMPONENTS, JUMPERS, AND CONNECTORS

This appendix contains physical diagrams showing component and jumper locations. It defines all physical user settings.

Figure A-1 on page A-2 labels the major board components and connectors.

Table A-2 on page A-4 lists the 80296SA evaluation board jumpers.

Figure A-2 on page A-5 shows power supply connector JP5. The flag is oriented toward the outer edge of the board.

Figure A-3 on page A-6 describes LED bank DP2 on the evaluation board.

Table A-3 on page A-7 describes the serial port connector and necessary cable for connecting the evaluation board to the host PC. If your PC has a 25-pin serial port, the wiring for an interface cable is shown in Figure A-4 on page A-8.

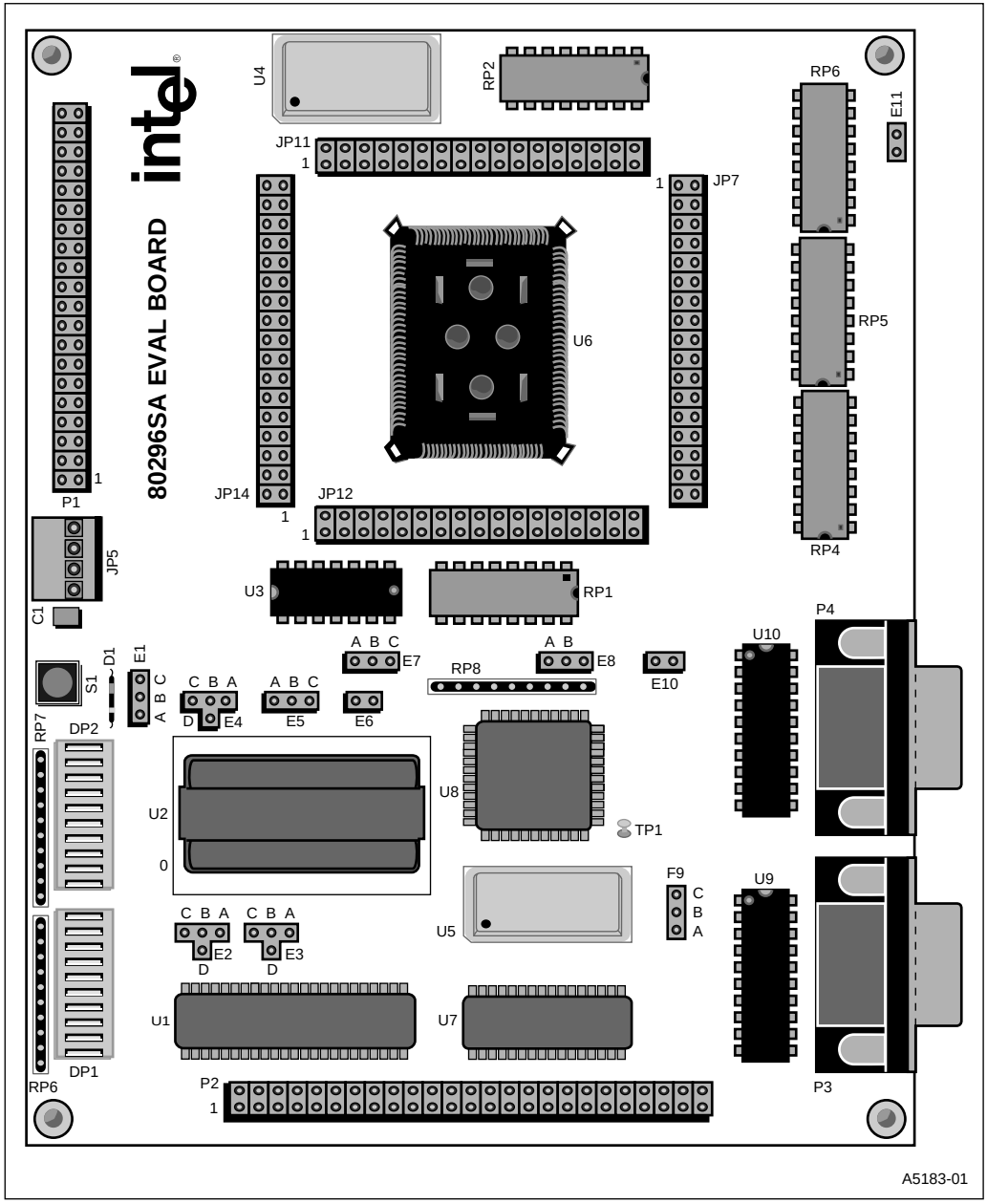


Figure A-1. 80296SA Evaluation Board Diagram

A.1 COMPONENT LIST

Table A-1 lists all 80296SA components.

Table A-1. Components List

Item #	Qty.	Description	Designators
1	1	Jumper	E11
2	1	0.1µF	C12
3	1	0.2µF	C24
4	13	0.1µF	C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C13 C14 C15
5	1	0.1µF	C16
6	4	0.1µF	C17 C18 C19 C20
7	1	0.1µF	C21
8	2	0.1µF	C22 C23
9	1	1.8432 MHz	U5
10	1	1N4148	D1
11	1	1µF	C1
12	2	2.2K	R1 R2
13	2	2.7KΩ SIP	RP6 RP7
14	2	5K	R3 R10
15	5	10K	R5 R7 R8 R9 R8
16	2	15K	R4 R6
17	1	16C550	U8
18	1	50 MHz Canned Osc	U4
19	5	50Ω	RP1 RP2 RP3 RP4 RP5
20	1	74LS04	U3
21	1	80C296SA-QFP	U6
22	4	CON 34	JP7 JP11 JP12 JP14
23	1	CONN 40POS	P1
24	1	CONN 50POS	P2
25	2	DB9 Female	P3 P4
26	2	HLCP-J100	DP1 DP2
27	2	JUMPER 2PIN	E6 E10
28	5	JUMPER 3PIN	E1 E5 E7 E8 E9
29	3	JUMPER 4PIN	E2 E3 E4
30	1	LH521007CK-20	U7

Table A-1. Components List (Continued)

Item #	Qty.	Description	Designators
31	2	MAX233	U9 U10
32	1	PA28F400BV-B	U2
33	1	PWR CONN	JP5
34	1	RESET	S1
35	1	TC55164J	U1
36	1	TEST POINT	TP1

A.2 JUMPER DEFINITIONS

Table A-2 contains the vital 80296SA jumper information.

Table A-2. Jumper Definitions

Jumper Label	Pin Number	Signal Name	Jumper Options
E1	U6:V_{CC}	3.3V/V_{CC}'/V_{CC1}	A-B = 3.3V B-C = V_{CC1}
E2	U2:Pin1	GND/V_{PP}/+12V/V_{CC}	A-B = GND B-C = +12V B-D = Vcc
E3	U2:Pin 2	V_{CC}/WP#/GND/A19	A-B = Vcc (Cutable Trace) B-C = GND B-D = A19
E4	U2:Pin 44	GND/RP#/+12/Vcc	A-B = GND B-C = +12V B-D = Vcc
E5	U2:Pin 43	WR#/WE#/Vcc	A-B = WR# B-C = Vcc
E6	U2:Pin 12	CS0#/CE#	A-B = CS0#
E7	JP7:Pin 30	CS4#/JP7/CS0#	A-B = CS4# B-C = CS0#
E8	U6:Pin 5	GND/EA#/Vcc'	A-B = Vcc' B-C = GND
E9	U6:Pin 43	U9.T2/P4.3/U10.T2	A-B = T2 OF U9 B-C = T2 OF U10
E10	Emulator POWER	Vcc1/Vcc	A-B = Vcc1
E11	U6:Pin 71	RPD/C24	A-B = RPD

A.3 POWER SUPPLY CONNECTOR

Figure A-2 depicts the orientation of the terminals with respect to the evaluation board. The flag on the JP5 connector is oriented to the inside of the board.

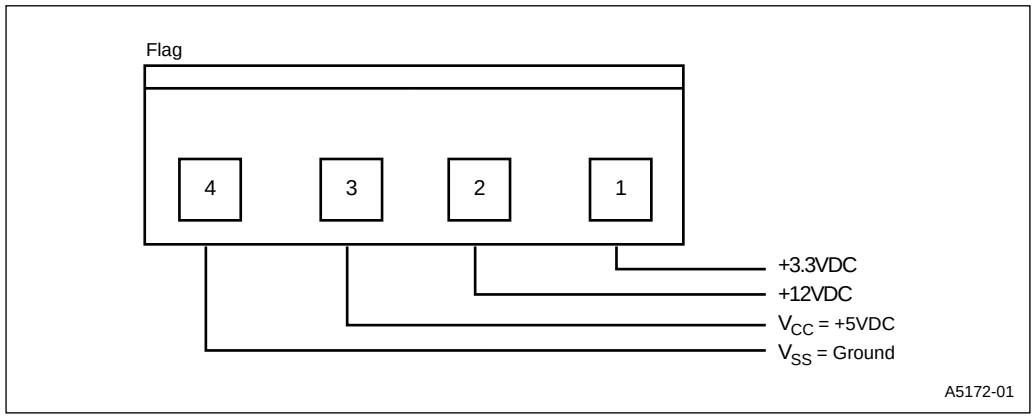


Figure A-2. Power Supply Connector JP5

A.4 LED BANK DESCRIPTIONS

Figure A-3 shows the LED banks DP1 and DP2. At power-on and whenever the board is reset, LEDs 1 through 8 turn on then off together. Then they blink on in sequence continuously until the host PC sends a command to the board, power is turned off, or the board is reset. LED 9 remains off during the entire power-up sequence. In the default configuration, LED 10 stays on continuously to indicate the board has +5 VDC applied. LEDs 1 through 8 can be programmed to display port 1.7:0.

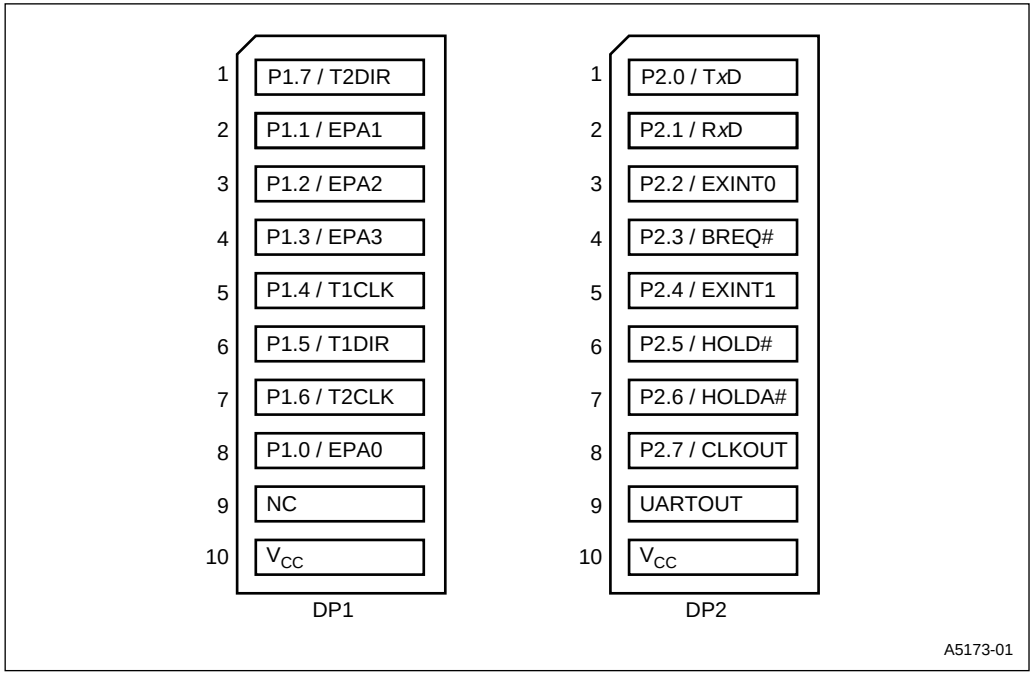
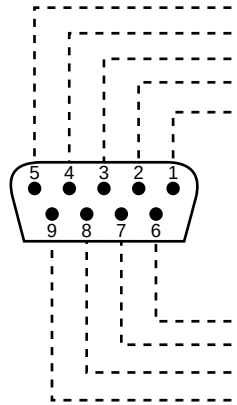


Figure A-3. LED Banks DP1 and DP2

A.5 HOST SERIAL CONNECTOR DESCRIPTION

Table A-3 shows a 9-pin serial connector and the connection of the signals to the 80296SA evaluation board. This connector is a DB-9S RS-232 (DB9); it is included as part of the 80296SA evaluation board kit. The cable must connect all nine signal lines, without loopbacks or crossed wires. The 80296SA evaluation board port P1 is pinned out for a standard IBM-PC/AT* type serial port.

Table A-3. P1 Host Serial Connector

P1 Connector	Pin Nos.	Host RS-232 Signal Name/Function	Connection on Evaluation Board
	5	SG – Signal Ground	V_{SS}
	4	DTR – Data Terminal Ready	INIT
	3	TXD – Transmit Data	RXD
	2	RXD – Receive Data	TXD
	1	DCD – Data Carrier Detect	INIT
	6	DSR – Data Set Ready	INIT
	7	RTS – Request To Send	CTS (pin 8)
	8	CTS – Clear To Send	RTS (pin 7)
	9	RI – Ring Indicator	Run Indicator

If your computer has a 25-pin serial port connector, we recommend you buy a standard RS-232 25-pin to 9-pin conversion adapter or cable. Figure A-4 on page A-8 shows you how to assemble a 25-pin to 9-pin serial port interface adapter cable for the correct connection to the 80296SA evaluation board.

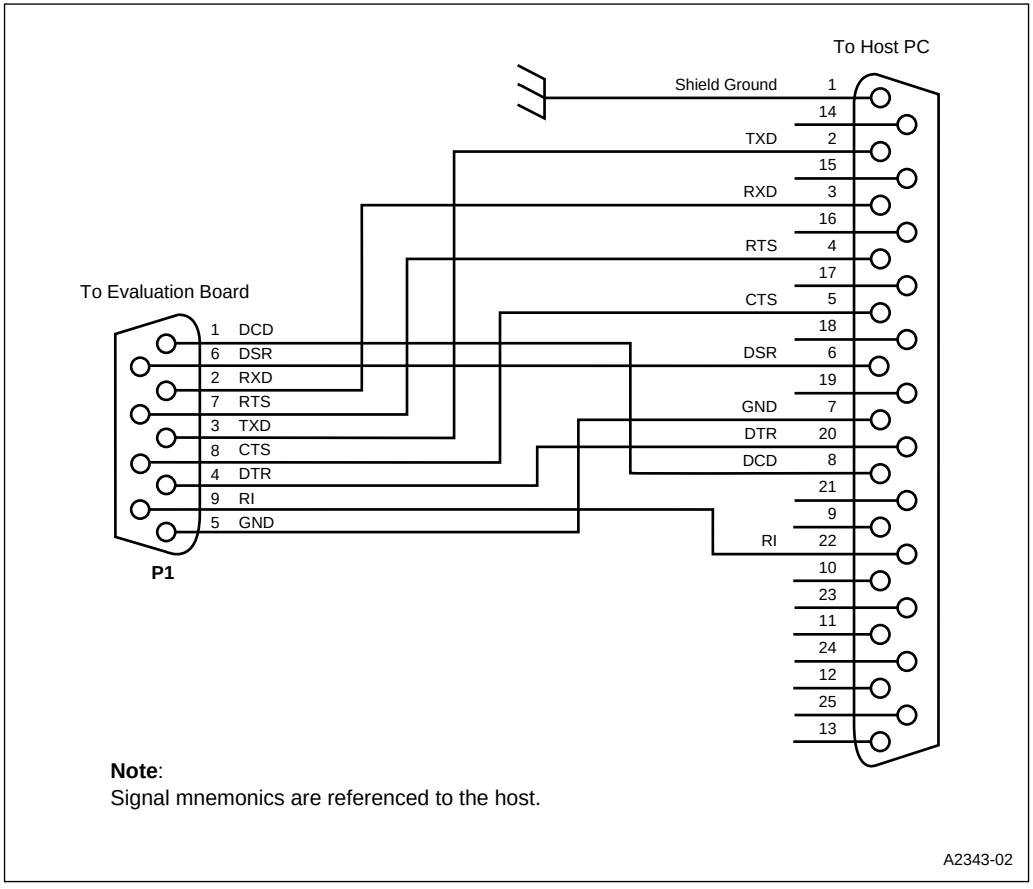


Figure A-4. Serial Interface

Index