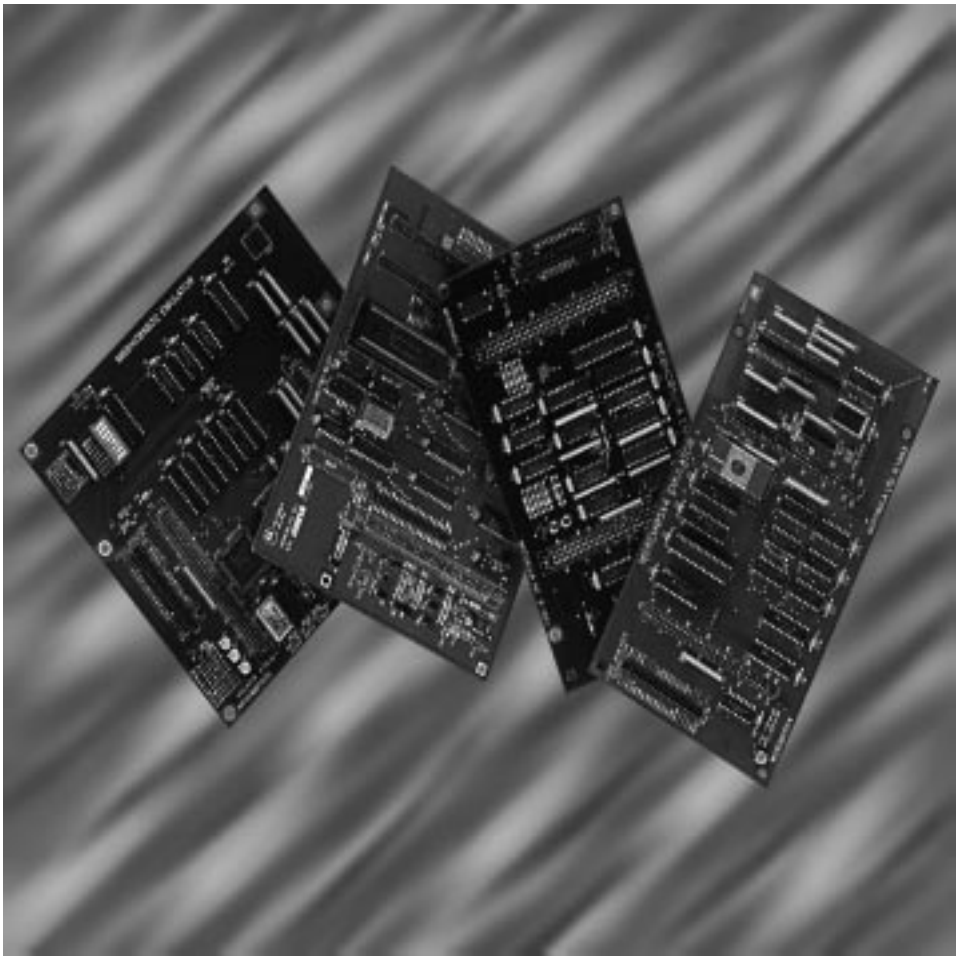
M68HC12B32EVBUM/D
October 1999

M68HC12B32EVB

EVALUATION BOARD
USER'S MANUAL



MOTOROLA

Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, Motorola assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. Motorola further assumes no liability arising out of the application or use of any information, product, or system described herein: nor any liability for incidental or consequential damages arising from the use of this document. Motorola disclaims all warranties regarding the information contained herein, whether expressed, implied, or statutory, *including implied warranties of merchantability or fitness for a particular purpose*. Motorola makes no representation that the interconnection of products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting or license to make, use or sell equipment constructed in accordance with this description.

Trademarks

This document includes these trademarks:

Motorola and the Motorola logo are registered trademarks of Motorola, Inc.

MCUez is a trademark of Motorola, Inc.

Apple, Macintosh, MacTerminal, and System 7 are registered trademarks of Apple Computer, Inc.

Windows and Windows 95 are registered trademarks of Microsoft Corporation in the U.S. and other countries.

Intel is a registered trademark of Intel Corporation.

Sun is a registered trademark of Sun Microsystems, Inc., in the United States and other countries.

Motorola, Inc., is an Equal Opportunity / Affirmative Action Employer.

List of Sections

List of Sections

Table of Contents

Section 1. General Information

Section 2. Configuration and Setup

Section 3. Operation

Section 4. Hardware Reference

Appendix A. S-Record Format

Appendix B. Communications Program Examples

Appendix C. D-Bug12 Startup Code

Appendix D. D-Bug12 Customization Data

Appendix E. EEPROM Bootloader

List of Figures

List of Figures

List of Tables

List of Tables

Section 1. General Information

1.1 Contents

1.2 Introduction

This user's manual provides the necessary information for using the M68HC12B32EVB evaluation board (EVB), an evaluation, debugging, and code-generation tool for the MC68HC912B32 microcontroller unit (MCU).

Reference items, such as schematic diagrams and parts lists, are shipped as part of the EVB package.

1.3 General Description and Features

The EVB is an economical tool for designing and debugging code for and evaluating the operation of the MC68HC912B32 MCU. By providing the essential MCU timing and input/output (I/O) circuitry, the EVB simplifies user evaluation of prototype hardware and software.

General Information

The board consists of a 5.15-inch by 3.4-inch (13.1-cm by 8.64-cm) double-sided printed circuit board (PCB) that provides the platform for interface and power connections to the MC68HC912B32 MCU chip.

Figure 1-1 shows the EVB's layout and locations of the major components, as viewed from the component side of the board.

Hardware features of the low-cost EVB include:

- Double-sided PCB
- Single-supply +3- to +5-Vdc power input (P5)
- RS-232C interface
- BDM (background debug mode) in and BDM out connectors for remote debugging of a user's target system
- Header footprints for access to all MCU pins
- 16-MHz crystal for 8-MHz bus operation
- Headers for jumper selection of and connection to hardware options (for full details of the jumper settings, refer to **Table 4-1**):
  - RS-232 isolation (W1 and W2)
  - EVB mode selection (W3 and W4)
  - MCU mode selection (W5 and W6)
  - V_{PP}/V_{DD} selection (W7)
  - V_{PP} input (W8)
  - BDM in (W9)
  - BDM out V_{DD}/reset disconnects (W10 and W11)
  - BDM out (W12)
  - Low-voltage inhibit (LVI) reset (W15)
  - EXTAL source and access (W16)
- Four 2-row x 20-pin header connectors for access to the MCU's I/O and bus lines (P2, P3, P4, and P6)
- Prototype expansion area for customized interfacing with the MCU
- Low-profile reset push-button switch (S1)
- LVI protection (U3)

Firmware features include:

- D-Bug12 monitor/debugger program, resident in on-chip FLASH EEPROM (electrically erasable programmable read-only memory)
- Full support for either dumb-terminal or host-computer terminal interface
- Single-line assembler/disassembler
- File transfer capability from a host computer to RAM or EEPROM, allowing off-board code generation
- Ability to program EEPROM on either the host EVB or a compatible target system

General Information

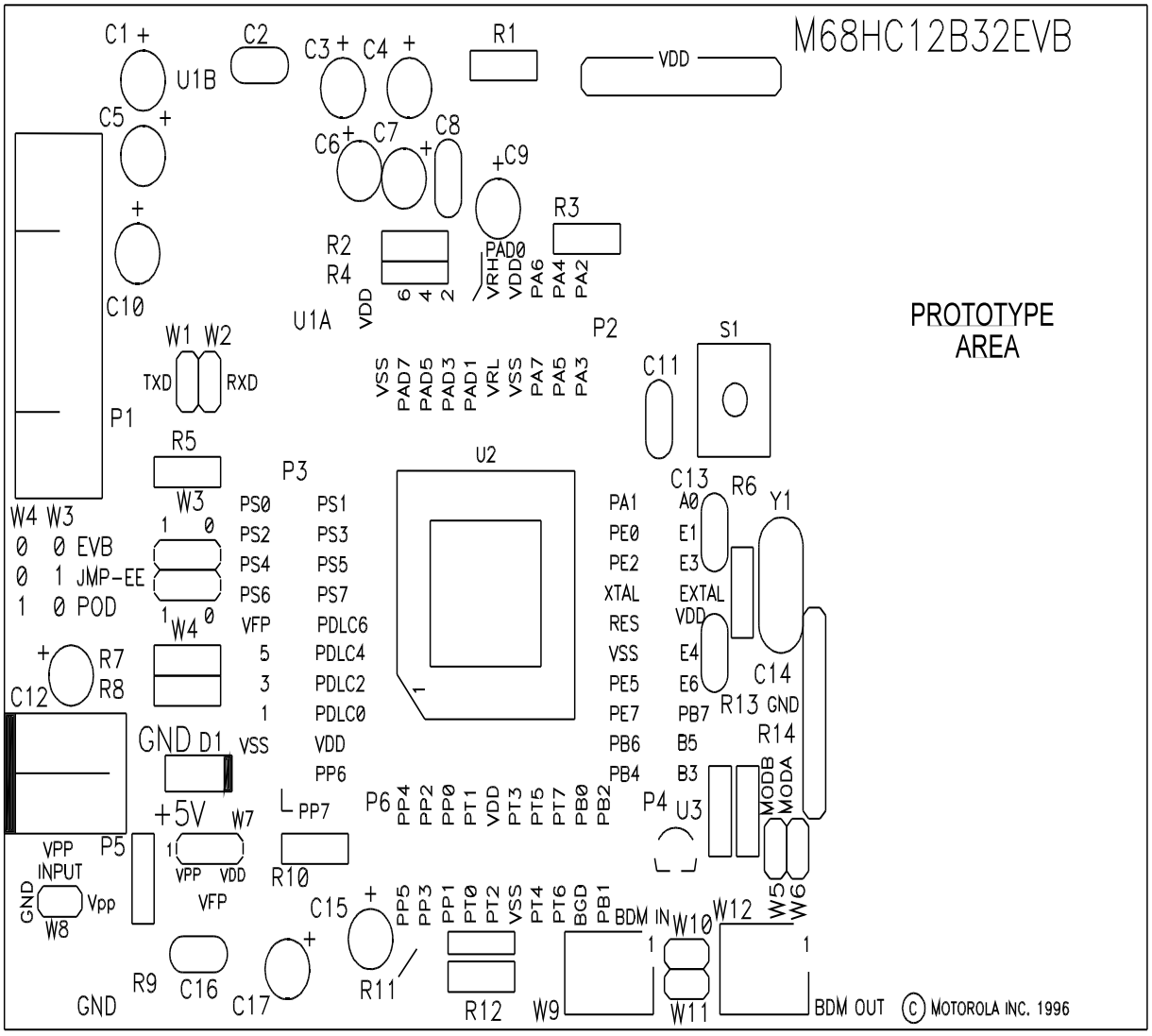


Figure 1-1. EVB Layout and Component Placement

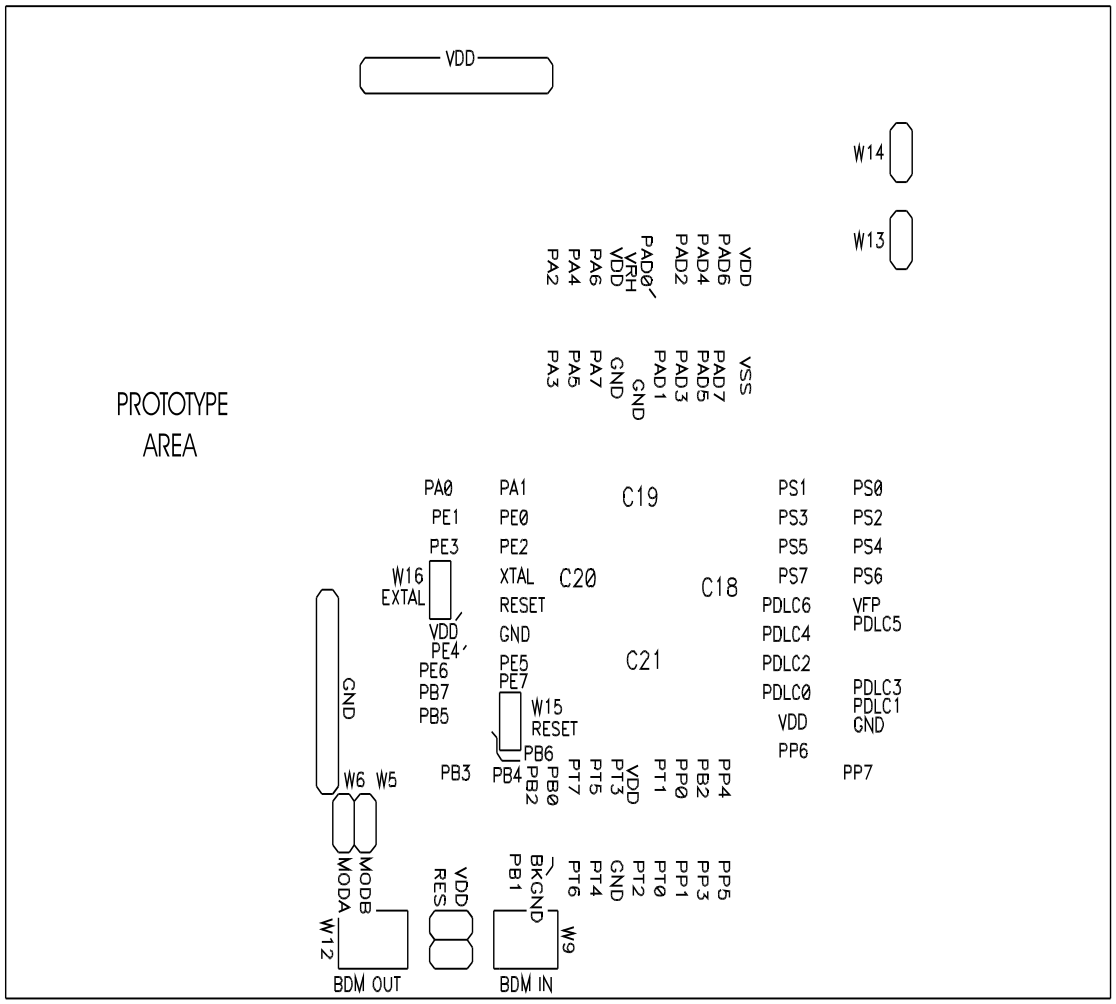


Figure 1-2. EVB Solder Side View

1.4 Functional Overview

The EVB is factory-configured to execute D-Bug12, the FLASH EEPROM-resident monitor program, without further configuration by the user. It is ready for use with an RS-232C terminal for writing and debugging user code. Follow the setup instructions in **Section 2. Configuration and Setup** to prepare for operation.

The EVB also can be jumper-configured to:

- Run a program directly out of EEPROM
- Control a remote “pod” MCU via the BDM interface
- Reprogram EEPROM on either the host EVB or the pod

For the correct jumper settings, refer to **4.3 Configuration Headers and Jumper Settings**.

NOTE: *The D-Bug12 operating instructions in this manual presume the factory-default memory configuration. Other configurations require different operating software arrangements.*

*EEPROM resides in two areas of memory (refer to **Table 3-6. Factory-Configuration Memory Map**), which are referred to in this manual as byte-erasable EEPROM and FLASH EEPROM. This distinction is necessary because of the different ways in which they may be programmed and used.*

D-Bug12 uses the MCU’s serial communications interface (SCI) for communications with the user terminal. For information on the port and its connector, refer to **2.5 EVB to Terminal Connection** and **4.5 Terminal Interface**.

If the MCU’s single-wire background debug mode (BDM) interface serves as the user interface, the SCI port becomes available for user applications. This mode requires either:

- Another M68HC12B32EVB and a host computer
- A background debug development tool, such as Motorola’s serial debug interface (SDI)

For more information, refer to the *SDI™ Interface User’s Manual*, Motorola document order number SDIUM/D.

Two methods may be used to generate EVB user code:

- For small programs or subroutines, D-Bug12’s single-line assembler/disassembler may be used to place object code directly into the EVB’s RAM or EEPROM.

- For larger programs, P&E Microcomputer Systems' IASM12 or Motorola's MCUez™ assembler may be used on a host computer to generate S-record object files, which then can be loaded into the EVB's memory using D-Bug12's LOAD command.

The EVB features a prototype area, which allows custom interfacing with the MCU's I/O and bus lines. These connections are broken out via headers P2, P3, P4, and P6, which are immediately adjacent to the MCU on the board. Wire-wrap pins may be placed in these headers to connect to the prototyping area, as shown in **Figure 1-1**.

An on-board push-button switch, S1, provides for resetting the EVB hardware and restarting D-Bug12.

The EVB can begin operation in any of four jumper-selectable (W3 and W4) modes at reset:

- In EVB mode, program execution begins in one of two ways:
  - If D-Bug 12 is resident in FLASH EEPROM (for example, if bootload mode has not been performed), D-Bug12 immediately issues its command prompt on the terminal display and waits for a user entry.
  - If D-Bug 12 has been replaced in FLASH EEPROM with user code (for example, bootload mode has been performed), execution begins with the user program.
- In jump-EEPROM mode, execution begins directly at location $0D00 with the user code in byte-erasable EEPROM.
- In pod mode, the board makes use of the BDM out header (W12) and uses the D-Bug12 commands to non-intrusively interrogate an external target MCU. Special prompts are displayed to let the user know if this mode is selected. If no external MCU is detected, the software informs the user.

  The target's EEPROM may be programmed while the host M68HC12B32EVB board is in EVB mode, using the D-Bug 12 commands BULK, LOAD, FBULK, and FLOAD.
- In bootload mode, the host EVB's byte-erasable or FLASH EEPROM may be reprogrammed with user code. This mode also may be used to reload or customize D-Bug12.

D-Bug12 allows programming of the MC68HC912B32's on-chip EEPROM through commands that directly alter memory. For full details of all the commands, refer to **3.7 D-Bug12 Command Set**.

When operating in EVB mode, the MCU must manage the EVB hardware and execute D-Bug12 in addition to serving as the user-application processor. There are a few restrictions on its use. For more information, refer to **3.10 Operational Limitations**.

1.5 External Equipment Requirements

In addition to the EVB, the following user-supplied external equipment is required:

- Power supply — See **Table 1-1** for voltage and current requirements.
- User terminal — Options:
  - RS-232C dumb terminal — Allows single-line on-board code assembly and disassembly
  - Host computer with RS-232C serial port — Allows off-board code assembly that can be loaded into the EVB's memory. Requires a user-supplied communications program capable of emulating a dumb terminal. Examples of acceptable communications programs are given in **Appendix B. Communications Program Examples**.
  - Host computer using the MCU's BDM interface — Frees the target MCU's SCI port for user applications. This requires another M68HC12B32EVB for use as the target or a background debug development tool, such as the Motorola serial debug interface (SDI).
- Power-supply and terminal interconnection cables as required

For full details of equipment setup, cabling, and special requirements, refer to **Section 2. Configuration and Setup**.

1.6 EVB Specifications

Table 1-1 lists the EVB specifications.

Table 1-1. EVB Specifications

Characteristic	Specifications
MCU	MC68HC912B32
MCU I/O ports	HCMOS compatible
BDM (in and out)	2-row x 3-pin headers
Communications port	RS-232C DCE port
Power requirements: 16-MHz clock source	 +2.7 to +5.0 Vdc @ 100 mA (max.) For low-voltage operation, refer to **4.9 Low-Voltage Inhibit (LVI)**
Prototype area: Area Holes	 Approximately 1.5 inches x 3 inches (3.8 cm x 7.6 cm) Approximately 15 wide x 31 high, on 0.1-inch (2.54 mm) centers
Board dimensions	5.15 inches x 3.4 inches (13.1 cm x 8.64 cm)

1.7 Typographic Conventions

This user's manual uses special typographical conventions to enhance readability. They are:

- Code, statements, confirmations, data entry, field text, parameters, and strings are indicated in regular Courier:

  ```
  $INCLUDE
  "INIT.AS"
  ```

  This option displays an `Exit Application` confirmation message.

  This new filename replaces the `[NONAME#1]` in the title bar.

  ```
  %FILE%
  ```

- When arguments in code are italicized, they are placeholders for values to be entered by the user:

  <n>
  <argument>

- In code, the user's entry is underlined. This underlining does not actually appear onscreen.

  A typical example looks like this:

`>baud 9600`	User's entry
`Change Terminal BR, Press Return`	D-Bug12's response
`>`	D-Bug12 prompt for next entry

- Window names and parts of windows are indicated in initial caps, unless the name of the window is capitalized in a unique way:

  Memory and Code windows

  CASM08W window

  WinIDE main window

- For usage in this manual, filenames are not case sensitive. But for consistency, they will always appear in all capital letters:

  SETUP.EXE
  MAP file

- Buttons, icons, functions, and keyboard keys are indicated in small caps:

  Press the ENTER key.

  Type CTRL + N or click on the NEW toolbar button.

  The RESET function is an input and output.

- Commands are not case sensitive. But for consistency, they will always appear in all capital letters, unless they contain some peculiarity:

  INPUTx
  UNDO
  LOADMAP

- Menu names, options, and tabs, and dialog, edit, text, and lists boxes are indicated in Times bold:

  Do this by checking the **Main File** option in the **Environment Settings** dialog's **General Options** tab.

  Open the **Open File** dialog.

  Select the filename in the **File Name** list, and use the filename in the **Main filename** edit box.

1.8 Customer Support

To obtain information about technical support or ordering parts, call the Motorola help desk at 800-521-6274.

General Information

Section 2. Configuration and Setup

2.1 Contents

2.2 Unpacking and Preparation

Before beginning configuration and setup of the EVB:

1. Verify that these items are present in the EVB package:
   - M68HC12B32EVB board assembly
   - Background debug mode (BDM) interface cable, 6-pin to 6-pin
   - Warranty and registration cards
   - EVB schematic diagram and parts list
   - *M68HC12B32EVB User's Manual*
   - *MC68HC912B32 Technical Summary*
   - *MC68HC912B32 Electrical Specifications Supplement*
   - *CPU12 Reference Manual*
   - *MC68HC12 Family Brochure*

- *Using D-Bug12 Callable Routines*
- Assembly language development toolset
- P&E Microcomputer Systems' IASM12 assembler and user's manual on 3.5-inch diskette

2. Remove the EVB from its anti-static shipping bag.
3. Save all packing materials for storing and shipping the EVB.

2.3 EVB Configuration

Because the EVB has been factory-configured to operate with D-Bug12, it is not necessary to change any of the jumper settings to begin operating immediately.

As shown in **Table 2-2**, only two jumpers (W3 and W4) should be changed during the course of factory-default EVB operation with D-Bug12.

Table 2-1. EVB Startup Mode Jumpers

Jumper Positions		Startup Mode
W3	W4	
0	0	EVB execution mode (default). D-Bug12 is executed from FLASH EEPROM upon reset. The D-Bug12 prompt appears immediately on the terminal display.
1	0	Jump-EEPROM mode. User code is executed from byte-erasable EEPROM upon reset. For more information, refer to **Section 3. Operation**.
0	1	Remote debugging through BDM out header (W12)
1	1	Bootload mode

Other jumper settings affect the hardware setup and/or MCU operational modes. For an overview of all jumper-selectable functions, refer to **1.3 General Description and Features**. For details of the settings, see **Table 4-1. Jumper-Selectable Functions**.

2.4 EVB to Power Supply Connection

The EVB requires a user-provided external power supply. See **Table 1-1. EVB Specifications** for the voltage and current specifications. For full details of the EVB's power-input circuitry, refer to **4.4 Power Input Circuitry**.

A power supply with current-limiting capability is desirable. If this feature is available on the power supply, set it at 200 mA.

Connect the external power supply to connector P5 on the EVB, using 20 AWG or smaller insulated wire. Strip each wire's insulation 1/4 inch from the end, lift the P5 contact lever to release tension on the contact, insert the bare end of the wire into P5, and close the lever to secure the wire. Observe the polarity carefully.

CAUTION: *Do not use wire larger than 20 AWG in connector P5. Larger wire could damage the connector.*

2.5 EVB to Terminal Connection

For factory-default operation, connect the terminal to P1 on the EVB, as shown in **Table 2-2**. This setup uses the MCU's SCI port and its associated RS-232C interface for communications with the terminal device.

Standard, commercially available cables may be used in most cases. Note that the EVB uses only three of the RS-232C signals. **Table 2-2** lists these signals and their pin assignments. Other signals have been routed through the RS-232C interface chip for proper levels. Some terminal interface programs require proper levels on all pins to function correctly.

The EVB's RS-232C connector, P1, is wired as data circuit-terminating equipment (DCE) and employs a 9-pin subminiature D (DB-9) receptacle.

Most terminal devices — whether dumb terminals or the serial ports on host computers — are wired as data terminal equipment (DTE) and employ 9- or 25-pin subminiature D (DB-9 or DB-25) plugs. In these cases, normal straight-through cabling is used between the EVB and the terminal. Adapters are readily available for connecting 9-pin cables to 25-pin terminal connectors.

If the terminal device is wired as DCE, the RXD and TXD lines must be cross-connected, as shown in **Table 2-2**. Commercial null modem adapter cables are available for this purpose.

Table 2-2. RS-232C Interface Cabling

EVB P1	DTE Signal	Terminal Pins			
DCE Receptacle		DTE[1] Plug		DCE[2] Receptacle	
		DB-9	DB-25	DB-9	DB-25
2	Receive data (RXD)	2	2	3	3
3	Transmit data (TXD)	3	3	2	2
5	Ground (GND)	5	7	5	7

(1) Normal (DCE-to-DTE) cable connections
(2) Null modem (DCE-to-DCE) cable connections

Optionally, the MCU's background debug mode (BDM in, W12) interface can serve as the user interface. This setup makes the SCI port available for user applications. Additional hardware and software are required. For more information, refer to the documentation for the background debug development tool being used. This can be another M68HC12B32EVB or a tool such as Motorola's serial debug interface (SDI).

2.6 Terminal Communications Setup

This section describes how communications among terminals are set up.

2.6.1 Communication Parameters

The EVB's serial communications port uses the communication parameters listed in **Table 2-3**. Of these, only the baud rate can be changed. For instructions on changing it, refer to **2.6.4 Changing the Baud Rate**.

Table 2-3. Communication Parameters

Baud Rate	9600
Data Bits	8
Stop Bits	1
Parity	None

2.6.2 Dumb-Terminal Setup

Configuring a dumb terminal for use with the EVB consists of setting its parameters as shown in **Table 2-3**. Many terminals are configurable with externally accessible switches, but the procedure differs between brands and models. Consult the manufacturer's instructions for the terminal being used.

2.6.3 Host-Computer Setup

One advantage of using a host computer as the EVB's terminal is the ability to generate code off-board, for subsequent loading into the EVB's memory. It is thus desirable for the host to be capable of running programs such as P&E Microcomputer Systems' IASM12 or Motorola's MCUez assembler. For more information, see **3.8 Off-Board Code Generation**.

To serve as the EVB's terminal, the host computer must have an RS-232C serial port and an installed communications program capable of operating with the parameters listed in **Table 2-3**.

Setting up the parameters is normally done within the communications program, after it has been started on the host. Usually, the setup can be saved in a configuration file so that it does not have to be repeated. Because procedures vary between programs, consult the user's guide for the specific program. **Appendix B. Communications Program Examples** provides examples of using some of the commonly available communications programs.

2.6.4 Changing the Baud Rate

The EVB's default baud rate for the RS-232C port is 9600. This can be changed in two ways:

- For temporary changes, use the D-Bug12 BAUD command. This change remains in effect only until the next reset or power-up, when the baud rate returns to 9600.
- For permanent changes, the D-Bug12 baud-rate initialization value stored in FLASH EEPROM must be modified. For instructions, refer to **Appendix D. D-Bug12 Customization Data** and **Appendix E. EEPROM Bootloader**.

Section 3. Operation

3.1 Contents

3.2 Operating Modes

The EVB can operate in one of four jumper-selectable modes:

- EVB mode — Either D-Bug12 or the user code in FLASH EEPROM executes.
- Jump-EEPROM mode — User code in byte-erasable EEPROM executes.
- Pod mode — D-Bug12 executes. EVB serves as the BDM probe for a target system.
- Bootload mode — The host EVB's EEPROM may be reprogrammed.

The operating mode is determined by jumper headers W3 and W4, as shown in **Table 4-1. Jumper-Selectable Functions**. The modes are described in the following three subsections.

NOTE: *When operating in EVB mode, the M68HC12B32EVB cannot fully emulate a target system. The limitations are described in **3.10 Operational Limitations**. Target system emulation may, however, be performed by using the EVB with D-Bug12 as an intelligent, non-intrusive BDM interface. This operation is described in **3.2.3 Pod (Probe) Mode**.*

3.2.1 EVB Mode

In the default EVB mode (W3-0 and W4-0), D-Bug12 begins execution immediately. The D-Bug12 prompt appears on the terminal and commands may be entered as described in **3.6 Using D-Bug12 Commands**.

If D-Bug12 has been replaced with user code in FLASH EEPROM, execution begins with the user's program.

3.2.2 Jump-EEPROM Mode

In this mode (W3-1 and W4-0), the EVB begins operation out of reset by executing the user program in byte-erasable EEPROM starting at address $0D00, as shown in **Table 3-6**.

This mode is made using the MCU's PAD0 line. User code may be programmed into byte-erasable EEPROM using the D-Bug12 commands listed in **3.6.3 Command Set Summary**.

Control can be returned to D-Bug12 two ways:

- Move the jumpers on headers W3 and W4 to position 0 and reset the EVB.
- Terminate the user program with code that returns to D-Bug12 after execution has finished.

To return to D-Bug12 automatically after a user program has finished, include these lines as the last instructions to be executed in the program:

```
STACKTOP:      equ   $0c00               ; stack at top
                                         ; of on-chip RAM
AltResetVect:  equ   $F7FE
                                         ;
               lds   #STACKTOP
               jmp   [AltResetVect,PCR] ; jump to start
                                         ; of D-Bug12
```

3.2.3 Pod (Probe) Mode

In this mode (W3-0 and W4-1), the EVB and D-Bug12 serve as a pod (probe) interface between a target system and the user. Communications between the EVB and the target are by means of the background debug mode (BDM) interface, using the EVB header W12 (BDM out).

This arrangement allows the target system to perform true emulation of an application, as the BDM interface is non-intrusive upon the target's foreground operation. The target's on-chip resources are all available for the application. The target may be a second M68HC12B32EVB board or any other M68HC12 system. D-Bug12 commands are entered as usual on the user terminal, which is served by the pod EVB.

3.2.3.1 Target Types Supported

All members of the M68HC12 Family may be used in the target system.

3.2.3.2 Target MCU Characteristics

These D-Bug12 commands must be used to inform D-Bug12 of the target MCU's essential operating characteristics to allow transparent modification of the target's EEPROM. For details, refer to the command descriptions in **3.7 D-Bug12 Command Set**.

DEVICE — Specifies the target's microprocessor type

EEBASE — Specifies the base address of the target's FLASH EEPROM

REGBASE — Specifies the base address of the target MCU's input/output (I/O) registers

3.2.3.3 Programming the Target's EEPROM

The target MCU's on-chip byte-erasable or FLASH EEPROM may be programmed from user-assembled S records on the host (terminal) computer by using the D-Bug12 commands BULK, LOAD, FBULK, and FLOAD. For details, refer to **3.7 D-Bug12 Command Set**.

3.2.3.4 Controlling Target Execution

All D-Bug12 commands that control the execution of user code may also be used in both EVB mode and pod mode. Two additional commands are available in pod mode:

- RESET — Resets the target MCU and places it in active background mode
- STOP — Halts program execution on the target

For details, refer to the command descriptions in **3.7 D-Bug12 Command Set**.

3.2.4 Bootload Mode

In this mode (W3-1 and W4-1), a user program may be loaded into the host EVB's byte-erasable or FLASH EEPROM. D-Bug12 may be replaced as the startup boot program. This mode also may be used to reload or customize D-Bug12. The procedures are described in **Appendix E. EEPROM Bootloader**.

3.3 Startup

To operate the M68HC12B32EVB, follow the startup procedure described here.

3.3.1 Startup Procedure

This startup procedure includes a checklist of configuration and setup items from **Section 2. Configuration and Setup**. To begin operating the M68HC12B32EVB, follow these steps:

1. Configure the EVB, if required. See **2.3 EVB Configuration**.
2. Determine whether execution should begin in EVB mode (**3.2.1 EVB Mode**), jump-EEPROM mode (**3.2.2 Jump-EEPROM Mode**), pod mode (**3.2.3 Pod (Probe) Mode**), or bootload mode (**3.2.4 Bootload Mode**). Set the jumpers on headers W3 and W4 accordingly. See **2.3 EVB Configuration** and **3.2 Operating Modes**.
3. Connect the EVB to the external power supply. See **2.4 EVB to Power Supply Connection**.
4. Connect the EVB to the terminal. See **2.5 EVB to Terminal Connection**.
5. Configure the terminal communications interface. See **2.6 Terminal Communications Setup**.
6. Apply power to the EVB and to the terminal. If the terminal is a host computer:
   a. Verify that it has booted correctly.
   b. Start the communications program for terminal emulation. See **2.6.3 Host-Computer Setup** and **Appendix B. Communications Program Examples**.
7. Reset the EVB by pressing and releasing the on-board reset switch (S1).

3.3.2 Operating Procedures

After starting the EVB in accordance with **3.3.1 Startup Procedure**, follow the operating procedure for the EVB mode that was selected: EVB, jump-EEPROM, pod, or bootload mode. These procedures are described in the following subsections.

3.3.2.1 EVB Mode

In EVB mode, the MC68HC912B32 begins executing code at the address contained in the alternate reset vector at $F7FE (for information on the alternate reset and interrupt vector table, see **E.4 Vector Jump Table: Interrupt and Reset Addresses**). The code pointed to by the alternate reset vector may either be D-Bug12 (factory default) or a user's program that has replaced D-Bug12 in FLASH EEPROM.

- **D-Bug12** — Upon reset, the D-Bug12 sign-on banner and prompt should appear on the terminal's display like this:

  ```
  D-Bug12 v 2.0.0
  Copyright 1996 - 1997 Motorola Semiconductor
  For Commands type "Help"
  >
  ```

  If the prompt does not appear, check all connections and verify that startup steps 1 through 7 in **3.3.1 Startup Procedure** have been performed correctly.

  When the prompt appears, D-Bug12 is ready to accept commands from the terminal as described in **3.6 Using D-Bug12 Commands**.

- **User boot program** — Upon reset, the user program executes immediately. D-Bug12 commands are not available. Terminal communications take place either via the SCI under control of the user program or via the BDM interface and a serial debug interface tool such as Motorola's SDI.

3.3.2.2 Jump-EEPROM Mode

In jump-EEPROM mode, the user code in byte-erasable EEPROM starting at address $0D00 is executed immediately. Terminal communications are controlled by the user code via the SCI or by an appropriate serial debug tool via the BDM interface. For more information, refer to **3.2.2 Jump-EEPROM Mode**. Control can be returned to the D-Bug12 terminal prompt by doing one of the following:

- Terminating the user code with appropriate instructions; see **3.2.2 Jump-EEPROM Mode**
- Pressing the reset button (S1)

3.3.2.3 Pod Mode

In pod mode, the host EVB serves as a non-intrusive controller for the target system via the BDM interface. The host EVB begins executing code at the address contained in the alternate reset vector at $F7FE. (For information on the alternate reset and interrupt vector table, see **E.4 Vector Jump Table: Interrupt and Reset Addresses**.) The code pointed to by the alternate reset vector may either be D-Bug12 (factory default) or a user's program that has replaced D-Bug12 in FLASH EEPROM.

- **D-Bug12** — Upon power-up or reset, D-Bug12 attempts to establish communications with a target system connected to BDM out (W11). Communications are first attempted without resetting the target system. If communications cannot be established, this message is displayed:

  ```
  Can't Communicate With The Target Processor
  To reset target, hit any key...
  ```

  Pressing any key on the terminal's keyboard causes D-Bug12 to assert the target's reset pin for approximately 2 ms and to try again to establish communications. If communications fail, the error message is redisplayed. Once communications have been established with the target system, the D-Bug12 sign-on banner and prompt should appear on the terminal's display like this:

  ```
  D-Bug12 v 2.0.0
  Copyright 1996 - 1997 Motorola Semiconductor
  For Commands type "Help"
  S>
  ```

If communications cannot be established with the target system after repeated attempts, check for these possible problems:

– The host EVB's BDM out (W11) must be properly connected to the target system's BDM connector. If the target system is another EVB, make sure that the host EVB's BDM out is connected to target EVB's BDM in (W9).

– If the target system is not another EVB, verify that its BDM connector is wired to the proper MCU signals on each pin.

– If the target MCU does not have any firmware to execute, it could "run away," possibly executing a STOP opcode and preventing BDM communications with the host EVB.

Thus, it is *strongly* recommended that, if the target system does not have firmware to execute at reset, the target MCU initially be configured to begin operation in special single-chip mode. Resetting the target MCU in special single-chip mode places it in active background mode. See the target MCU's technical summary for details on setting the MCU operating mode.

Special D-Bug12 command-line prompts indicate the status of the target system:

```
S>          target is in active background mode
R>          target is running a user program
```

In addition to the normal D-Bug12 commands that control execution of user code, the RESET and STOP commands are available in pod mode. These commands are described in **3.7 D-Bug12 Command Set**.

D-Bug12 must be informed of the target MCU's basic operating parameters. Refer to **3.2.3 Pod (Probe) Mode** for more information about setting up and using pod mode.

- **User boot program** — Upon reset, the user program executes immediately. D-Bug12 commands are not available. Communications with the user terminal and with the target system are controlled by the user program.

3.3.2.4 Bootload Mode

In bootload mode, a user program may be loaded into the host EVB's byte-erasable or FLASH EEPROM. If the user code replaces D-Bug12 in FLASH EEPROM, it serves as the "boot" program when the EVB is restarted in EVB or pod mode. This procedure is described in **Appendix E. EEPROM Bootloader**.

3.4 Reset

EVB operation can be restarted at any time by activating the hardware reset function. Do this in one of two ways:

- Press and release the on-board reset switch, S1 (always applicable).
- Activate the external reset input if one has been provided for operation below 3.0 Vdc.

Note that the EVB's reset circuitry is associated with the low-voltage inhibit (LVI) protection. For more information, refer to **4.8 Reset** and **4.9 Low-Voltage Inhibit (LVI)**.

3.5 Aborting a User Program

When operating in EVB mode, the only way to recover from an erroneous or runaway user program is to press the reset switch (S1). If this becomes necessary, the jumpers on headers W3 and W4 should be set to execute D-Bug12 at reset instead of the flawed user program.

When operating in pod mode, the D-Bug12 RESET or STOP command can be used to regain control of the target system.

3.6 Using D-Bug12 Commands

D-Bug12, the EVB's firmware-resident monitor program, provides a self-contained operating environment that allows writing, evaluation, and debugging of user programs.

3.6.1 Command-Line Prompt

D-Bug12 displays one of three command-line prompts, depending upon its operating mode and/or the state of the target system. When D-Bug12 is operating in EVB mode, it displays the greater than (>) single character at the beginning of a line when it is waiting for the user to enter a command. When a command is issued that causes user code to run, D-Bug12 places the terminal cursor on a blank line, where it remains until control returns to D-Bug12.

When operating in pod mode, D-Bug12 displays one of two prompts, depending upon the state of the attached target system. When the target system is in active background mode (not running a user program), the 2-character prompt S> is displayed, indicating that the target is stopped and not running a user program. When the target system is running a user program, the 2-character prompt R> is displayed, indicating that the target is running a user program.

Because the M68HC12 background debug mode interface allows the reading and writing of target system memory even when the target is running a user's program, the probe microcontroller is always available for the entry of commands. D-Bug12 commands that examine or modify target system memory may be issued when either the S> or R> prompt is displayed.

3.6.2 Entering Commands

Commands are typed on the terminal's D-Bug12 prompt line and executed when the carriage-return ENTER key is pressed. D-Bug12 then displays either the appropriate response to the command or an error indication.

Operation

The D-Bug12 command-line prompt is the greater than sign (>). Type the command and any other required or optional fields immediately after the prompt, like this:

Command-Line Syntax:

<*command*> [<*parameter*>] ...[<*parameter*>]ENTER

Where:

<*command*>	is the command mnemonic.
<*parameter*>	is an expression or address.
ENTER	is the terminal keyboard's carriage-return or ENTER key.

- The command-line syntax is illustrated using the following special characters for clarification. Do not type these characters on the command line:

< >	required syntactical element
[ ]	optional field
...[ ]	repeated optional fields

- Fields are separated by any number of space characters.
- All numeric fields, unless noted otherwise, are interpreted as hexadecimal.
- Command-line entries are case-insensitive and may be typed using any combination of upper- and lower-case letters.
- A maximum of 80 characters, including the terminating carriage return, may be entered on the command line. After the 80th character, D-Bug12 automatically terminates the command-line entry and processes the characters entered to that point.
- Before the ENTER key is pressed, the command line may be edited using the BACKSPACE key. Receiving the backspace character causes D-Bug12 to delete the previously received character from its input buffer and erase the character from the display.

3.6.3 Command Set Summary

Table 3-1 summarizes the D-Bug12 commands. For detailed descriptions of each command, refer to 3.7 D-Bug12 Command Set.

Table 3-1. D-Bug12 Command Set Summary

Command	Description
ASM <address>	Single-line assembler/disassembler
BAUD <BAUDRate>	Set the SCI communications baud rate
BF <StartAddress><EndAddress> [<Data>]	Block fill user memory with data
BR [<Address><Address>...]	Set/display user breakpoints
BULK	Bulk erase on-chip EEPROM
CALL [<Address>]	Execute a user subroutine; return to D-Bug12 when finished
DEVICE	Select/define a new target MCU device
EEBASE <Address>	Inform D-Bug12 of the target's EEPROM base address
FBULK	Erase the target processor's on-chip FLASH EEPROM
FLOAD <AddressOffset>	Program the target processor's on-chip FLASH EEPROM from S-records
G [<Address>]	Go — Begin execution of user program
GT <Address>	Go Till — Set a temporary breakpoint and begin execution of user program
HELP	Display D-Bug12 command set and command syntax
LOAD [<AddressOffset>]	Load user program in S-record format*
MD <StartAddress> [<EndAddress>]	Memory Display — Display memory contents in hex bytes/ASCII format
MDW <StartAddress> [<EndAddress>]	Display Memory as 16-Bit Word — Display memory contents in hex words/ASCII format
MM <Address> [<data>]	Memory Modify — Interactively examine/change memory contents
MMW <address> [<data>]	Modify 16-Bit Memory Word — Interactively examine/change memory contents
MOVE <StartAddress> <EndAddress> <DestAddress>	Move a block of memory
NOBR [<Address> <Address>...]	Remove individual user breakpoints

Table 3-1. D-Bug12 Command Set Summary (Continued)

Command	Description
RD	Register Display — Display the CPU register contents
REGBASE	Inform D-Bug12 of the target I/O register's base address
RESET	Reset the target CPU
RM	Register Modify — Interactively examine/change CPU register contents
STOP	Stop execution of user code on the target processor and place it in background mode
T [<*Count*>]	Trace — Execute an instruction, disassemble it, and display the CPU registers
UPLOAD <*StartAddress*> <*EndAddress*>	Display memory contents in S-record format*
VERF [<*AddressOffset*>]	Verify memory contents against S-record data
<*RegisterName*> <*RegisterValue*>	Set CPU <*RegisterName*> to <*RegisterValue*>

* Refer to **Appendix A. S-Record Format** for S-record information.

3.7 D-Bug12 Command Set

In the following command descriptions, the examples represent what is seen on the terminal display.

NOTE: *For clarity, the user's entry is underlined. This underlining does not actually appear onscreen.*

A typical example looks like this:

```
>baud 9600                                  User's entry.
Change Terminal BR, Press Return            D-Bug12 response.
>                                           D-Bug12 prompt for
                                            next entry.
```

Assemble Instructions ASM

Syntax: ASM *<Address>*

Where:

<Address> is a 16-bit hexadecimal number.

The assembler/disassembler is an interactive memory editor that allows memory contents to be viewed and altered using assembly language mnemonics. Each entered source line is translated into object code and placed into memory at the time of entry. When displaying memory contents, each instruction is disassembled into its source mnemonic form and displayed along with the hexadecimal object code and any instruction operands.

Assembler mnemonics and operands may be entered in any mix of upper- and lower-case letters. Any number of spaces may appear between the assembler prompt and the instruction mnemonic or between the instruction mnemonic and the operand. Numeric values appearing in the operand field are interpreted as *signed* decimal numbers. Placing a $ (dollar sign) in front of any number will cause the number to be interpreted as a hexadecimal number.

When an instruction is disassembled and displayed, the D-Bug12 prompt is displayed following the disassembled instruction. If a carriage return is the first non-space character entered following the prompt, the next instruction in memory is disassembled and displayed on the next line.

If a CPU12 instruction is entered following the prompt, the entered instruction is assembled and placed into memory. The line containing the new entry is erased and the new instruction is disassembled and displayed on the same line. The next instruction location is then disassembled and displayed on the screen.

The instruction mnemonics and operand formats accepted by the assembler follow the syntax as described in the *CPU12 Reference Manual,* Motorola document order number CPU12RM/AD.

ASM Assemble Instructions (Continued)

A number of M68HC11 instruction mnemonics appear in the *CPU12 Reference Manual* that do not have directly equivalent CPU12 instructions. These mnemonics, listed in **Table 3-2**, are translated into functionally equivalent CPU12 instructions. To aid the current M68HC11 users who may desire to continue using the M68HC11 mnemonics, the disassembler portion of the assembler/disassembler recognizes the functionally equivalent CPU12 instructions and disassembles those instructions into the equivalent M68HC11 mnemonics.

When entering branch instructions, the number placed in the operand field should be the absolute destination address of the instruction. The assembler calculates the two's-complement offset of the branch and places the offset in memory with the instruction.

The assembly/disassembly process may be terminated by entering a period (.) as the first non-space character following the assembler prompt.

Table 3-2. M68HC11 to CPU12 Instruction Translation

M68HC11 Mnemonic	CPU12 Instruction
CLC	ANCC # $FE
CLI	ANCC # $EF
CLV	ANCC # $FD
SEC	ORCC # $01
SEI	ORCC # $10
SEV	ORCC # $02
ABX	LEAX B, X
ABY	LEAY B, Y
DES	LEAS –1, S

M68HC11 Mnemonic	CPU12 Instruction
INS	LEAS 1, S
TAP	TFR A, CC
TPA	TFR CC, A
TSX	TFR S, X
TSY	TFR S, Y
XGDX	EXG D, X
XGDY	EXG D, Y
SEX R_8, R_{16}	TFR R_8, R_{16}

Restrictions: None

Assemble Instructions (Continued) ASM

Example:

```
>ASM 800

0800 CC1000          LDD    #$1000
0803 1803123401FE    MOVW   #$1234,$01FE
0809 0EF9800001F1    BRSET  -32768,PC,$01,$0700
080F 18FF            TRAP   $FF
0811 183FE3          ETBL   <Illegal Addr Mode>
>.
>
```

Assembly Operand Format:

This section describes the operand format used by the assembler when assembling CPU12 instructions. The operand format accepted by the assembler is described separately in the *CPU12 Reference Manual*. Rather than describe the numeric format accepted for each instruction, some general rules are used. Exceptions and complicated operand formats are described separately.

In general, anywhere the assembler expects a numeric value in the operand field, either a decimal or hexadecimal value may be entered. Decimal numbers are entered as signed constants having a range of –32,768 to 65,535. A leading minus sign (–) indicates negative numbers; the absence of a leading minus sign indicates a positive number. A leading plus sign (+) is not allowed. Hexadecimal numbers must be entered with a leading dollar sign ($) followed by one to four hexadecimal digits. The default number base is decimal.

For all branching instructions (Bcc, LBcc, BRSET, BRCLR, DBEQ, DBNE, IBEQ, IBNE, TBEQ, and TBNE), the number entered as the branch address portion of the operand field is the absolute address of the branch destination. The assembler calculates the two’s-complement offset to be placed in the assembled object code.

Disassembly Operand Format:

The operand format used by the disassembler is described separately in the *CPU12 Reference Manual*. Rather than describing the numeric format used for each instruction, some general rules are applied. Exceptions and complicated operand formats are described separately.

All numeric values disassembled as hexadecimal numbers are preceded by a dollar sign ($) to avoid being confused with values disassembled as signed decimal numbers.

ASM Assemble Instructions (Continued)

For all branching instructions (Bcc, LBcc, BRSET, BRCLR, DBEQ, DBNE, IBEQ, IBNE, TBEQ, and TBNE), the numeric value of the address portion of the operand field is displayed as the hexadecimal absolute address of the branch destination.

All offsets used with indexed addressing modes are disassembled as signed decimal numbers.

All addresses, whether direct or extended, are disassembled as 4-digit hexadecimal numbers.

All 8-bit mask values (BRSET/BRCLR/ANDCC/ORCC) are disassembled as 2-digit hexadecimal numbers.

All 8-bit immediate values are disassembled as hexadecimal numbers.

All 16-bit immediate values are disassembled as hexadecimal numbers.

Set Baud Rate — BAUD

Syntax: BAUD <*BAUDRate*>

Where:

<*BAUDRate*> is an unsigned 16-bit decimal number.

The BAUD command is used to change the communications rate of the SCI used by D-Bug12 for the terminal interface.

Restrictions: Because the <*BAUDRate*> parameter supplied on the command line is a 16-bit unsigned integer, baud rates greater than 65,535 baud cannot be set using this command. The SCI baud rate divider value for the requested baud rate is calculated using the E-clock value supplied in the customization data area. Because the SCI baud rate divider is a 13-bit counter, certain baud rates may not be supported at particular E-clock frequencies. If the value calculated for the SCI's baud rate divider is equal to 0 or greater than 8191, command execution is terminated and the communications baud rate is not changed.

Example:

```
>BAUD 50
Invalid BAUD Rate

>BAUD 38400
Change Terminal BR, Press Return

>
```

BF Block Fill

Syntax: BF <*StartAddress*> <*EndAddress*> [<*Data*>]

Where:

<*StartAddress*>	is a 16-bit hexadecimal number.
<*EndAddress*>	is a 16-bit hexadecimal number.
<*Data*>	is an 8-bit hexadecimal number.

The BLOCK FILL command is used to place a single 8-bit value into a range of memory locations. <*StartAddress*> is the first memory location written with <*data*> and <*EndAddress*> is the last memory location written with <*data*>. If the <*data*> parameter is omitted, the memory range is filled with the value $00.

Restrictions: None

Example:

```
>BF 6400 6fff 0
>BF 6f00 6fff 55
>
```

Breakpoint Set — BR

Syntax: BR [<*Address*> <*Address*> ...]

Where:

<*Address*> is an optional 16-bit hexadecimal number.

The BR command is used to set a software breakpoint at a specified address or to display any previously set breakpoints. The function of a breakpoint is to halt user program execution when the program reaches the breakpoint address. When a breakpoint address is encountered, D-Bug12 disassembles the instruction at the breakpoint address, prints the CPU12's register contents, and waits for a D-Bug12 command to be entered by the user.

Breakpoints are set by typing the breakpoint command followed by one or more breakpoint addresses. Entering the breakpoint command without any breakpoint addresses will display all the currently set breakpoints.

Restrictions: D-Bug12 implements the breakpoint function by replacing the opcode at the breakpoint address in the user's program with an SWI instruction when operating in EVB mode or with the BGND instruction when operating in pod mode. A breakpoint may not be set on a user SWI instruction when operating in EVB mode. In either mode, breakpoints may be set only at an opcode address, and breakpoints may be placed only at memory addresses in alterable memory.

Even though D-Bug12 supports a maximum of 10 user-defined breakpoints, a maximum of nine breakpoints may be set on the command line at one time. This restriction is due to the limitation of the command line processor, which allows a maximum of 10 command line arguments including the command string.

When operating in pod mode, new breakpoints may not be set with the BR command when the R> prompt is displayed. However, the BR command may be used to display breakpoints that currently are set in the user's running program.

BR Breakpoint Set (Continued)

D-Bug12 version 2.0.0 does not support the MC68HC912B32's hardware breakpoint (H/W) function. Later versions of D-Bug 12, which may support this function, can be obtained from the sources listed in **E.5.1 Obtaining D-Bug12 Upgrades**.

Example:

```
>BR 35ec 2f80 c592
Breakpoints: 35EC 2F80 C592

>BR
Breakpoints: 35EC 2F80 C592

>
```

Bulk Erase On-Chip EEPROM — BULK

Syntax: BULK

The BULK command is used to erase the entire contents of byte-erasable EEPROM in a single operation. After the bulk erase operation has been performed, each on-chip EEPROM location is checked for an erased condition.

Restrictions: To erase EEPROM, the EEPROM block-protect control bits must be cleared. Refer to *MC68HC912B32 Technical Summary*, Motorola document order number MC68HC912BC32TS/D, for locations and operation of the block-protect controls.

Example:

```
>BULK
F/EEPROM Failed To Erase
>BULK
>
```

CALL — Call Subroutine

Syntax: CALL [<*Address*>]

Where:

<*Address*> is an optional 16-bit hexadecimal number.

The CALL command is used to execute a subroutine and return to the D-Bug12 monitor program when the final RTS of the subroutine is executed. When control is returned to D–Bug12, the CPU register contents are displayed. All CPU registers contain the values at the time the final RTS instruction was executed, with the exception of the program counter (PC). The PC contains the starting address of the subroutine. If a subroutine address is not supplied on the command line, the current value of the PC is used as the starting address.

NOTE: *No user breakpoints are placed in memory before execution is transferred to user code.*

Restrictions: If the called subroutine modifies the value of the stack pointer during its execution, it must restore the stack pointer's original value before executing the final RTS of the called subroutine. This restriction is required because a return address is placed on the user's stack that returns to D-Bug12 when the final RTS of the subroutine is executed. Any subroutine must obey this restriction to execute properly.

The CALL command cannot be issued when the R> prompt is displayed, indicating that the target system is already running a user program.

Example:

```
>CALL 820
Subroutine Call Returned

 PC    SP    X     Y      D = A:B     CCR = SXHI  NZVC
0820  0A00  057C  0000       0F:F9          1001  0000
>
```

Specify Target MCU Device — DEVICE

Syntax:

```
DEVICE
DEVICE <DeviceName> [<EEStart> <EEEnd>
       <FStart> <FEnd> <RAMStart> <RAMEnd>
        <IOBase>]
```

Where:

<DeviceName>	is the maximum of seven ASCII characters used to select/define a target MCU device.
<EEStart>	is the on-chip EEPROM starting address, a 16-bit hexadecimal number.
<EEEnd>	is the on-chip EEPROM ending address, a 16-bit hexadecimal number.
<FStart>	is the on-chip FLASH EEPROM starting address, a 16-bit hexadecimal number.
<FEEnd>	is the on-chip FLASH EEPROM ending address, a 16-bit hexadecimal number.
<RAMStart>	is the on-chip RAM starting address, a 16-bit hexadecimal number.
<RAMEnd>	is the on-chip RAM ending address, a 16-bit hexadecimal number.
<IOBase>	is the base address of the on-chip I/O registers, a 16-bit hexadecimal number.

Selecting the proper target MCU with the DEVICE command provides D-Bug12 the information necessary to allow transparent alteration of the target MCU's on-chip EEPROM using any D-Bug12 commands that modify memory. It also provides the necessary information to allow the programming and erasure of on-chip FLASH EEPROM. In addition, it allows D-Bug12 to initialize the stack pointer to the top of on-chip RAM when the target MCU is reset by use of the RESET command. The DEVICE command has three separate command line formats that allow for the display, selection and/or definition of target device parameters.

DEVICE Specify Target MCU Device (Continued)

Entering DEVICE on the command line followed by a carriage return displays the name of the currently selected device, the on-chip EEPROM's starting and ending address, the on-chip FLASH EEPROM's starting and ending address, the on-chip RAM's starting and ending address, and the I/O base address. This form of the command may be used when D-Bug12 is operating in either EVB or pod mode.

When D-Bug12 is operated in pod mode, the DEVICE command also may be used to select or define a new target device. Entering the DEVICE command followed only by a device name configures D-Bug12 for operation with the selected target device. The default device list contains entries for the MC68HC912B32 and the MC68HC812A4. **Table 3-3** shows the command line name to use for the two default MCU devices.

Table 3-3. Command Line Names

Device Name	Target MCU
912B32	MC68HC912B32
812A4	MC68HC812A4

Entering the DEVICE command followed by a device name and seven hexadecimal parameters allows new devices to be added to the target device table or existing device table entries to be modified. When a new device is added or when an existing device entry is modified, it becomes the currently selected device. If a new device does not contain a particular on-chip resource, such as FLASH EEPROM, a value of 0 should be entered for the starting and ending addresses.

Specify Target MCU Device (Continued) DEVICE

Because the target device data and the current device selection are stored in the probe MCU's on-chip EEPROM, new device information and the device selection are retained when power is removed from the pod.

NOTE: *If the M68HC12B32EVB is operated in EVB mode and the contents of any locations of the on-chip EEPROM are altered, it is strongly recommended that the on-chip EEPROM be completely erased by using the BULK command before using the EVB in pod mode again. Erasing the on-chip EEPROM causes D-Bug12 to reinitialize the device table with the two default MCU devices. The information for any new devices that were added to the table will be lost.*

Restrictions: When operating the M68HC12B32EVB in EVB mode, the DEVICE command may be used only to display the current device information.

The DEVICE command maintains a 16-bit checksum on the contents of the entire on-chip EEPROM to maintain the integrity of the device table. If any of the on-chip EEPROM locations are altered while operating the M68HC12B32EVB in EVB mode, D-Bug12 will reinitialize the device table with the default device information contained in the on-chip FLASH EEPROM. However, it is possible for the checksum verification to fail (for instance, if the entire contents of the on-chip EEPROM is programmed with 0s). Therefore, it is *strongly* recommended that the on-chip EEPROM be completely erased by using the BULK command before using the EVB in pod mode again. Using the EVB in probe mode with a corrupt device data table may cause D-Bug12 to operate in an unpredictable manner.

The 768 bytes of on-chip EEPROM allow a total of 34 entries in the device table. *Do not* exceed this number.

When adding a new device to the device table, the addresses provided for the on-chip FLASH EEPROM, on-chip RAM, and I/O registers should reflect the locations of these resources when the part is reset. This requirement is necessary for the FBULK and FLOAD commands to work properly.

DEVICE Specify Target MCU Device (Continued)

Example:

```
>DEVICE

Device: 912B32
EEPROM: $0D00 - $0FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $0000

S>DEVICE 912b32 1d00 1fff 8000 ffff 800 bff 0

Device: 912B32
EEPROM: $1D00 - $1FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $0000

S>DEVICE 812a4

Device: 812A4
EEPROM: $1000 - $1FFF
RAM: $0800 - $0BFF
I/O Regs: $0000

S>
```

Specify Target EEPROM Base Address EEBASE

Syntax: EEBASE *<Address>*

Where: *<Address>* is an optional 16-bit hexadecimal number.

Each time D-Bug12 performs a memory write, it automatically performs the necessary register manipulations to program the on-chip EEPROM if the write operation falls within the address range of the target's on-chip EEPROM. Because user code may change the EEPROM's base address by writing to the INITEE register, D-Bug12 must be informed of the EEPROM's location if automatic EEPROM writes are to occur. The EEBASE command is used to specify the base address of the target processor's on-chip EEPROM.

When operating in EVB mode, the default EEPROM base address and range are specified in the customization data variables `CustomData.EEBase` and `CustomData.EESize`. The value in `CustomData.EEBase` is used by the startup code to remap the EEPROM. The EEBASE command may not be used to relocate the I/O registers.

When operating in pod mode, the target's default EEPROM base address and range are specified by the currently selected device. (See the DEVICE command for additional details.)

The EEBASE command does *not* check to ensure that the parameter is a valid base address for the selected M68HC12 Family member. If an improper base address is provided, automatic programming of the on-chip EEPROM will not operate properly.

NOTE: *The EEBASE command does not automatically modify the INITEE register. The user is responsible for ensuring that the INITEE register is modified either manually or through the execution of user code.*

Restrictions: The EEBASE command may not be used when D-Bug12 is operated in EVB mode.

EEBASE Specify Target EEPROM Base Address (Continued)

Example:

```
S>DEVICE

Device: 912B32
EEPROM: $0D00 - $0FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $0000

S>EEBASE 1d00

Device: 912B32
EEPROM: $1D00 - $1FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $0000

S>MM 12
0012 01 11
0013 0F .
S>MD 1d00

1D00 FF FF FF FF - FF FF FF FF - FF FF FF FF - FF FF FF FF  ................
S>
```

Erase Target FLASH EEPROM — FBULK

Syntax: FBULK

The FBULK command is used to erase the entire contents of the target MCU's on-chip FLASH EEPROM in a single operation. After the bulk erase operation has been performed, each on-chip FLASH location is verified. If the contents are not $FF, an error message is displayed.

The target processor's FLASH EEPROM is erased by resetting the target processor and then loading a small driver program into the target processor's on-chip RAM. For this reason, the previous contents of the target processor's on-chip RAM are lost.

Restrictions: When operating in EVB mode, the FBULK command cannot be used. If the FBULK command is entered while in EVB mode, an error message is displayed and command execution is terminated.

Before using the FBULK command, a target device must have been selected with the DEVICE command that reflects the locations of the target's on-chip FLASH EEPROM, on-chip RAM, and the I/O registers when the part is reset. Failure to follow this restriction will cause the FBULK command to fail and may require that the EVB be reset.

FLASH EEPROM programming voltage (V_{PP}) must be applied to the target MCU. If the target system is another M68HC12B32EVB board, V_{PP} may be supplied via header W8, with header W7 set accordingly. For more information on these EVB headers, see **Table 4-1**.

Because the FBULK command downloads a small "driver" program into the target MCU's on-chip RAM, D-Bug12's breakpoint table is cleared before beginning execution of the "driver." This is necessary to prevent previously set breakpoints from accidentally halting the execution of the driver program.

FBULK Erase Target FLASH EEPROM (Continued)

Example:

```
S>FBULK
Flash Programming Voltage Not Present
S>FBULK
F/EEPROM Failed To Erase
S>FBULK
S>

>FBULK
Command Not Allowed In EVB Mode
>
```

Program Target FLASH EEPROM — FLOAD

Syntax: FLOAD [<*AddressOffset*>]

Where: <*AddressOffset*> is a 16-bit hexadecimal number.

The FLOAD command is used to program a target device's FLASH EEPROM memory with the data contained in S-record object files. The address offset, if supplied, is added to the load address of each S record before the S record's data bytes are placed in memory. Providing an address offset other than 0 allows object code or data to be programmed into memory at a location other than that for which it was assembled or compiled.

The programming of the on-chip FLASH EEPROM uses an algorithm where the time required to program each byte or word can vary from as little as 60 sto as long as 3.5 ms. (Note, however, that the programming time for each byte or word typically should take no more than 120 to 180 µs). Because of this variability, the FLOAD command uses a software handshaking protocol to control the flow of S-record data from the host computer. When the FLOAD command is ready to receive an S record, an ASCII asterisk character (*) is sent to the host computer. The host computer should respond by sending a single S record. The S record may include a carriage return and/or line feed character(s). Most commercial terminal programs that are capable of sending ASCII text files have the ability to wait for a specific character or characters before sending a line of text.

The FLOAD command is terminated when D-Bug12 receives an S9 end-of-file record. If the object file being loaded does not contain an S9 record, D-Bug12 does not return its prompt and continues to wait for the end-of-file record. Pressing the reset switch returns D-Bug12 to its command line prompt.

Restrictions: The host program used to send the S-record data must be capable of waiting for an ASCII asterisk character (*) before sending each S-record line.

Because the on-chip FLASH EEPROM is only bulk erasable, the FBULK command should be used before loading new data into FLASH EEPROM with the FLOAD command.

FLOAD Program Target FLASH EEPROM (Continued)

The FLOAD command cannot be used with S records that contain a code/data field longer than 64 bytes. Sending an S record with a longer field may cause D-Bug12 to crash or load incorrect data into the FLASH EEPROM.

Before using the FLOAD command, a target device must have been selected using the DEVICE command that reflects the locations of the on-chip FLASH EEPROM, on-chip RAM, and the I/O registers when the part is reset. Failure to follow this restriction will cause the FLOAD command to fail and may require that the EVB be reset.

FLASH EEPROM programming voltage (V_{PP}) must be applied to the target MCU. If the target system is another M68HC12B32EVB board, V_{PP} may be supplied via header W8, with header W7 set accordingly. For more information on these EVB headers, see **Table 4-1**.

Because the FLOAD command downloads a small “driver” program into the target MCU’s on-chip RAM, D-Bug12’s breakpoint table is cleared before beginning execution of the “driver.” This is necessary to prevent previously set breakpoints from accidentally halting the execution of the driver program.

Example:

```
S>FLOAD
Flash Programming Voltage Not Present
S>FLOAD
************************************************************************
************************************************************************
**********************
S>
```

Go Execute a User Program G

Syntax: G [<*Address*>]

Where:

<*Address*> is an optional 16-bit hexadecimal number.

The G command is used to begin the execution of user code in real time. Before beginning execution of user code, any breakpoints that were set with the BR command are placed in memory. Execution of the user program continues until a user breakpoint is encountered, a CPU exception occurs, the STOP or RESET command is entered, or the EVB's reset switch is pressed.

When user code halts for any of these reasons (except reset, which wipes the slate clean) and control is returned to D-Bug12, a message is displayed explaining the reason for user program termination. In addition, D-Bug12 disassembles the instruction at the current program counter (PC) address, prints the CPU12's register contents, and waits for the next D-Bug12 command to be entered by the user.

If a starting address is not supplied in the command line parameter, program execution will begin at the address defined by the current value of the PC.

Restrictions: The G command cannot be issued when the R> prompt is displayed, indicating that the target system is already running a user program.

Example:

```
S>G 800
R>MD 1000

1000 FF FF FF FF - FF FF FF FF - FF FF FF FF - FF FF FF FF  ............
R>
User Breakpoint Encountered

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
0820  09FE  057C  0000     00:00         1001 0100
0820  08                    INX
S>
```

GT Go Till

Syntax: GT *<Address>*

Where:

<Address> is a 16-bit hexadecimal number.

The GT command is similar to the G command except that a temporary breakpoint is placed at the address supplied on the command line. Any breakpoints that were set by the use of the BR command are not placed in the user code before program execution begins. Program execution begins at the address defined by the current value of the program counter. When user code reaches the temporary breakpoint and control is returned to D-Bug12, a message is displayed explaining the reason for user program termination. In addition, D-Bug12 disassembles the instruction at the current PC address, prints the CPU12's register contents, and waits for a command to be entered by the user.

Restrictions: The GT command cannot be issued when the R> prompt is displayed, indicating that the target system is already running a user program.

Example:

```
S>GT 820
R>
Temporary Breakpoint Encountered

 PC    SP    X     Y      D = A:B   CCR = SXHI  NZVC
0820  09FE  057C  0000       00:00        1001  0100
0820  08                     INX
S>
```

Onscreen Help Summary HELP

Syntax: HELP

The HELP command is used to display a summary of the D-Bug12 command set. Each command is shown along with its command line format and a brief description of its function.

Restrictions: None

HELP Onscreen Help Summary (Continued)

Example:

```
>HELP
ASM <Address>  Single line assembler/disassembler
    <CR>       Disassemble next instruction
    <.>        Exit assembly/disassembly
BAUD <baudrate>  Set communications rate for the terminal
BF <StartAddress> <EndAddress> [<data>]  Fill memory with
data
BR [<Address>]  Set/Display user breakpoints
BULK  Erase entire on-chip EEPROM contents
CALL [<Address>]  Call user subroutine at <Address>
DEVICE [<DevName> [<Address>...<Address>]]
display/select/add target device
EEBASE <Address>  Set base address of on-chip EEPROM
FBULK  Erase entire target Flash contents
FLOAD [<AddressOffset>]  Load S-Records into target Flash
G [<Address>]  Begin/continue execution of user code
GT <Address>  Set temporary breakpoint at <Address> &
execute user code
HELP  Display this D-Bug12 command summary
LOAD [<AddressOffset>]  Load S-Records into memory
MD <StartAddress> [<EndAddress>]  Memory Display Bytes
MDW <StartAddress> [<EndAddress>]  Memory Display Words
MM <StartAddress>   Modify Memory Bytes
   <CR>             Examine/Modify next location
   </> or <=>       Examine/Modify same location
   <^> or <->       Examine/Modify previous location
   <.>              Exit Modify Memory command
MMW <StartAddress>  Modify Memory Words (same subcommands
as MM)
MOVE <StartAddress> <EndAddress> <DestAddress> Move a
block of memory
NOBR [<address>]  Remove One/All Breakpoint(s)
RD  Display all CPU registers
REGBASE <Address>  Set base address of I/O registers
RESET  Reset target CPU
RM  Modify CPU Register Contents
STOP  Stop target CPU
T  [<count>]  Trace <count> instructions
UPLOAD  <StartAddress> <EndAddress>  S-Record Memory
display
VERF  [<AddressOffset>]  Verify S-Records against memory
contents
<Register Name> <Register Value>  Set register contents
       Register Names: PC, SP, X, Y, A, B, D
       CCR Status Bits: S, XM, H, IM, N, Z, V, C
>
```

Load S-Record File — LOAD

Syntax: LOAD [<*AddressOffset*>]
{*Send File*}

Where:

<*AddressOffset*> is an optional 16-bit hexadecimal number.

{*Send File*} is the host-computer communications program's utility for sending an ASCII (text) file. Refer to **Appendix B. Communications Program Examples** for examples.

The LOAD command is used to load S-record object files into memory from an external device. The address offset, if supplied, is added to the load address of each S record before its data bytes are placed in memory. Providing an address offset other than 0 allows object code or data to be loaded into memory at a location other than that for which it was assembled. During the loading process, the S-record data is not echoed to the control console. However, for each 10 S records that are successfully loaded, an ASCII asterisk character (*) is sent to the control console. When an S-record file has been loaded successfully, control returns to the D-Bug12 prompt.

The LOAD command is terminated when D-Bug12 receives an S9 end of file record. If the object file being loaded does not contain an S9 record, D-Bug12 does not return its prompt and continues to wait for the end of file record. Pressing the reset switch returns D-Bug12 to its command line prompt.

Restrictions: When operating in pod mode, the LOAD command does not support standard baud rates above 38,400. This is due to the overhead involved in the implementation of the custom serial protocol required by the single-wire background debug mode interface.

Example:

```
>LOAD 1000
********************
>
```

MD — Memory Display

Syntax: MD <*StartAddress*> [<*EndAddress*>]

Where:

<*StartAddress*> is a 16-bit hexadecimal number.

<*EndAddress*> is an optional 16-bit hexadecimal number.

The MEMORY DISPLAY command displays the contents of memory as both hexadecimal bytes and ASCII characters, 16 bytes on each line. The <*StartAddress*> parameter must be supplied; the <*EndAddress*> parameter is optional. When the <*EndAddress*> parameter is not supplied, a single line is displayed.

The number supplied as the <*StartAddress*> parameter is rounded down to the next lower multiple of 16, while the number supplied as the <*EndAddress*> parameter is rounded up to the next higher multiple of 16, minus 1. This causes each line to display memory in the range of $xxx0 through $xxxF. For example, if $205 is entered as the start address and $217 as the ending address, the actual memory range displayed would be $200 through $21F.

Restrictions: None

Example:

```
>MD 800
0800  AA 04 37 6A - 00 06 27 F9 - 35 AE 78 0D - B7 56 78 20  ..7j..'.5.x..Vx

>MD 800 87f
0800  AA 04 37 6A - 00 06 27 F9 - 35 AE 78 0D - B7 56 78 20  ..7j..'.5.x..Vx
0810  B6 36 27 F9 - 35 AE 27 F9 - 35 9E 27 F9 - 35 BE B5 28  .6'.5.'.5.'.5..(
0820  27 F9 35 D6 - 37 B8 00 0F - 37 82 01 0A - 37 36 FF F0  '.5.7...7...76..
0830  7C 10 37 B3 - 00 00 37 B6 - 00 0F AA 04 - A5 02 37 B6  |.7...7.......7.
0840  00 0F 27 78 - 37 6A 00 06 - 27 F9 35 78 - 27 F9 35 56  ..'x7j..'.5x'.5V
0850  78 0D B7 10 - 78 3B 37 86 - 00 DC 27 F9 - 35 48 78 57  x...x;7...'.5HxW
0860  37 86 00 DE - F5 01 EA 09 - 37 B5 0D 0A - 27 F9 36 2A  7.......7...'.6*
0870  A5 00 37 65 - 00 02 27 F9 - 35 E8 37 9C - 37 4C F5 02  ..7e..'.5.7.7L..
>
```

Display Memory as 16-Bit Word — MDW

Syntax: MDW <*StartAddress*> [<*EndAddress*>]

Where:

<*StartAddress*> is a 16-bit hexadecimal number.

<*EndAddress*> is an optional 16-bit hexadecimal number.

The MDW command displays the contents of memory as hexadecimal words and ASCII characters, 16-bytes on each line. The <*StartAddress*> parameter must be supplied; the <*EndAddress*> parameter is optional. When the <*EndAddress*> parameter is not supplied, a single line is displayed.

The number supplied as the <*StartAddress*> parameter is rounded down to the next lower multiple of 16, while the number supplied as the <*EndAddress*> parameter is rounded up to the next higher multiple of 16, minus 1. This causes each line to display memory in the range of $xxx0 through $xxxF. For example, if $205 is entered as the start address and $217 as the ending address, the actual memory range displayed would be $200 through $21F.

Restrictions: None

Example:

```
>MDW 800
0800  AA04 376A - 0006 27F9 - 35AE 780D - B756 7820  ..7j..'.5.x..Vx

>MDW 800 87f
0800  AA04 376A - 0006 27F9 - 35AE 780D - B756 7820  ..7j..'.5.x..Vx
0810  B636 27F9 - 35AE 27F9 - 359E 27F9 - 35BE B528  .6'.5.'.5.'.5..(
0820  27F9 35D6 - 37B8 000F - 3782 010A - 3736 FFF0  '.5.7...7...76..
0830  7C10 37B3 - 0000 37B6 - 000F AA04 - A502 37B6  |.7...7.......7.
0840  000F 2778 - 376A 0006 - 27F9 3578 - 27F9 3556  ..'x7j..'.5x'.5V
0850  780D B710 - 783B 3786 - 00DC 27F9 - 3548 7857  x...x;7...'.5HxW
0860  3786 00DE - F501 EA09 - 37B5 0D0A - 27F9 362A  7.......7...'.6*
0870  A500 3765 - 0002 27F9 - 35E8 379C - 374C F502  ..7e..'.5.7.7L..
>
```

MM — Memory Modify

Syntax: `MM <Address> [<Data>]`

Where:

`<Address>`	is a16-bit hexadecimal number.
`<Data>`	is an optional 8-bit hexadecimal number.

The MEMORY MODIFY command allows the contents of memory to be examined and/or modified as 8-bit hexadecimal data. If the 8-bit data parameter is present on the command line, the byte at memory location `<Address>` is replaced with `<Data>` and the command is terminated. If not, D-Bug12 enters the interactive memory modify mode. In the interactive mode, each byte is displayed on a separate line following the data's address. Once the MEMORY MODIFY command has been entered, single-character subcommands are used for the modification and verification of memory contents. These subcommands have this format:

`[<Data>] <CR>`	Optionally, update current location and display the next location.
`[<Data>] </>` or `<=>`	Optionally, update current location and redisplay the current location.
`[<Data>] <^>` or `<->`	Optionally, update current location and display the previous location.
`[<Data>] <.>`	Optionally, update current location and exit MEMORY MODIFY.

With the exception of the carriage return, the subcommand must be separated from any entered data with at least one space character. If an invalid subcommand character is entered, an appropriate error message is issued and the contents of the current memory location are redisplayed.

Restrictions: While there are no restrictions on the use of the MM command, be careful when modifying target memory while user code is running. Accidentally modifying target memory containing program code could lead to program runaway.

Memory Modify (Continued) MM

Example:

```
>MM 800
0800  00  <CR>
0801  F0  FF
0802  00  ^
0801  FF  <CR>
0802  00  <CR>
0803  08  55 /
0803  55  _.
>
```

MMW — Modify 16-Bit Memory Word

Syntax: `MMW <Address> [<Data>]`

Where:

<Address>	is a 16-bit hexadecimal number.
<Data>	is an optional 16-bit hexadecimal number.

The MMW command allows the contents of memory to be examined and/or modified as 16-bit hexadecimal data. If the 16-bit data parameter is present on the command line, the word at memory location *<Address>* is replaced with *<Data>* and the command is terminated. If not, D-Bug12 enters the interactive memory modify mode. In the interactive mode, each word is displayed on a separate line following the data's address. Once the MMW command has been entered, single-character subcommands are used for the modification and verification of memory contents. These subcommands have this format:

`[<Data>] <CR>`	Optionally, update current location and display the next location.
`[<Data>] </>` or `<=>`	Optionally, update current location and redisplay the current location.
`[<Data>] <^>` or `<->`	Optionally, update current location and display the previous location.
`[<Data>] <.>`	Optionally, update current location and exit MMW.

With the exception of the carriage return, the subcommand must be separated from any entered data with at least one space character. If an invalid subcommand character is entered, an appropriate error message is issued and the contents of the current memory location are redisplayed.

Restrictions: While there are no restrictions on the use of the MMW command, caution should be used when modifying target memory while user code is running. Accidentally modifying target memory containing program code could lead to program runaway.

Modify 16-Bit Memory Word (Continued) MMW

Example:

```
>MMW 800
0800  00F0  <CR>
0802  0008  AA55 /
0804  843F  ^
0802  AA55  <CR>
0804  843F  <CR>
0806  C000  .
>
```

MOVE — Move Memory Block

Syntax: MOVE <*StartAddress*> <*EndAddress*> <*DestAddress*>

Where:

<*StartAddress*>	is a 16-bit hexadecimal number.
<*EndAddress*>	is a 16-bit hexadecimal number.
<*DestAddress*>	is a 16-bit hexadecimal number.

The MOVE command is used to move a block of memory from one location to another, one byte at a time. The number of bytes moved is one more than the <*EndAddress*> – <*StartAddress*>. The block of memory beginning at the destination address may overlap the memory block defined by the <*StartAddress*> and <*EndAddress*>.

One of the uses of the MOVE command might be to copy a program from RAM into the on-chip EEPROM memory.

Restrictions: A minimum of one byte may be moved if the <*StartAddress*> is equal to the <*EndAddress*>. The maximum number of bytes that may be moved is $2^{16} - 1$.

Be careful when moving target memory while user code is running. Accidentally modifying target memory containing program code could lead to program runaway.

Example:

```
>MOVE 800 8ff 1000
>
```

Remove Breakpoints — NOBR

Syntax: NOBR [<*Address*> <*Address*> ...]

Where:

<*Address*> is an optional 16-bit hexadecimal number.

The NOBR command can be used to remove one or more previously entered breakpoints. If the NOBR command is entered without any arguments, all user breakpoints are removed from the breakpoint table.

Restrictions: When operating in pod mode, breakpoints may not be removed with the NOBR command when the R> prompt is displayed.

Example:

```
>BR 800 810 820 830
Breakpoints: 0800  0810  0820  0830

>NOBR 810 820
Breakpoints: 0800  0830

>NOBR
All Breakpoints Removed

>
```

RD Register Display

Syntax: RD

The REGISTER DISPLAY command is used to display the CPU12's registers.

Restrictions: When operating in pod mode, the CPU registers may not be displayed when the R> prompt is displayed.

Example:

```
>RD
 PC    SP     X    Y      D = A:B   CCR = SXHI NZVC
0206  03FF  1000  3700       27:FF       1001 0001
>
```

Specify Target EEPROM Register Address REGBASE

Syntax: REGBASE *<Address>*

Where:

<Address> is a 16-bit hexadecimal number.

Because D-Bug12 supports the ability to transparently program the on-chip EEPROM of the target MCU, it must know the base address of the I/O registers. Because user code may change the register block's base address by writing to the INITRG register, D-Bug12 must be informed of the register block's base address for transparent EEPROM writes to occur. The REGBASE command is used to specify the base address of the target processor's on-chip registers.

The REGBASE command does not check to ensure that the *<Address>* parameter is a valid base address for the selected M68HC12 Family member. If an improper register base address is provided, automatic programming of the on-chip EEPROM will not operate properly.

When operating in EVB mode, the default register base address is specified in the customization data variable `CustomData.IOBase`. This value is used by the startup code to remap the I/O registers. The REGBASE command may not be used to relocate the I/O registers.

NOTE: *The REGBASE command does not automatically modify the INITRG register. The user is responsible for ensuring that the INITRG register is modified either manually or through the execution of user code.*

Restrictions: The REGBASE command may not be used when D-Bug12 is operated in the EVB mode.

REGBASE Specify Target EEPROM Register Address (Continued)

Example:

```
S>DEVICE

Device: 912B32
EEPROM: $0D00 - $0FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $0000

S>REGBASE 2000

Device: 912B32
EEPROM: $0D00 - $0FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $2000

S>
```

Reset Target MCU — RESET

Syntax: RESET

The RESET command is used to reset the target system processor when operating in D-Bug12's pod mode. The target processor's reset pin is held byte-erasable for approximately 2 ms. When the reset line is released, BDM commands are sent to the target processor to place it in active background mode. The target processor's registers are initialized with the same values used for the registers when operating in EVB mode.

The effects of the RESET command may be different from a user assertion of the target's $\overline{\text{RESET}}$ pin:

- When the RESET command is issued, the host EVB controls the state of the target's BKGD pin, placing the target processor in special mode and active background execution.
- When a user assertion of the target's $\overline{\text{RESET}}$ pin occurs, the target processor may enter either special or normal mode, depending on the state of its BKGD pin. D-Bug12 displays a message indicating that the target processor has been reset.

Restrictions: When operating in EVB mode, the RESET command cannot be used. If the RESET command is entered while in EVB mode, an error message is displayed and command execution is terminated.

Example:

```
S>RESET
Target Processor Has Been Reset
S>G 4000
R>RESET
Target Processor Has Been Reset
S>
```

RM Register Modify

Syntax: RM

The REGISTER MODIFY command is used to examine and/or modify the contents of the CPU12's registers in an interactive manner. As each register and its contents are displayed, D-Bug12 allows the user to enter a new value in hexadecimal for the register. If modification of the displayed register is not desired, entering a carriage return will cause the next CPU12 register and its contents to be displayed on the next line. When the last of the CPU12's registers has been examined and/or modified, the RM command displays the first register, giving the user an opportunity to make additional modifications to the CPU12's register contents.

Typing a period (.) as the first non-space character on the line will exit the interactive mode of the REGISTER MODIFY command and return to the D-Bug12 prompt.

The registers are displayed in this order, one register per line: PC, SP, X, Y, A, B, and CCR.

Restrictions: When operating in pod mode, the CPU registers may not be modified when the R> prompt is displayed.

Example:

```
>RM
PC=0206 200
SP=03FF <CR>
X=1000 1004
Y=3700 <CR>
A=27 <CR>
B=FF <CR>
CCR=D0 D1
PC=0200 .
>
```

Stop Execution on Target MCU — STOP

Syntax: STOP

When operating in D-Bug12's pod mode, the STOP command is used to halt target program execution and place the target processor in active background debug mode.

Restrictions: When operating in EVB mode, the STOP command cannot be used. If the STOP command is entered while in EVB mode, an error message is displayed and command execution is terminated.

Example:

```
S>ASM 4000
4000  CCFFFF         LDD   #$FFFF
4003  830001         SUBD  #$0001
4006  26FB           BNE   $4003
4008  20F6           BRA   $4000
400A  00             BGND
>.
S>G 4000
R>STOP
Target Processor Has Been Stopped

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
4003  0A00  0000  0000     37:3F        1101 0000
4003  830001         SUBD  #$0001
S>
```

T Trace

Syntax: T [<*Count*>]

Where:

<*Count*> is an optional 8-bit decimal number in the range 1 to 255.

The TRACE command is used to execute one or more user program instructions beginning at the current program counter (PC) location. As each program instruction is executed, the CPU12's register contents are displayed and the *next* instruction to be executed is displayed. A single instruction may be executed by entering the TRACE command immediately followed by a carriage return.

Restrictions: Because of the method used to execute a single instruction, branch instructions (Bcc, LBcc, BRSET, BRCLR, DBEQ/NE, IBEQ/NE, and TBEQ/NE) that contain an offset that branches back to the instruction opcode do not execute. D-Bug12 appears to become stuck at the branch instruction and does not execute the instruction even if the condition for the branch instruction is satisfied. This limitation can be overcome by using the GT (GO TILL) command to set a temporary breakpoint at the instruction following the branch instruction.

In EVB mode, the TRACE command may be used only for code located in alterable memory.

These restrictions *do not* apply when using D-Bug12 on a target system in pod mode.

Trace (Continued) T

Example:

```
>T

 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0803  09FE  057C  0000      10:00      1001 0000
0803  830001        SUBD    #$0001
>T 3

 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0806  09FE  057C  0000      0F:FF      1001 0000
0806  26FB          BNE     $0803

 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0803  09FE  057C  0000      0F:FF      1001 0000
0803 830001         SUBD    #$0001


 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0806  09FE  057C  0000      0F:FE      1001 0000
0806 26FB           BNE     $0803
>
```

UPLOAD Display Memory in S-Record Format

Syntax: UPLOAD *<StartAddress>* *<EndAddress>*

Where:

<StartAddress> is a 16-bit hexadecimal number.

<EndAddress> is a 16-bit hexadecimal number.

The UPLOAD command is used to display the contents of memory in Motorola S-record format. In addition to displaying the specified range of memory, the UPLOAD command also outputs an S9 end-of-file record. The output of this command may be captured by the user's terminal program and saved to a disk file.

Restrictions: None

Example:

```
>upload 400 5ff
S123040000F0000843FC0000F50F379F37BF43FCF50F27FA757F177AFA047504177AFA21C5
S123042037B500FF37FAFB0437B5400037FAFB061735FB0037B500C137FAFA003715379C01
S1230440F50F379D37BC012C37BD400085009A003C023D02377C0140B6EE7A0F400037B583
S1230460000337FAFA4C37FAFA5037FAFA5437B5502037FAFA4E37B5302037FAFA5237B58A
S1230480682037FAFA5637BD014037BC000095008A003C023D02377D0172B6EE37BD017259
S12304A037BC020095008A003C023D02377D018EB6EE27F937B0F50F379C37BC00CE27F901
S12304C000FC27F9104C27F90E68378000BE0A0D442D4275673136205631 2E3033202D20E3
S12304E04465627567204D6F6E69746F7220466F7220546865204D36384843313620 4661ED
S12305006D696C790A0D2843292031393932204D6F746F726F6C612053656D69636F6E64BD
S12305207563746F7220496E632E000037B5FF0237FAFA4837B578B037FAFA4A7A0F005E52
S1230540000000000000000000200020402080 20C021000000000000000000000002144F
S123056000000000000000000000000000000000002187A0F3BAC7A0F3BBC7A0F11E87A0F62
S12305803C727A0F3C847A0F3C967A0F3CA8F50F379C379D379E27FAF50F379F37BF43FCE8
S12305A07501177A4054173540523604361C27F90088B0D637BC01BC360227F70A0D3E00A9
S12305C04500B70427F936BC3C01B0F027F7277537BC400017BC405027F936CC780DB60477
S12305E027F936A0274A27F77803B6FEB03A7808B6162776B7DE3730000127F93686752002
S9030000FC
>
```

Verify S-Record File Against Memory VERF

Syntax: VERF [*<AddressOffset>*]
{*Send File*}

Where:

<AddressOffset>	is an optional 16-bit hexadecimal number.
{*Send File*}	is the host computer communications program's utility for sending an ASCII (text) file. Refer to **Appendix B. Communications Program Examples**.

The VERF command is used to compare the data contained in an S-record object file to the contents of EVB memory. The address offset, if supplied, is added to the load address of each S record before an S record's data bytes are compared to the contents of memory. Providing an address offset other than 0 allows the S record's object code or data to be compared against memory other than that for which the S record was assembled.

During the verification process, an ASCII asterisk character (*) is sent to the control console for each 10 S records that are successfully verified. When an S-record file has been verified successfully, control returns to the D-Bug12 prompt.

If the contents of EVB memory do not match the corresponding data in the received S records, an error message is displayed and the VERIFY command is terminated. D-Bug12 then returns to its command-line prompt. If the host computer continues to send S records to the EVB, D-Bug12 tries to interpret each S record as a command and issues an error message for each S record received.

If the contents of EVB memory match the contents of the received S records, the VERF command terminates when D-Bug12 receives an S9 end-of-file record. If the object file being verified does not contain an S9 record, D-Bug12 continues to wait for an S9 record without returning to the command-line prompt. Pressing the reset switch, S1, returns D-Bug12 to its command-line prompt.

VERF Verify S-Record File Against Memory (Continued)

Restrictions: None

Example:

```
>VERF 1000
**********
>
```

Modify Register Value *<RegisterName>*

Syntax: *<RegisterName>* *<RegisterValue>*

Where:

<RegisterName> is one of the CPU12 registers listed in **Table 3-4**.

<RegisterValue> is an 8- or 16-bit hexadecimal number.

Table 3-4. CPU12 Registers

Register Name	Description	Legal Range
PC	Program counter	$0 to $FFFF
SP	Stack pointer	$0 to $FFFF
X	X-index register	$0 to $FFFF
Y	Y-index register	$0 to $FFFF
A	A accumulator	$0 to $FF
B	B accumulator	$0 to $FF
D	D accumulator (A:B)	$0 to $FFFF
CCR	Condition code register	$0 to $FF

Each of the fields in the condition code register (CCR) may be modified by using the bit names in **Table 3-5**.

Table 3-5. Condition Code Register Bits

CCR Bit Name	Description	Legal Values
S	STOP enable	0 or 1
H	Half carry	0 or 1
N	Negative flag	0 or 1
Z	Zero flag	0 or 1
V	Two’s complement overflow flag	0 or 1
C	Carry flag	0 or 1
IM	IRQ interrupt mask	0 or 1
XM	XIRQ interrupt mask	0 or 1

<RegisterName> Modify Register Value (Continued)

This set of “commands” uses a CPU12 register name as the command name to allow changing the register’s contents. Each register name or CCR bit name is entered on the command line followed by a space, then followed by the new register or bit contents. After successful alteration of a CPU register or CCR bit, the entire CPU register set is displayed.

Restrictions: None

Example:

```
>PC 700e

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  7315  7D62        47:44    1001   0000
>X 1000

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62        47:44    1001   0000
>C 1

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62        47:44     1001   0001
>Z 1

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62        47:44    1001   0101
>D adf7

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62        AD:F7    1001   0101
>
```

3.8 Off-Board Code Generation

Code developed outside the EVB environment should be generated with an M68HC12-compatible assembler or C compiler that can generate object files in S-record format. The recommended assembler, P&E Microcomputer Systems' IASM12, is supplied with the EVB package on the diskette labeled "IASM12." The IASM12 user's manual, IASM12.DOC, is also on the diskette.

S-records are described in **Appendix A. S-Record Format**.

When the S-record file has been generated, it may be loaded from the host computer into EVB memory in these ways:

- Into the host EVB's byte-erasable EEPROM or RAM, using the D-Bug12 commands BULK and LOAD when the host EVB is in EVB mode
- Into the host EVB's byte-erasable or FLASH EEPROM, using the EEPROM bootloader when the host EVB is in BOOTLOAD mode
- Into a target MCU's byte-erasable EEPROM or RAM, using the D-Bug12 commands BULK and LOAD when the host EVB is in pod mode
- Into a target MCU's FLASH EEPROM, using the D-Bug12 commands FBULK and FLOAD when the host EVB is in pod mode

More information on the EVB operating modes, the D-Bug12 commands, and the EEPROM bootloader can be found in **3.2 Operating Modes**, **3.3.2 Operating Procedures**, **3.7 D-Bug12 Command Set**, and **Appendix E. EEPROM Bootloader**.

3.9 Memory Usage

The EVB's memory usage and requirements are described here and are summarized in **Table 3-6**.

3.9.1 Description

The monitor program, D-Bug12, occupies the 32-Kbyte FLASH EEPROM area of the MCU's memory map. To use the FLASH EEPROM area for custom programs, refer to **Appendix E. EEPROM Bootloader**.

When operating in EVB mode, D-Bug12 requires 512 bytes of on-chip RAM, from $0A00 to $0BFF, for stack and variable storage. The remaining 512 bytes of on-chip RAM, from $0800 to $0900, are available for variable storage and stack space by user programs.

NOTE: *D-Bug12 sets the default value of the user's stack pointer to $0A00. This is not a mistake. The M68HC12 Family's stack pointer points to the last byte that was pushed onto the stack, rather than to the next available byte on the stack, as the M68HC11 Family does. The M68HC12 Family first decrements its stack pointer, then stores data on the stack. The M68HC11 Family stores data on the stack and then decrements its stack pointer.*

3.9.2 Memory Map

The information in **Table 3-6** describes address ranges and locations.

Table 3-6. Factory-Configuration Memory Map

Address Range	Usage	Description
$0000 – $01FF	CPU registers	On-chip registers
$0800 – $09FF $0A00 – $0BFF	User code/data Reserved for D-Bug12	1-Kbyte on-chip RAM
$0D00 – $0FFF	User code/data	768 bytes on-chip EEPROM
$8000 – $F67F $F680 – $F6BF $F6C0 – $F6FF $F700 – $F77F $F780 – $F7FF $F800 – $FBFF $FC00 – $FFBF $FFC0 – $FFFF	D-Bug12 code User-accessible functions D-Bug12 customization data D-Bug12 startup code Interrupt vector jump table Reserved for bootloader expansion EEPROM bootloader Reset and interrupt vectors	32 Kbytes on-chip FLASH EEPROM

3.10 Operational Limitations

In EVB mode, D-Bug12 requires many of the MC68HC912B32's resources for execution. In this mode, the EVB cannot provide true emulation of a target system. These limitations are described in the following subsections.

If target-system emulation is required, the EVB may be reprogrammed and controlled via the BDM interface. Operation as a target is described in **3.2.3 Pod (Probe) Mode**.

3.10.1 On-Chip RAM

D-Bug12 requires 512 bytes of on-chip RAM for stack and variable storage. This usage is shown in **Table 3-6**.

3.10.2 On-Chip EEPROM

D-Bug12 occupies FLASH EEPROM starting at address $8000, as shown in **Table 3-6**. This area is thus not available for emulation of a target application.

3.10.3 SCI Port Usage

D-Bug12 requires the MCU's serial communications interface (SCI) port for the terminal interface. The SCI port is either connected (default) or disconnected from the RS-232C RXD and TXD signals by means of jumpers W1 and W2.

3.10.4 Dedicated MCU Pins

As used on the EVB with D-Bug12, the following MCU lines perform specific functions. If an application requires their use, the EVB hardware and/or operating software must be custom-configured or special precautions must be taken in the application code to avoid conflicts with the D-Bug12 usage.

PAD0 — EVB mode select pin (W3)

PAD1 — EVB mode select pin (W4)

PE5/MODA and PE6/MODB — Sets MCU chip mode, normally single chip

3.10.5 Terminal Communications

High baud rates occasionally result in dropped characters on the terminal display. This is not the result of a baud rate mismatch, but is due to the host processor being too busy or too slow to process incoming data at the selected baud rate. The D-Bug12 MD, MDW, T, and HELP commands may be affected by this problem. Sometimes the problem can be ignored without harm.

If it requires correcting, try:

- Using a slower baud rate
- A different communications program
- Closing unnecessary applications or exiting Windows. In multitasking environments such as Windows® and the Macintosh System 7®, the problem can occur when several applications are running at once.
- Displaying fewer address locations or tracing fewer instructions at a time when using the MD, MDW, or T commands

Section 4. Hardware Reference

4.1 Contents

4.2 Printed Circuit Board (PCB) Description

The EVB printed circuit board (PCB) is an 5.15-inch by 3.4-inch (13.1-cm by 8.64-cm) board with two layers.

Most of the connection points on the EVB use headers spaced on 1/10-inch (2.54-mm) centers, with these exceptions:

- Subminiature D connector for the RS-232C interface
- External power-supply connections

4.3 Configuration Headers and Jumper Settings

For maximum flexibility, the EVB uses two types of jumper headers:

- Factory-installed headers are those most likely to be used for configuration without major alteration of the EVB's hardware operation. These headers are populated, and the factory-installed jumpers on them are preset for the default EVB hardware and firmware (D-Bug12) configurations. **Table 4-1** lists these headers by function and describes their default and optional jumper settings.
- Cut-trace header footprints offer EVB hardware options that are less likely to be changed. These footprints are not populated. The default connection between pins is a trace on the PCB. To change a cut-trace footprint, the PCB trace must be cut. To return to the original configuration, a header and a jumper must be installed to re-establish the shunt.

NOTE: *Use of the cut-trace header footprints requires a thorough understanding of the MCU and of the EVB hardware. Refer to the MC68HC912B32 Technical Summary, Motorola document order number MC68HC912BC32TS/D, and to the EVB schematic diagram (**4.13 Schematics**) for design information.*

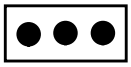
CAUTION: *When cutting a PCB trace to customize a header footprint, use a sharp blade. Be careful to avoid personal injury and not to cut adjacent traces.*

Key to **Table 4-1**: Headers are depicted as viewed from either the component side as shown in **Figure 1-1. EVB Layout and Component Placement** or the solder side as shown in **Figure 1-2. EVB Solder Side View**.

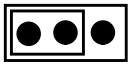
2-pin header with no jumper installed or
2-pin cut-trace header with trace cut

2-pin header with jumper installed

2-pin cut-trace header with default trace intact



3-pin header with no jumper installed



3-pin header with jumper installed on left 2 pins

1–2 **bold** pin numbers indicate factory-default settings
1–2, cut *italics* indicate alternate settings

Table 4-1. Jumper-Selectable Functions (Sheet 1 of 3)

Diagram	Setting	Description
W1 RS-232C TXD Connection to SCI PS1		
1 2 	**1–2** *cut*	**TXD enabled.** TXD disconnected from SCI port.
W2 RS-232C RXD Connection to SCI PS0		
1 2 	**1–2** *cut*	**RXD enabled.** RXD disconnected from SCI port.

Table 4-1. Jumper-Selectable Functions (Sheet 2 of 3)

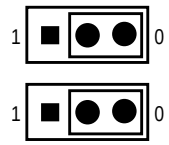
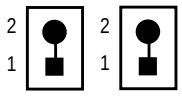
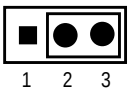
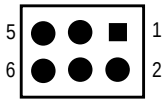
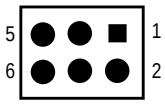
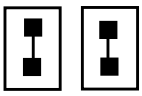
Diagram	Setting	Description
W3, W4 EVB Mode Selection		
	**W3–0** **W4–0**	**EVB mode — Execution from FLASH EEPROM (D-Bug12 default)**
	W3–1 *W4–0*	Jump to EEPROM mode
	W3–0 *W4–1*	Pod mode — Remote BDM
	W3–1 *W4–1*	Bootload mode
W5, W6 MCU Mode Selection: MODB (W5), MODA (W6)		
	**W5 in** **W6 in**	MODB MODA **0 0 Single-chip mode** Note: If cut, these headers must be wired to external circuitry that provides the desired levels for MODA and MODB. Refer to **Table 4-2**.
W7 V_{PP}/V_{DD} Selection		
	**2–3** *1–2*	**Connects MCU's V_{PP} pin to V_{DD} (non-programming mode)** Connects MCU's V_{PP} pin to V_{PP} input header (programming mode)
W8 V_{PP} Input Header		
	1 2	V_{PP} input Ground
W9 BDM In		
	1 2 3 4 5 6	Input to MCU BKGD Ground NC $\overline{\text{RESET}}$ input to MCU NC V_{DD} Note: At reset, the BKGD input serves with MODA and MODB to determine the CPU mode. Refer to **Table 4-2**.

Table 4-1. Jumper-Selectable Functions (Sheet 3 of 3)

Diagram	Setting	Description
W10 V_{DD} Connection to BDM Out		
	**1–2** *Cut*	**Connects V_{DD} to BDM out pin 6** BDM out pin 6 open
W11 Reset Connection to BDM Out		
	**1–2** *Cut*	**Connects MCU-generated reset (PT6) to BDM out pin 4** BDM out pin 4 open
W12 BDM Out		
	1 2 3 4 5 6	BKGD output from MCU PT7 Ground NC Reset output from MCU PT6 NC V_{DD}
W13, W14 RS-232C Configuration (Reserved)		
	**In**	**Reserved**
W15 LVI Reset Enable		
	**In** *Cut*	**On-board low-voltage reset enabled** On-board LVI reset disabled; provide external reset
W16 On-Board Crystal Enable		
	**In** *Cut*	**On-board crystal connected to MCU EXTAL** On-board crystal disabled; use W16 to provide external clock to EXTAL

4.4 Power Input Circuitry

The input power connector on the EVB is a 2-pin, lever-actuated connector (P5). Decoupling capacitors filter ripple and noise from the supply voltage.

4.5 Terminal Interface

An RS-232C transceiver (U1A or U1B) links the MCU's serial communications interface to the RS-232C DB-9 receptacle, P1. The communications parameters for this port are described in **2.6 Terminal Communications Setup**.

4.6 Microcontroller

The MC68HC912B32 is one of the first of a family of next generation M68HC11 microcontrollers with both on-chip memory and peripheral functions. The CPU12 is a high-speed, 16-bit processing unit. The programming model and stack frame are identical to those of the standard M68HC11 CPU. The CPU12 instruction set is a proper superset of the M68HC11 instruction set. All M68HC11 instruction mnemonics are accepted by CPU12 assemblers with no changes.

The EVB-resident MC68HC912B32 (U2) has seven modes of operation. These modes are determined at reset by the state of three mode pins — BKGD, MODB, and MODA — as shown in **Table 4-2**.

The EVB is factory-configured for MCU operation in the normal single-chip mode. In this mode of operation, all port pins are available to the user. On-chip FLASH EEPROM is used for program execution, with byte-erasable EEPROM and some RAM available for user code/data. Although other MCU modes are available, the EVB was designed for the single-chip mode of operation. There is no provision for external memory.

For more information on the CPU, refer to the *CPU12 Reference Manual*, Motorola document order number CPU12RM/AD.

Table 4-2. CPU Mode Selection

BKGD Through BDM In	MODB Header W5	MODA Header W6	Mode Description
0	0	0	Special single chip
0	0	1	Special expanded narrow
0	1	0	Special peripheral
0	1	1	Special expanded wide
1	**0**	**0**	**Normal single chip**
1	0	1	Normal expanded narrow
1	1	0	Reserved (currently defaults to peripheral mode)
1	1	1	Normal expanded wide

4.7 Clock Circuitry

The EVB comes with a 16-MHz crystal, Y1, with appropriate startup capacitors. The board should be able to accommodate most crystals and ceramic resonators.

Header W16 may be used to disconnect Y1 from the MCU's on-chip oscillator. An external clock may then be supplied to EXTAL through W16.

4.8 Reset

The reset circuit includes a pullup resistor, reset switch (S1), and a low-voltage inhibit device with a toggle voltage of 3.0 Vdc. This reset circuit drives the MCU's $\overline{\text{RESET}}$ pin directly. Note that header W15 may be used to provide an alternate reset input.

4.9 Low-Voltage Inhibit (LVI)

Low-voltage inhibit (LVI) uses a Motorola undervoltage sensing device (U1) to automatically drive the MCU's $\overline{\text{RESET}}$ pin low when V_{DD} falls below U1's threshold. This prevents the accidental corruption of EEPROM data if the power-supply voltage should drop below the allowable level..

Depending on the date of manufacture, the sensing device installed on the EVB may have either a 2.7-volt or 4.5-volt threshold. U1 may be identified by part number:

- MC34164P-3 — 2.7 Vdc
- MC34164P-5 — 4.5 Vdc

If operation below U1's threshold (but no less than 2.7 Vdc) is required, one of two methods can be used:

- Replace U1 with a device that has the required threshold voltage.
- Cut the trace on header W15 to disconnect U1 from the $\overline{\text{RESET}}$ line. If this is done, an external reset signal should be provided via W15 in case the supply voltage falls below the acceptable level.

4.10 Background Debug Mode (BDM) Interface

The MCU's serial BDM interface can be accessed through two 2-row x 3-pin headers, BDM in (W9) and BDM out (W12). The pin assignments are shown in **Table 4-3**.

The BDM interface may serve in two ways:

- As the "probe" interface through which a host EVB in pod mode controls a target system (see **3.2.3 Pod (Probe) Mode**)
- As the user interface with the EVB. This requires a development tool such as Motorola's serial debug interface. For more information, refer to the *SDI™ Interface User's Manual*, Motorola document order number SDIUM/D.

Table 4-3. BDM Connector J5 Pin Assignments

Pin Number	Description	
	W9 (In)	W12 (Out)
1	BKGD input to MCU	BKGD output from MCU PT7
2	V_{SS}	V_{SS}
3	No connection	No connection
4	$\overline{\text{RESET}}$ input to MCU	$\overline{\text{RESET}}$ output from MCU PT6 (1)
5	No connection	No connection
6	V_{DD}	V_{DD} (1)

(1) Refer to **Table 4-1**.

4.11 Prototype Area

The EVB's prototype area allows construction of custom I/O circuitry that can be connected to the MCU's I/O lines through connectors P2, P3, P4, and P6. This area is a grid of holes (approximately 15 by 31) on 1/10-inch (2.54 mm) centers. This spacing accommodates most sockets, headers, and device packages.

Figure 1-1. EVB Layout and Component Placement shows the component-side view of the prototype area. Adjacent V_{SS} (ground) and V_{DD} footprints are provided for wire-wrap pins.

4.12 MCU Connectors

Four 2-row x 20-pin header footprints, P2, P3, P4, and P6, surround the MCU and provide access to its I/O and bus lines. They may be populated with wire-wrap pins or strip headers for use as I/O connectors, connection points for instrumentation probes and target hardware, and connections to the prototype area described in **4.10 Background Debug Mode (BDM) Interface**. **Figure 4-1** and **Figure 4-2** depict the pin assignments for these headers.

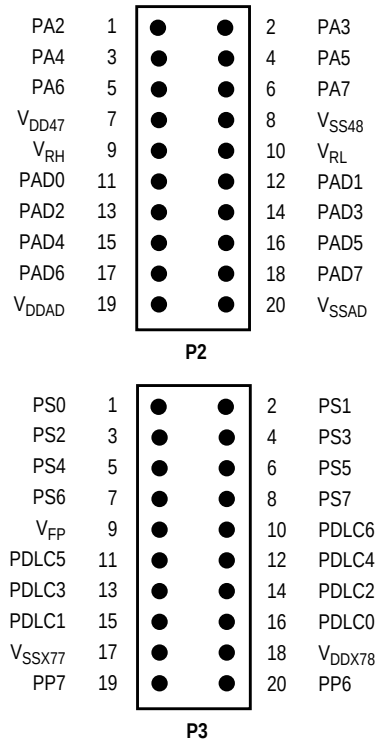


Figure 4-1. MCU I/O Headers P2 and P3

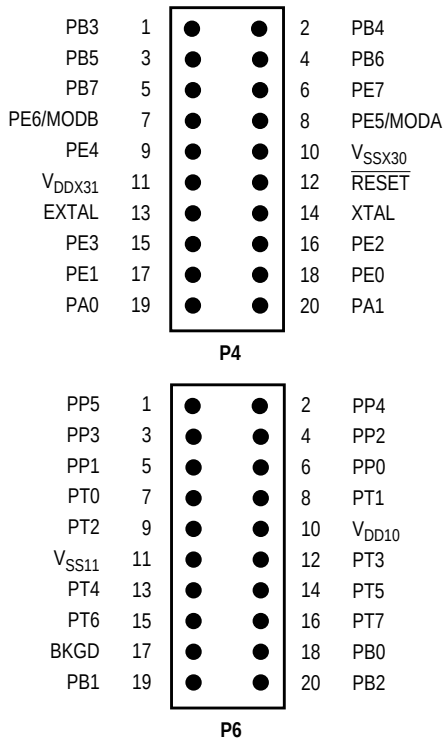


Figure 4-2. MCU I/O Headers P4 and P6

4.13 Schematics

The schematics are available here for your reference.

EVB912B32

EVALUATION BOARD

NOTES, UNLESS OTHERWISE SPECIFIED

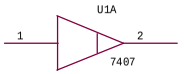
1. VCC PIN LOCATIONS :

   VCC IS APPLIED TO PIN 8 OF ALL 8-PIN IC's, PIN 14 OF ALL 14-PIN IC's, PIN 16 OF ALL 16-PIN IC's, PIN 20 OF ALL 20-PIN IC's, ETC.

2. GROUND PIN LOCATIONS :

   GROUND IS APPLIED TO PIN 4 OF ALL 8-PIN IC's, PIN 7 OF ALL 14-PIN IC's, PIN 8 OF ALL 16-PIN IC's, PIN 10 OF ALL 20-PIN IC's, ETC.

3. DEVICE TYPE, PIN NUMBERS, AND REFERENCE DESIGNATOR OF GATES ARE SHOWN AS FOLLOWS :



   7407 = DEVICE TYPE
   1 AND 2 = PIN NUMBERS
   U1A = REFERENCE DESIGNATORS

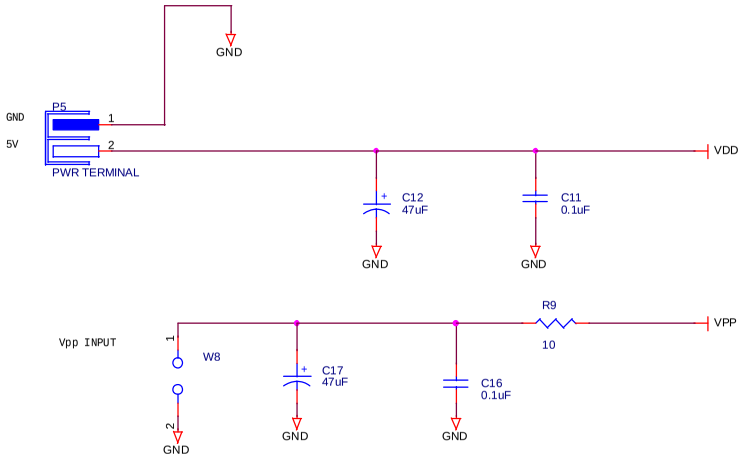
4. RESISTANCE VALUES ARE IN OHMS.

5. RESISTORS ARE 1/4 WATT, 5%.

6. CAPACITANCE VALUES ARE IN MICROFARADS.

REVISIONS

1	MC68HC912B32 EVALUATION BOARD	AS DATE
2	ADD VRH LPF, 5407 BYPASS, CUT JUMPERS	10-4-96
3	Remove 80QFP Socket, jmprs, new rs-232	10-18-96
4	Add cut jumps for Reset & Extal. Change SW1	11-1-96
5	Added jumps to W5,W6,W10,W11,W15 and W16. Changed all 47K Ohm resistors to 15K Ohm.	12-17-96



NOTE:

After PCB DRCs have been done, MANUALLY place cut jump on W5, W6, W10, W11, W15 AND W16.

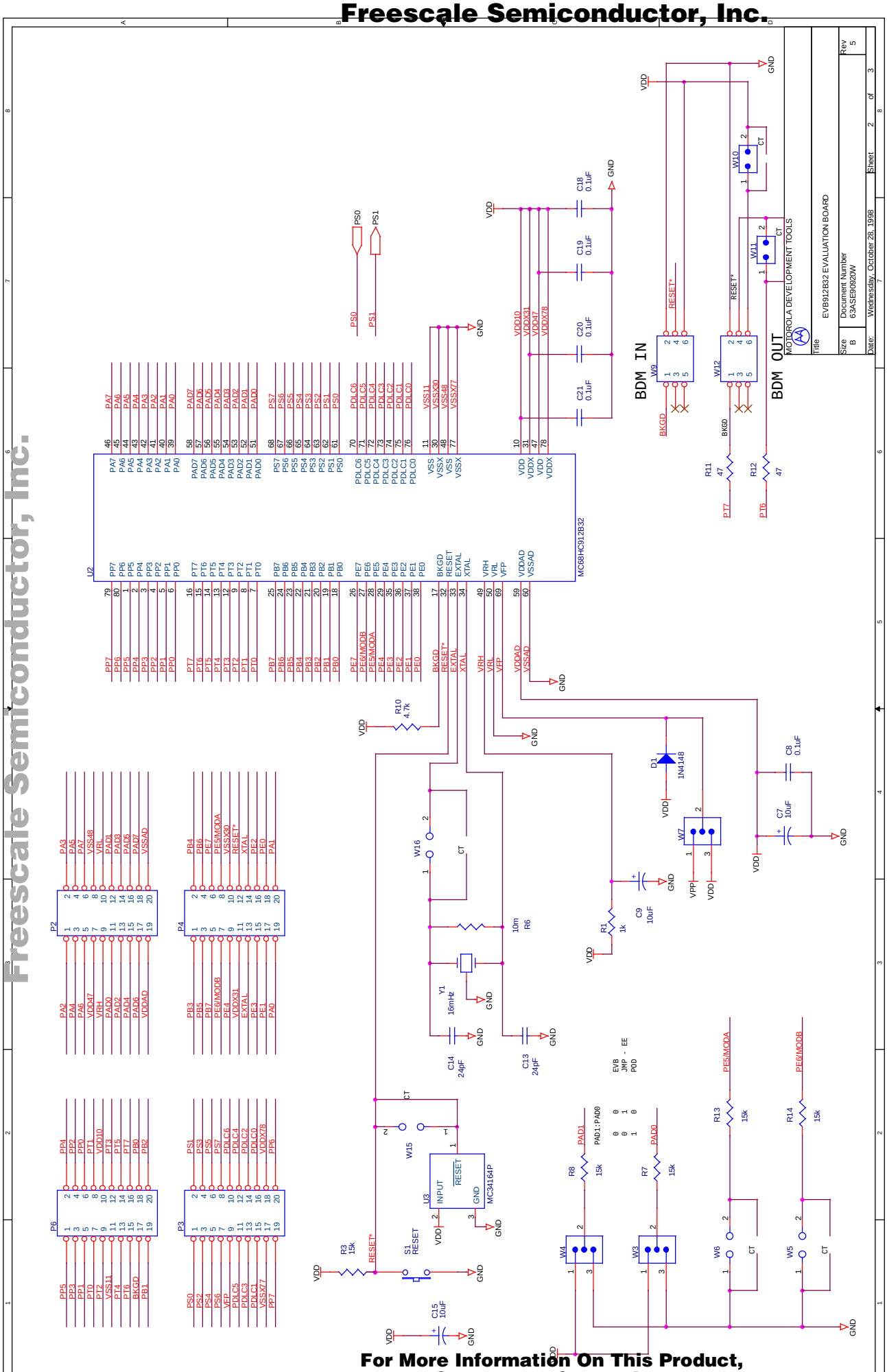
COMPUTER GENERATED DRAWING : DO NOT REVISE MANUALLY

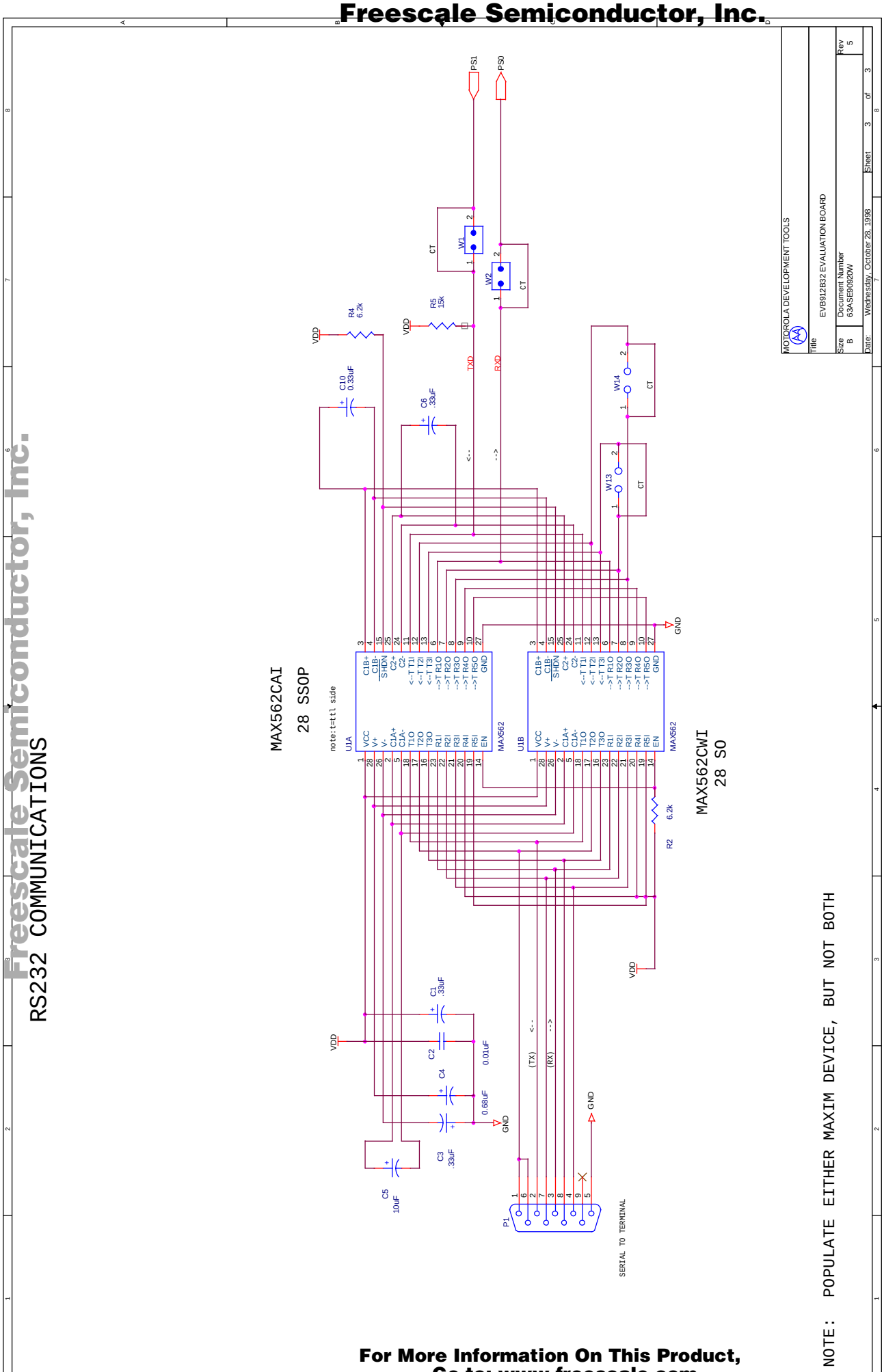
MOTOROLA DEVELOPMENT TOOLS

Title: EVB912B32 EVALUATION BOARD

Size	Document Number	Rev
B	63ASE90920W	5

Date: Wednesday, October 28, 1998 Sheet 1 of 3





NOTE: POPULATE EITHER MAXIM DEVICE, BUT NOT BOTH

Appendix A. S-Record Format

A.1 Contents

A.2 Overview

The Motorola S-record format was devised to encode programs or data files in a printable format for transport between computer platforms. The format also provides for editing of the S records and monitoring the cross-platform transfer process.

A.3 S-Record Contents

Each S record is a character string composed of several fields which identify:

- Record type
- Record length
- Memory address
- Code/data
- Checksum

S-Record Format

Each byte of binary data is encoded in the S record as a 2-character hexadecimal number:

- The first character represents the high-order four bits of the byte.
- The second character represents the low-order four bits of the byte.

The five fields that comprise an S record are shown in **Table A-1**.

Table A-1. S-Record Fields

Type	Record Length	Address	Code/Data	Checksum

The S-record fields are described in **Table A-2**.

Table A-2. S-Record Field Contents

Field	Printable Characters	Contents
Type	2	S-record type — S0, S1, etc.
Record Length	2	Character pair count in the record, excluding the type and record length.
Address	4, 6, or 8	2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
Code/Data	0 – 2n	From 0 to n bytes of executable code, memory loadable data, or descriptive information. For compatibility with teletypewriter, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S record).
Checksum	2	Least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S record may have an initial field to accommodate other data such as line number generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

A.4 S-Record Types

Eight types of S records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross assemblers, linkers, and other file-creating or debugging programs, utilize only those S records that serve the purpose of the program.

For specific information on which S records are supported by a particular program, the user manual for that program must be consulted.

NOTE: *D-Bug12 supports only the S0, S1, and S9 records. All data before the first S1 record is ignored. Thereafter, all records must be S1 type until the S9 record terminates data transfer.*

An S-record format may contain the record types listed in **Table A-3**.

Table A-3. S-Record Types

Type	Description
S0	Header record for each block of S records. The code/data field may contain any descriptive information identifying the following block of S records. The address field is normally 0s.
S1	Record containing code/data and the 2-byte address at which the code/data is to reside
S2 – S8	Ignored by the EVB
S9	Termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S records. Normally, only one header record is used, although it is possible for multiple header records to occur.

A.5 S Record Creation

S-record format programs may be produced by dump utilities, debuggers, cross assemblers, or cross linkers. Several programs are available for downloading a file in the S-record format from a host system to an 8- or 16-bit microprocessor-based system.

A.6 S-Record Example

A typical S-record format, as printed or displayed, is shown in this example:

Example:

```
S00600004844521B
S1130000285F245F2212226A00042429008237C2A
S11300100002000800082529001853812341001813
S113002041E900084#42234300182342000824A952
S107003000144ED492
S9030000FC
```

In the example, the format consists of:

- An S0 header
- Four S1 code/data records
- An S9 termination record

A.6.1 S0 Header Record

The S0 header record is described in **Table A-4**.

Table A-4. S0 Header Record

Field	S-Record Entry	Description
Type	S0	S-record type S0, indicating a header record
Record Length	06	Hexadecimal 06 (decimal 6), indicating six character pairs (or ASCII bytes) follow
Address	00 00	4-character, 2-byte address field; zeroes

Table A-4. S0 Header Record (Continued)

Field	S-Record Entry	Description
Code/Data	48 44 52	Descriptive information identified these S1 records: ASCII H D R — “HDR”
Checksum	1B	Checksum of S0 record

A.6.2 First S1 Record

The first S1 record is described in **Table A-5**.

Table A-5. S1 Header Record

Field	S-Record Entry			Description	
Type	S1			S-record type S1, indicating a code/data record to be loaded/verified at a 2-byte address	
Record Length	13			Hexadecimal 13 (decimal 19), indicating 19 character pairs, representing 19 bytes of binary data, follow	
Address	0000			4-character, 2-byte address field; hexadecimal address 0000 indicates location where the following data is to be loaded	
Code/Data	Opcode			Instruction	
	28 24 22 22 00 29 08	5F 5F 12 6A 04 00 23	 24 7	BHCC BCC BHI BHI BRSET BHCS BRSET	$0161 $0163 $0118 $0172 0, $04, $012F $010D 4, $23, $018C
Checksum	2A			Checksum of the first S1 record	

The 16 character pairs shown in the code/data field of **Table A-5** are the ASCII bytes of the actual program.

The second and third S1 code/data records each also contain $13 (19T) character pairs and are ended with checksum 13 and 52, respectively. The fourth S code/data record contains 07 character pairs and has a checksum of 92.

A.6.3 S9 Termination Record

The S9 termination record is described in **Table A-6**.

Table A-6. S9 Header Record

Field	S-Record Entry	Description
Type	S9	S-record type S9, indicating a termination record
Record Length	03	Hexadecimal 04, indicating three character pairs (three bytes) follow
Address	00 00	4-character, 2-byte address field; zeroes
Code/Data		There is no code/data in an S9 record.
Checksum	FC	Checksum of S9 record

A.6.4 ASCII Characters

Each printable ASCII character in an S record is encoded in binary. **Table A-5** gives an example of encoding for the S1 record. The binary data is transmitted during a download of an S record from a host system to a 9- or 16-bit microprocessor-based system. For example, the first S1 record in **Table A-5** is sent as shown here.

TYPE				LENGTH				ADDRESS								CODE/DATA								...	CHECKSUM			
S		1		1		3		0		0		0		0		2		8		5		F		...	2		A	
5	3	3	1	3	1	3	3	3	0	3	0	3	0	3	0	3	2	3	8	3	5	4	6	...	3	2	4	1
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	...	0011	0010	0100	0001

Appendix B. Communications Program Examples

B.1 Contents

B.2 Introduction

In all of these examples, first follow the EVB startup procedure in **3.3 Startup**. When the startup procedure calls for setting up the host computer's communications program for terminal emulation, follow the steps in the examples.

Keyboard entries are illustrated in this appendix using these conventions:

ENTER	Press the keyboard's ENTER, CARRIAGE RETURN, or RETURN key.
ALT-P	While holding down the ALTERNATE key, press the P key.
CTRL-\	While holding down the CONTROL key, press the BACKSLASH key.
<filename>	Supply the appropriate filename when required.

The stepwise procedures in this appendix are as accurate as possible. However, it is not feasible to document all of the communications programs that are available or to guarantee that a newer revision of a program behaves in exactly the same way as the version used to develop the procedure. For this reason, the steps are as generic as possible in their descriptions. They can thus serve as guidelines for programs not exemplified in this manual. Always consult the documentation for the program being used.

B.3 Procomm for DOS — IBM PC

B.3.1 Setup

To set up Procomm using DOS on an IBM®-compatible PC for use as the EVB terminal, first refer to **3.3 Startup** for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. At the DOS prompt, invoke the Procomm program by typing:

   `PROCOMM`

   And then pressing the RETURN key.

2. Enter the **Setup** menu by pressing ALT-S.

3. From the **Terminal Setup** submenu, select:

Terminal emulation	WYSE 100
Duplex	FULL
Flow control	NONE
CR translation (in)	CR
CR translation (out)	CR
BS translation	DEST
BS key definition	BS
Line wrap	OFF
Scroll	ON
Break Length (ms)	350
Enquiry (CTRL-E)	OFF

4. From the **ASCII Transfer Setup** submenu, select:

Echo locally	YES
Expand blank lines	YES
Pace character	0 (ASCII)
Character pacing	25 (1/1000th sec)
Line pacing	10 (1/10th sec)
CR translation	NONE
LF translation	NONE

5. Enter the **Line Settings** menu by pressing ALT-P. Select:

baud rate	9600 (or the customized EVB setting)
data bits	8
stop bits	1
parity	none
COM port	Host port used as the EVB terminal interface

6. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry.

7. Press ENTER. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in **3.3 Startup**.

B.3.2 S-Record Transfers to EVB Memory

To load an S-record file from the host computer into EVB memory using Procomm on an IBM-compatible host computer, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. Instruct Procomm to send the S-record file by pressing the PAGE UP key. Follow the onscreen instructions to select the S-record file for transfer, using ASCII transfer protocol.

Upon completion of the S-record file transfer, the D-Bug12 prompt is displayed.

B.4 Kermit for DOS — IBM PC

B.4.1 Setup

To set up Kermit using DOS on an IBM-compatible PC for use as the EVB terminal, first refer to section **3.3 Startup** for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. At the DOS prompt, invoke Kermit by typing:

   `kermit` ENTER
2. Set the baud rate to 9600 or the customized EVB setting by typing:

   `set baud 9600` ENTER
3. Connect to the EVB by typing:

   `connect` ENTER
4. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in **3.3 Startup**.

B.4.2 S-Record Transfers to EVB Memory

To load an S-record file from the host computer into EVB memory using Kermit on an IBM-compatible host computer, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. "Escape" from the D-Bug12 prompt and start the Kermit file transfer by typing:

   CTRL-] `c` ENTER

   `<`*`filename`*`> > com1` ENTER

Upon completion of the S-record file transfer, the D-Bug12 prompt is displayed.

B.5 Kermit — Sun Workstation

B.5.1 Setup

To set up Kermit on the Sun® Workstation for use as the EVB terminal, first refer to section **3.3 Startup** for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. In a shell window, invoke Kermit by typing:

   `kermit` ENTER
2. Set the serial port to the one in use for the EVB (ttya, ttyb, etc.) by typing:

   `set line /dev/ttya` ENTER
3. Set the baud rate to 9600 or the customized EVB setting by typing:

   `set speed 9600` ENTER
4. Connect to the EVB by typing:

   `connect` ENTER
5. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in **3.3 Startup**.

B.5.2 S-Record Transfers to EVB Memory

To load an S-record file from the host computer into EVB memory using Kermit on a Sun Workstation, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. In the shell window being used for the EVB terminal interface, at the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. Open a shell window separate from the one being used for the EVB terminal interface. In this window, type:

   `cat <filename> > /dev/ttya` ENTER

Upon completion of the S-record file transfer, the D-Bug12 prompt is displayed in the shell window being used for the EVB terminal interface.

B.6 MacTerminal — Apple Macintosh

Using MacTerminal® on an Apple® Macintosh® computer is described here.

B.6.1 Setup

To set up MacTerminal on an Apple Macintosh computer for use as the EVB terminal, first refer to **3.2 Startup** for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. Select the following from the **Terminal Settings** menu:

Terminal:	TTY
Cursor Shape:	Underline
Line Width:	80 Columns
Select:	On Line
	Auto Repeat
Click on:	**OK**

2. Select the following from the **Compatibility Settings** menu:

Baud Rate:	9600 or the customized EVB setting
Bits per Character:	8 Bits
Parity:	None
Handshake:	None
Connection:	Modem or Another Computer
Connection Port:	Modem or Printer
Click on:	**OK**

3. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry.
4. Press ENTER. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in **3.3 Startup**.

B.6.2 S-Record Transfers to EVB Memory

To load an S-record file from the host computer into EVB memory using MacTerminal, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. From the **Macintosh File** menu, select **Send File - ASCII**.
3. From the dialog box, select the S-record file to be transferred.
4. Click on **Send**.

NOTE: *S records are not displayed during the file transfer.*

NOTE: *Following the file transfer, MacTerminal sends a carriage return-line feed pair, which D-Bug12 interprets as an erroneous command. To return to the D-Bug12 prompt, reset the EVB.*

B.7 Red Ryder — Apple Macintosh

B.7.1 Setup

To set up Red Ryder on an Apple Macintosh computer for use as the EVB terminal, first refer to **3.3 Startup** for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. Launch the Red Ryder program.
2. Set up the Red Ryder parameters:

   9600 baud or the customized EVB setting

   8 data bits

   1 stop bit

   no parity

   full duplex
3. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry.
4. Press ENTER. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in **3.3 Startup**.

B.7.2 S-Record Transfers to EVB Memory

To load an S-record file from the host computer into EVB memory using Red Ryder, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. From the **Macintosh File** menu, select **Send File - ASCII**.
3. From the dialog box, select the S-record file to be transferred.
4. Click on **Send**.

NOTE: *S records are not displayed during the file transfer.*

Upon completion of the S-record file transfer, the D-Bug12 prompt is displayed.

Appendix C. D-Bug12 Startup Code

The D-Bug12 startup code is located in the EPROMs, U7 and U9A, in the address range $FD80 to $FDFF, as shown in **Table 3-6. Factory-Configuration Memory Map**.

To customize this startup code, it is necessary to alter the startup code in FLASH EEPROM. For more information, refer to **Appendix E. EEPROM Bootloader**.

```
;
MAP_PAGE:     equ  $0000
;
;
PORTE:        equ  $0008+MAP_PAGE
DDRE:         equ  $0009+MAP_PAGE
PEAR:         equ  $000a+MAP_PAGE
MODE:         equ  $000b+MAP_PAGE
INITRG:       equ  $0011+MAP_PAGE
INITEE:       equ  $0012+MAP_PAGE
COPCTL:       equ  $0016+MAP_PAGE
CSCTL0:       equ  $003c+MAP_PAGE
CSCTL1:       equ  $003d+MAP_PAGE
CSSTR0:       equ  $003e+MAP_PAGE
PORTAD:       equ  $006f+MAP_PAGE
EEMCR:        equ  $00f0+MAP_PAGE
BPROT:        equ  $00f1+MAP_PAGE
FEELCK:       equ  $00f4+MAP_PAGE
;
MonRAMStart:  equ  $0A00
MonRAMSize:   equ  $0200


RAM_START:    equ  $0800

RAMSize:      equ  $0400

STACKTOP:     equ  RAM_START+RAMSize    ; stack at top of internal RAM

EE_START:     equ  $0d00                ; EEPROM located here out of reset
```

D-Bug12 Startup Code

```
CustData:    equ  $f6c0
IOBase:      equ  CustData+15          ; location of user supplied base address
                                       ; of I/O registers
EEBase:      equ  CustData+19          ; location of user supplied base
                                       ; address of EEPROM
;
             xref    _BootLoad, _UserFnTable
             xdef    __MonStartup, _EEDelay
;
             switch  .text
;
;*****************************************************************************
; INITIALIZATION
;
; The code in this section is initialization for the monitor on the EVB12B32
;*****************************************************************************

__MonStartup:
      lds    #STACKTOP                ; initialize monitor stack pointer
;
;     Disable the COP watchdog by CR2:CR1:CR0 = 0:0:0
;     COPCTL = $07 when reset in normal modes
;     FCME and CRx bits are write once in normal modes
;     COPCTL  [ CME  :FCME  : FCM  : FCOP ! DISR : CR2  : CR1  : CR0  ] $--16
;
      clr    COPCTL                   ; disable watchdog
;
;     Clear all monitor RAM to start from a known state
;
      ldx    #MonRAMStart
ClrRAM:      clr  1,x+                ; clear one and inc pointer
      cpx    #MonRAMStart+MonRAMSize
      bne    ClrRAM                   ; loop till RAM clear

;     Enable pipe signals, E, low strobe and read/write in port E
;     PIPOE, NECLK, LSTRE and RDWE are write once in normal modes
;     PEAR    [ARSIE :CDLTE :PIPOE :NECLK !LSTRE : RDWE :  0   :  0   ] $--0A
;
      ldaa   #$2c                     ; prevent later protection lock
      staa   PEAR                     ; PROTLK is write-once

;     Without changing modes, enable internal visibility
;     MODE    [SMODN : MODB : MODA : ESTR ! IVIS :  0   : EMD  : EME  ] $--0B
;
      bset   MODE,$08                 ; set IVIS

;     Enable EEPROM so monitor can program/erase bytes
;     EEMCR   [  1   :  1   :  1   :  1   !  1   :  1   :PROTLK: EERC ] $--F0
;     BPROT   [  1   :BPROT6:BPROT5:BPROT4!BPROT3:BPROT2:BPROT1:BPROT0] $--F1
;
      ldaa   #$fc                     ; prevent later protection lock
```

```
      staa    EEMCR                   ; PROTLK is write-once
      clr     BPROT                   ; allow EE program and erase
;
;     Disable writing to the on-chip FLASH EEPROM
;     FEELCK   [  0   :  0   :  0   :  0   :  0   :  0   :  0   : LOCK] $--F4
;
      ldaa    #$01                ; write a 1 to the FLASH LOCK bit to
                                  ; disable accidential reprogramming of
      staa    FEELCK              ; the FLASH memory (where we're located)
;
      ldd     EEBase              ; get the user supplied base address of the
                                  ; on-chip EEPROM
      oraa    #1                  ; make sure that the EEON bit remains set.
      staa    INITEE              ; re-map the on-chip EEPROM.
;
      ldd     IOBase              ; get the user supplied base address of the
                                  ; on-chip I/O registers
      staa    INITRG              ; re-map the on-chip registers.

      ldx     #_UserFnTable       ; point to the table of user accessible
                                  ; routines.
      jmp     [0,x]               ; the first entry is a pointer to main.
                                  ; GO.........
;
;
;
;     This small subroutine is used to produce a delay of approximately 10 ms.
;     This delay is based on the following conditions:
;
;            1.) An 8.00 MHz E-clock
;            2.) Subroutine located in internal memory
;
;     This routine is called by D-Bug12's WriteEEByte() function (through a
;     pointer stored in the Customization Data Table).
;
_EEDelay:
      ldx     #20000                  ; load delay count into x
DlyLoop:      dex                     ; decrement count
      bne     DlyLoop                 ; loop till done.
      rts                             ; return.
;
;

      end
```

D-Bug12 Startup Code

Appendix D. D-Bug12 Customization Data

D.1 Contents

D.2 Customization Data Area

The customization data area, located in FLASH EEPROM from $FC60 to $F6FF, allows users to change default data parameters used by D-Bug12. The data contained in this area is described by C data structure. The `CustomData typedef` is shown here.

For those unfamiliar with C, an assembly language equivalent is also shown here. The purpose of each field also is explained.

D.2.1 C Format

```
typedef struct {
                Byte UserCCR;                   /* User CPU Condition Code Register */
                Byte UserB;                     /* User CPU B-accumulator */
                Byte UserA;                     /* User CPU A-accumulator */
                Address UserX;                  /* User CPU X-index register */
                Address UserY;                  /* User CPU Y-index register */
                Address UserPC;                 /* User CPU Program Counter */
                Address UserSP;                 /* User CPU Stack Pointer */
                unsigned long SysClk;           /* System Clock frequency (in Hz) */
                Address IOBase;                 /* Base address of the I/O registers */
                unsigned int SCIBaudRegVal;     /* Initial SCI BAUD register value */
                Address EEBase;                 /* Base address of on-chip EEPROM */
                unsigned int EESize;            /* size of the on-chip EEPROM */
                void (*Delay)(void);            /* pointer to EEPROM delay routine */
                int AuxCmdCount;                /* number of commands in the */
                                                /* auxiliary command table */
                CmdTblEntryP AuxCmdTableP;      /* pointer to the auxiliary command */
                                                /* table */
                } CustomData;
```

D.2.2 Assembly Format

```
              org   $F6C0
;
CustData       equ   *
UserCCR        dc.b  $90      ; User CPU Condition Code Register
UserB          dc.b  $00      ; User CPU B-accumulator
UserA          dc.b  $00      ; User CPU A-accumulator
UserX          dc.w  $0000    ; User CPU X-index register
UserY          dc.w  $0000    ; User CPU Y-index register
UserPC         dc.w  $0000    ; User CPU Program Counter
UserSP         dc.w  $0A00    ; User CPU Stack Pointer
SysClk         dc.l  8000000  ; System Clock frequency (in Hz)
IOBase         dc.w  $0000    ; Base address of the I/O registers
SCIBaudRegVal  dc.w  52       ; Initial SCI BAUD register value
EEBase         dc.w  $0D00    ; Base address of the on-chip EEPROM
EESize         dc.w  768      ; size of the on-chip EEPROM
EEDelay        dc.w  _EEDelay ; address of EEPROM program/erase delay routine
AuxCmdCount    dc.w  0        ; number of commands in auxiliary command table
AuxCmdTableP   dc.w  $0000    ; pointer to the auxiliary command table
;
```

D.2.3 Initial User CPU Register Values

The first seven fields in the `CustomData typedef struct` are used to provide default values for the user CPU12 registers. In this version of D-Bug12, he user CCR value is set to 0x90. This sets the S bit, disabling the STOP instruction, and the I bit, inhibiting IRQ interrupts. The X bit is cleared to allow the use of the XIRQ interrupt as a user-supplied programmer's switch when operating in EVB mode. The user SP value is set to 0x0a00 when operating in

EVB mode, which is one byte beyond the last on-chip RAM location available to the user. The CPU12 stack pointer points to the last byte pushed onto the stack.

When operating the M68HC12B32EVB in pod mode, the values in the table for the CCR and the stack pointer are not used. Instead, when the target processor is reset by using the RESET command, the target's CCR is set to 0xd0. The stack pointer is set to one byte beyond the end of the target system's RAM, as specified by the DEVICE command.

In both operating modes, all of the other registers are initialized with the values contained in the customization data table.

D.2.4 SysClk Field

The `SysClk` field is used to inform D-Bug12 of the system clock frequency, M. Its value, in hertz, is set to 8,000,000. In this implementation, the E-clock frequency is the same as the system clock frequency, M. `SysClk` is used by the D-Bug12 BAUD command in calculating the new value of the SCI baud register for the requested baud rate.

NOTE: *It is the responsibility of the startup code to perform any actions necessary to set the system clock frequency. D-Bug12 does not set or change the system clock frequency using the* `SysClk` *value.*

D.2.5 IOBase Field

The `IOBase` field defines the base address of the input/output (I/O) registers. This address is used by D-Bug12 when accessing the I/O registers associated with the SCI and when programming or erasing the on-chip EEPROM. On the MC68HC912B32, the I/O registers are mappable to any 2-Kbyte memory space. Therefore, the `IOBase` entry should be only a multiple of 2048. The value of `IOBase` is set to 0x0000 which is the default address of the I/O registers for the MC68HC912B32.

NOTE: *It is the responsibility of the startup code to set the base address of the I/O registers. D-Bug12 does not set or change the I/O register base address.*

D.2.6 SCIBaudRegVal Field

The `SCIBaudRegVal` field is used to set the initial baud rate of the SCI used for console I/O by D-Bug12. Note that the value in `SCIBaudRegVal` is written directly to the baud register of the EVB being used for the SCI terminal interface.

Note that the value in `SCIBaudRegVal` is not the desired baud rate. The calculation of the actual baud rate is not made by D-Bug12 because of the possibility of an invalid baud register value. Without a valid baud register value during SCI initialization, D-Bug12 would have no way to inform the user that a problem exists. Not all combinations of baud rates and system clock frequencies produce a valid baud register value.

The formula used to calculate the baud register value is:

```
SCIBaudRegVal = MCLK   ( 16 * SCIBaudRate)
```

The initial baud register value for this version of D-Bug12 is 52 (0x0034). At a system clock frequency of 8.0 MHz, this sets the EVB-to-terminal baud rate at 9600 baud.

NOTE: *D-Bug12 takes care of initializing the SCI registers. The startup code should not initialize the SCI. The SCI data format is set to 8 data bits, 1 start bit, 1 stop bit, and no parity.*

D.2.7 EEBase and EESize Fields

The `EEBase` and `EESize` fields are used to describe the base address and range of the M68HC12's on-chip byte-erasable EEPROM. This information is used by D-Bug12's `WriteMem()` function to determine when a byte is being written to the on-chip EEPROM. D-Bug12 then calls its `WriteEEByte()` function to place the data in the on-chip EEPROM. On the MC68HC912B32, the EEPROM base address is mappable to any 4-Kbyte memory space and resides in the upper 768 bytes of the 4-Kbyte block. Therefore, the `EEBase` entry should be only a multiple of 0x1000. The value of `EEBase` is set to 0x0d00 which is the default base address of the on-chip EEPROM for the MC68HC912B32. The value of `EESize` is also set to 0x0300 (768) which is the size of the on-chip EEPROM. Setting the value of `EESize` to 0 disables the `WriteMem()` function's ability to write to on-chip EEPROM.

NOTE: *It is the responsibility of the startup code to set the base address of the EEPROM. D-Bug12 does not set or change the EEPROM base address.*

D.2.8 EEPROM Erase/Program Delay Function Pointer Field

The `(void)(* Delay)(void)` field is a function pointer that points to an EEPROM program/erase delay routine. For the MC68HC912B32, the routine should produce a delay of 10 ms before it returns. The current implementation of the delay routine is nothing more than a software delay loop. The subroutine is located in the startup code area of the D-Bug12 FLASH EEPROM from $F700 to $F77F.

D.2.9 Auxiliary Command Table Entries

The last two entries in this table provide a mechanism to extend D-Bug12's command set. The `AuxCmdTableP` points to an auxiliary command table, and `AuxCmdCount` contains the number of entries in the auxiliary command table. The table is an array of entries of type `CmdTblEntry.` Each `CmdTblEntry` in the auxiliary command table has this structure:

```
typedef struct {
      const char *CommandStr;                   /* pointer to the  command */
                                                /* string */
      int (*ExecuteCmd)(int argC, char *argV[]);/* pointer to function that*/
                                                /* implements the command */
               } CmdTblEntry, * CmdTblEntryP;
```

As the `typedef` shows, the first field is a character pointer pointing to a null terminated character array containing the command name. The command name string *must be in upper case*. The second field, a function pointer, points to a function that implements the new D-Bug12 command. The first parameter to this function is a count of the number of command line arguments that the command line interpreter found on the command line. This count *includes* the command name itself. The command line may contain no more than a total of 10 parameters. The second function parameter is a pointer to an array of `char *`. Each `char *` points to one of the command line parameters parsed by the command line interpreter.

The function implementing the new command can report any error conditions to the user in one of two ways.

- If the error condition can be described by one of the error messages in the enumerated constant list here, the user-defined command should return the appropriate constant.
- If some other message text needs to be conveyed to the user, the command should communicate the error message directly to the user by using the `printf()` function which is one of the available user-callable C functions. In this case, the user-defined command should return an error code of `noErr`.

```
enum Error {
            noErr = 0            /* Define No Error */
            WrongNumArgs = 6,    /* Wrong Number of Arguments */
            BadStartAddress = 7, /* Invalid Starting Address */
            BadEndAddress = 8,   /* Invalid Ending Address */
            StartEndError = 9,   /* Start Address Greater Than End Address */
            BadHexData = 10,     /* Invalid Hex Data */
            DataSizeError = 11,  /* Data Out Of Range */
            NoTargetWrite = 12,  /* Can't Write Target Memory */

           };
```

Appendix E. EEPROM Bootloader

E.1 Contents

E.2 Introduction

The EEPROM bootloader occupies 1 Kbyte of erase-protected FLASH EEPROM starting at address $FC00. It is invoked when the EVB is started in bootload mode (W3-1 and W4-1).

The bootloader may be used to program user code into byte-erasable EEPROM starting at address $0D00 and/or FLASH EEPROM starting at address $8000. The user program in FLASH EEPROM may then serve as the “boot” (startup) code when the board is placed in EVB mode (W3-0 and W4-0) or POD mode (W3-0 and W4-1) and reset.

D-Bug12 is overwritten when using FLASH EEPROM for user code. But since the bootloader itself cannot be overwritten, it is always available for loading new user code or reloading D-Bug12.

NOTE: *An additional 1 Kbyte of FLASH EEPROM, starting at address $F800, is reserved for future expansion of the bootloader. Thus, user code may only occupy the 30 Kbytes from $8000 to $F7FF.*

Programs loaded and used in this manner cannot be used for true emulation of an application. Refer to the restrictions in ***3.9 Memory Usage****.*

E.3 Serial S-Record Bootloader

The bootloader contains a serial S-record loader that can load assembled code from the host computer into either FLASH EEPROM or byte-erasable EEPROM. It uses the SCI for communications with the host computer via the EVB’s RS-232C interface.

The only special requirements for the host computer’s communications program are:

- It must operate at 9600 baud.
- It must wait for the prompt string * (the ASCII asterisk character) before sending a line of text to the EVB. This “handshaking” is necessary because of the variable amount of time required to program each S record into byte-erasable or FLASH EEPROM. Byte-erasable EEPROM requires 10 ms per byte. FLASH EEPROM typically requires less than 180 µs per byte but can take as long as 3.5 ms.

When the EVB is restarted with jumpers W3 and W4 set for bootload mode, the EEPROM bootloader executes immediately. The bootloader's prompt appears on the host terminal:

```
(E)rase, (P)rogram or (L)oadEE:
```

Select the desired function by typing an upper- or lower-case E, P, or L.

NOTE: *Before selecting the `Erase` or `Program` function, apply V_{PP} to the EVB via header W8. Then move the jumper on header W7 to position 1-2. After programming is completed, remove V_{PP} and return W7 to position 2-3.*

The starting address of the user code must be placed in the reset vector position ($F7FE) of the alternate reset/interrupt vector jump table. For more information, see ***E.4 Vector Jump Table: Interrupt and Reset Addresses****.*

The bootloader cannot be used with S records containing a code/data field longer than 64 bytes (S-record length field greater than 67 bytes). Longer S records will cause the bootloader to crash and/or program incorrect data into EEPROM.

S records may contain ASCII CR and/or LF characters.

CAUTION: *If an `Erase` or `Program` operation is unsuccessful after one or two attempts, check the V_{PP} connection on header W8 and measure the value of V_{PP} to verify compliance with the MC68HC912B32 Electrical Specifications Supplement. A V_{PP} voltage lower than that specified may cause the erase or program operation to fail. A V_{PP} voltage higher than that specified may cause permanent damage to the device.*

E.3.1 (E)rase

This selection causes a bulk erase of FLASH EEPROM except for the erase-protected area starting at address $F800, which contains the bootloader program, the area reserved for bootloader expansion, and the reset/interrupt vector table. After the erase operation, a verify operation checks for proper erasure of all locations.

If the erase operation was successful, the message `Erased` is displayed and the bootloader's prompt is redisplayed.

If any locations were found to contain a value other than $FF, the message `Not Erased` is displayed and the bootloader's prompt is redisplayed. If an error occurs, see the previous caution.

E.3.2 (P)rogram

In FLASH programming mode, the bootloader sends an ASCII * (asterisk character) to the host computer, indicating that it is ready to receive an S record. The host then sends a single S record and waits for the * prompt from the bootloader before sending the next S record.

This process is repeated until the bootloader receives an end-of-file (S9) record from the host computer. If no S9 record is received, the bootloader continues to wait for another S record indefinitely. In this situation, the EVB must be reset to return to the bootloader's prompt. (S records already loaded into FLASH EEPROM are unaffected by the missing S9 record; reprogramming is not necessary.)

If a FLASH EEPROM location fails to program properly, the message `Not Programmed` is displayed, and the bootloader's prompt is redisplayed.

If an error occurs during programming, see the previous caution. If errors persist, the problem may be caused by an S record containing data that is outside the range of the available FLASH EEPROM. The S record data must be within the range $8000 – $F7FF.

E.3.3 (L)oadEE

This selection causes a bulk erase of byte-erasable EEPROM in the address range $0D00 – $0FFF. After the erase operation, a verify operation checks for proper erasure of all locations. If any locations were found to contain a value other than $FF, the message `Not Erased` is displayed, and the bootloader's prompt is redisplayed.

If the erase operation was successful, the bootloader sends an ASCII * (asterisk character) to the host computer, indicating that it is ready to receive an S record. The host then sends a single S record and waits for the * prompt from the bootloader before sending the next S record.

This process is repeated until the bootloader receives an end-of-file (S9) record from the host computer. If no S9 record is received, the bootloader continues to wait for another S record indefinitely. In this situation, the EVB must be reset to return to the bootloader's prompt. (S records already loaded into EEPROM are unaffected by the missing S9 record; reprogramming is not necessary.)

In case of errors during the `(L)oadEE` procedure, repeat the process several times. If the errors persist, it is possible that the MCU may be damaged.

E.4 Vector Jump Table: Interrupt and Reset Addresses

The CPU's interrupt and reset vectors are located in the erase-protected area of FLASH EEPROM and thus cannot be reprogrammed with the S record bootloader.

To allow the user code to specify interrupt and reset addresses, each member of the erase-protected vector table starting at address $FFC0 contains a *pointer* to a vector jump table, which is located in user-programmable FLASH EEPROM starting at address $F7C0.

Each entry in the vector jump table occupies two bytes of memory, which is adequate for the addresses of user reset and interrupt service routines. The interrupt vector mapping is shown in **Table E-1**.

Table E-1. Vector Jump Table

Vector Address	CPU Interrupt	Jump Table Address
$FFC0–FFCF	Reserved	$F7C0 – $F7CF
$FFD0	BDLC (J1850)	$F7D0
$FFD2	ATD	$F7D2
$FFD4	Reserved	$F7D4
$FFD6	SCI0	$F7D6
$FFD8	SPI	$F7D8
$FFDA	Pulse Acc. Input Edge	$F7DA
$FFDC	Pulse Acc. Overflow	$F7DC
$FFDE	Timer Overflow	$F7DE
$FFE0	Timer Channel 7	$F7E0
$FFE2	Timer Channel 6	$F7E2
$FFE4	Timer Channel 5	$F7E4
$FFE6	Timer Channel 4	$F7E6
$FFE8	Timer Channel 3	$F7E8
$FFEA	Timer Channel 2	$F7EA
$FFEC	Timer Channel 1	$F7EC
$FFEE	Timer Channel 0	$F7EE
$FFF0	Real-Time Interrupt	$F7F0
$FFF2	$\overline{\text{IRQ}}$	$F7F2
$FFF4	$\overline{\text{XIRQ}}$	$F7F4
$FFF6	$\overline{\text{SWI}}$	$F7F6
$FFF8	Illegal Opcode Trap	$F7F8
$FFFA	COP Failure Reset	$F7FA
$FFFC	Clock Mon. Fail Reset	$F7FC
$FFFE	$\overline{\text{RESET}}$	$F7FE

E.5 Reloading and Customizing D-Bug12

D-Bug12 should be reloaded into FLASH EEPROM when:

- User code has been programmed into FLASH EEPROM, and it is desired to restore D-Bug12 as the boot program
- Upgrading to a newer version of D-Bug12
- Modifying the D-Bug12 startup code or customization data

E.5.1 Obtaining D-Bug12 Upgrades

Upgrades to D-Bug12 are made available for electronic downloading. S-record files containing the latest version may be obtained from http://mot-sps.com.

E.5.2 Reloading D-Bug12

Whether reloading D-Bug12 from an upgrade file or from the file shipped with the EVB package (on the IASM12 diskette), the S-record file requires editing before programming it into FLASH EEPROM. This is necessary to remove the S records containing the bootloader and vectors, which reside in erase-protected areas of FLASH EEPROM. Failure to remove them will cause errors.

Using a text editor, prepare the D-Bug12 S-record file as follows:

1. Search for the S-record line that begins with S123FC00.
2. Delete this line and *all* remaining S records *except for* the last line in the file, which is the S9 end-of-file record. This removes the bootloader program and vector table from the file.
3. Make sure that no blank lines remain in the file, as they may cause the loading process to fail.

The S record file may now be programmed into FLASH EEPROM, using the E and P bootloader procedures described in **E.3 Serial S-Record Bootloader**.

E.5.3 Customizing D-Bug12

Two areas within D-Bug12 may be customized by the user:

- Customization data — Located from $F6C0 – $F6FF. This area contains default data parameters that D-Bug12 uses for device initialization (for instance, baud rate for the communications interface).
- Startup code — Located from $F700 – $F77F. This area contains program code used by D-Bug12 to initialize the MC68HC912B32's hardware.

Appendix C. D-Bug12 Startup Code and **Appendix D. D-Bug12 Customization Data** contain detailed explanations and source listings for these two areas.

First, generate S record files for the new data, using an M68HC12-compatible assembler or C compiler.

Next, prepare the D-Bug12 S-record file for loading and add the customized S records to it. Using a text editor, perform these steps:

1. Search for the S-record line that begins with S123FC00.
2. Delete this line and *all* remaining S records *except for* the last line in the file, which is the S9 end-of-file record. This removes the bootloader program and vector table from the file.
3. Search for the S-record line that begins with S120F6C0. *Replace* this line with the S record containing the new customization data.
4. Search for the S-record line that begins with S123F700. *Replace* this line *and* the next one, S11FF720, with the S records containing the new startup code.
5. Make sure that no blank lines remain in the file, as they may cause the loading process to fail.

The S-record file may now be programmed into FLASH EEPROM using the E and P bootloader procedures described in **E.3 Serial S-Record Bootloader**.

Glossary

8-bit MCU — A microcontroller whose data is communicated over a data bus made up of eight separate data conductors.

assembler — A software program that translates source code mnemonics into opcodes that can then be loaded into the memory of a microcontroller.

assembly language — Instruction mnemonics and assembler directives that are meaningful to programmers and can be translated into an object code program that a microcontroller understands. The CPU uses opcodes and binary numbers to specify the operations that make up a computer program. Humans use assembly language mnemonics to represent instructions. Assembler directives provide additional information such as the starting memory location for a program. Labels are used to indicate an address or binary value.

ASCII — American Standard Code for Information Interchange. A widely accepted correlation between alphabetic and numeric characters and specific 7-bit binary numbers.

breakpoint — During debugging of a program, it is useful to run instructions until the CPU gets to a specific place in the program, and then enter a debugger program. A breakpoint is established at the desired address by temporarily substituting a software interrupt (SWI) instruction for the instruction at that address. In response to the SWI, control is passed to a debugging program.

byte — A set of exactly eight binary bits.

clock — A square wave signal that is used to sequence events in a computer.

command set — The command set of a CPU is the set of all operations that the CPU knows how to perform. One way to represent an instruction set is with a set of shorthand mnemonics such as LDA meaning load A. Another representation of an instruction set is the opcodes that are recognized by the CPU.

Glossary

CPU — Central processor unit. The part of a computer that controls execution of instructions.

CPU cycles — A CPU clock cycle is one period of the internal bus-rate clock. Normally, this clock is derived by dividing a crystal oscillator source by two or more so the high and low times will be equal. The length of time required to execute an instruction is measured in CPU clock cycles.

CPU registers — Memory locations that are wired directly into the CPU logic instead of being part of the addressable memory map. The CPU always has direct access to the information in these registers. The CPU registers in an MC68HC908 are A (8-bit accumulator), X (8-bit index register), CCR (condition code register containing the H, I, N, Z, and C bits), SP (stack pointer), and PC (program counter).

cycles — See CPU cycles.

data bus — A set of conductors that are used to convey binary information from a CPU to a memory location or from a memory location to a CPU.

development tools — Software or hardware devices used to develop computer programs and application hardware. Examples of software development tools include text editors, assemblers, debug monitors, and simulators. Examples of hardware development tools include simulators, logic analyzers, and PROM programmers.

EPROM — Erasable, programmable read-only memory. A non-volatile type of memory that can be erased by exposure to an ultra-violet light source. MCUs that have EPROM are easily recognized by their packaging: a quartz window allows exposure to UV light. If an EPROM MCU is packaged in an opaque plastic package, it is termed a one-time-programmable OTP MCU, since there is no way to erase and rewrite the EPROM.

EEPROM — Electrically erasable, programmable read-only memory.

input-output (I/O) — Interfaces between a computer system and the external world. For example, a CPU reads an input to sense the level of an external signal and writes to an output to change the level on an external signal.

instructions — Instructions are operations that a CPU can perform. Instructions are expressed by programmers as assembly language mnemonics. A CPU interprets an opcode and its associated operand(s) as an instruction.

listing — A program listing shows the binary numbers that the CPU needs alongside the assembly language statements that the programmer wrote. The listing is generated by an assembler in the process of translating assembly language source statements into the binary information that the CPU needs.

LSB — Least significant bit.

MCU – Microcontroller unit — Microcontroller. A complete computer system including CPU, memory, clock oscillator, and I/O on a single integrated circuit.

MSB — Most significant bit.

object code file — A text file containing numbers that represent the binary opcodes and data of a computer program. An object code file can be used to load binary information into a computer system. Motorola uses the S-record file format for object code files.

operand — An input value to a logical or mathematical operation.

opcode — A binary code that instructs the CPU to do a specific operation in a specific way.

OTPROM — A non-volatile type of memory that can be programmed but cannot be erased. An OTPROM is an EPROM MCU that is packaged in an opaque plastic package. It is called a one-time-programmable MCU because there is no way to expose the EPROM to a UV light.

program counter — The CPU register that holds the address of the next instruction or operand that the CPU will use.

RAM — Random access memory. Any RAM location can be read or written by the CPU. The contents of a RAM memory location remain valid until the CPU writes a different value or until power is turned off.

registers — Memory locations that are wired directly into the CPU logic

instead of being part of the addressable memory map. The CPU always has direct access to the information in these registers. Memory locations that hold status and control information for on-chip peripherals are called I/O and control registers.

reset — Reset is used to force a computer system to a known starting point and to force on-chip peripherals to known starting conditions.

S record — A Motorola standard format used for object code files.

simulator — A computer program that copies the behavior of a real MCU.

source code — See source program.

source program — A text file containing instruction mnemonics, labels, comments, and assembler directives. The source file is processed by an assembler to produce a composite listing and an object file representation of the program.

stack pointer — A CPU register that holds the address of the next available storage location on the stack.

TTL — Transistor-to-transistor logic.

V_{DD} — The positive power supply to a microcontroller (typically 5 volts dc).

V_{SS} — The 0-volt dc power supply return for a microcontroller.

Word — A group of binary bits. Some larger computers consider a set of 16 bits to be a word but this is not a universal standard.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Index

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution, P.O. Box 5405, Denver, Colorado 80217. 1-303-675-2140 or 1-800-441-2447. Customer Focus Center, 1-800-521-6274

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3-20-1, Minami-Azabu, Minato-ku, Tokyo 106-8573 Japan. 81-3-3440-8573

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852-26668334

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/; TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: http://motorola.com/sps/

MOTOROLA

M68HC12B32EVBUM/D