Reference Guide
For
D–Bug12 Version 4.x.x

A Debug Monitor
For
The MC9S12DP256 Microcontroller

Written By
Gordon Doughman
Field Applications Engineer
Software Specialist

1.0 Introduction

D–Bug12 has undergone considerable revision since the introduction of version 1.0.x for the M68HC812A4 EVB and version 2.x.x for the MC68HC912B32 EVB. Version 4.x.x of D–Bug12 was developed for the MC9S12DP256 to provide an economical yet powerful debugging tool that can be utilized to develop M68HC12 applications or simply to evaluate the M68HC12 architecture. The MC9S12DP256 was chosen for this latest version of D-Bug12 to provide more Flash memory for additional features and programming support for new M68HC12 Flash memory technologies. While this new version has the capability to run as a simple ROM monitor in an MC9S12DP256 EVB, most of the improvements are related to D–Bug12's operation in 'POD' mode. In this operating mode, D–Bug12 communicates with a target M68HC12 microcontroller through the Single–Wire Background Debug Mode (BDM) Interface to allow true emulation of an application in the target microcontroller's operating environment.

Target MCU Flash and EEPROM programming support has been dramatically improved, especially for those family members utilizing their on-chip PLLs. D-Bug12 now supports buffered, interrupt driven SCI communications utilizing XOn/XOff software handshaking for all serial communications. This feature allows D-Bug12 to continue receiving S-Record data from the host computer while it is sending data from the previously received S-Record to the target MCU.

Version 4.x.x of D-Bug12 utilizes the same variable speed BDM communications primitives as version 2.1.x allowing the target MCU to be operated with a bus clock between 16.384 kHz and the bus frequency of the MC9S12DP256 on the EVB (normally 24.0 MHz).

Note: It is strongly recommended that current D-Bug12 users read Section 2 and 3 of this reference guide. Section 2 explains the new features in version 4.x.x. Section 3 explains the new requirements for the terminal emulator program used with D-Bug12.

2.0 New Features

In addition to several new commands that have been added to D-Bug12, many of the it's commands have been updated or enhanced. The following sections describe the updated features of D-Bug12's command set.

- Supports on-chip hardware breakpoint modules. Including the UDR HC12 and Star12 products providing two program only hardware breakpoints.
- EVB or target hardware breakpoints are enabled by default. Software breakpoints may be enabled by using the USEHBR command.
- FBulk command supports Motorola's newly specified erase pulse timing of 10 mS for UDR M68HC12 devices.
- Improved target memory read and write routines supporting aligned word access of 16-bit wide memory and peripherals.
- D–Bug12 utilizes the XIRQ interrupt input for a program abort function when operating in EVB mode.
- Maximum S-Record code/data field length was increased to 64 bytes to support programming of M68HC12 family members utilizing SST Flash.
- Support for all MC9S12DP256 interrupt vectors when operating in EVB mode.
- Improved VERF command reports the target & S-Record data when two memory locations don't match.
- Arithmetic expressions involving CPU register names and numeric constants are permitted in place of a simple hexadecimal address for most commands.
- The TCONFIG command can be used to configure target hardware, typically I/O ports that control V_{FP} circuitry, before erasing or programming target Flash memory.
- UPLOAD command improved to allow S-Record target memory display for M68HC12 family members containing more than 64K of flash memory.
- DEVICE command can no longer be used to specify a new target MCU device.
- Disassembler now displays indexed addressing modes using the program counter with a 5-, 9-, or 16-bit offset as: `<AbsoluteAddress>,PCR`.
- Target EEPROM may be programmed using the LOAD command without changing D–Bug12's baud rate.
- For devices containing more than 64K of address space, the value of the PPAGE register can be changed using the PP command.
- Improved support for expanded addressing for HC12 and Star12 devices containing more than 64K of program memory.

- The PCALL command, similar in operation to the CALL command, was added to allow subroutines ending with the RTC instruction to be executed from the command line.

- The FLOAD, VERF and LOAD commands have been enhanced to accept either linear or 'banked' S-Records.

2.1 Hardware Breakpoint Support

Earlier versions of D–Bug12 supported 10 software breakpoints allowing developers to halt program execution on instruction opcode boundaries. Unfortunately, the placement of software breakpoints are restricted to programs that reside in alterable memory. This restriction is not a problem for small programs placed in the on-chip RAM or EEPROM when operating the EVB in EVB mode. However, when the EVB is utilized in POD mode to test and debug code in a target M68HC12's Flash, software breakpoints cannot be used.

To facilitate debugging in an M68HC12's on-chip flash, many M68HC12 family members include an on-chip hardware breakpoint module. D–Bug12 supports the hardware breakpoint module by providing two hardware breakpoints in place of the 10 software breakpoints. Even though the breakpoint module is capable of providing data access breakpoints, D–Bug12 currently supports only the module's dual address program breakpoint operating mode. In this operating mode, the hardware breakpoints utilize the CPU12's instruction tagging mechanism. Much like software breakpoints, this restricts the placement of the hardware breakpoints to the address of an instruction opcode.

The hardware breakpoints may be used in both POD and EVB operating modes. Utilizing the hardware breakpoints in EVB mode is especially important when developing code in the on-chip EEPROM. The hardware breakpoints prevent D–Bug12 from erasing and reprogramming the EEPROM each time an instruction is traced or breakpoints are placed in memory.

D-Bug12's 10 software breakpoints are still available to the programmer, however, the default operating mode uses the two hardware breakpoints. Refer to the USEHBR command documentation for details on changing the breakpoint operating mode.

2.2 FBULK Erase Pulse Time Reduced for UDR Flash Devices

Motorola has recently made a change to the erase pulse timing specification, t_{EPULSE}, reducing it from a nominal value of 100 mS to a nominal value of 10 mS. The FBULK command has been modified to reflect this change. This change applies only to supported 'UDR 1.5T' family Flash devices. M68HC12 family devices utilizing the SST Flash technology ('A' suffix devices) and the Star12 devices are supported with the appropriate erase and programming algorithms.

2.3 Additional Flash Programming Support

The FLOAD, FBULK, VERIFY and DEVICE commands have been enhanced to support on-chip Flash programming for additional M68HC12 family members - the MC68HC912D60, MC68HC912DA/DG128, MC68HC912GA32, MC68HC912KD/K128, MC68HC912DP256 and the MC68HC912DT128A. In addition, the LOAD command, which supports loading of S-Records into RAM, supports the extended memory space of the MC68HC812A4. For details of the S–Record format required for parts supporting greater than 64K bytes of program memory, refer to the LOAD, FLOAD and VERIFY commands.

Note: The ‘A’ suffix devices, such as the MC68HC912DT128A, utilize Flash memory technology licensed from SST Corporation. Unlike the ‘UDR 1.5T’ that may be programmed a byte or aligned word at a time, the SST Flash must be programmed 64 bytes (32 aligned words) at a time. To program ‘A’ series devices requires S-Records that have a code/data field of EXACTLY 64 bytes and the load address must begin on a 64 byte boundary. For compilers or assemblers that do not have the capability to produce S–Records in this format, a utility named `SRecCvt.exe` supplied with D-Bug12, may be used to reformat any S–Record file.

Note: The Star12 devices, such as the MC9S12DP256, utilize Flash memory technology licensed from SST Corporation. Unlike the ‘UDR 1.5T’ that may be programmed a byte or aligned word at a time, the SST Flash must be programmed an aligned word at a time. To programming these devices requires S-Records that have a code/data field consisting of an even number of bytes. In addition, the load address must begin on an even byte boundary. For compilers or assemblers that do not have the capability to produce S–Records in this format, a utility named `SRecCvt.exe` supplied with D-Bug12, may be used to reformat any S–Record file.

Note: Please refer to the section titled “FLOAD, LOAD and VERIFY S-Record Format” at the end of this document for a complete description of the S-Record Format utilized by these commands for M68HC12 devices supporting more than 64K bytes of memory.

2.4 16-bit Aligned Target Memory Access Supported

All versions of D–Bug12 prior to 2.1.x access memory a byte at a time through low–level drivers. Because all on-chip memory modules support byte access, utilizing this method simplified the low level driver code. However, this access method presents some potential problems for 16-bit registers that reside in the on-chip peripherals. Because the data bus connection to the on-chip peripherals is 16-bits wide, with a few exceptions, the peripherals are designed in such a way that 16-bit registers must be read or written with a single 16-bit access to ensure data coherency.

D–Bug12’s low level memory access drivers have been rewritten to perform aligned word reads whenever possible. For instance, if the Memory Modify Word (MMW) command is used with an even address, all reads and writes will be performed as aligned word accesses. However, if the MMW command is used with an odd address, each memory access will be performed as two individual byte read or write operations. Because the Memory Display commands (MD and MDW) always display an even multiple of 16 bytes, all memory read operations are performed as aligned word accesses.

2.5 XIRQ Interrupt Usable As Program Abort Input

When testing and debugging programs that reside in the internal RAM or EEPROM of the MC9S12DP256 when operating in EVB mode, it is possible for the program to become ‘hung-up’ and never return to the D–Bug12 prompt. In these cases, it is desirable to terminate user program execution and return control to D–Bug12. Unfortunately, pressing the reset switch, S1, causes a

reinitialization of D–Bug12 resulting in a complete loss of information about the state of the executing user code. D–Bug12 utilizes the XIRQ interrupt input to terminate the execution of a user program and return control to D-Bug12. Even though a program abort switch is not present on the MC9S12DP256 EVB, the XIRQ interrupt input (PE0) may be utilized for a program abort function. One side of a normally open, momentary contact push button can be wired to the XIRQ input, the other side of the push button should be wired to Vss.

Utilizing the program abort function will return control back to D–Bug12, displaying the CPU register contents at the point where the users program was terminated.

2.6 MC9S12DP256 Interrupt Vector Support

When D-Bug12 is operated in EVB mode, it provides default interrupt handlers for all of the on–chip peripherals. Version 4.x.x now fully supports the use of the MC9S12DP256 as a host CPU by providing default interrupt handlers for all of the MC9S12DP256's interrupt vectors.

2.7 Maximum S-Record Code/Data Field Length Increased

To support the programming of the M68HC12 'A' suffix devices that utilize Flash memory technology licensed from SST Corporation, the maximum S-Record code/data field length was increased to 64 bytes.

2.8 Improved VERF Command

In previous versions of D-Bug12, the VERF command terminated execution with an error message when target memory contents did not match the received S-Record. To improve support for debugging memory related problems, the VERF command now reports the S-Record address, S-Record data, Target memory address and target memory contents for **each** target memory location that does not match the S-Record contents.

2.9 Simple Arithmetic Expressions

Many of D-Bug12's commands accept one or more 16-bit hexadecimal addresses as command line parameters. To simplify the debugging process, these parameters may be supplied as a 16-bit hexadecimal number **or** a simple numeric expression. The simple numeric expression may consist of one or more CPU register names (PC, X, Y, SP, A, B or D) or numeric constants separated by the addition (+) or subtraction (–) operator. The numeric constants may be supplied in one of four number bases by using one of three number base prefix characters. In expressions, hexadecimal numbers must be preceded by the dollar sign character ($), octal numbers must be preceded by the commercial at character (@) and binary numbers must be preceded by the percent character (%). Within expressions, numbers not preceded by one of the number base character designators is interpreted as a decimal number.

Using numeric expressions in command line parameters can be advantageous in a number of debugging circumstances. For example, when using the Trace command (T) to step through program code, it is often desirable to skip execution of subroutines that are known to function properly. This can be easily accomplished using a simple numeric expression in the operand field of the GoTill (GT) command, setting a temporary breakpoint at the instruction following a `JSR`. In this instance entering the command line `GT PC+3` places a temporary breakpoint at the current location of the Program counter plus three, the length of the `JSR` instruction utilizing extended addressing. Obviously, `JSR` instructions using other addressing modes or `BSR` instructions would require an offset other than three.

Another important use of expressions in command line parameters is for examining data accessed with a reference to one or more of the CPU12's index address registers. This can be especially advantageous for instructions such as the MOVB or MOVW instructions where the data does not pass through one of the CPU registers. Consider the case where a MOVW instruction is used to move data from one data table to another utilizing accumulator offset indexed for both the source and destination address (`MOVW A,X,B,Y`). To examine or change the data at the destination address either before or after the execution of the MOVW instruction, entering the command line `MMW Y+B` would display the data at the destination address without having to perform any hexadecimal arithmetic. In a similar situation where data in a stack frame is accessed relative to the stack pointer, using an expression for the memory display (MD) or memory modify (MM) command line parameter can greatly simplify the task of examining passed parameters or local variables.

Note: Simple numeric expressions may not contain space characters.

2.10 TCONFIG Command

Some target systems contain their own V_{FP} generation circuitry to allow in–circuit programming via CAN, J1850 or even the SCI. For these systems, it is desirable to utilize the target V_{FP} generation circuitry for programming of the on-chip Flash rather than an externally supplied programming voltage. In most cases, the application of the target generated programming voltage to the V_{FP} pin is controlled by one or more I/O pins of the target M68HC12. The TCONFIG command can be used to specify up to eight one byte values that will be written to the target memory just before the execution of the FBULK and FLOAD commands. For additional details on the syntax and features of the TCONFIG command please refer to the detailed command description in Section 5.

2.11 UPLOAD Command Improvements

In earlier versions of D-Bug12, the UPLOAD command did not support the display of memory contents for devices containing more than 64K of Flash memory. The UPLOAD command now supports devices containing more than 64K of Flash memory. In addition, a command line option has been added that allows the size of the S-Record code/data field to be specified. For additional details on the syntax and features of the UPLOAD command refer to the detailed command description in Section 5.

2.12 DEVICE Command Changes

Version 2.x.x of D-Bug12 allowed new MCU device definitions to be added to D-Bug12's device table. This feature was initially provided to allow support for devices that utilized the UDR 1.5T Flash technology with less than 64K of Flash memory. Because the M68HC12 family now utilizes four different Flash memory technologies and has planned devices with up to 256K of Flash memory, it is not practical to support new family members by simply defining new devices using the DEVICE command. Instead, the D-Bug12 device table contained in Flash and may not be altered. As new M68HC12 family members are developed, new versions of D-Bug12 will be released to support Flash and EEPROM programming of the new devices.

2.13 Disassembler Improvements

When writing position independent code, it is necessary to access global variables and/or data utilizing program counter relative indexed addressing. Earlier versions of D-Bug12 disassembled instructions using this form of indexed addressing as: `<signed offset>,PC`. This form of disassembly made debugging position independent code difficult because the signed offset to a memory location changes depending where it is referenced within a program. To make debugging position independent code easier, instructions using the program counter as an index register are now disassembled as: `<AbsoluteAddress>,PCR`. The absolute address is displayed as a four digit hexadecimal number preceded by a dollar sign ($).

2.14 PP Command

The PPAGE register is present on M68HC12 family devices that have a program memory space greater than 64K. The PPAGE register, located in the I/O register block, is used in conjunction with the CALL and RTC instructions to address expanded program memory through the memory expansion window ($8000 - $BFFF). Because the PPAGE register is located at a different I/O register address in various M68HC12 family members, the PP command can be used to change its value. See Section 5 for a complete description of the PP command.

2.15 Expanded Addressing Support

D-Bug12 now fully supports the expanded address space of M68HC12 family members containing more than 64K of program memory. The display of register values now includes the PPAGE register if the selected device contains more than 64K of program memory. The new register display format is shown below. Note that the disassembly line displays the characters '`xx`' in place of the PPAGE register value if the program counter value is outside the range of the PPAGE window ($8000 - $BFFF) indicating that the PPAGE register value is not pertinent to program execution in this memory range.

```
PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
30 C00A  4000  0000  0000     00:00        1101 0000
xx:C00A  CF3FFF        LDS   #$3FFF
```

In addition to the new register display and disassembly format, the Go (G), GoTill (GT), ASM, BR and NOBR commands accept an additional address format that includes a PPAGE value. The expanded address format consists of an 8-bit PPAGE number and a 16-bit PPAGE window address separated by the colon character (':'). The general format of an expanded address is:

`<PPAGENum>:<PPAGEWinAddr>`

Both the PPAGE number and the PPAGE window address may consist of a simple expression. `<PPAGENum>` must be in a valid range for the selected device and `<PPAGEWinAddr>` must be an address in the PPAGE window ($8000 - $BFFF).

Note: Because the GoTill (GT), BR and NOBR commands involves setting breakpoints, the target MCU must contain a breakpoint module that supports the PPAGE number as part of the address comparison. The original UDR M68HC12 family members do not posses this capability. Alternatively, if the UDR target system contains RAM in place of the Flash (such as an EVS system), the GoTill (GT), BR and NOBR commands may be used in conjunction with software breakpoints (see the USEHBR command description in Section 5).

2.16 PCALL Command

The PCALL command is used to execute a subroutine ending with an RTC instruction, returning to the D-Bug12 monitor program when the final RTC of the subroutine is executed. For complete information, see the PCALL command description in Section 5.

2.17 FLOAD, VERF and LOAD command Enhancements

The FLOAD, VERF and LOAD commands

3.0 Terminal Communications Setup

Version 4.x.x of D-Bug12 requires a host terminal program that supports XOn/XOff software handshaking for proper operation. Many popular terminal emulation programs for the Windows™ operating system meet this requirement. However, because the HyperTerminal terminal emulation program is supplied with Windows 95/98/NT it is recommended for use with D-Bug12. The factory configured default communications parameters used by D-Bug12 are 9600 baud, eight data bits, one stop bit, XOn/XOff handshaking and no parity.

When configured for EVB mode, the communications parameters will revert to the factory default settings whenever the board is powered down or reset. However, when operating the EVB in POD mode, baud rate changes made using the BAUD command are stored in the MC9S12DP256's on-chip EEPROM, making the entered baud rate the new default communication rate. If communication cannot be established with the EVB and random characters are displayed on the terminal screen, it is possible that the baud rate was changed to something other than the current settings of the terminal program.

If attempting communications at various baud rates does not result in D-Bug12's prompt being displayed, the EVB's on-chip EEPROM should be erased to reconfigure the baud rate to the factory default of 9600. This task can be accomplished by configuring the EVB for EVB mode, setting the terminal emulator program to 9600 baud and entering the BULK command on the command line. If communication cannot be established when the EVB is configured for EVB mode, check all communications and power connections to the EVB board.

3.1 Configuring HyperTerminal

The HyperTerminal terminal emulation program is supplied with Windows 95/98/NT and is recommended for use with D-Bug12 version 4.x.x. For those not familiar with HyperTerminal, this section describes the setup procedure necessary to use HyperTerminal with D-Bug12. For most Windows configurations, a short cut to the HyperTerminal program or the HyperTerminal folder can be found in the Start menu under Programs/Accessories/Communications. If a short cut for the HyperTerminal program or folder does not exist in the communications menu, use the Find item in the Start menu to search for the program.

After locating the HyperTerminal program or short cut icon, double click on the icon to start the program. When presented with the Configuration Dialog click cancel to dismiss the dialog box. From the **File menu** select **Properties**. Make sure that the **Connect To** tab is selected. From the **Connect using** drop down list, select a direct connection using one of the computers COM ports. After selecting the com port, click on the **Configure…** button to reveal the Properties Dialog. Use the drop down lists to set the baud rate to 9600 (factory default), one stop bit, no parity and XOn/XOff handshaking. Click the OK button in the Properties and connection dialog box to confirm the settings.

In the Call menu select the Call item to establish a connection through the selected COM port to the EVB. Pressing the reset button on the EVB should display one of the D-Bug12 sign-on messages and prompt. If nothing appears on the screen, check the connection between the computer and EVB to ensure that the EVB is connected to the proper serial port. If random characters appear on the screen, it is most likely that an incorrect baud rate was selected.

Note: Before selecting a new baud rate, HyperTerminal must be disconnected from the communication source by selecting **Disconnect** from the **Call** menu. Failure to disconnect from the communication source will cause HyperTerminal to ignore any new port settings.

4.0 Operating Modes

The D-Bug12 firmware has four operating modes controlled by the logic level present on the PAD0 and PAD1 pins at power up or reset. The operating modes for the four logic level combinations are presented in Figure 1.

PAD1	PAD0	OperatingMode
0	0	D-Bug12;EVB
0	1	Jump to internal EEPROM
1	0	D-Bug12;POD
1	1	SerialBootloader

Figure 1, D-Bug12's Operating Modes

4.1 EVB Mode

In D-Bug12's 'EVB' mode the monitor firmware operates as a ROM resident monitor/debugger executing from the MC9S12DP256's internal Flash. While this mode provides an excellent environment for silicon evaluation, a stable evaluation environment for testing new algorithms or conducting performance benchmarks, it does have some limitations. Because the monitor/debugger program executes out of the MC9S12DP256's internal memory, the Flash memory, 1024 bytes of the on-chip RAM, and one of the the SCI serial ports are not available to the developer. As shown in Figure 2, D-Bug12 runs on the target system.

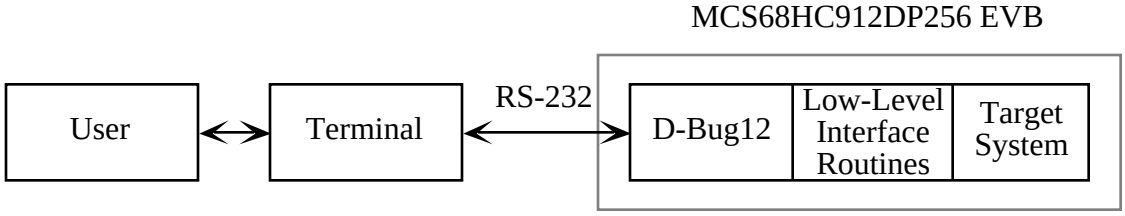

Figure 2, EVB Mode Conceptual Model

When operating in the 'EVB' mode, D-Bug12 is not capable of supporting true emulation of a target system. However, programs may be downloaded and executed from the 3072 bytes of the on-chip EEPROM or the portion of on-chip RAM not utilized by D-Bug12. The portion of the RAM that may be used by developer programs begins at $1000 and ends at $3BFF. D-Bug12 utilizes the remainder of the RAM which begins at $3C00 and extends through $3FFF.

After selecting the EVB operating mode and applying power or pressing the reset button, the sign on message in Figure 3 will be displayed on the screen:

```
D-Bug12 v4.0.0
Copyright 1996 - 2000 Motorola Semiconductor
For Commands type "Help"

>
```

Figure 3, EVB Mode Sign on Message

D-Bug12 displays the ASCII greater than character (>) indicating that it is ready to accept a command. When issuing a command that causes a program to run from the internal RAM or EEPROM, D-Bug12 will place the terminal cursor on a blank line where it will remain until control is returned to D-Bug12. If a running program fails to return to D-Bug12, pressing the EVB's reset switch will cause the running program to halt execution and initiate the D-Bug12 initialization sequence. Using this method to regain control of an executing program fails to report any information to the programmer on why or how the program may have failed.

Alternately, if an optional S.P.S.T. normally open switch has been wired to the XIRQ interrupt input pin, pressing it generates an XIRQ interrupt that causes the running program to halt execution and returns control back to D-Bug12 where the CPU register contents are displayed.

4.2 Interrupts in EVB Mode

D-Bug12 contains default interrupt handlers for all of the implemented MC9S12DP256 interrupt vectors. However, to allow a programmer to utilize peripherals in an interrupt driven manner, a RAM based interrupt vector table is provided by D-Bug12. Each of the 64 entries in the table consists of a two byte address with the table beginning at $3E00. Initially, all entries in the table have an address of $0000. Storing a value other than $0000 in any of the RAM interrupt vector table entries causes execution of the interrupt service routine pointed to by the address when an associated interrupt occurs. The user supplied interrupt service routine must end with an `RTI` instruction to ensure that execution of the main program continues where it was interrupted.

If an unmasked interrupt occurs and a table entry contains the default address of $0000, program execution is returned to D-Bug12 where a message is displayed indicating the source of the interrupt and displays the CPU registers at the point where the program was interrupted.

Figure 4 shows the correspondence between the interrupt source and the two byte RAM interrupt vector. Note that even though there is an entry in the table for SCI0, replacing the default value with the address of an interrupt service routine will be ignored since D-Bug12 requires the SCI0 for all of its communication.

Interrupt Source	RAM Vector Address	Interrupt Source	RAM Vector Address
Reserved $FF80	$3E00	IIC Bus	$3E40
Reserved $FF82	$3E02	DLC	$3E42
Reserved $FF84	$3E04	SCME	$3E44
Reserved $FF86	$3E06	CRG Lock	$3E46
Reserved $FF88	$3E08	Pulse Accumulator B Overflow	$3E48
Reserved $FF8A	$3E0A	Modulus Down Counter Underflow	$3E4A
PWM Emergency Shutdown	$3E0C	Port H Interrupt	$3E4C
Port P Interrupt	$3E0E	Port J Interrupt	$3E4E
MSCAN 4 Transmit	$3E10	ATD1	$3E50
MSCAN 4 Receive	$3E12	ATD0	$3E52
MSCAN 4 Errors	$3E14	SCI1	$3E54
MSCAN 4 Wake-up	$3E16	SCI0	$3E56
MSCAN 3 Transmit	$3E18	SPI0	$3E58
MSCAN 3 Receive	$3E1A	Pulse Accumulator A Input Edge	$3E5A
MSCAN 3 Errors	$3E1C	Pulse Accumulator A Overflow	$3E5C
MSCAN 3 Wake-up	$3E1E	Timer Overflow	$3E5E
MSCAN 2 Transmit	$3E20	Timer Channel 7	$3E60
MSCAN 2 Receive	$3E22	Timer Channel 6	$3E62
MSCAN 2 Errors	$3E24	Timer Channel 5	$3E64
MSCAN 2 Wake-up	$3E26	Timer Channel 4	$3E66
MSCAN 1 Transmit	$3E28	Timer Channel 3	$3E68
MSCAN 1 Receive	$3E2A	Timer Channel 2	$3E6A
MSCAN 1 Errors	$3E2C	Timer Channel 1	$3E6C
MSCAN 1 Wake-up	$3E2E	Timer Channel 0	$3E6E
MSCAN 0 Transmit	$3E30	Real Time Interrupt	$3E70
MSCAN 0 Receive	$3E32	IRQ	$3E72
MSCAN 0 Errors	$3E34	XIRQ	$3E74
MSCAN 0 Wake-up	$3E36	SWI	$3E76
Flash	$3E38	Unimplemented Instruction Trap	$3E78
EEPROM	$3E3A	N/A	$3E7A
SPI2	$3E3C	N/A	$3E7C
SPI1	$3E3E	N/A	$3E7E

Figure 4, RAM Interrupt Vector Addresses

4.3 POD Mode

In the POD operating mode, none of the MC9S12DP256's resources are available to the developer. Instead, D-Bug12 communicates with the developer's M68HC12 target system through the Single Wire Background Debug interface. This arrangement, as shown in Figure 5, allows access to a developer's target system in a non-intrusive manner. All of the target MCU's resources are available to the developer, providing a non-invasive development environment for the target system.

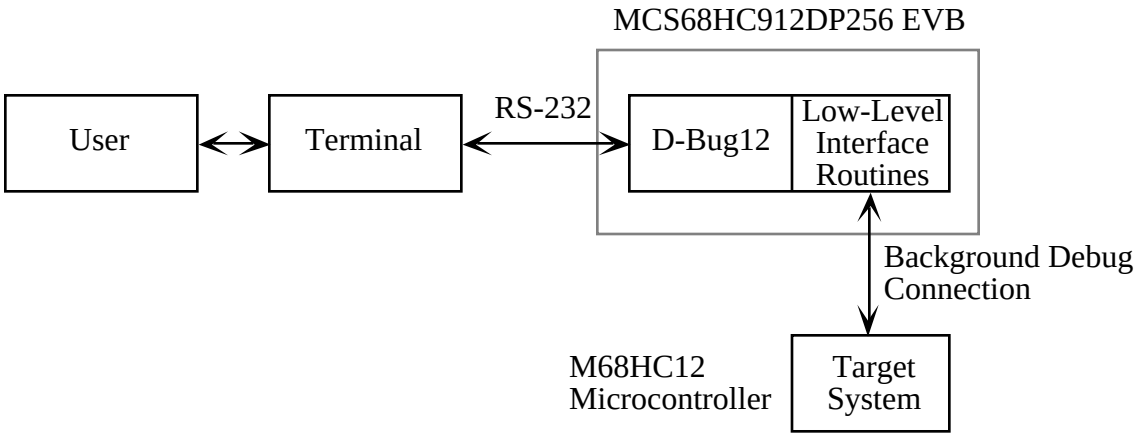

Figure 5, POD Mode Conceptual Model

On power-up or reset D–Bug12 attempts to establish communications with a target system. Initially, communications is attempted without resetting the target system. This feature allows the POD EVB to be 'hot connected' to a running system without disturbing the target microcontroller. If communications cannot be established, the message shown in Figure 6 is displayed.

```
Can't Communicate With Target CPU

1.) Set Target Speed (4000 KHz)
2.) Reset Target
3.) Reattempt Communication
4.) Erase & Unsecure
?
```

Figure 6, Failed Target Communications Prompt

Entering the number '1', '2', '3' or '4' from the keyboard allows the developer to configure D–Bug12 for an alternate target frequency, reset the target, attempt to establish communications without resetting the target M68HC12 or erase and unsecure an MC9S12 target device. Entering a character other than the choices provided will result in the target being reset and an attempt to establish communications. The frequency displayed in parenthesis is the current setting for the target crystal frequency.

Entering the number one causes the prompt shown in Figure 7 to be displayed.

```
Enter Target Crystal Frequency (KHz):
```

Figure 7, Request for Target Frequency Prompt

The entered number must be the target's crystal frequency and **not** the target's E–clock frequency. The entered frequency must be in kilohertz and not hertz. Valid target frequencies range from a low of 32 KHz to a high equal to the crystal frequency of the EVB being used as the POD. Numbers outside this range will result in an error message being displayed and cause the menu of choices to be redisplayed. Each time a valid target crystal frequency is entered, the new value is saved in the EVB's on-chip EEPROM. The saved value is used to initiate communications each time the EVB is powered-up or connected to a new target system.

Note: Because of the timing tolerance inherent in the BDM communications protocol and the implementation of the BDM firmware communications primitives, an exact value for the target crystal need not be specified. However, the entered value should be as accurate as possible. For very low frequencies, such as a 32.768 KHz crystal, a value of 32 or 33 will result in proper communication. In reality, the BDM firmware communications primitives will communicate properly with the target microcontroller even if the entered crystal frequency is as much as ± 20% different from the actual target crystal frequency.

After a valid target crystal frequency has been entered, D–Bug12 will attempt to establish communications with the target processor **without** resetting the target. If the menu of choices is redisplayed, communication could not be established. If communication cannot be established after several attempts, check for the following possible problems:

- The EVB's BDM OUT connector must be properly connected to the target systems BDM connector. If the target system is another MC9S12DP256 EVB, make sure that the POD EVB's BDM OUT connector is connected to the target EVB's BDM IN connector.

- Check for the proper orientation of the BDM cable with the BDM connectors on both the EVB and the target.

- 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, the CPU will most likely "run away", possibly executing a STOP instruction, preventing BDM communications with the target MCU. Thus it is strongly recommended that if a target system does not have firmware to execute at power-up or reset, that the target MCU be configured to operate in Special Single Chip mode.

- If the target MCU is a member of the MC9S12 family, has it's security feature enabled and the Flash and EEPROM are not blank, normal BDM communication cannot be established with the device. Option '4' must first be used to erase the Flash and EEPROM and disable security.

When communications with a target MCU is properly established, either initially or after setting the target crystal frequency, the sign on message in Figure 8 is displayed. Note that this sign on

message is identical to that of the sign on message for EVB mode except for the prompt.

```
D-Bug12 v4.0.0
Copyright 1996 - 2000 Motorola Semiconductor
For Commands type "Help"

S>
```

Figure 8, Successful POD Mode Sign on Message

When operating in the POD mode, D-Bug12 will display one of two command prompts depending on the state of the attached target system. When the target system is in active background mode (not running a user program), a two character prompt of 'S>' is displayed. The 'S' in the prompt indicates that the target is Stopped and not running a user program. When the target system is running a user program, a two character prompt of 'R>' is displayed. The 'R' indicates that the target is Running a user program.

Because the M68HC12 Single Wire Background 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.

4.3.1 The Erase & Unsecure Option

The security of a microcontroller's program and data memories has long been a concern of companies for one main reason. Because of the considerable time and money that is invested in the development of proprietary algorithms and firmware, it is extremely desirable to keep the firmware and associated data from prying eyes. The MC9S12 (Star12) family members have been designed with a device security mechanism that makes it nearly impossible to access the Flash or EEPROM contents. Once the security mechanism has been enabled, access to the Flash and EEPROM either through the BDM or the expanded bus is inhibited. Gaining access to either of these resources may only be accomplished by erasing the contents of the Flash and EEPROM or through a built in back door mechanism. While having a back door mechanism may seem to be a weakness of the security mechanism, the target application must specifically support this feature for it to operate.

When a secured device is reset in Special Single-chip mode, a special BDM security ROM becomes active. The program in this small ROM performs a blank check of the Flash and EEPROM memories. If both memory spaces are erased, the BDM firmware temporarily disables device security, allowing full BDM functionally. However, if the Flash or EEPROM are not blank, security remains active and only the BDM hardware commands remain functional. In this mode the BDM commands are restricted to reading and writing the I/O register space. Because all other BDM commands and on-chip resources are disabled, the contents of the Flash and EEPROM remain protected. This functionality is adequate to manipulate the Flash and EEPROM control registers to erase their contents.

Note: Use of the BDM interface to erase the Flash and EEPROM memories is not present in the initial mask set (0K36N) of the MC9S12DP256. Great care must be exercised to ensure that the microcontroller is not programmed in a secure state unless the back door mechanism is supported by the target firmware.

Because normal BDM communication cannot be established with a secured MC9S12 device whose Flash and/or EEPROM are not erased, the Erase and Unsecure option can be used to erase the Flash and EEPROM of a target MC9S12 device and place it in the unsecured state.

4.4 Jump to EEPROM Mode

This operating mode allows a small program (4096 bytes or less) to be executed from the on-chip EEPROM whenever the EVB is powered up or reset. D-Bug12's startup code jumps directly to address $0400 without performing any initialization of the CPU registers or peripherals. This mode provides a convenient way to execute a program in a standalone manner without having to erase and program the on-chip Flash using the Bootloader. Code and data can be programmed into the EEPROM using D-Bug12's LOAD command.

Note: The MC9S12DP256 contains 4096 bytes of small sector Flash used to 'emulate' EEPROM memory. The default address range of the EEPROM is $0000 - $0FFF. Because the default location of the I/O register block occupies the address range from $0000 - $03FF, the lower 1024 bytes of the EEPROM is not initially accessible. This is why the 'Jump to EEPROM Mode' jumps to $0400 instead of the start of the EEPROM. Note that the I/O register block may be relocated by writing to the INITRG register, thus providing access to the lower 1024 bytes of EEPROM memory.

4.5 Serial Bootloader Mode

The on-chip Flash memory includes a boot block area from $F000 - $FFFF containing an S-Record bootloader program. The bootloader can be used to erase and reprogram the remainder of on-chip Flash memory or erase the on-chip byte erasable EEPROM. The bootloader program utilizes the on-chip SCI for communications and does not require any special programming software on the host computer. The only host software required is a simple terminal program that is capable of communicating at 9600 - 115,200 baud and supports XOn/XOff handshaking.

Invoking the bootloader causes the prompt shown in Figure 9 to be displayed on the host terminal's screen.

```
HCS912DP256 Bootloader

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?
```

Figure 9, Serial Bootloader Prompt

4.5.1 Erase Flash Command

Selecting the Erase function by typing a lower case 'a' on the terminal will cause a bulk erase of all four 64K Flash arrays except for the 4k boot block in the upper 64K array where the S-Record bootloader resides. After the erase operation is completed, a verify operation is performed to ensure that all locations were properly erased. If the erase operation is successful, the bootloader's prompt is redisplayed.

If any locations were found to contain a value other than $FF, an error message is displayed on the screen and the bootloader's prompt is redisplayed. If the MC9S12DP256 device will not erase after one or two attempts the device may be damaged.

4.5.2 Program Flash Command

To increase the efficiency of the programming process, the S-Record bootloader uses interrupt driven, buffered serial I/O in conjunction with XOn/XOff software handshaking to control the flow of S-Record data from the host computer. This allows the bootloader to continue receiving S-Record data from the host computer while the data from the previously received S-Record is programmed into the Flash. The terminal program **must** support XOn/XOff handshaking to properly reprogram the MC9S12DP256's Flash memory.

Typing a lower case 'b' on the terminal causes the bootloader to enter the programming mode and wait for S-Records to be sent from the host computer. The bootloader will continue to receive and process S-Records until it receives an 'S8 or 'S9' end of file record. If the object file being sent to the bootloader does not contain an 'S8' or 'S9' record, the bootloader will not return its prompt and will continue to wait for the end of file record. Pressing the EVB's reset switch, will cause the bootloader to return to its prompt.

If a Flash memory location will not program properly, an error message is displayed on the terminal screen and the bootloader's prompt is redisplayed. If the MC9S12DP256 device will not program after one or two attempts the device may be damaged or an S-Record with a load address outside the range of the available on-chip Flash may have been received. The S-Record data must have load addresses in the range $C0000 - $FFFFF. This address range represents the upper 256K bytes of the 1MB address space of the MC9S12DP256.

4.5.3 Set Baud Rate Command

While the default communications rate of the bootloader is 9600 baud, this speed is much too slow if the majority of the MC9S12DP256's Flash is to be programmed, however, it provides the best compatibility for initial communications with most terminal programs. The Set Baud Rate command allows the bootloader communication rate to be set to one of four standard baud rates. Using a baud rate of 57,600 allows the entire 256K of flash to be programmed in just under two minutes.

Typing a lower case 'c' on the terminal causes the prompt shown in Figure 10 to be displayed on the host terminal's screen. Entering a number '1' through '4' on the keyboard will select the associated baud rate and issue a secondary prompt indicating that the terminal baud rate should be changed. After changing the terminal baud rate, pressing the enter or return key will return to the main bootloader prompt.

```
1) 9600
2) 38400
3) 57600
4) 115200
? 3
Change Terminal BR, Press Return
```

Figure 10, Change Baud Rate Prompt

4.5.4 Reloading D-Bug12

When features or enhancements are added to D-Bug12 it may be desirable to replace the version that was shipped with the MC9S12DP256 with the latest version. An S-Record file containing the current version of D-Bug12 can be obtained electronically from the Advanced Microcontroller Division Freeware Data Systems at the following locations:

- World Wide Web - ????

4.5.5 Loading User programs into Flash

While the MC9S12DP256 EVB was designed to be used with the D-Bug12 software to evaluate the MC9S12DP256 device, the board may also be used with user supplied software and hardware to prototype an embedded application. When using the board in this manner the user supplied code may occupy all of the on-chip Flash memory except the address range from $F000 - $FFFF in the fixed Flash memory page that begins at $C000. To begin execution of the users application program, PAD0 and PAD8 must both have jumpers placed in the '0' or 'OFF' position. This will cause the bootloader startup code to jump to the address in the alternate reset vector at $EFFE. When the user code is programmed into Flash, an address **MUST** be placed in the Reset vector position ($EFFE) of the alternate interrupt vector table.

4.5.6 Erasing the On-chip EEPROM

When D-Bug12 operates in POD mode, it saves various operating parameters and data in the MC9S12DP256's on-chip EEPROM. One of the parameters is the default baud rate. If communication cannot be established with the EVB and random characters are displayed on the terminal screen, it is possible that the baud rate was changed to something other than the current settings of the terminal program. If attempting communications at various baud rates does not result in D-Bug12's prompt being displayed, the EVB's on-chip EEPROM should be erased to reconfigure the baud rate to the factory default of 9600.

5.0 D-Bug12 Command Summary

The following list summarizes the D-Bug12 command set. Each command's function and command line syntax are described in detail.

- ALTCLK - Specify an alternate BDM communications rate
- ASM - Single line assembler/disassembler.
- BAUD - Set the SCI communications BAUD rate
- BF - Block Fill user memory with data.
- BR - Set/Display user breakpoints.
- BULK - Bulk erase on-chip EEPROM
- CALL - Execute a user subroutine, return to D-Bug12 when finished.
- DEVICE - Select/define a new target MCU device.
- EEBASE - Inform D-Bug12 of the target's EEPROM base address
- FBULK - Erase the target processor's on-chip Flash EEPROM
- FLOAD - Program the target processor's on-chip Flash EEPROM from S-Records
- G - Go. Begin execution of user program.
- GT - Go Till. Set a temporary breakpoint and begin execution of user program.
- HELP - Display D-Bug12 command set and command syntax.
- LOAD - Load user program in S-Record format.
- MD - Memory Display. Display memory contents in hex bytes/ASCII format.
- MDW - Memory Display Words. Display memory contents in hex words/ASCII format.
- MM - Memory Modify. Interactively examine/change memory contents.
- MMW - Memory Modify Words. Interactively examine/change memory contents.
- MOVE - Move a block of memory.
- NOBR - Remove one/all user breakpoints.
- PCALL - Execute a user subroutine in expanded memory, return to D-Bug12 when finished.
- RD - Register Display. Display the CPU register contents.
- REGBASE - Inform D-Bug12 of the target's I/O register's base address
- RESET - Reset the target CPU
- RM - Register Modify. Interactively examine/change CPU register contents.
- SO - Step over subroutine calls.
- STOP - Stop the execution of user code in the target processor and place the target processor in background mode.
- T - Trace. Execute an instruction, disassemble it, and display the CPU registers.
- TCONFIG - Configure target before erasing or programming target Flash
- UPLOAD - Display memory contents in S-Record format.
- USEHBR - Use EVB/Target Hardware breakpoints
- VERF - Verify memory contents against S-Record Data.
- <RegisterName> <RegisterValue> - Set CPU <RegisterName> to <RegisterValue>

ALTCLK - Specify An Alternate BDM Communications Rate

Command Line Format

ALTCLK [<AltBDMRate>]

Parameter Description

<AltBDMRate> - A 16-bit decimal number

Command Description

An errata was introduced in the BDM module on the MC9S12DP256 (Barracuda II) 0K79X mask set and the MC9S12H256 (Mako) 0K78X mask set. When using EXTAL ÷ 2 as the BDM clock source (default) and the PLL is selected as the bus clock source, BDM communications will be lost when the PLL multiplier is greater than 2 ((synr+1))/(refdv+1)). Once communication is lost, the only way to regain communications is to reset the target MCU.

D-Bug12 version 4.0.0b8 and later has been modified to configure the BDM to use the target bus clock (BDM Status Register CLKSW=1) if either of these parts is connected as the target device. Because the BDM interface is being driven by the target bus clock, BDM communication will be lost if the target firmware changes the bus clock frequency using the PLL. To prevent the loss of communications from disrupting a debug session, a command has been added to D-Bug12 that allows an alternate BDM communication clock frequency to be specified. If BDM communication with the target is lost, D-Bug12 will automatically attempt communication at the alternate frequency without notifying the user.

The ALTCLK command is used to specify the alternate BDM communication frequency, which should be equal to the target bus frequency with the PLL engaged as the bus clock. For example, if a 4 MHz crystal/oscillator is being used in a target application and the firmware programs the PLL to generate a 24 MHz bus clock, the ALTCLK command should be used to specify an alternate bus frequency of 24000 KHz. The ALTCLK command must be used to specify the alternate BDM communication frequency **before** executing the target code that engages the PLL as the bus clock. Note that the alternate BDM communication rate specified using the ALTCLK command is saved in D-Bug12's host MCU EEPROM so that it does not have to be reentered each time the development tool is powered up.

Entering the ALTCLK command without an alternate BDM communications frequency will display the current alternate clock setting.

Restrictions

Switching between the two BDM communications rates is completely transparent to the developer with one exception. If D-Bug12's memory modify command (MM) is used to engage the PLL as the bus clock by setting the PLLSEL bit in the CLKSEL register, D-Bug12 will report that the target memory could not be modified because of the temporary loss of communications. However, after displaying the error message, D-Bug12 will resynchronize to the new BDM communications rate and show that the target memory was properly modified.

The ALTCLK command can only be used if the MC9S12DP256 (Barracuda II) 0K79X mask set or the MC9S12H256 (Mako) 0K78X mask set device is connected as the target device.

Example

```
S>ALTCLK 24000
S>ALTCLK
Alternate BDM Clock Frequency (KHz): 24000
S>
```

ASM - Single Line Assembler/Disassembler Command

Command Line Format

ASM <Address> | <PPAGENum>:<PPAGEWinAddr>

Parameter Description

<Address> - A 16-bit hexadecimal number or simple expression
<PPAGENum> - An 8-bit hexadecimal number or simple expression
<PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression

Command Description

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 machine language 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 machine 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. By default, numeric values appearing in the operand field are interpreted as *signed* decimal numbers. Placing a $ in front of a number will cause the number to be interpreted as a hexadecimal number.

When an instruction has been disassembled and displayed, the D-Bug12 prompt is displayed following the disassembled instruction. If a carriage return is entered immediately 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 in memory. The line containing the new entry is erased and the new instruction is disassembled and displayed on the same line. The contents of the next memory location(s) is disassembled and displayed on the screen.

The instruction mnemonics and operand formats accepted by the assembler follow the syntax as described in the *M68HC12 Family CPU12 Reference Manual*.

There are a number of M68HC11 instruction mnemonics that appear in the *M68HC12 Family CPU12 Reference Manual* that do not have direct equivalent CPU12 instructions. These mnemonics, listed in the table below, are translated into functionally equivalent CPU12 instructions. To aid the current M68HC11 users that may desire continue to use 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 will calculate the twos compliment offset of the branch.

The assembly/disassembly process may be terminated by entering a period (.) following the assembler prompt.

Restrictions

None.

M68HC11 Mnemonic	CPU12 Instruction	M68HC11 Mnemonic	CPU12 Instruction
CLC	ANDCC #$FE	INS	LEAS 1,S
CLI	ANDCC #$EF	TAP	TFR A,CC
CLV	ANDCC #$FD	TPA	TFR CC,A
SEC	ORCC #$01	TSX	TFR S,X
SEI	ORCC #$10	TSY	TFR S,Y
SEV	ORCC #$02	XGDX	EXG D,X
ABX	LEAX B,X	XGDY	EXG D,Y
ABY	LEAY B,Y	SEX R_8,R_{16}	TFR R_8,R_{16}
DES	LEAS -1,S		

M68HC11 to CPU12 Instruction Translation

Example

```
>ASM 700

0700  CC1000         LDD   #4096
0703  1803123401FE   MOVW  #$1234,$01FE
0709  0EF9800001F1   BRSET $003F,PCR,#$01,$0700
070F  18FF           TRAP  $FF
0711  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 that describe the numeric format accepted for each instruction, some general rules will be 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 -32768..65535. 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, TBNE) the number entered in the address portion of the operand field must be the *absolute address of the branch destination*. The assembler will calculate the two's compliment offset to be placed in the assembled object code.

The D–Bug12 assembler allows an optional # symbol to precede the 8-bit mask value in all bit manipulation instructions (BSET, BCLR, BRSET, BRCLR).

Disassembly Operand Format

This section describes the operand format for the disassembler that is used in conjunction with the single line assembler. The operand format used by the disassembler is described separately in the *CPU12 Reference Manual*. Rather that describe the numeric format used for each instruction, some general rules will be applied. Exceptions and complicated operand formats will be described separately.

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

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

All offsets used with indexed addressing modes will be disassembled as *signed* decimal numbers with the following exception. When an instruction is disassembled utilizing the program counter as an index register, the offset field will contain an *absolute hexadecimal address* rather than a decimal offset. The address is calculated by adding the offset in the object code to the value of the program counter at the end of the instruction. Rather than displaying the index register name as 'PC' the mnemonic 'PCR' is used to indicate that the offset field contains an absolute address.

All addresses, whether direct or extended, will be disassembled as four digit hexadecimal numbers.

All 8-bit mask values (BRSET/BRCLR/ANDCC/ORCC) will be disassembled as two digit hexadecimal numbers.

For bit manipulation instructions (BSET, BCLR, BRSET, BRCLR), the disassembler always displays the # symbol preceding the 8-bit mask value.

All 8-bit immediate values will be disassembled as hexadecimal numbers.

All 16-bit immediate values will be disassembled as hexadecimal numbers.

BAUD - Change The Communications BAUD Rate

Command Line Format

BAUD <BAUDRate> [;t]

Parameter Description

<BAUDRate>	An unsigned 32-bit decimal number
;t	The ASCII string '`;t`' or '`;T`'

Command Description

The BAUD command is used to change the communications rate of the SCI that is used by D-Bug12 to communicate with the user.

Normally, the newly specified baud rate is saved in non-volitle memory so that it is used the next time the hardware running D-Bug12 is powered up or reset. The `;t` command line option may be used to make the baud rate change temporary. The next time the hardware is powered up or reset D-Bug12 will revert to the previously saved baud rate.

Restrictions

Because the <BAUDRate> parameter supplied on the command line is a 32-bit unsigned integer, BAUD rates greater than 65535 baud may be set using this command. The SCI BAUD rate divider value for the requested BAUD rate is calculated using the bus clock value that is supplied in the *CustomizationData* area. Because the SCI BAUD rate divider is a 13-bit counter, certain BAUD rates may not be supported at particular MCU clock frequencies.

Example

```
>baud 50


Invalid BAUD Rate
>baud 115200
Change Terminal BR, Press Return
>
```

BF - Fill memory with data

Command Line Format

BF <StartAddress> <EndAddress> [<Data>] [;nv]

Parameter Description

<StartAddress>	A 16-bit hexadecimal number or simple expression
<EndAddress>	A 16-bit hexadecimal number or simple expression
<Data>	An 8-bit hexadecimal number
;nv	The ASCII string ‘;nv’ or ‘;NV’

Command Description

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.

Normally the Block Fill command verifies each memory location as it is written. The ‘;nv’ option prevents the Block Fill command from verifying writes to the specified memory range. This option can be useful for testing a range of memory, especially RAM or EEPROM, for defective locations.

Restrictions

None.

Example

```
>bf 400 fff 0
>bf x x+$ff 55
>
```

BR - Set/Display User Breakpoints

Command Line Format

BR [<Address> | <PPAGENum>:<PPAGEWinAddr>…]

Parameter Description

<Address>A 16-bit hexadecimal number or simple expression
<PPAGENum> - An 8-bit hexadecimal number or simple expression
<PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression

Command Description

The BR command is used to set a 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 will disassemble the instruction at the breakpoint address, print the CPU12's register contents, and wait for the next D-Bug12 command to be entered by the user.

Breakpoints are set by entering 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.

A maximum of 10 breakpoints may be set at one time when using software breakpoints (default). A maximum of 2 breakpoints may be set when using the EVB or target CPU's hardware breakpoint capability. For additional information on D-Bug12's hardware breakpoint support, see the USEHBR command description.

Restrictions

D-Bug12 implements the software breakpoint function by replacing the opcode at the breakpoint address with an SWI instruction when operating in the EVB mode or the BGND instruction when operating in the POD mode. A breakpoint may not be set on a user SWI instruction when operating in EVB mode. In either mode breakpoints may only be set at an opcode address and breakpoints may only be placed at memory addresses implemented as RAM.

When using the on-chip hardware breakpoints, D–Bug12 utilizes the the breakpoint module in either SWI Dual Address (EVB) or BDM Dual Address (POD) mode. Both of these breakpoint module modes utilize the CPU12 instruction fetch tagging mechanism which only allows breakpoints to be set on instruction opcodes.

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

Example

```
>br 35ec 2f80 c592
Breakpoints: 35ec 2f80 c592

>br
Breakpoints: 35EC 2F80 C592

>
```

BS - Block Search, Search an Address Range For A Data Pattern

Command Line Format

BS <StartAddress> <EndAddress> '<ASCIIString>' | <Data8> [<Data8>]

Parameter Description

<StartAddress>	A 16-bit hexadecimal number or simple expression OR an expanded memory address
<EndAddress>	A 16-bit hexadecimal number or simple expression OR an expanded memory address
<ASCIIString>	An ASCII string consisting of any printable characters EXCEPT the single quote (') character
<Data8>	An 8-bit hexadecimal number or simple expression

Command Description

The block search command can be used to search an address range for a data pattern. The specified data can be supplied as a quoted ASCII string or up to eight hexadecimal bytes. If the data pattern is found in the specified memory range, the address of the first byte of the data pattern is displayed. Note that when using an ASCII string as the data pattern, only printable ASCII characters excluding the single quote (') character may be used. The search start and end addresses for parts containing less than 64K of memory are specified using a 16-bit hexadecimal number or simple expression and can span the entire 64K memory map.

Start and end addresses for parts containing 64K or more of (Flash) memory within the PPAGE window range ($8000 - $BFFF) must be specified as an expanded memory address range. The expanded address format consists of an 8-bit PPAGE number and a 16-bit PPAGE window address separated by the colon character (':'). The general format of an expanded address is:

```
<PPAGENum>:<PPAGEWinAddr>
```

Both the PPAGE number and the PPAGE window address may consist of a simple expression. `<PPAGENum>` must be in a valid range for the selected device and `<PPAGEWinAddr>` must be an address in the PPAGE window ($8000 - $BFFF).

For example, to search the entire contents of the on-chip Flash memory of the MC68HC912DA128 for the data pattern $47, $6f, $72, $64, the following command line would be used:

```
bs 00:8000 07:bfff 47 6f 72 64
```

Note that start and end addresses outside the PPAGE window address **may** include the PPAGE window in the search range. For example, using a start address of $4000 and an end address of $F000 for a part containing 64K or more of Flash would include a search of the PPAGE window. However, the data in the PPAGE window range would depend on the value in the PPAGE register at the time of the search.

Restrictions

If the memory range specified by <StartAddress> and <EndAddress> does not contain at least as many bytes as the specified data, the command will be terminated and an appropriate error message displayed.

The <StartAddress> and <EndAddress> must both be the same address type, i.e. both must be a 16-bit address or both must be an expanded address. If the address types are different, the command will be terminated and an appropriate error message displayed.

When operating in the POD mode, a block search using an expanded memory address range is not permitted when the 'R>' prompt is being displayed.

When operating in the EVB mode, a block search using an expanded memory address range is not permitted.

Example

```
S>bs 3e:8000 3f:bfff 'HCS'
Data Found at: $3F:B0AD
S>mm 30 3f
S>md b0ad

B0A0  D6 33 3D F3 - 19 F2 62 F2 - 23 F1 DD 0D - 0A 48 43 53    .3=...b.#....HCS
S>bs c000 ffff 'HCS'
Data Found at: $F0AD
S>
```

BULK - Bulk Erase on-chip EEPROM

Command Line Format

BULK

Parameter Description

No parameters are required

Command Description

The BULK command is used to erase the entire contents of the on-chip EEPROM in a single operation. After the bulk erase operation has been performed, each on-chip EEPROM location shall be checked for contents of $FF.

Restrictions

None.

Example

```
>BULK

F/EEPROM Failed To Erase
>BULK

>
```

CALL - Execute A User Subroutine

Command Line Format

CALL [<Address>]

Parameter Description

<Address> A 16-bit hexadecimal number or simple expression

Command Description

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 will be 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 will contain the starting address of the subroutine. If a subroutine address is not supplied on the command line, the current value of the Program Counter (PC) will be used as the starting address.

NOTE: No 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 D–Bug12 places four bytes of data on the users stack that causes control to return to D-Bug12 when the final RTS of the subroutine is executed. Obviously, any subroutine must obey this restriction to execute properly.

The CALL command cannot be issued when the 'R>' prompt is being 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
0820  CCFFFF        LDD   #$0FFF

>
```

DEVICE - Specify a target MCU device type

Command Line Format

```
DEVICE
DEVICE  ?
DEVICE  <DeviceName>
```

Parameter Description

<DeviceName> Maximum of 9 ASCII characters used to select a target MCU device

Command Description

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 using the RESET command. The DEVICE command has three command line formats allowing for the display and/or selection of target device parameters.

Entering "DEVICE" on the command line followed by a carriage return will display 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 the POD mode, the device command may also be used to select a new target device. Entering the DEVICE command followed only by a device name will configure D-Bug12 for operation with the selected target device. The table below shows the command line name to use for the various MCU devices.

Device Name	Target MCU
912B32	MC68HC912B32 MC68HC912BC32
812A4	MC68HC812A4
912D60	MC68HC912D60
912D60A	MC68HC912D60A
DA128	MC68HC912DA128 MC68HC912DG128
DT128A	MC68HC912DT128A MC68HC912DG128A
KD128	MC68HC912KD128 MC68HC912K128
DP256	MC9S12DP256 MC9S12DG256 MC9S12DJ256 MC9S12H256

Device Name	Target MCU
S12DB128	MC9S12DB128 MC9S12DT128 MC9S12DJ128 MC9S12DG128
GA32	MC68HC912GA32

Restrictions

When operating the MC9S12DP256 EVB in EVB mode, the DEVICE command may only be used to display the current device information.

Example

```
>device

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

S>device 812a4

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

S>device da128

Device: DA128
EEPROM: $0800 - $0FFF
Flash: $8000 - $BFFF  Pages: 8  PPAGE at: $00FF
RAM: $2000 - $3FFF
I/O Regs: $0000

S>device ?

Supported Devices:
DP256
912D60
912B32
DA128
DT128A
912D60A
GA32
KD128
S12DB128
812A4

S>
```

EEBASE - Specify the EEPROM base address

Command Line Format

EEBASE <Address>

Parameter Description

<Address> A 16-bit hexadecimal number

Command Description

Each time D-Bug12 performs a memory write, it will automatically perform 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 may be changed 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 description 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. It is the responsibility of the programmer to ensure that the INITEE register is modified either manually or through the execution of code.

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>
```

FBULK - Erase target on-chip Flash EEPROM Memory

Command Line Format

FBULK [;np]

Parameter Description

;np	The ASCII string '`;np`' or '`;NP`'

Command Description

The FBULK command is used to erase the entire contents of the on-chip Flash EEPROM of a target MCU in a single operation. After the bulk erase operation has been performed, each on-chip Flash location is checked for contents of $FF. The target processor's Flash memory 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 is lost.

The '`;np`' option is used to force the target NOT to use the target PLL during the erase or programming process. The D-Bug12 firmware and the erase drivers for the UDR (0.65u and 0.5u) HC12 parts are written to automatically detect if the PLL is enabled (the state of the Vddpll pin is reflected in the PLLON bit in the PLLCR register) and utilize the maximum bus frequency of the selected device to reduce the erase time. Because Star12 devices must have power applied to Vddpll even if the target application does not use the PLL (because Vddpll supplies power to the oscillator circuitry), the Star12 drivers have no way to detect if they may use the PLL to reduce the erase or program time.

Restrictions

When operating in the '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 will be terminated.

Before using the FBULK command, a target device must be selected (see the DEVICE command description) 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 FBULK command to fail and may require that the EVB be reset.

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.

Example

```
S>fbulk
Vfp Not Present
S>fbulk
F/EEPROM Failed To Erase
S>fbulk
S>
```

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

FLOAD - Program on-chip Flash memory from S-Records

Command Line Format

FLOAD [<AddressOffset> | ;b] [;np] [;nf]

Parameter Description

<AddressOffset>	A 32-bit hexadecimal number
;b	The ASCII string '`;b`' or '`;B`'
;np	The ASCII string '`;np`' or '`;NP`'
;nf	The ASCII string '`;nf`' or '`;NF`'

Command Description

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 an S-Record's data bytes are placed in memory. Providing an address offset other than zero allows object code or data to be programmed into memory at a location other than the address for which it was assembled or compiled. An offset greater than $FFFF may only be used with devices that support more than 64K bytes of memory.

The time required to program target on-chip Flash memory varies with the different Flash memory technologies used on the M68HC12 family. Because of this variability, D–Bug12 uses XOn/XOff handshaking to control the flow of S-Record data between the host computer and the EVB. As each S-Record is received and processed, an ASCII asterisk character (*) is sent to the screen. Note that this is only to indicate programming progress and is **NOT** used for handshaking purposes.

The FLoad command is terminated when D-Bug12 receives an 'S8' or 'S9' end of file record. If the object file being loaded does not contain an 'S8' or 'S9' record, D–Bug12 will not return its prompt and will continue to wait for the end of file record. Pressing a system Reset will return D–Bug12 to its command line prompt.

The ;b option, which is mutually exclusive with the address offset option, is used to allow direct loading of 'banked' or 'paged' S-Records produced by some compilers and assemblers.

Note: Please refer to the section titled "FLOAD, LOAD and VERIFY S-Record Format" at the end of this document for a complete description of the S-Record Formats utilized by this command for M68HC12 devices supporting more than 64K bytes of memory.

The '`;np`' option is used to force the target NOT to use the target PLL during the erase or programming process. The D-Bug12 firmware and the erase drivers for the UDR (0.65u and 0.5u) HC12 parts are written to automatically detect if the PLL is enabled (the state of the Vddpll pin is reflected in the PLLON bit in the PLLCR register) and utilize the maximum bus frequency of the selected device to reduce the erase time. Because Star12 devices must have power applied to Vddpll even if the target application does not use the PLL (because Vddpll supplies power to the oscillator circuitry), the Star12 drivers have no way to detect if they may use the PLL to reduce the erase or program time.

The ‘;nf’ option is used to prevent the ASCII asterisk character (*) from being sent to the screen as each S-Record is received.

Restrictions

Because the on-chip Flash EEPROM is only bulk erasable, the FBULK command should be used before attempting to program the Flash memory using the FLOAD command.

The FLOAD command cannot be used with target MCUs operating with crystal speeds lower than 3.0 MHz (E-clock speeds less than 1.5 MHz) unless the target MCU contains an on–chip PLL that is used in the target application.

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 code/data field longer than 32 bytes will cause D-Bug12 to terminate the FLOAD command the issue an error message.

Before using the FLOAD command, a target device must be selected (see the DEVICE command description) 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.

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.

Supplying an address offset greater than $FFFF for an M68HC12 family member that contains less than 64K of addressable program memory will result in termination of the FLOAD command and an error message being issued.

S-Record object files used with the M68HC12 ‘A’ family parts (MC68HC912D60A, MC68HC912DT128A, MC68HC912DG128A) must contain S-Records consisting of a 64 byte code/data field with a load address that begins on a 64 byte boundary. This restriction is necessary due to the programming requirements of the on-chip Flash memory. The supplied SRecCvt utility can be used to reformat S-Record files to meet these requirements.

S-Record object files used with the MC68HC912GA32 must contain only S-Records with a code/data field that is a multiple of 8 bytes and a load address that begins on an 8 byte boundary. This restriction is necessary due to the programming requirements of the on-chip Flash memory. The supplied SRecCvt utility can be used to reformat S-Record files to meet these requirements.

Example

```
S>fload
Vfp Not Present
S>fload
************************************************************************
************************************************************************
**********************
S>
```

Go, begin execution of user code

Command Line Format

G [<Address> | <PPAGENum>:<PPAGEWinAddr>]

Parameter Description

<Address>A 16-bit hexadecimal number or simple expression
<PPAGENum> - An 8-bit hexadecimal number or simple expression
<PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression

Command Description

The G command is used to begin the execution of user code in real time. Before beginning execution of user code, any breakpoints set using the BR command are placed in memory. Execution of the user program will continue until a user breakpoint is encountered, a CPU exception occurs or the reset switch on the HC12EVB is pressed. When user code halts for one of these reasons and control is returned to D-Bug12, a message is displayed explaining the reason for program termination. In addition, D-Bug12 displays the CPU12's register contents, disassembles the instruction at the current PC address, 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 Program Counter.

Restrictions

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

In EVB mode if the program counter is pointing to a CALL instruction, the Go command cannot be used to continue program execution. This limitation is due to the fact that D-Bug12 traces one instruction before continuing program execution. The CALL instruction cannot be traced because it interferes with the operation of D-Bug12.

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 Until, Execute user code until temporary breakpoint is reached

Command Line Format

GT <Address> | <PPAGENum>:<PPAGEWinAddr>

Parameter Description

<Address>A 16-bit hexadecimal number or simple expression
<PPAGENum> - An 8-bit hexadecimal number or simple expression
<PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression

Command Description

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 set by the BR command are NOT placed in the user's 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 displays the CPU12's register contents, disassembles the instruction at the current PC address, and waits for the next D-Bug12 command to be entered by the user.

Restrictions

The GT command cannot be issued when the 'R>' prompt is being 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>
```

HELP - Display D-Bug12 command summary

Command Line Format

HELP

Parameter Description

No parameters are required

Command Description

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 the command's function. The commands are listed in alphabetical order.

Restrictions

None.

Error Conditions

None.

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 breakpoints
BULK  Erase entire on-chip EEPROM contents
CALL [<Address>]  Call user subroutine at <Address>
DEVICE [<DevName>]  display/select 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 D-Bug12 command summary
LOAD [<AddressOffset>] [;f]  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 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
TCONFIG  [<Address>=<Data8>] | [DLY=<mSDelay>] | NONE  Configure Target Device
UPLOAD  <StartAddress> <EndAddress> [;f] [<SRecSize>]  S-Record Memory display
USEHBR  [ON | OFF]  Use Hardware/Software Breakpoints
VERF  [<AddressOffset>] [;f]  Verify S-Records against memory contents
<Register Name> <Register Value>  Set register contents
       Register Names: PC, SP, X, Y, A, B, D, PP
       CCR Status Bits: S, XM, H, IM, N, Z, V, C
>
```

LOAD - Load user program in S-Record format

Command Line Format

LOAD [[<AddressOffset>] [;f]] | [;b]

Parameter Description

<AddressOffset>	A 32-bit hexadecimal number
;f	The ASCII string '`;f`' or '`;F`'
;b	The ASCII string '`;b`' or '`;B`'

Command Description

The Load command is used to load S-Record object files into user memory from an external device. The address offset, if supplied, is added to the load address of each S-Record before an S-Record's data bytes are placed in memory. Providing an address offset other than zero allows object code or data to be loaded into memory at a location other than that for which it was assembled. An offset greater than $FFFF may only be used with devices that contain more than 64K bytes of memory.

Note: Please refer to the section titled "FLOAD, LOAD and VERIFY S-Record Format" at the end of this document for a complete description of the S-Record Format utilized by this command for M68HC12 devices supporting more than 64K bytes of memory.

During the loading process, the S-Record data is not echoed to the control console. However, for each ten S-Records that are successfully loaded, an ASCII asterisk character (*) is sent to the control console. When an S-Record file has been successfully loaded, D-Bug12 will issue its prompt.

The Load command is terminated when D-Bug12 receives an 'S8' or 'S9' end of file record. If the object file being loaded does not contain an 'S8' or 'S9' record, D–Bug12 will not return its prompt and will continue to wait for the end of file record. Pressing a systems Reset button will return D–Bug12 to its command line prompt.

The '`;f`' option is used to load S-Records into target memory normally occupied by on chip Flash memory for devices having a program memory space greater than 64K. This option is only required by devices that support more than 64K bytes of memory and have a device definition where the number of 16K memory pages is greater than zero. This option allows the S-Record loader to distinguish between S-Records that are to be loaded into paged program memory and those destined for other areas of on-chip or off-chip memory.

When the '`;f`' option is not used, the load command accepts only S1 S-Records. This allows program or data to be loaded into any memory locations visible in the 64K memory map, including the PPAGE window address range ($8000 - $BFFF) for devices containing more than 64K of paged program memory.

The ;b option, which is mutually exclusive with the address offset and ;f option, is used to allow direct loading of 'banked' or 'paged' S-Records produced by some compilers and assemblers.

Restrictions

None.

Example

```
>load 1000
********************
>
```

MD - Display memory in hexadecimal bytes and ASCII format

Command Line Format

MD <StartAddress> [<EndAddress>]

Parameter Description

<StartAddress>	A 16-bit hexadecimal number or simple expression
<EndAddress>	A 16-bit hexadecimal number or simple expression

Command Description

The memory display command displays the contents of memory in both hexadecimal bytes and ASCII, 16-bytes on each line. The <StartAddress> parameter must be supplied, however, 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 - 1. This causes each line to display memory in the range of $xxx0 through $xxxF. For example if the user entered $205 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..
```

MDW - Display memory in hexadecimal words and ASCII format

Command Line Format

MDW <StartAddress> [<EndAddress>]

Parameter Description

<StartAddress>	A 16-bit hexadecimal number or simple expression
<EndAddress>	A 16-bit hexadecimal number or simple expression

Command Description

The memory display command displays the contents of memory in both hexadecimal words and ASCII, 16-bytes on each line. The <StartAddress> parameter must be supplied, however, 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 - 1. This causes each line to display memory in the range of $xxx0 through $xxxF. For example if the user entered $205 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 - Modify memory bytes in hexadecimal format

Command Line Format

MM <Address> [<data>]

Parameter Description

<Address>	A 16-bit hexadecimal number or simple expression.
<data>	An 8-bit hexadecimal number.

Command Description

The memory modify word 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 at <Address> is replaced with <data>. If not, D-Bug12 will enter 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, several sub-commands are used for the modification and verification of memory contents. These sub-commands have the following 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 sub-command must be separated from any entered data with at least one space character. If an invalid sub-command character is entered, an appropriate error message will be issued and the contents of the current memory location shall be redisplayed.

Restrictions

While there are no restrictions regarding the use of the MM 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 run away.

Example

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

MMW - Modify memory words in hexadecimal format

Command Line Format

MMW <Address> [<data>]

Parameter Description

<Address>	A 16-bit hexadecimal number or simple expression
<data>	A 16-bit hexadecimal number

Command Description

The memory modify word 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 at <Address> is replaced with <data>. If not, D-Bug12 will enter 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, several sub-commands are used for the modification and verification of memory contents. These sub-commands have the following 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 sub-command must be separated from any entered data with at least one space character. If an invalid sub-command character is entered, an appropriate error message will be issued and the contents of the current memory location shall be redisplayed.

If the <Address> parameter corresponds to an even byte address, values read from and/or written to memory will be performed as aligned word accesses. This guarantees data coherency for peripherals that require a single access to their 16-bit registers.

Restrictions

While there are no restrictions regarding 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 run away.

Example

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

MOVE - Move a Block of Memory

Command Line Format

MOVE <StartAddress> <EndAddress> <DestAddress>

Parameter Description

<StartAddress>	A 16-bit hexadecimal number or simple expression
<EndAddress>	A 16-bit hexadecimal number or simple expression
<DestAddress>	A 16-bit hexadecimal number or simple expression

Command Description

The MOVE command is used to move a block of memory from one location to another a byte at a time. The number of bytes moved is one more than the <EndAddress> - <StartAddress>. The block of memory created 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 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. In addition, caution should be exercised when moving target memory while user code is running. Accidentally modifying target memory containing program code could lead to program run away.

Example

```
>move 800 8ff 1000
>
```

NOBR - Remove one/all user breakpoints

Command Line Format

NOBR [<Address> | <PPAGENum>:<PPAGEWinAddr>…]

Parameter Description

<Address>A 16-bit hexadecimal number or simple expression
<PPAGENum> - An 8-bit hexadecimal number or simple expression
<PPAGEWinAddr> - A 16-bit hexadecimal number or simple expression

Command Description

The NOBR command is used to remove one or more of 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 the POD mode, breakpoints may not be removed with the NOBR command when the 'R>' prompt is being displayed.

Example

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

>nobr 810 820
Breakpoints: 0800  0830

>nobr
All Breakpoints Removed

>
```

PCALL - Execute A User Subroutine Ending with RTC

Command Line Format

PCALL [<PPAGENum>:<PPAGEWinAddr>]

Parameter Description

<PPAGENum>	An 8-bit hexadecimal number or simple expression
<PPAGEWinAddr>	A 16-bit hexadecimal number or simple expression

Command Description

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

NOTE: No 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 RTC of the called subroutine. This restriction is required because D–Bug12 places five bytes of data on the users stack that causes control to return to D-Bug12 when the final RTC of the subroutine is executed. Obviously, any subroutine must obey this restriction to execute properly.

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

Example

```
S>pcall 30:8006

Subroutine Call Returned

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
30 8006  3FFF  0000  0000     00:00        1101 1000
30:8006  4D0101        BCLR  $0001,#$01

S>
```

RD - Display CPU12 Register Contents

Command Line Format

RD

Parameter Description

No parameters are required. Any parameters on the command line will be ignored.

Command Description

The Register Display command is used to display the CPU12's registers. The registers are displayed in the same format used when a breakpoint is encountered.

Restrictions

When operating in the POD mode, the CPU registers may not be displayed when the 'R>' prompt is being displayed.

Example

```
S>rd

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
C028  4000  0000  0000     00:00        1101 0000
C028  790016        CLR   $0016
S>device da128

Device: DA128
EEPROM: $0800 - $0FFF
Flash: $8000 - $BFFF  Pages: 8  PPAGE at: $00FF
RAM: $2000 - $3FFF
I/O Regs: $0000

S>rd

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
03 C00A  4000  0000  0000     00:00        1101 0000
xx:C00A  CF4000        LDS   #$4000
S>
```

REGBASE - Specify the Register base address

Command Line Format

REGBASE <Address>

Parameter Description

<Address> A 16-bit hexadecimal number

Command Description

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 variables `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. It is the responsibility of the user to ensure 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.

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
```

RESET - Reset the target system MCU

Command Line Format

RESET

Parameter Description

No parameters are required. Any parameters on the command line will be ignored.

Command Description

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 low for approximately 2 mS. When the reset line is released, BDM commands are are sent to the target processor to place it in active background mode. With the exception of the program counter (PC), the target processor's registers are initialized with the same values used for the registers when operating in EVB mode. The PC is initialized with the contents of the target processor's reset vector, memory locations $FFFE and $FFFF

Restrictions

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

Example

```
S>reset
Target Processor Has Been Reset
S>g 4000
R>reset
Target Processor Has Been Reset
S>
```

RM - Interactively Modify CPU12 Register Contents

Command Line Format

RM

Parameter Description

No parameters are required. Any parameters on the command line will be ignored.

Command Description

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 is displayed, D-Bug12 allows the user to enter a new value for the register in hexadecimal. If modification of the displayed register is not desired, entering a carriage return causes 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 will redisplay 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 the following order, one register per line: PC, SP, X, Y, A, B, CCR.

Restrictions

When operating in the POD mode, the CPU registers may not be modified when the 'R>' prompt is being 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 .
>
```

SO - Step Over Subroutine Calls

Command Line Format

SO

Parameter Description

No parameters are required. Any parameters on the command line are ignored.

Command Description

When tracing through code it is often unnecessary to trace through subroutines that are known to be bug free. The SO command is similar to the Trace command (T) except that subroutine calls (BSR, JSR and CALL) are traced as a single instruction. When the SO command encounters one of the subroutine call instructions, it places a temporary breakpoint at the instruction following the BSR, JSR or CALL. It then issues a Go command, executing the subroutine at full speed. All other instructions are executed the same as if the Trace command were used.

When operating in POD mode, if the subroutine requires more than a handful of cycles to execute, D-Bug12 will display its R> prompt indicating the target is running. When the subroutine returns, the temporary breakpoint is removed, the CPU12's register contents are displayed and the *next* instruction to be executed is displayed.

Restrictions

None.

Example

```
S>so

PP  PC   SP   X    Y   D = A:B   CCR = SXHI NZVC
00 1000  4000  0000  0000    FF:FE      1101 1100
xx:1000  CCFFFF      LDD   #$FFFF
S>so

PP  PC   SP   X    Y   D = A:B   CCR = SXHI NZVC
00 1003  4000  0000  0000    FF:FF      1101 1000
xx:1003  161100      JSR   $1100
S>so
R>

PP  PC   SP   X    Y   D = A:B   CCR = SXHI NZVC
00 1006  4000  0000  0000    FF:FE      1101 1100
xx:1006  A7          NOP

S>
```

STOP - Stop Execution of user code in the target MCU

Command Line Format

STOP

Parameter Description

No parameters are required. Any parameters on the command line are ignored.

Command Description

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 the '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 will be 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 (Execute) CPU12 Instruction(s)

Command Line Format

T [<Count>]

Parameter Description

<Count> An 8-bit decimal number in the range 1..255

Command Description

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 followed immediately by a carriage return.

Restrictions

When operating in 'EVB' mode using software breakpoints, all branch instructions (`Bcc`, `LBcc`, `BRSET`, `BRCLR`, `DBEQ/NE`, `IBEQ/NE`, `TBEQ/NE`) containing an offset that branches back to the instruction opcode will NOT execute because of the method used to execute a single instruction. The monitor will appear to become 'stuck' at the branch instruction and will not execute the instruction even if the condition for the branch instruction is satisfied. This limitation can be overcome by using the GT (GoTill) command to set a temporary breakpoint at the instruction following the branch instruction.

This restriction **DOES NOT** apply when using D-Bug12 on a target system in POD mode or when utilizing the hardware breakpoints of the EVB's MCU in either EVB or POD mode. See the USEHBR command for additional information on the use of hardware breakpoints.

In EVB mode the `CALL` instruction cannot be executed using the trace command because it interferes with the operation of D-Bug12.

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 2

 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

 >
```

TCONFIG - Configure target system

Command Line Format

TCONFIG <Address>=<Data> [<Address>=<Data>] [DLY=<mSDelay>]
TCONFIG NONE
TCONFIG

Parameter Description

<Address>	A 16-bit hexadecimal number.
<Data>	An 8-bit hexadecimal number.
<mSDelay>	A 16-bit unsigned decimal number.

Command Description

Some target systems contain their own V_{FP} generation circuitry to allow in–circuit programming via CAN, J1850 or even the SCI. For these systems, it is desirable to utilize the target V_{FP} generation circuitry for programming of the on-chip Flash rather than an externally supplied programming voltage. In most cases, the application of the target generated programming voltage to the V_{FP} pin is controlled by one or more I/O pins of the target M68HC12. The TCONFIG command can be used to specify up to eight one byte values that will be written to the target memory just before the execution of the FBULK and FLOAD commands.

To allow time for the target V_{FP} circuitry to stabilize after it is enabled, an optional delay between 1 and 65535 mS may be specified. The specified delay does not have to appear as the last parameter on the command line, however, a delay may not be specified without also supplying values to be written to the target memory.

Entering the TCONFIG command without supplying any parameters reports the address, data and delay time previously specified using the TCONFIG command. If no address, data and delay time have been specified, a message is displayed indicating that no data has been supplied.

To disable the function of the TCONFIG command, a single parameter ‘NONE’ is entered on the command line. When a new target device is specified using the DEVICE command, any previously entered address, data and delay information is discarded.

Restrictions

The TCONFIG command may not be used when the EVB is operating in ‘EVB’ mode. If the TCONFIG command is entered while in ‘EVB’ mode, an error message is displayed and command execution will be terminated.

Example

```
>tconfig 1=01 3=fe dly=20
>tconfig
$0001=$01 $0003=$FE Delay=20 mS
>
```

UPLOAD - Display Memory In S-Record Format

Command Line Format

UPLOAD <StartAddress> <EndAddress> [;f] [;<SRecSize>]

Parameter Description

<StartAddress>	A 32-bit hexadecimal number
<EndAddress>	A 32-bit hexadecimal number
;f	The ASCII string ‘;f’ or ‘;F’
[;<SRecSize>]	Decimal number specifying the S-Record data field length.

Command Description

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 a terminal program and saved to a disk file.

When the ‘;f’ option is not used, the upload command accepts a 16-bit <StartAddress> or <EndAddress> parameter in the range $0000 through $FFFF. This allows any program or data visible in the 64K memory map to be displayed in S-Record format, including the PPAGE window address range ($8000 - $BFFF) for devices containing more than 64K of paged program memory.

The ‘;f’ option is used to upload S-Records into target memory normally occupied by on chip Flash memory for devices having a program memory space greater than 64K. This option is only required by devices that support more than 64K bytes of memory and have a device definition where the number of 16K memory pages is greater than zero. This option allows the S-Record loader to distinguish between S-Records that are to be uploaded from paged program memory and those destined for other areas of on-chip or off-chip memory.

The optional <SRecSize> parameter may be used to specify the length of the S-Record data field. Permissible values for <SRecSize> range from 16 through 64. If the <SRecSize> parameter is not specified, the default S-Record length is 32.

Restrictions

None.

Example

```
>upload 400 4ff
S123040000F0000843FC0000F50F379F37BF43FCF50F27FA757F177AFA047504177AFA21C5
S123042037B500FF37FAFB0437B5400037FAFB061735FB0037B500C137FAFA003715379C01
S1230440F50F379D37BC012C37BD400085009A003C023D02377C0140B6EE7A0F400037B583
S1230460000337FAFA4C37FAFA5037FAFA5437B5502037FAFA4E37B5302037FAFA5237B58A
S1230480682037FAFA5637BD014037BC000095008A003C023D02377D0172B6EE37BD017259
S12304A037BC020095008A003C023D02377D018EB6EE27F937B0F50F379C37BC00CE27F901
S12304C000FC27F9104C27F90E68378000BE0A0D442D42756731362056312E3033202D20E3
S12304E04465627567204D6F6E69746F7220466F7220546865204D3638484331362046 61ED
S9030000FC
>
```

USEHBR - Use EVB/Target Hardware Breakpoints

Command Line Format

USEHBR [ON | OFF]

Parameter Description

[ON | OFF] The ASCII string '`ON`' or '`OFF`'

Command Description

Entering the USEHBR command causes D-Bug12 to use the hardware breakpoint capability of the MC9S12DP256 on the EVB, in EVB mode, or the breakpoint capability of the target microcontroller in POD mode. Using hardware breakpoints allows two, program only breakpoints to be set in Flash or other non-volatile memory. By default, D-Bug12 uses the hardware breakpoint capability of the MC9S12DP256 in EVB mode or the hardware breakpoint capability of the target microcontroller. To utilize D-Bug12's 10 software breakpoints, the USEHBR command should be entered with the '`OFF`' parameter on the command line.

Using the hardware breakpoints of the MC9S12DP256 when operating in EVB mode allows the developer to trace through the user accessible routines in D-Bug12 that are located in the on-chip Flash memory. Further, when debugging small programs located in the MC9S12DP256's on-chip EEPROM, it is recommended that hardware breakpoints be used. Using hardware breakpoints will prevent D-Bug12 from repeatedly erasing and reprogramming the on-chip EEPROM when using the T, G or GT commands or when setting breakpoints.

Entering the USEHBR command will reinitialize the breakpoint table causing any previously set breakpoints to be removed from the breakpoint table.

Restrictions

When operating in the POD mode, the USEHBR command cannot be issued when the 'R>' prompt is being displayed indicating that the target system is running a user program.

Example

```
S>usehbr
Using Hardware Breakpoints
S>br 810 835
Breakpoints: 0810  0835
S>br 957
Breakpoint Table Full
S>
```

VERF - Compare S-Record File To The Contents of Memory

Command Line Format

VERF [[<AddressOffset>] [;f]] | [;b]

Parameter Description

<AddressOffset>	A 16-bit hexadecimal number
;f	The ASCII string '`;f`' or '`;F`'
;b	The ASCII string '`;b`' or '`;B`'

Command Description

The VERF command is used to compare the data contained in an S-Record object file to the contents of target 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 zero allows the S-Record's object code or data to be compared against memory other than that for which the S-Record was assembled. An offset greater than $FFFF may only be used with devices that support more than 64K bytes of memory.

The '`;f`' option is used to verify S-Records against target memory locations normally occupied by on chip Flash memory for devices having a program memory space greater than 64K. This option is only required by devices that support more than 64K bytes of memory and have a device definition where the number of 16K memory pages is greater than zero. This option allows the S-Record loader to distinguish between S-Records that are to be verified against paged program memory and those to compare against other areas of on-chip or off-chip memory.

When the '`;f`' option is **not** used, the verify command accepts only S1 S-Records. This allows S-Record data to be verified against any memory locations visible in the 64K memory map. This includes the PPAGE window address range ($8000 - $BFFF) for devices containing more than 64K of paged program memory.

Note: Please refer to the section titled "FLOAD, LOAD and VERIFY S-Record Format" at the end of this document for a complete description of the S-Record Format utilized by this command for M68HC12 devices supporting more than 64K bytes of memory.

During the verification process, the ASCII characters '|', '/', '-' and '\' are sent one at a time to the control console to indicate that the verify process is proceeding. Before each character is sent, an ASCII backspace character is sent to the console so that the previously sent progress character is effectively erased from the screen. The displayed effect is a rotating bar. When an S-Record file has been successfully verified, D-Bug12 will issue its prompt.

If the contents of target memory does not match the corresponding data in the S–Record, an informational line is displayed on the console showing the S-Record address, the S-Record data and the data at the corresponding target memory location. Note that the displayed S-Record address includes the optional address offset that may have been entered on the command line.

The VERF command is terminated when D-Bug12 receives an ‘S8’ or ‘S9’ end of file record. If the object file being loaded does not contain an ‘S8’ or ‘S9’ record, D–Bug12 will not return its prompt and will continue to wait for the end of file record. Pressing system Reset will return D–Bug12 to its command line prompt.

Restrictions

None.

Example

```
S>verf
S-Rec Address  S-Rec Data  Target Address  Target Data
$00200         $00         $0200           $C6
$002C3         $00         $02C3           $87
$00397         $00         $0397           $EB
$005BA         $00         $05BA           $A2
$007DF         $00         $07DF           $61

S>
```

<RegisterName> - Modify a CPU12 Register Value

Command Line Format

<RegisterName> <RegisterValue>

Parameter Description

Where <RegisterName> is one of the following CPU12 register names:

Register Name	Description	Legal Range
PC	Program Counter	$0..$FFFF
SP	Stack Pointer	$0..$FFFF
X	X-Index Register	$0..$FFFF
Y	Y-Index Register	$0..$FFFF
A	A Accumulator	$0..$FF
B	B Accumulator	$0..$FF
D	D Accumulator (A:B)	$0..$FFFF
CCR	Condition Code Register	$0..$FF
PP	PPAGE Register	$0..$FF

Each of the fields in the CCR may be modified by using the following field Names:

CCR Bit Name	Description	Legal Range
S	STOP Enable	0..1
H	Half Carry	0..1
N	Negative Flag	0..1
Z	Zero Flag	0..1
V	Twos Complement Overflow Flag	0..1
C	Carry Flag	0..1
IM	IRQ Interrupt Mask	0..1
XM	XIRQ Interrupt Mask	0..1

For each of the CPU register names, <RegisterValue> may be a hexadecimal number or a simple expression. For the CCR bit names only a value of zero or one may be supplied for the <RegisterValue> parameter.

Command Description

This set of “commands” uses the CPU12 register names as individual commands to allow changing the contents of individual registers. Each register name or Condition Code Register bit name is entered on the command line followed by a space, then followed by the new register or bit value. The successful alteration of a CPU register or CCR will cause the CPU12’s register contents to be displayed.

Restrictions

When operating in POD mode, these commands may not be used when the ‘R>’ is being displayed.

If a value outside the range for a given register is entered, an error message is displayed and command execution is terminated leaving the register contents unaltered.

Example

```
>pc 700e

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
700E  0A00  7315  7D62     47:44        1001 0000
700E  790016        CLR   $0016
>x 1000

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
700E  0A00  1000  7D62     47:44        1001 0000
700E  790016        CLR   $0016
>c 1

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
700E  0A00  1000  7D62     47:44        1001 0001
700E  790016        CLR   $0016
>z 1

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
700E  0A00  1000  7D62     47:44        1001 0101
700E  790016        CLR   $0016
>d adf7

 PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
700E  0A00  1000  7D62     AD:F7        1001 0101
700E  790016        CLR   $0016
>
```

Appendix A

FLOAD, LOAD and VERIFY S-Record Format

The S-Record object file format was designed to allow binary object code and/or data to be represented in printable ASCII hexadecimal format to allow easy transportation between computer systems and development tools. For M68HC12 family members supporting less than 64K bytes of address space, S1 records, which contain a 16-bit address, are sufficient to specify the location in the device's memory space where code and/or data are to be loaded. The load address contained in the S1 record generally corresponds directly to the address of on-chip or off-chip memory device. For M68HC12 devices that support an address space greater than 64K bytes, S1 records are not sufficient.

Because the M68HC12 family is a 16-bit microcontroller with a 16-bit program counter, it cannot directly address a total of more than 64K bytes of memory. To enable the M68HC12 family to address more than 64K bytes of program memory, a paging mechanism was designed into the architecture. Program memory space expansion provides a window of 16K byte pages that are located from $8000 through $BFFF. An 8-bit paging register, called the PPAGE register, provides access to a maximum of 256, 16K byte pages or 4 megabytes of program memory. While there may never be any devices that contain this much on-chip memory, the MC68HC812A4 is capable of addressing this much external memory. In addition, the MC9S12DP256 contains 256K bytes of on-chip Flash that resides in a 1MB address space.

While many high-level debuggers are capable of directly loading linked, absolute binary object files into a target system's memory, D–Bug12 does not have that capability. D–Bug12 is only capable of loading object files that are represented in the S-Record format. As mentioned previously, because S1 records contain a 16-bit address, they are inadequate to specify a load address for a memory space greater than 64K bytes. S2 records, which contain a 24-bit address, were originally defined for loading object files into the memory space of the M68000 family. It would seem that S2 records would provide the necessary load address information required for M68HC12 object files. However, as those who are familiar with the M68000 family know, the M68000 has a linear (non-paged) address space. Thus, development tools, such as non-volitle memory device programmers, interpret the 24-bit address as a simple linear address when placing program data into memory devices.

Unfortunately, because no S-Record standard existed to support paged memory MCU devices, software vendors have devised two different methods of utilizing existing S-Record definitions to support the paged memory architecture of the M68HC12 family.

Linear S-Record Format

The 'linear S-Record format', utilizes the linear load address of S1 and/or S2 S-Records in a standard manner to define the placement of code and/or data in the MCU's Flash memory space. Instead of defining a new S–Record type or utilizing an existing S–Record type in a non-standard manner, the programmers LOAD and VERF commands view M68HC12 Flash memory spaces larger than 64K bytes as a simple linear array of memory beginning at an address of $00000. This is the same format in which S–Records would need to be presented to a stand alone non-volitle memory device programmer.

The MC9S12DP256, and other Star12 family devices, implement 6 bits of the PPAGE register giving it a 1MB program memory address space accessed through the PPAGE window at addresses $8000 through $BFFF. The lower 768K portion of the address space, accessed through the PPAGE window using PPAGE values $00 through $2F, are reserved for external memory

when the part is operated in expanded mode. The upper 256K of the address space, accessed using PPAGE values $30 through $3F, is occupied by the on-chip Flash memory. The mapping between the linear address contained in the S–Record and the 16K byte page viewable through the PPAGE window is shown in Figure A-1 below. Notice that because the linear load addresses associated with the on-chip Flash memory of the MC9S12DP256 are greater than $FFFF, a linear S-Record file for this device should only contain S2 S-Records.

The last two entries in the table do not associate linear address ranges with PPAGE window addresses, instead, these entries correspond to the two fixed page memory address ranges. Observe that while the addresses in the Window Address Range correspond to the fixed page memory addresses, the addresses in the Linear Address Range column correspond to the same PPAGE window addresses as those for PPAGE $3E and $3F.

Linear Address Range	PPAGE Value	Window Address Range	Memory Type
$00000 - $BFFFF	$00 - $2F	$8000 - $BFFF	off-chip memory
$C0000 - $C3FFF	$30	$8000 - $BFFF	on-chip Flash
$C4000 - $C7FFF	$31	$8000 - $BFFF	on-chip Flash
$C8000 - $CBFFF	$32	$8000 - $BFFF	on-chip Flash
$CC000 - $CFFFF	$33	$8000 - $BFFF	on-chip Flash
$D0000 - $D3FFF	$34	$8000 - $BFFF	on-chip Flash
$D4000 - $D7FFF	$35	$8000 - $BFFF	on-chip Flash
$D8000 - $DBFFF	$36	$8000 - $BFFF	on-chip Flash
$DC000 - $DFFFF	$37	$8000 - $BFFF	on-chip Flash
$E0000 - $E3FFF	$38	$8000 - $BFFF	on-chip Flash
$E4000 - $E7FFF	$39	$8000 - $BFFF	on-chip Flash
$E8000 - $EBFFF	$3A	$8000 - $BFFF	on-chip Flash
$EC000 - $EFFFF	$3B	$8000 - $BFFF	on-chip Flash
$F0000 - $F3FFF	$3C	$8000 - $BFFF	on-chip Flash
$F4000 - $F7FFF	$3D	$8000 - $BFFF	on-chip Flash
$F8000 - $FBFFF	$3E	$8000 - $BFFF	on-chip Flash
$FC000 - $FFFFF	$3F	$8000 - $BFFF	on-chip Flash
$F8000 - $FBFFF	N/A	$4000 - $7FFF	on-chip Flash
$FC000 - $FFFFF	N/A	$C000 - $FFFF	on-chip Flash

Figure A-1, MC9S12DP256 PPAGE to S-Record Address Mapping

The MC68HC912DA/DG128 only implements 3 bits of the PPAGE register giving it a total of 128K bytes of program memory address space accessed through the PPAGE window. The mapping between the linear address contained in the S–Record and the 16K byte page viewable through the PPAGE is shown in Figure A-2 below. Figure A-4 provides a graphical representation of the DA/DG128 mapping.

Again, notice that the last two entries in the table do not associate linear address ranges with PPAGE window addresses, instead, these entries correspond to the two fixed page memory address ranges. Observe that while the addresses in the Window Address Range correspond to the fixed page memory addresses, the addresses in the Linear Address Range column correspond to the same PPAGE window addresses as those for PPAGE $06 and $07. Notice that because the linear load addresses associated with the on-chip Flash memory of the MC68HC912DA/DG128 span a range from $0000 through $1FFFF, a linear S-Record file for this device can contain both S1 and S2 S-Records.

Linear Address Range	PPAGE Value	Window Address Range	Memory Type
$00000 - $03FFF	$00	$8000 - $BFFF	on-chip Flash
$04000 - $07FFF	$01	$8000 - $BFFF	on-chip Flash
$08000 - $0BFFF	$02	$8000 - $BFFF	on-chip Flash
$0C000 - $0FFFF	$03	$8000 - $BFFF	on-chip Flash
$10000 - $13FFF	$04	$8000 - $BFFF	on-chip Flash
$14000 - $17FFF	$05	$8000 - $BFFF	on-chip Flash
$18000 - $1BFFF	$06	$8000 - $BFFF	on-chip Flash
$1C000 - $1FFFF	$07	$8000 - $BFFF	on-chip Flash
$18000 - $1BFFF	N/A	$4000 - $7FFF	on-chip Flash
$1C000 - $1FFFF	N/A	$C000 - $FFFF	on-chip Flash

Figure A-2, MC68HC912DA/DG128 PPAGE to S-Record Address Mapping

The conversion of the linear S-Record load address to a PPAGE number and a PPAGE window address can be performed by the two formulas shown in Figure A-3. In the first formula `PageNum` is the value written to the `PPAGE` register, `PPAGEWinSize` is the size of the PPAGE window which is $4000. In the second formula `PPAGEWinAddr` is the address within the PPAGE window where the S-Record code/data is to be loaded. `PPAGEWinStart` is the beginning address of the PPAGE window which is $8000.

```
PageNum = SRecLoadAddr / PPAGEWinSize;

PPAGEWinAddr = (SRecLoadAddr % PPAGEWinSize) + PPAGEWinStart;
```

Figure A-3, PPAGE Number and Window Address Formulas

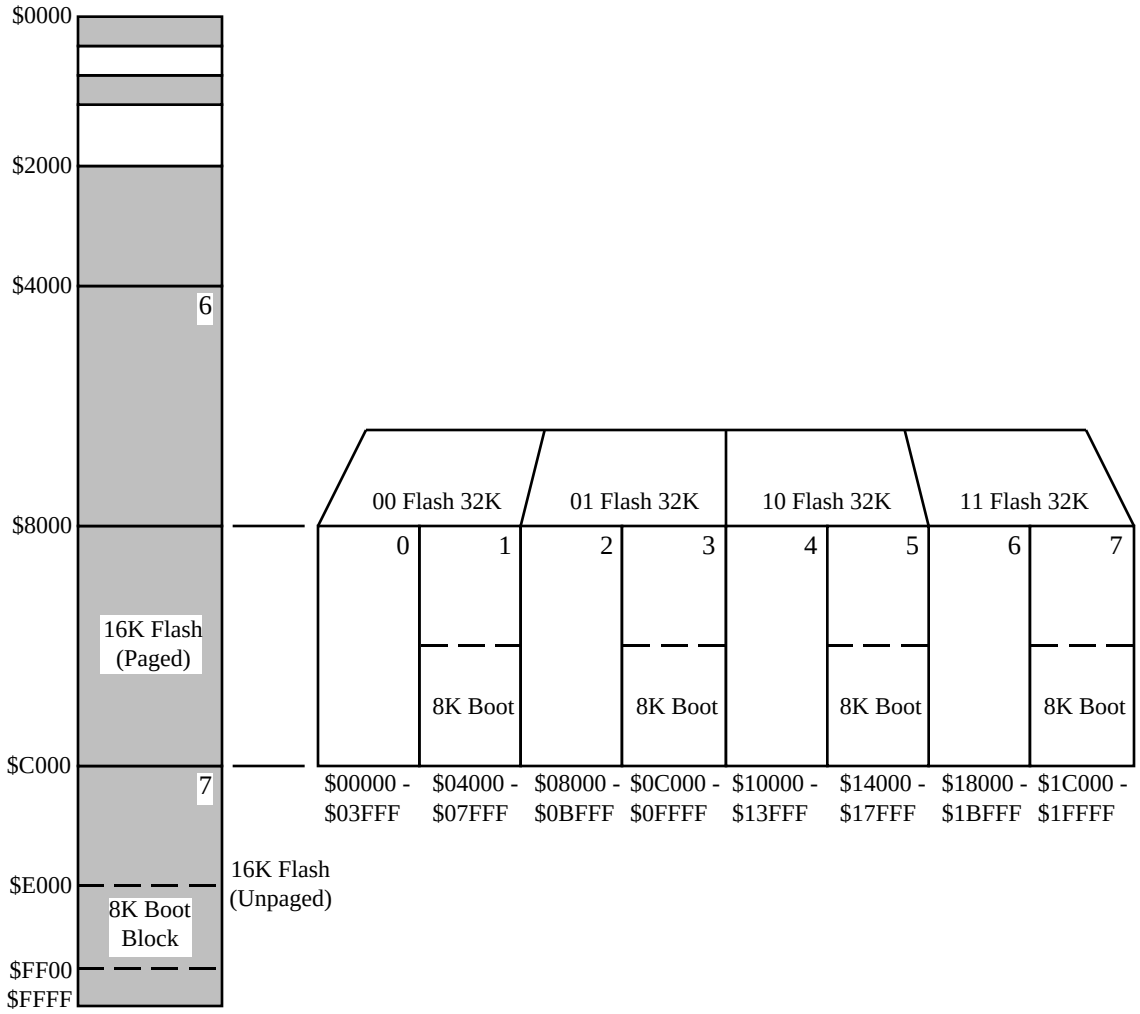

Figure A-4, MC68HC912DA/DG128 Flash Memory Paging

Banked or Paged S-Record Format

The banked or paged S-Record file format can also utilize a combination of S1 and S2 S-Records to directly specify the PPAGE number and the PPAGE window address. However, the 24-bit load address of S2 S-Records is used in a non-standard manner by encoding the PPAGE number in the upper 8-bits of the 24-bit address while placing the PPAGE window address in the lower 16-bits. While this method of encoding makes decoding convenient for an S-Record loader, it is specific to the M68HC12 family and these S2 S-Records would not be understood by a standalone device programmer. In addition, banked S-Record files are not accepted by Motorola for masked ROM generation.

The table in Figure A-5 illustrates the direct correspondence between the banked S-Record load addresses and the PPAGE register and PPAGE window address for the MC68HC912DA/DG128 device. Notice that the first entry in the table does not indicate that a bank number is included in the load address. As this illustrates, S-Records for PPAGE zero are usually encoded using an S1 record where the PPAGE value is implied to be $00. However, an S2 S-Record can also be used with the upper 8-bits of the 24-bit address explicitly explicitly set to $00. As the first eight table entries show, the lower 16-bits of a banked S2 S-Record must be in the range of $8000 - $BFFF.

However, as the last two entries show, the code and/or data for the two fixed page Flash memory areas may be specified using S1 S-Records.

Banked S-Rec Load Address	PPAGE Value	Window Address Range	Memory Type
$8000 - $BFFF	$00	$8000 - $BFFF	on-chip Flash
$018000 - $01BFFF	$01	$8000 - $BFFF	on-chip Flash
$028000 - $02BFFF	$02	$8000 - $BFFF	on-chip Flash
$038000 - $03BFFF	$03	$8000 - $BFFF	on-chip Flash
$048000 - $04BFFF	$04	$8000 - $BFFF	on-chip Flash
$058000 - $05BFFF	$05	$8000 - $BFFF	on-chip Flash
$068000 - $06BFFF	$06	$8000 - $BFFF	on-chip Flash
$078000 - $07BFFF	$07	$8000 - $BFFF	on-chip Flash
$4000 - $7FFF	N/A	$4000 - $7FFF	on-chip Flash
$C000 - $FFFF	N/A	$C000 - $FFFF	on-chip Flash

Figure A-5, MC68HC912DA/DG128 Banked S-Record Address Mapping

Appendix B

Adapting D-Bug12 to Use Alternate Crystal Frequencies

As supplied, D-Bug12 v4.x.x is configured to operate with an 8.0 MHz crystal or oscillator. The 8.0 MHz reference frequency is used by the PLL to generate a bus frequency of 24.0 MHz. Using a 24.0 MHz bus speed (rather than the DP256's rated speed of 25 MHz) allows communication with a host terminal program at 115,200 baud with very little speed mismatch (approximately 0.16%). To use D-Bug12 with a crystal or oscillator frequency other than 8.0 MHz, one area of D–Bug12 and the bootloader need to be modified.

Bootloader Modifications

Even though the bootloader is separate from D-Bug12, it requires modification so that the FCLKDIV, ECLKDIV and PLL registers can be programmed with the proper values when the bootloader is used. Figure B-1 shows an excerpted portion of the supplied bootloader source code used to calculate the register constants. OscClk is the crystal or oscillator frequency in MHz and should be changed to the desired value. In addition, depending on the value used for OscClk, a change may have to be made to the RefClk value to obtain a 24 MHz bus frequency. The equation used to calculate the value for REFDVVal and SYNRVal **MUST** produce integer results. The RefClk value must always be less than or equal to the OscClk value.

```
OscClk:     equ   8000000             ;crystal or oscillator frequency.
Eclock:     equ   24000000            ;final E-clock frequency (PLL).
RefClock:   equ   8000000             ; frequency used by the PLL to generate E-clock.
;
REFDVVal:   equ   (OscClk/RefClock)-1 ; value for REFDV register.
SYNRVal:    equ   (Eclock/RefClock)-1 ; value for SYNR register.
            if    OscClk>12800000
FCLKDIVVal: equ   (OscClk/200000/8)+FDIV8;valueforFCLKDIV/ECLKDIVregister.
            else
FCLKDIVVal: equ   (OscClk/200000)     ;value for FCLKDIV/ECLKDIV register.
            endif
```

Figure B-1, Bootloader FCLKDIV, ECLKDIV and PLL Register Values

For example, if a 16 MHz crystal or oscillator were used, the only required change would be to the value of OscClk because OscClk would be an integer multiple of RefClock and Eclock is an integer multiple of RefClock. If, however, a 12.0 MHz crystal or oscillator were used, both the OscClk and RefClock values would need to be changes to 12000000 because the E-clock frequency of 24.0 MHz is an even multiple of the OscClk.

As a final example, if a 10 MHz crystal or oscillator were used, the following changes would have to be made. In this case, because Eclock is not an integer multiple of OscClk, a reference clock must be generated that is an even multiple of Eclock. A reference clock of 2.0 MHz could be multiplied by 12 to obtain a 24 MHz bus clock. Therefore, specifying a RefClock value of 2000000 will cause the calculation of the proper integer values for both the FCLKDIV and ECLKDIV registers.

D-Bug12 Modifications

The changes required to D-Bug12 are isolated to a single data table residing at a fixed address. Among other data, this table contains constant values used to initialize the FCLKDIV, ECLKDIV and PLL registers. The values in the table related to the oscillator clock frequency are all calculated from the three constants at the beginning of the listing. The values of these three constants have the same restrictions as the constants described in the previous section.

```
;
;*************************************************************************
; Customization Data for D-Bug12 v4.x.x
;
; This data MUST reside at address $eec0
;*************************************************************************
;
FDIV8:      equ    $40
;
            org    $eec0
;
OscClk:     equ    16000000
Eclock:     equ    24000000              ; PLL E-clock frequency.
RefClock:   equ    8000000               ; reference frequency used by PLL.
;
UserCCR:    dc.b   $90                   ; initial CCR register for EVB mode.
UserB:      dc.b   $00                   ; initial B acc. for EVB or POD mode.
UserA:      dc.b   $00                   ; initial A acc. for EVB or POD mode.
UserX:      dc.w   $0000                 ; initial X register for EVB or POD mode.
UserY       dc.w   $0000                 ; initial Y register for EVB or POD mode.
UserPC:     dc.w   $0000                 ; initial PC value for EVB or POD mode.
BusClk:     dc.l   Eclock                ; system bus (E-clock) frequency.
REFDVVal:   dc.b   (OscClk/RefClock)-1
SYNRVal:    dc.b   (Eclock/RefClock)-1
            if     OscClk>12800000
FCLKDIVVal: dc.b   (OscClk/200000/8)+FDIV8 ; value for FCLKDIV & ECLKDIV register.
            else
FCLKDIVVal: dc.b   (OscClk/200000)       ; value for FCLKDIV & ECLKDIV register.
            endif
IOBase:     dc.w   $0000                 ; I/O register base address.
SCIBRegVal: dc.w   Eclock/16/9600        ; initial baud register value (9600 baud)
EEBase:     dc.w   $0400                 ; on-chip EEPROM base addr avail to D-Bug12.
EESize:     dc.w   3072                  ; on-chip EEPROM size avail to D-Bug12.
EEDelay:    dc.w   $ef02                 ; adress of 1 mS delay routine.
DlyCnt:     dc.w   Eclock/4000
;
```

Figure B-2, D-Bug12 Customization Data

Note: While the listing in Figure B-2 shows that the start of the data table must reside in the upper fixed page at $EEC0, the S-Record load address must be $FEEC0. If the assembler used to assemble the data table allows addresses greater than 16-bits, the address in the ORG statement should be changed to $FEEC0. If not, the resulting S-Record file with a 16-bit load address can be converted to have the proper load address using the supplied SRecCvt utility using the following command line:

```
sreccvt -m c0000 fffff 32 -of f0000 <inputfilename>
```

The file containing the converted S-Record will be named Out.S19 and reside in the same directory as the input file.

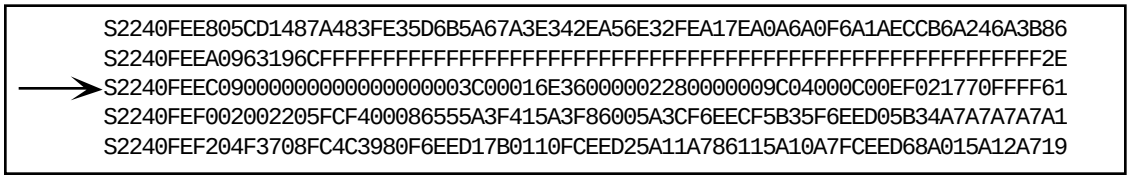

Figure B-3, S-Record Line Containing Customization Data

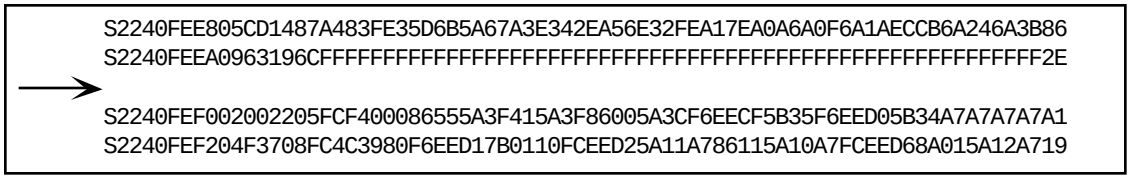

Figure B-4, Line Containing Customization Data Removed

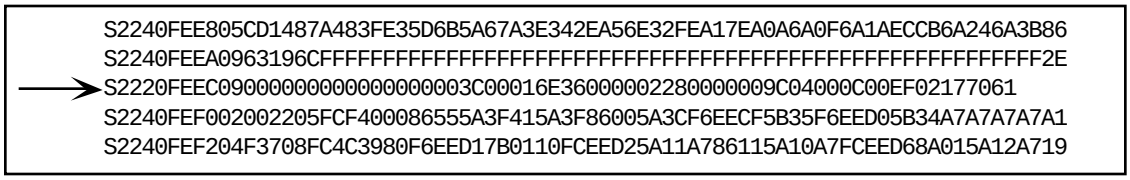

Figure B-5, Replacement of S-Record Header, Byte Count and Load Address

After making the necessary changes and successfully assembling the source file, a single line in the D-Bug12 S-Record file must be replaced. Using a text editor search for the S-Record line that begins with "`S2240FEEC0`" as shown in Figure B-3. Remove this line from the file leaving a space for the S-Record containing the newly generated customization data as shown in Figure B-4. Finally, paste in the S-Record containing the new customization data as shown in Figure B-5 and save the result.

Note: The new S-Record shown in Figure B-5 was generated directly by an assembler. An S-Record generated by the SRecCvt program will be the same length as the other S-records in the file with the last two data bytes in the being $FF.

Appendix C

User Accessible Utility Routines

When operating in EVB mode, D-Bug12 v4.x.x provides access to 18 utility routines through an array of function pointers (addresses) beginning at $EE80. Placing the table at a fixed address, allows access to the individual functions to remain constant even though the actual address of the routines may move when changes are made to D-Bug12.

Because D-Bug12 was written almost entirely in C, the utility routines are presented as C function definitions. However, this does not mean that the utility routines are usable only when programming in C. They may easily be accessed when programming in assembly language. Figure C-1 shows a summary of the available utility routines. A complete description of each utility routine is provided later in Appendix C.

Function	Description	Pointer Address
`far main()`	Start of D-Bug12	$EE80
`getchar()`	Get a character from SCI0 or SCI1	$EE84
`putchar()`	Send a character out SCI0 or SCI1	$EE86
`printf()`	Formatted Output - Translates binary values to characters	$EE88
`far GetCmdLine()`	Obtain a line of input from the user	$EE8A
`far sscanhex()`	Convert an ASCII hexadecimal string to a binary integer	$EE8E
`isxdigit()`	Checks for membership in the set [0..9, a..f, A..F]	$EE92
`toupper()`	Converts lower case characters to upper case	$EE94
`isalpha()`	Checks for membership in the set [a..z, A..Z]	$EE96
`strlen()`	Returns the length of a null terminated string	$EE98
`strcpy()`	Copies a null terminated string	$EE9A
`far out2hex()`	Displays 8-bit number as 2 ASCII hex characters	$EE9C
`far out4hex()`	Displays 16-bit number as 4 ASCII hex characters	$EEA0
`SetUserVector()`	Setup user interrupt service routine	$EEA4
`far WriteEEByte()`	Write a data byte to on-chip EEPROM	$EEA6
`far EraseEE()`	Bulk erase on-chip EEPROM	$EEAA
`far ReadMem()`	Read data from the M68HC12 memory map	$EEAE
`far WriteMem()`	Write data to the M68HC12 memory map	$EEB2

Figure C-1, Utility Routines Summary

User Accessible Function Calling Conventions

All of the user accessible routines were written in C. In general, parameters are passed to the user callable functions on the stack. Parameters must be pushed onto the stack in the reverse order they are listed in the function declaration (right-to-left) *except* for the last parameter (the first parameter listed in the C function declaration). The last parameter is passed to the function in the D-accumulator. Functions having only a single parameter pass it in the D-accumulator. `char` parameters must always be converted to an `int`. This means that even if a parameter is declared as

a `char` it will occupy two bytes of stack space as a parameter. `char` parameters should occupy the low order byte (higher byte address) of a word pushed onto the stack or the B-accumulator if the parameter is passed in D.

Parameters pushed onto the stack before the function is called remain on the stack when the function returns. It is the responsibility of the *calling* routine to remove passed parameters from the stack.

All 8- and 16-bit function results are returned in the D-accumulator. `char` values returned in the D-accumulator are located in the 8-bit B-accumulator. `Boolean` function results are zero for False and non-zero values for True.

None of the CPU12's register contents, except the Stack Pointer, are preserved by the called functions. If any of the register values need to be preserved, they should be pushed onto the stack before any of the parameters and restored after deallocating the parameters.

Functions listed in Figure C-1 with the `far` qualifier are located in the MC9S12DP256's banked memory and must be accessed using the `CALL` instruction. Note that each of the far function pointers occupies four bytes of memory instead of two.

Assembly Language Interface

Calling the functions from assembly language is a simple matter of pushing the parameters onto the stack in the proper order, loading the first or only function parameter into the D-accumulator. The function can then be called with a `CALL` or `JSR` instruction. The code following the `CALL` or `JSR` instruction should remove any parameters pushed onto the stack If a single parameter was pushed onto the stack, a simple `PULX` or `PULY` instruction is on of the most efficient ways to remove the parameter from the stack. If two or more parameters were pushed on to the stack, the `LEAS` instruction is the most efficient way to remove the parameters. Any of the CPU12's registers may have been saved on the stack before the function parameters should be restored with corresponding `PULx` instructions. An example of calling the `WriteEEByte()` function is shown in Figure C-2.

```
WriteEEByte:  equ   $EEA6              ; address of function pointer.
;
              .
              .
              ldab  #$55               ; write $55 to EEPROM.
              pshd                     ; place the data on the stack.
              ldd   EEAddress          ; EEaddress to write data.
              call  [WriteEEByte,pcr]  ; Call the routine.
              pulx                     ; remove the parameter from stack.
              beq   EEWError           ; zero return value means error.
              .
              .
```

Figure C-2, Calling the `WriteEEByte` Subroutine

Callable Routine Descriptions

The following paragraphs contain complete descriptions and usage notes for D-Bug12's user callable routines. In addition, the amount of stack space required by each routine and the routine's pointer address are also supplied.

`void far main(void);`

Pointer Address: $EE80

The first field in the table contains a pointer to D-Bug12's `main()` function. This entry is provided for two purposes. First, the Reset vector does not point to `main()` but rather to code that is contained in the file `Startup.s`. This file contains assembly language code that is required to initialize various hardware modules of the MC9S12DP256 before proper execution of the monitor can occur. As the monitor code is changed, the address of `main()` will change. Because the user may replace the supplied startup routines with their own startup code, the user would have to examine the supplied D-Bug12 startup object code to determine the address of `main()`.

Placing the address of the `main()` function at first location in the user callable routines table allows user supplied startup code to easily begin execution of the monitor with the following instructions:

```
ldx   #main          ; point to the main() far pointer
movb  2,x,PPAGE      ; move main's PPAGE number into the PPAGE register.
jmp   [main,pcr]     ; start D-Bug12 from main().
```

In addition, the user may want to execute a program stored in EEPROM or other non-volatile memory before entering the monitor. Again, for the same reasons listed above, providing the address of the monitor's `main()` function at a fixed address allows the location of main() to change without having to change the users code.

Note: When executing a user program from powerup or reset that is stored in the on-chip EEPROM, the users program should enter D-Bug12 through the secondary reset vector at $EFFE rather than through main(). The main() function does not perform any hardware initialization and does not clear D-Bug12's variable memory.

When calling the `main()` function from a user program that began execution from D-Bug12, the user's program should first load the CPU12's stack pointer (SP) with the value located in the Customization Data area described in Appendix A.

Note: Reentering D-Bug12 from a user program through the `main()` function reinitializes all of D-Bug12's internal tables and variables. Any previously set breakpoints will be lost and any breakpoint SWI's will remain in the users program.

`int getchar(void);`

Pointer Address: $EE84

The `getchar()` function provides the ability to retrieve a single character from the control terminal SCI. If a character is not available in the SCI's Receive Data Register when the function is called, the `getchar()` will wait until one is received. Because the character is returned as an `int`, the 8-bit character is placed in the B-accumulator.

`int putchar(int);`

Pointer Address: $EE86

The `putchar()` function provides the ability to send a single character to the control terminal SCI. If the SCI's Transmit Data Register is full when the function is called, `putchar()` will wait until the Transmit Data Register is empty before sending the character. No buffering of characters is provided. `putchar()` returns the character that was sent. However, it does not detect any error conditions that may occur in the process and therefore will never return `EOF`. Because the character is returned as an `int`, the 8-bit character is placed in the B-accumulator.

`int printf(char *format, ...);`

Pointer Address: $EE88

The `printf()` function is used to convert, format, and print its arguments on the standard output under the control of the format string pointed to by `format`. It returns the number of characters that were sent to standard output. The version of `printf()` included as part of the monitor supports the formatted printing of all data types *except* floating point numbers.

The format string can contain two basic types of objects: ASCII characters which are copied directly from the format string to the display device, and conversion specifications that cause succeeding `printf()` arguments to be converted, formatted, and sent to the display device. Each conversion specification begins with a percent sign (%) and ends with a single conversion character. Optional formatting characters may appear between the percent sign and the conversion character in the following order:

[-][<FieldWidth>][.][<Precision>][h | l]

Character	Description
- (minus sign)	Left justifies the converted argument.
FieldWidth	Integer number that specifies the minimum field width for the converted argument. The argument will be displayed in a field at least this wide. The displayed argument will be padded on the left or right if necessary.
. (period)	Separates the field width from the precision.
Precision	Integer number that specifies the maximum number of characters to display from a string or the minimum number of digits for an integer.
h	To have an integer displayed as a short.

l (letter ell) To have an integer displayed as a long.

The FieldWidth or Precision field may contain an asterisk (*) character instead of a number. The asterisk will cause the value of next argument in the argument list to be used instead.

Figure C-3, shown below, contains the conversion characters supported by the `printf()` function included in D-Bug12. If the conversion character(s) following the percent sign are not one of the formatting characters shown above or the conversion characters shown in Table 2 below, the behavior of the `printf()` function is undefined.

Character	Argument Type; Displayed As
d, i	int; signed decimal number
o	int; unsigned octal number (without a leading 0)
x	int; unsigned hexadecimal number using abcdef for 10..15
X	int; unsigned hexadecimal number using ABCDEF for 10..15
u	int; unsigned decimal number
c	int; single character
s	char *; display from the string until a '\0'
p	void *; pointer (implementation-dependent representation)
%	no argument is converted; print a %

Figure C-3, printf() Conversion Characters

For those unfamiliar with C or the `printf()` function the following examples show the results produced by the `printf()` function for several different format strings.

Example 1.

```
printf("Signed Decimal: %d  Unsigned Decimal: %u/n", Num, Num);
```

Where Num has the value $FFFF

Displays the result:

```
Signed Decimal: -1  Unsigned Decimal: 65535
```

Example 2.

```
printf("Hexadecimal: %H  Hexadecimal: %4.4H/n", Num, Num);
```

Where Num has the value $FF

Displays the result:

```
Hexadecimal: FF  Hexadecimal: 00FF
```

Example 3.

```
printf("This is a %s/n", TestStr);
```

Where TestStr is a *pointer* to (address of) the first byte of a null (zero) terminated character array containing "Test".

Displays the result:

```
This is a Test
```

`int GetCmdLine(char *CmdLineStr, int CmdLineLen);`

Pointer Address: $EE8A

The `GetCmdLine()` function is used to obtain a line of input from the user. `GetCmdLine()` accepts input from the user a single character at a time by calling getchar(). As each character is received it is echoed back to the users terminal by calling putchar() and placed in the character array pointed to by `CmdLineStr`. A maximum of `CmdLineLen - 1` printable characters may be entered. Only printable ASCII characters are accepted as input with the exception of the ASCII backspace character ($08) and the ASCII carriage return character ($0D). All other non-printable ASCII characters are ignored by the function.

The ASCII backspace character ($08) is used by the `GetCmdLine()` function to delete the previously received character from the command line buffer. When `GetCmdLine()` receives the backspace character, it will echo the backspace to the terminal, print the ASCII space character, $20, and then send a second backspace character to the terminal. This action will cause the previous character to be erased from the screen of the terminal device. At the same time the character is deleted from the command line buffer. If a backspace character is received when there are no characters in `CmdLineStr`, the backspace character is ignored.

The reception of an ASCII carriage return character ($0D) terminates the reception of characters from the user. The carriage return, however, is not placed in the command line buffer. Instead an ASCII NULL character ($00) is placed in the next available buffer location.

Before returning, all the entered characters are converted to upper case. `GetCmdLine()` always returns an error code of `noErr`.

`char * far sscanhex(char *HexStr, unsigned int *BinNum);`

Pointer Address: $EE8E

The `sscanhex()` function is used to convert an ASCII hexadecimal string to a binary integer. The hexadecimal string pointed to by `HexStr` may contain any number of ASCII hexadecimal characters. However, the converted value must be no greater than $FFFF. The string must be terminated by either an ASCII space ($20) or an ASCII NULL ($00) character.

The value returned by `sscanhex()` is either a pointer to the terminating character or a NULL pointer. A NULL pointer indicates that either an invalid hexadecimal character was found in the string or that the converted value of the ASCII hexadecimal string was greater than $FFFF.

`int isxdigit(int c);`

Pointer Address: $EE92

The `isxdigit()` function tests the character passed in `c`, for membership in the character set [0..9, a..f, A..F]. If the character `c` is part of this set, the function returns a non-zero (true) value otherwise, a value of zero is returned.

`int toupper(int c);`

Pointer Address: $EE94

If `c` is a lower-case character, [a..z], `toupper()` will return the corresponding upper-case letter. If the character is upper-case, it simply returns `c`.

`int isalpha(int c);`

Pointer Address: $EE96

The `isalpha()` function tests the character passed in `c`, for membership in the character set [a..z, A..Z]. If the character `c` is part of this set, the function returns a non-zero (true) value otherwise, a value of zero is returned.

`unsigned int strlen(const char *cs);`

Pointer Address: $EE98

The `strlen()` function returns the length of the string pointed to by `cs`. A string is an array of characters that is terminated by a `'\0'` character.

`char * strcpy(char *s1, char *s2);`

Pointer Address: $EE9A

The `strcpy()` function copies the contents of string `s2` into the string pointed to by `s1` including the `'\0'`. A pointer to s1 is returned.

`void far out2hex(unsigned int num);`

Pointer Address: $EE9C

The `out2hex()` function displays the lower byte of `num` on the control terminal as two hexadecimal characters. The upper byte of `num` is ignored. This function is provided for those that may not know how to use the `printf()` function. `out2hex()` simply calls printf() with a format string of "`%2.2X`".

`void far out4hex(unsigned int num);`

Pointer Address: $EEA0

`out4hex()` displays `num` on the control terminal as four hexadecimal characters.This function is provided for those that may not know how to use the `printf()` function. `out4hex()` simply calls printf() with a format string of "`%4.4X`".

`int SetUserVector (int VectNum,`
`Address UserAddress);`

Pointer Address: $EEA4

Note: The `SetUserVector()` function is included with v4.x.x only for compatibility with earlier versions of D-Bug12. The RAM vector table is located at $3E00 and may be accessed directly as described in Section 4.2

The function `SetUserVector()` allows the user to substitute their own interrupt service routines for the default interrupt service routines provided by D-Bug12. Providing access to the RAM interrupt vector table through this routine provides flexibility for future implementations of interrupt handling in D-Bug12. In addition the memory location of the table may be changed without having to change a users code. The address of the user's interrupt service routine, passed in `UserAddress`, should point to a routine that ends with an M68HC12 RTI instruction.

The following `enum typedef` defines the valid constants for `VectNum`. If an invalid constant is passed in `VectNum`, a value of -1 will be returned by `SetUserVector()` otherwise a value of zero is returned.

```
typedef Address char *;
typedef enum Vect {
                    UserRsrv0x80 = 0,
                    UserRsrv0x82 = 1,
                    UserRsrv0x84 = 2,
                    UserRsrv0x86 = 3,
                    UserRsrv0x88 = 4,
                    UserRsrv0x8a = 5,
                    UserPWMShDn  = 6,
                    UserPortP    = 7,
                    UserMSCAN4Tx = 8,
                    UserMSCAN4Rx = 9,
                    UserMSCAN4Errs = 10,
                    UserMSCAN4Wake = 11,
                    UserMSCAN3Tx = 12,
                    UserMSCAN3Rx = 13,
                    UserMSCAN3Errs = 14,
                    UserMSCAN3Wake = 15,
                    UserMSCAN2Tx = 16,
                    UserMSCAN2Rx = 17,
                    UserMSCAN2Errs = 18,
                    UserMSCAN2Wake = 19,
```

```
UserMSCAN1Tx = 20,
UserMSCAN1Rx = 21,
UserMSCAN1Errs = 22,
UserMSCAN1Wake = 23,
UserMSCAN0Tx = 24,
UserMSCAN0Rx = 25,
UserMSCAN0Errs = 26,
UserMSCAN0Wake = 27,
UserFlash = 28,
UserEEPROM = 29,
UserSPI2 = 30,
UserSPI1 = 31,
UserIIC = 32,
UserDLC = 33,
UserSCME = 34,
UserCRG = 35,
UserPAccBOv = 36,
UserModDwnCtr = 37,
UserPortH = 38,
UserPortJ = 39,
UserAtoD1 = 40,
UserAtoD0 = 41,
UserSCI1 = 42,
UserSCI0 = 43,
UserSPI0 = 44,
UserPAccEdge = 45,
UserPAccOvf = 46,
UserTimerOvf = 47,
UserTimerCh7 = 48,
UserTimerCh6 = 49,
UserTimerCh5 = 50,
UserTimerCh4 = 51,
UserTimerCh3 = 52,
UserTimerCh2 = 53,
UserTimerCh1 = 54,
UserTimerCh0 = 55,
UserRTI = 56,
UserIRQ = 57,
UserXIRQ = 58,
UserSWI = 59,
UserTrap = 60,
RAMVectAddr = -1 };
```

Once set, all of the addresses of the user's interrupt service routines will remain in the RAM vector table until D-Bug12 is restarted by a hardware reset. Alternately, individual interrupt service routine addresses may be removed by passing a null pointer in the `UserAddress` parameter.

Passing the constant '`RAMVectAddr`' in the `VectNum` parameter will return the base address of the RAM interrupt vector table instead of an error code. This will allow the user to make numerous changes to the RAM vector table without having to call the `SetUserVector()` function for each interrupt vector change. When accessing the RAM vector table by using the base address, the Vect enumerated constants must be multiplied by two before being used as an offset into the RAM vector table.

Note: Care should be used when allowing addresses of user interrupt service routines to remain in the RAM vector table. If the addresses of interrupt service routines change during program development, D-Bug12's interrupt handler will most likely jump to an incorrect program address resulting in loss of CPU/monitor control.

`Boolean far WriteEEByte(Address EEAddress, Byte EEData);`

Pointer Address: $EEA6

The `WriteEEByte()` provides a mechanism to program individual bytes of the on-chip EEPROM without having to manipulate the EEPROM programming control registers. `WriteEEByte()` does not perform any range checking on `EEAddress` to ensure that it falls within the address range of the on-chip EEPROM. A users program can determine the start address and size of the on-chip EEPROM array by examining the data contained in the custom data area fields `CustData.EEBase` and `CustData.EESize`.

A byte erase operation is performed before the programming operation and a verify is performed after the programming operation. If the EEPROM data does not match `EEData`, false (zero value) is returned by the function.

`int far EraseEE(void);`

Pointer Address: $EEAA

The `EraseEE()` `function` provides a mechanism to bulk Erase the on-chip EEPROM without having to manipulate the EEPROM programming control registers. After the bulk erase operation is performed, a check of the memory range described by `CustData.EEBase` and `CustData.EESize` is checked for erasure. If any of the bytes does not contain `0xff`, a non-zero error code is returned.

`int far ReadMem (Address StartAddress, Byte *MemDataP, unsigned int NumBytes);`

Pointer Address: $EEAE

The `ReadMem()` function is used internally by D-Bug12 for all memory read accesses. For this implementation of the monitor, the `ReadMem()` function simply reads `NumBytes` of data directly from the target memory and places it in a buffer pointed to by `MemDataP`. A user implemented command would probably not benefit from the use of this function. Instead, it could read values directly from memory. A non-zero error code is returned if a problem occurs while reading target memory.

```
int WriteMem (Address StartAddress, Byte *MemDataP,
              unsigned int NumBytes);
```

Pointer Address: $EEB2

The `WriteMem()` function is used internally by D-Bug12 for all memory write accesses. `WriteMem()` is different from `ReadMem()` in the fact that it is aware of the on-chip EEPROM memory. If a byte is written to the memory range described by `CustData.EEBase` and `CustData.EESize`, `WriteMem()` calls the `WriteEEByte()` function to program the data into the on-chip EEPROM memory. A non-zero error code is returned if a problem occurs while writing target memory.