Reference Guide
For
D–BugS08 Version 1.x.x

A Debug Monitor
For
The MC9S08 Family of Microcontrollers
That Runs On The
MC9S12Dx256, MC9S12A256, MC9S12Dx128,
MC9S12H256 & MC9S12A128
Microcontrollers

Written By
Gordon Doughman
Field Applications Engineer
Software Specialist

1.0 Introduction

D–BugS08 is a simple but powerful command line debugger for the MC9S08 family of 8-bit microcontrollers. Unlike D-Bug12, which operates as both a ROM resident and BDM POD debugger, D–BugS08 runs on an MC9S12 family device and only operates in POD mode. Like D-Bug12, D-BugS08 has the capability to erase and program target Flash memory, examine and change RAM and I/O register contents and hot connect to a running target without disturbing the state of the target MCU.

2.0 Hardware Requirements

D-BugS08 was designed to run on an MC9S12 family device containing at least 128K of Flash program memory, 4K of RAM, 1K of EEPROM and a single SCI port with RS-232 level translator. Hardware meeting these requirements would include Freescale Semiconductor's MC9S12DP256 EVB and the Serial BDM Programmer. Other evaluation boards from various manufacturers may also meet these requirements.

In addition to these hardware requirements, it may be necessary for the POD hardware to provide voltage level translators on the Reset and BKGD pins of the BDM target (OUT) connector depending on the operating voltage range of the POD and target MCU. For example, the MC9S12DP256 used on the MC9S12DP256 EVB has an operating voltage range of 5 volts ±10%. If the MC9S12DP256 EVB were used as the POD with a target system containing an

MC9S08 device operating outside that range, voltage level translators would be required to ensure proper communication and prevent damage to the target S08 device. Note that the Freescale Serial BDM Programmer hardware version 2, revision 2 contains level translator circuitry on the Target BDM connector that will automatically track the target V_{DD} levels as long as target V_{DD} is supplied to pin 6 of the target's BDM connector.

For hardware not containing voltage level translators, the Maxim/Dallas MAX3373E bidirectional level translators may be added to existing hardware as shown in Figure 2-1.

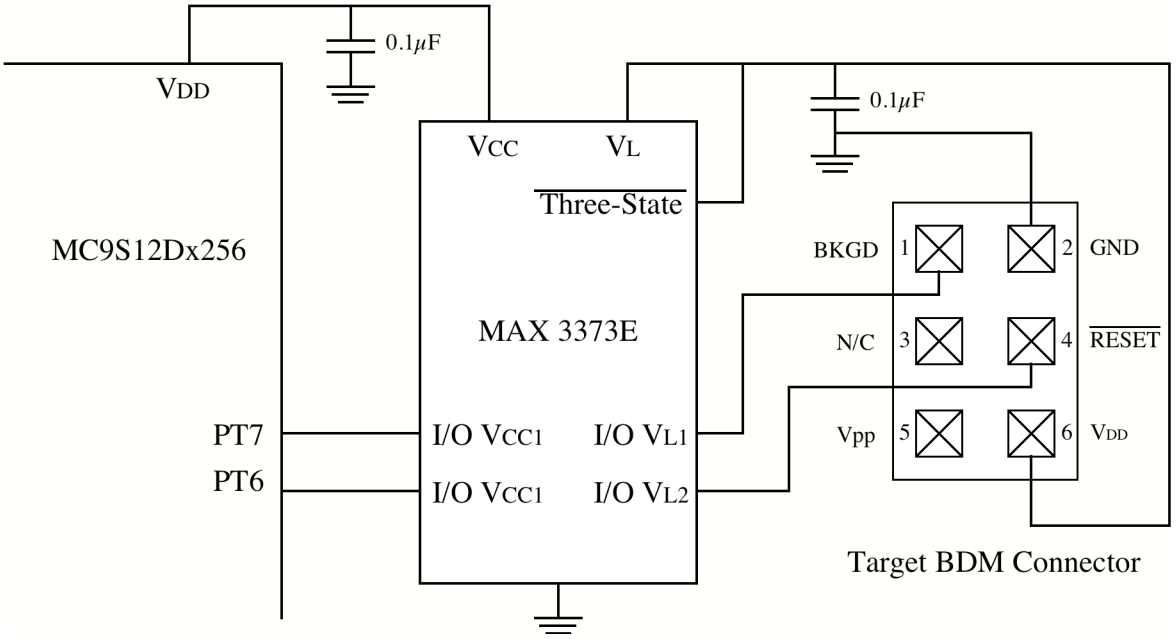

Figure 2-1, Bidirectional Level Shifters

In addition, D-BugS08 requires a bootloader that supports a secondary interrupt/reset vector table that begins at $EFFF and grows downward (i.e. the secondary reset vector resides at $EFFE and $EFFF). A 4K bootloader meeting these requirements is included with the D-BugS08 firmware distribution. This bootloader is suitable for use on the MC9S12DP256 EVB.

3.0 Terminal Communications Setup

D-BugS08 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 of extremely slow text file transfer rates, the Hyperterminal terminal emulation program supplied with Windows™ is **NOT** recommended. Instead, a free terminal emulator program, named TeraTerm, is recommended for use with D–BugS08. TeraTerm can be downloaded from the TeraTerm home page at: `http://hp.vector.co.jp/authors/VA002416/teraterm.html`.

Baud rate changes made using the BAUD command are stored in the programmer's nonvolatile memory making the entered baud rate the new default communication rate. If communication cannot be established with the programmer 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.

The factory configured default communications parameters used by D-BugS08 is 9600 baud, eight data bits, one stop bit, XOn/XOff handshaking and no parity. If attempting communications at various baud rates does not result in the display of the programmer's prompt, the programmer's on-chip EEPROM should be erased to reconfigure the baud rate to the factory default of 9600. This can be accomplished by the invoking the Serial bootloader as described in Section 4.0, "Serial Bootloader" and selecting the Erase EEPROM command as described in Section 4.4, "Erasing the On-chip EEPROM".

3.1 Configuring TeraTerm

The TeraTerm terminal emulation program is recommended for use with D-BugS08. For those not familiar with TeraTerm, this section describes the setup procedure necessary to use TeraTerm with the programmer. After downloading and installing the TeraTerm terminal emulator program, locate the TeraTerm program and double click on the icon to start the program.

When presented with the "TeraTerm: New connection" dialog box, select the Serial connection, choose the appropriate Port from the pop up menu and click the 'OK' button. Before proceeding, the serial port must be configured by selecting "Serial Port…" from the "Setup" menu. The Port pop up menu should be configured with the serial port chosen when the application was initially launched. The Baud rate pop up menu should be set to the baud rate last set by the BAUD command. If the programmer has not been used before, set the baud rate to the factory default of 9600. Make sure that the Data pop up is set to' 8 bit', Parity to 'none', Stop to '1 bit' and Flow control to 'Xon/Xoff'. Click 'OK' to accept the settings.

Pressing the SYSTEM RESET button on the programmer should display the programmer's sign-on messages and prompt. If nothing appears on the screen, check the connection between the computer and programmer to ensure that the programmer 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.

4.0 BDM Target Connection

D-BugS08 communicates with the developer's MC9S08 target system through the Single Wire Background Debug interface. This arrangement, as shown in Figure 4-1, allows access to a developer's target system in a nonintrusive manner. All of the target MCU's resources are available to the developer, providing a noninvasive development environment for the target system.

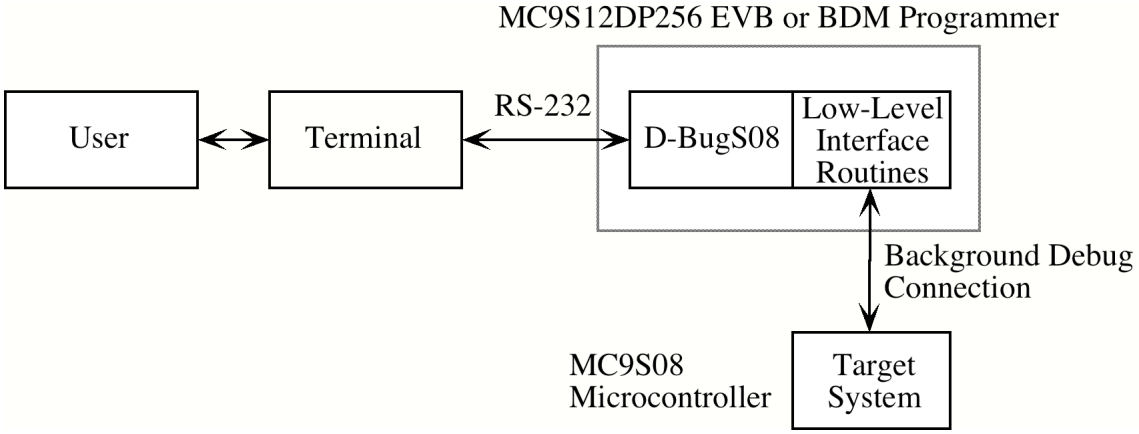

Figure 4-1, BDM Target Connection

On power-up or reset D–BugS08 attempts to establish communications with a target system. Initially, communications with the target is attempted without resetting the target system. This feature allows D–BugS08 to be 'hot connected' to a running system without disturbing the target microcontroller. Because the MC9S08 BDM interface has the capability to send a timing reference pulse to D–BugS08, it is never necessary for the developer to specify the target operating frequency. D-BugS08 will automatically measure the timing reference pulse and configure its BDM communications drivers accordingly.

If the hardware running D-BugS08 is powered up without being connected to a target system, the message shown in Figure 4-2 is displayed.

```
D-BugS08 1.0.0b1
Copyright 1996 - 2005 Freescale Semiconductor
For Commands type "Help"


Can't Communicate With Target CPU

1.) Reattempt Communication
2.) Reset Target
3.) Erase & Unsecure
4.) Enter BDM debugger
5.) Load BDM Programmer Firmware From Flash
?
```

Figure 4-2, Failed Target Communications Prompt

Entering option '1', causes D-BugS08 to issue the BDM SYNC command which requests the target MCU to issue a timed reference pulse on the BDM pin allowing D-BugS08 to determine the target's operating frequency. If a target MCU is properly connected and has power applied, D-BugS08 should display either the '`S>`' or '`R>`' prompt. 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.

If communication cannot be established after several attempts, check for the following possible problems:

- The D-BugS08 POD connector must be properly connected to the target systems BDM connector. Make sure that the D-BugS08 POD's BDM OUT connector is connected to the target's BDM IN connector.

- Check for the proper orientation of the BDM cable with the BDM connectors on both the D-BugS08 POD and the target.

- Verify that the target's BDM connector is wired to the proper MCU signals on each pin.

- If the target MCU does not have any firmware to execute, the CPU08 will most likely "run away", possibly executing a STOP instruction, preventing BDM communications with the target MCU. In this case, Option 2 should be selected which will reset the target device in Special Single Chip mode, placing it in Active Background.

- If the target MCU has it's security feature enabled and the Flash is not blank, normal BDM communication cannot be established with the device. Option '3' must first be used to erase the target Flash and disable security.

Because the MC9S08 Single Wire Background interface allows the reading and writing of target

system memory even when the target is running a user's program, the D-BugS08 is always available for the entry of commands. D-BugS08 commands that examine or modify target system memory may be issued when either the 'S>' or 'R>' prompt is displayed.

4.1 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 MC9S08 family members have been designed with a device security mechanism that makes it nearly impossible to access the Flash contents. Once the security mechanism has been enabled, access to the Flash through the BDM is inhibited. Gaining access to either of these resources may only be accomplished by erasing the contents of the Flash 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 S08 device is reset in Special Single-chip mode, all on-chip resources are disabled except for the I/O registers. 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 RAM remains protected. This functionality is adequate to manipulate the Flash control registers to erase its contents.

The Erase and Unsecure option can be used to erase the Flash of a target MC9S08 device and place it in the unsecured state. If the selected target device contains the Internal Clock Generator Module (ICG), the Erase and Unsecure option will perform a calibration operation to determine the correct ICGTRM register value to trim the 243 KHz on-chip oscillator to within a couple of tenths of a percent of its nominal frequency. This value is programmed into Flash memory at address $FFBE where an application can easily retrieve the value and initialize the ICGTRM register during the ICG initialization.

4.2 BDM Debugger Option

The BDM debugger included with D-BugS08 is intended to be used by tool developers or factory engineers when evaluating new silicon or debugging BDM communication problems, however, it may also be useful to end customers. See Appendix B and the documentation associated with the BDMDB command for additional information on the BDM debugger.

4.3 Load BDM Programmer Firmware Option

5.0 D-BugS08 Command Summary

The following list summarizes the D-BugS08 command set. Each command's function and command line syntax are described in detail.

- ASM - Single line assembler/disassembler.
- BAUD - Set the SCI communications BAUD rate.
- BDMDB - Enter the BDM command debugger.
- BF - Block Fill user memory with data.
- BR - Set/Display user breakpoints.
- CAL – Calibrate the target MCU's internal clock reference
- CALL - Execute a user subroutine, return to D-BugS08 when finished.
- DEVICE - Select/define a new target MCU device.
- FBULK - Erase the target processor's on-chip Flash EEPROM
- FLOAD - Program the target processor's on-chip Flash EEPROM from S-Records
- FSERASE - Erase one or more sectors of target Flash
- G - Go. Begin execution of user program.
- GT - Go Till. Set a temporary breakpoint and begin execution of user program.
- HELP - Display D-BugS08 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.
- RD - Register Display. Display the CPU register contents.
- 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.
- VER - Display the running version of D-BugS08
- VERF - Verify memory contents against S-Record Data.
- <RegisterName> <RegisterValue> - Set CPU <RegisterName> to <RegisterValue>

ASM - Single Line Assembler/Disassembler Command

Command Line Format

ASM <Address>

Parameter Description

<Address> - 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-BugS08 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 CPU08 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 *HCS08 Family Reference Manual*.

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.

Example

```
S>asm 1080
1080  A603     LDA   #$03         >
1082  C71802   STA   $1802        >
1085  C6FFBE   LDA   $FFBE        >
1088  B74E     STA   $4E          >
108A  A608     LDA   #$08         >
108C  B748     STA   $48          >
108E  A670     LDA   #$70         >
1090  B749     STA   $49          >.
S>
```

Assembly Operand Format

This section describes the operand format used by the assembler when assembling CPU08 instructions. The operand format accepted by the assembler is described separately in the *HCS08 Family 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, BRSET, BRCLR, DBNZ, CBEQ) 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.

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 *HCS08 Family 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, BRSET, BRCLR, DBNZ, CBEQ) 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 *unsigned* decimal numbers with the following exception.

Instructions using direct addressing will have the direct address disassembled as a two digit hexadecimal number.

Instructions using extended addressing will have the direct address disassembled as a four digit hexadecimal number.

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 nonvolatile 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 *Customization Data* 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
>
```

BDMDB - Enter BDM Debugger

Command Line Format

BDMDB

Parameter Description

None.

Command Description

The BDMDB command halts normal D-BugS08 operation and enters the BDM debugger. Using D-BugS08's low level BDM driver routines, the BDM debugger allows individual BDM commands to be sent to a target device directly from the command line. As shown in the example, upon entering the BDM debugger, the command line prompt changes to a question mark (?). Note that while running the BDM debugger, no target BDM communication occurs other than during the execution of a command. Unlike D-BugS08, when running the BDM debugger, no checks are performed to ensure a valid target connection with a target MCU exists before executing a command.

See Appendix B for a complete description of the BDM debugger commands.

Restrictions

None.

Example

```
>bdmdb
BDM Command Debugger
For Commands type "HELP"

?
```

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`' in upper, lower or mixed case

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>…]

Parameter Description

<Address> 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-BugS08 will disassemble the instruction at the breakpoint address, print the CPU08's register contents, and wait for the next D-BugS08 command to be entered by the developer.

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. A maximum of 3 breakpoints may be set when using the target CPU's hardware breakpoint capability. For additional information on D-BugS08's hardware breakpoint support, see the USEHBR command description.

Restrictions

D-BugS08 implements the software breakpoint function by replacing the opcode at the breakpoint address with a BGND instruction. Breakpoints may only be set at an opcode address and breakpoints may only be placed at memory addresses implemented as RAM.

When using the target hardware breakpoints, D–BugS08 utilizes the the BDC's single hardware breakpoint and the two breakpoints available in the target MCU's Debug (DBG) module. Both of these breakpoint modules utilize the CPU08 instruction fetch tagging mechanism which only allows breakpoints to be set on instruction opcodes.

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 are specified using a 16-bit hexadecimal number or simple expression and can span the entire 64K memory map.

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.

Example

```
S>bs c000 ffff 'HCS'
Data Found at: $F0AD
S>
```

CAL – Calibrate the target MCU's internal clock reference

Command Line Format

CAL [<TargetBusFreq>][;P]

Parameter Description

< TargetBusFreq >	A 16-bit decimal number representing the desired target bus speed in KHz.
;P	The ASCII string ";P" in upper, lower or mixed case.

Command Description

On S08 devices containing an internal clock reference generator, the CALibrate command can be used to trim the reference to within ±0.2%, or better, of the supplied target bus frequency. The allowable values for the <TargetBusFreq> parameter will vary depending on specifications of the selected target device. On devices such as the S08DZ and S08QG family members, the internal reference generator can be trimmed to produce a range of bus frequencies. The table below shows examples for some of the currently available S08 devices.

Device Family	Max. Bus Frequency (KHz)	Min. Bus Frequency (KHz)
MC9S08DZxx	20000	16000
MC9S08QGx	10000	8000
MC9S08QD8	8000	8000
MC9S08AWxx	4443	4443

The MC9S08AWxx family, and all other S08's using the Internal Clock Generator (ICG) module, is a special case since the FLL in the ICG itself may be used to generate a wide range of bus frequencies using the Multiplication Factor (MFD) and Reduced Frequency Divisor (RFD) bits. For the purpose of calibrating the internal reference generator, the listed maximum and minimum bus frequency of 4443 KHz represents the bus frequency produced by the FLL with the minimum MFD multiplication factor of 4 and the minimum RFD reduction factor of 1 with the internal reference trimmed to 243 KHz.

If the <TargetBusFreq> parameter is omitted from the command line, the target clock reference generator will be calibrated to produce the maximum bus frequency.

The ;P option, if present, will cause the calculated trim value(s) to be programmed into the the Flash locations on the target device reserved for that purpose. Note that all other data the Flash sector containing the trim information is preserved.

Note: The clock calibration procedure utilizes the BDC SYNC pulse to indirectly measure the target clock frequency. To accurately measure the target clock using this method, D-BugS08 configures the output of the clock module to use the internal reference clock as the system clock source. Because the bus clock is the output of the clock module divided by two, the clock frequency measured is actually one half the reference frequency. The CAL command reports double the measured frequency as the value of the internal reference clock. The reported bus clock frequency is calculated based on the measured reference frequency.

Note: The ***absolute*** accuracy of the calibration performed by D-BugS08 on the internal clock reference of the target device is dependent on the absolute accuracy of the clock source of the MCU running D-BugS08. Calibration accuracy ***relative*** to D-BugS08's clock source will be within the target device's specified trimming accuracy specification.

Restrictions

None.

Example

```
S>device

Device: S08QG8, MC9S08QG8
Flash: $E000 - $FFFF
RAM: $0060 - $025F
I/O Regs: $0000 - $005F, $1800 - $184F
Target Bus Frequency: 10000 KHz

TRIM Register Value $93
FTRIM Register Value $00
Measured reference clock frequency: 31254 Hz
Calculated bus frequency:  8001024 Hz

Programming trim value(s) into Flash...

S>
```

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-BugS08 monitor program when the final RTS of the subroutine is executed. When control is returned to D–BugS08, 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–BugS08 places four bytes of data on the users stack that causes control to return to D-BugS08 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 1080
Subroutine Call Returned

 PC    SP    HX    A   CCR = V--H INZC
1080  00FF  0000   00        0110 1000
1080  A603     LDA   #$03
S>
```

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

Entering "DEVICE" on the command line followed by a carriage return will display the name of the currently selected device the on-chip Flash's starting and ending address, the on-chip RAM's starting and ending address, and the Direct Page and High Page I/O Register address ranges.

Entering the DEVICE command followed by the ASCII question mark character (?) will display a list of all the supported HCS08 target devices.

Note that D-BugS08 will automatically select the proper device by comparing the value in the target MCU's SDIDH and SDIDL registers with the values stored in its device table.

Restrictions

None.

Example

```
R>device

Device: S08GB60, MC9S08GB60, MC9S08GT60
Flash: $1080 - $17FF, $182C - $FFFF
RAM: $0080 - $107F
I/O Regs: $0000 - $007F, $1800 - $182B
Target Bus Frequency: 20000 KHz

R>
```

FBULK - Erase target on-chip Flash EEPROM Memory

Command Line Format

FBULK [;<SecByteValue>]

Parameter Description

<SecByteValue> An 8-bit hexadecimal number

Command Description

The FBULK command is used to erase the entire contents of the on-chip Flash memory 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 optional <SecByteValue> allows a value other than $FE to be programmed into the security byte of HCS08 target device. Programming a value other than 1:0 into the lower two bits of the security byte will cause the security feature to be enabled causing a loss of BDM communications when the Flash is not blank.

If the selected target device contains the Internal Clock Generator Module (ICG), Internal Clock Source Module (ICS) or Multi-purpose Clock Generator (MCG), the FBULK command will preserve any clock trim information stored in the Flash locations reserved for this purpose. If the reserved Flash locations are blank before the erasure of the Flash, the FBULK command will perform a calibration operation of the on-chip clock reference. This information is then programmed into the reserved Flash location(s) on the target device.

Note: The ***absolute*** accuracy of the calibration performed by D-BugS08 on the internal clock reference of the target device is dependent on the absolute accuracy of the clock source of the MCU running D-BugS08. Calibration accuracy ***relative*** to D-BugS08's clock source will be within the target device's specified trimming accuracy specification.

Restrictions

None.

Example

```
S>fbulk
S>
```

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

Command Line Format

FLOAD [<AddressOffset>] [;nf]

Parameter Description

<AddressOffset>	A 16-bit hexadecimal number
;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.

D–BugS08 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-BugS08 receives an 'S9' end of file record. If the object file being loaded does not contain an 'S9' record, D–BugS08 will not return its prompt and will continue to wait for the end of file record. Pressing a system Reset will return D–BugS08 to its command line prompt.

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 S-Records that contain a code/data field longer than 64 bytes. Sending an S-Record with a code/data field longer than 64 bytes will cause D–BugS08 to terminate the FLOAD command the issue an error message.

Because the FLOAD command downloads a small 'driver' program into the target MCU's on chip RAM, D-BugS08's breakpoint table is cleared before beginning execution of the 'driver'. This is necessary to prevent previously set breakpoints from accidentally halting the

execution of the driver program.

Example

```
S>fload
************************************************************************
************************************************************************
**********************
S>
```

FSERASE - Erase one or more sectors of target Flash

Command Line Format

FSERASE <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 `FSERASE` command allows one or more sectors of target Flash memory to be erased. Supplying only the <StartAddress> parameter will erase a single Flash sector associated with the supplied address. Supplying the optional <EndAddress> will erase a range of sectors encompassed by the <StartAddress> and <EndAddress> parameters. Note that the range of Flash memory erased is `<StartAddress> & SectorMask` through `<EndAddress> & SectorMask + (SectorSize - 1)`. Where `SectorMask` has a value of $FE00 for Flash memory containing 512 byte sectors.

Before the sector erase operation is performed, the target device is reset and the target's FPROT register is written to $FF, allowing any Flash sectors to be erased regardless of the state of the Flash protection byte.

Note: If the FSERASE command is used to erase the sector containing the security byte, the security byte will be programmed to $FE to prevent the target device from being left in the secured state. If the selected target device contains the Internal Clock Generator Module (ICG), Internal Clock Source Module (ICS) or Multi-purpose Clock Generator (MCG), the FBULK command will preserve any clock trim information stored in the Flash locations reserved for this purpose. If the reserved Flash locations are blank before the erasure of the Flash, the FBULK command will perform a calibration operation of the on-chip clock reference. This information is then programmed into the reserved Flash location(s) on the target device.

Note: The ***absolute*** accuracy of the calibration performed by D-BugS08 on the internal clock reference of the target device is dependent on the absolute accuracy of the clock source of the MCU running D-BugS08. Calibration accuracy ***relative*** to D-BugS08's clock source will be within the target device's specified trimming accuracy specification.

Restrictions

The FSERASE command may not be used when the 'R>' prompt is being displayed indicating that the target system is running a user program.

Example

```
S>fserase 8000 8fff
S>
```

Go, begin execution of user code

Command Line Format

G [<Address>]

Parameter Description

<Address> 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 or a CPU exception occurs. When user code halts for one of these reasons and control is returned to D–BugS08, a message is displayed explaining the reason for program termination. In addition, D-BugS08 displays the CPU08's register contents, disassembles the instruction at the current PC address, and waits for the next D–BugS08 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.

Example

```
S>g 1080
R>md 8000

8000  FF FF FF FF - FF FF FF FF - FF FF FF FF - FF FF FF FF   ................
R>
User Breakpoint Encountered

 PC    SP    HX    A   CCR = V--H INZC
10A2  00FF  0000   01        0110 1000
10A2  1000     BSET  0,$00
S>
```

GT - Go Until, Execute user code until temporary breakpoint is reached

Command Line Format

GT <Address>

Parameter Description

<Address> 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 executing code reaches the temporary breakpoint and control is returned to D-BugS08, a message is displayed explaining the reason for user program termination. In addition, D-BugS08 displays the CPU08's register contents, disassembles the instruction at the current PC address, and waits for the next D-BugS08 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 10AB
R>
Temporary Breakpoint Encountered

 PC    SP    HX    A   CCR = V--H INZC
10AB  00FF  0000  01        0110 1000
10AB  20F3     BRA   $10A0

S>
```

HELP - Display D-BugS08 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-BugS08 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> [;t]  Set communications rate for the terminal
BDMDB  Enter the BDM command debugger
BF <StartAddress> <EndAddress> [<data>] [;nv]  Fill memory with data
BR [<Address>]  Set/Display breakpoints
BS <StartAddress> <EndAddress> '<String>' | <Data8> [<Data8>]  Block Search
CALL [<Address>]  Call user subroutine at <Address>
DEVICE [<DevName>] | [?]  Display/select target device
FBULK [;<SecByteVal>] Erase entire target FLASH contents
FLOAD [<AddressOffset>] [;nf]  Load S-Records into target Flash
FSERASE <StartAddress> [<EndAddress>] Erase one or more sectors of 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>]  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
RESET  Reset target CPU
RM  Modify CPU Register Contents
SO  Step Over subroutine calls
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
VER  Display D-BugS08's Version Number
VERF  [[<AddressOffset>] [;f]] | [;b]  Verify S-Records against memory contents
<Register Name> <Register Value>  Set register contents
       Register Names: PC, SP, HX, A, CCR
       CCR Status Bits: H, I, N, Z, V, C
>
```

LOAD - Load user program in S-Record format

Command Line Format

LOAD [[<AddressOffset>]

Parameter Description

<AddressOffset> A 16-bit hexadecimal number

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.

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-BugS08 will issue its prompt.

The Load command is terminated when D-BugS08 receives an 'S9' end of file record. If the object file being loaded does not contain an 'S9' record, D–BugS08 will not return its prompt and will continue to wait for the end of file record. Pressing a system's Reset button will return D–BugS08 to its command line prompt.

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 $205 was entered as the start address and $217 as the ending address, the actual memory range displayed would be $200 through $21F.

Restrictions

None.

Example

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

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

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 $205 were entered as the start address and $217 as the ending address, the actual memory range displayed would be $200 through $21F.

Restrictions

None.

Example

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

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

MM - 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-BugS08 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 subcommands are used for the modification and verification of memory contents. These subcommands 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 subcommand must be separated from any entered data with at least one space character. If an invalid subcommand character is entered, an appropriate error message 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-BugS08 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 subcommands are used for the modification and verification of memory contents. These subcommands 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 subcommand must be separated from any entered data with at least one space character. If an invalid subcommand character is entered, an appropriate error message 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>…]

Parameter Description

<Address> 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

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

>
```

RD - Display CPU08 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 CPU08's registers. The registers are displayed in the same format used when a breakpoint is encountered.

Restrictions

The CPU registers may not be displayed when the '`R>`' prompt is being displayed.

Example

```
S>rd

 PC    SP    HX    A   CCR = V--H INZC
10A2  00FF  0000  01        0110 1000
10A2  1000     BSET  0,$00
S>
```

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 and place it in active background mode. For S08 devices having a dedicated reset pin, D-BugS08 forces the target MCU into active background mode by asserting the Reset and BKGD pins and then releasing the reset pin while keeping the BKGD pin asserted. Communication with the target is then attempted.

For S08 devices not having a dedicated reset pin, D-BugS08 writes a value to the target MCU's SBDFR register to force a system reset. Immediately after the BDM write operation, the BKGD pin is driven low so that the target device enters active background mode.

With the exception of the program counter (PC), the target processor's registers are initialized with the values stored in D-BugS08's Customization Data area. The PC is initialized with the contents of the target processor's reset vector, memory locations $FFFE and $FFFF.

Restrictions

None.

Example

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

RM - Interactively Modify CPU08 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 CPU08's registers in an interactive manner. As each register and its contents is displayed, D-BugS08 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 CPU08 register and its contents to be displayed on the next line. When the last of the CPU08'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. 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-BugS08 prompt.

The registers are displayed in the following order, one register per line: PC, SP, HX, A, CCR.

Restrictions

The CPU registers may not be modified when the '`R>`' prompt is being displayed.

Example

```
>RM
PC=1206 1200
SP=17FF <CR>
HX=1000 1004
A=27 <CR>
CCR=68 69
PC=1200 .
>
```

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) 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 or JSR. 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.

If the subroutine requires more than a handful of cycles to execute, D-BugS08 will display its R> prompt indicating the target is running. When the subroutine returns, the temporary breakpoint is removed, the CPU08's register contents are displayed and the *next* instruction to be executed is displayed.

Restrictions

None.

Example

```
S>pc 100

 PC    SP    HX    A   CCR = V--H INZC
0100  00FF  0000   00        0110 1000
0100  A6FF     LDA   #$FF
S>so

 PC    SP    HX    A   CCR = V--H INZC
0102  00FF  0000   FF        0110 1100
0102  CD0200   JSR   $0200
S>so


 PC    SP    HX    A   CCR = V--H INZC
0105  00FF  0000   FE        0110 1100
0105  26FB     BNE   $0102
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

The STOP command is used to halt target program execution and place the target processor in active background debug mode.

Restrictions

If the STOP command is entered while the 'S>' prompt is displayed, an error message is displayed and command execution will be terminated.

Example

```
S>asm 1080
1080  A603     LDA   #$03         >
1082  C71802   STA   $1802        >
1085  C6FFBE   LDA   $FFBE        >
1088  B74E     STA   $4E          >
108A  A608     LDA   #$08         >
108C  B748     STA   $48          >
108E  A670     LDA   #$70         >
1090  B749     STA   $49          >
1092  B64A     LDA   $4A          >
1094  A448     AND   #$48         >
1096  A148     CMP   #$48         >
1098  26F8     BNE   $1092        >.
S>g
R>stop
Target Processor Has Been Stopped

 PC    SP    HX    A   CCR = V--H INZC
10A0  00FF  0000   01        0110 1000
10A0  9D       NOP
S>
```

T - Trace (Execute) CPU08 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 CPU08'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

None.

Example

```
S>t

 PC    SP    HX    A   CCR = V--H INZC
1082  00FF  0000   03        0110 1000
1082  C71802   STA   $1802

S>t

 PC    SP    HX    A   CCR = V--H INZC
1085  00FF  0000   03        0110 1000
1085  C6FFBE   LDA   $FFBE

S>t

 PC    SP    HX    A   CCR = V--H INZC
1088  00FF  0000   64        0110 1000
1088  B74E     STA   $4E

S>t

 PC    SP    HX    A   CCR = V--H INZC
108A  00FF  0000   64        0110 1000
108A  A608     LDA   #$08

S>
```

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 may contain external circuitry such as voltage regulators or peripherals that need to be enabled before performing in–circuit programming operations. 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 external 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 no data has previously 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

None.

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> [;<SRecSize>]

Parameter Description

<StartAddress>	A 32-bit hexadecimal number
<EndAddress>	A 32-bit hexadecimal number
[;<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.

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.

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`' in upper, lower or mixed case

Command Description

Entering the USEHBR command causes D-BugS08 to use the hardware breakpoint capability of the target microcontroller. Using hardware breakpoints allows up to three, program only breakpoints to be set in Flash or other nonvolatile memory. By default, D-BugS08 uses the hardware breakpoint capability of the target MCU. To utilize D-BugS08's 10 software breakpoints, the USEHBR command should be entered with the '`OFF`' parameter on the command line.

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

Restrictions

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 8A4
Breakpoints: 0810  0835  08A4
S>br 957
Breakpoint Table Full
S>
```

VER - Display the D-BugS08 Version Number

Command Line Format

VER

Parameter Description

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

Command Description

The `VER` command that displays the current version of D-BugS08. This allows high level debuggers to determine what version of D-BugS08 is running so they know what features they can be utilized.

Restrictions

None.

Example

```
S>ver
1.0.0b1
S>
```

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

Command Line Format

VERF [<AddressOffset>]

Parameter Description

<AddressOffset> A 16-bit hexadecimal number

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.

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.

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-BugS08 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-BugS08 receives an 'S9' end of file record. If the object file being loaded does not contain an 'S9' record, D–BugS08 will not return its prompt and will continue to wait for the end of file record. Pressing system Reset will return D–BugS08 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 CPU08 Register Value

Command Line Format

<RegisterName> <RegisterValue>

Parameter Description

Where <RegisterName> is one of the following CPU08 register names:

Register Name	Description	Legal Range
PC	Program Counter	$0..$FFFF
SP	Stack Pointer	$0..$FFFF
HX	HX-Index Register	$0..$FFFF
A	A Accumulator	$0..$FF
CCR	Condition Code 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
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
I	IRQ 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 CPU08 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 CPU08's register contents to be displayed.

Restrictions

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

```
S>pc 1080

 PC    SP    HX    A   CCR = V--H INZC
1080  00FF  0000   00        0110 1000
1080  A603     LDA   #$03
S>hx 1234

 PC    SP    HX    A   CCR = V--H INZC
1080  00FF  1234   00        0110 1000
1080  A603     LDA   #$03
S>c 1

 PC    SP    HX    A   CCR = V--H INZC
1080  00FF  1234   00        0110 1001
1080  A603     LDA   #$03
S>v 1

 PC    SP    HX    A   CCR = V--H INZC
1080  00FF  1234   00        1110 1001
1080  A603     LDA   #$03
S>
```

Appendix A

Adapting D-BugS08 to Use Alternate Crystal Frequencies

As supplied, D-BugS08 v1.x.x is configured to operate with an 8.0 or 16.0 MHz crystal or oscillator. The 8.0 or 16.0 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-BugS08 with a crystal or oscillator frequency other than 8.0 or 16.0 MHz, one area of D-BugS08 and the bootloader need to be modified.

Bootloader Modifications

Even though the bootloader is separate from D-BugS08, 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 ;value for FCLKDIV/ECLKDIV register.
             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-BugS08 Modifications

The changes required to D-BugS08 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-BugS08 v1.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   $68                   ; initial CCR register value.
Reserved1:  dc.b   $00
UserA:      dc.b   $00                   ; initial A accumulator value.
UserHX:     dc.w   $0000                 ; initial HX register value.
Reserved2:  dc.w   $0000
UserPC:     dc.w   $0000                 ; initial PC value.
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   $0000                 ; on-chip EEPROM base addr avail to D-BugS08.
EESize:     dc.w   0                     ; on-chip EEPROM size avail to D-BugS08.
EEDelay:    dc.w   $ef02                 ; adress of 1 mS delay routine.
DlyCnt:     dc.w   Eclock/4000
;
```

Figure B-2, D-BugS08 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.

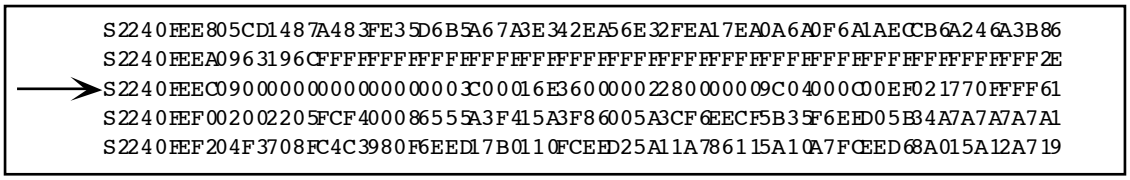

Figure B-3, S-Record Line Containing Customization Data

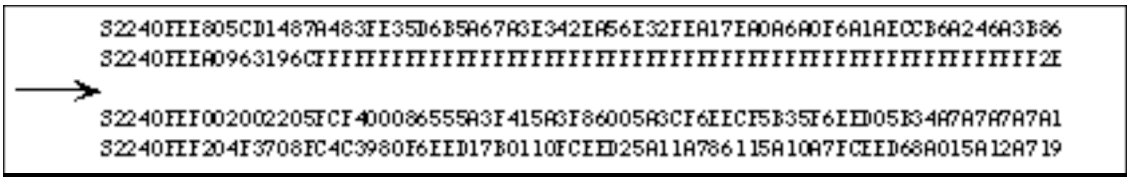

Figure B-4, Line Containing Customization Data Removed

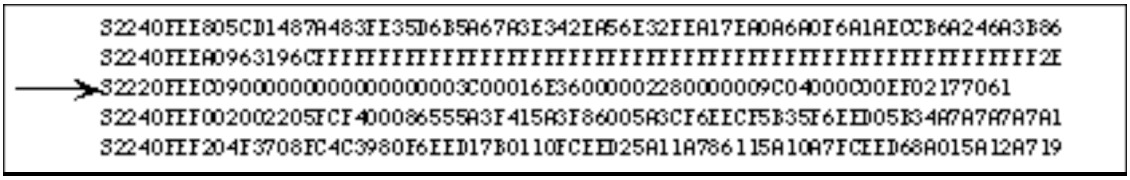

Figure B-5, Replacement of S-Record Header, Byte Count and Load Address

After making the necessary changes and successfully assembling the source file, a single line in the D-BugS08 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 B

BDM Debugger

The BDM debugger contained within D-BugS08 is intended to be used by tool developers or factory engineers when evaluating new silicon or debugging BDM communication problems, however, it may also be useful to end customers. Using D-BugS08's low level BDM driver routines, the BDM debugger allows individual BDM commands to be sent to a target device directly from the command line. Unlike D-BugS08, which maintains constant communications with a connected target, no BDM communication occurs other than during the execution of a command. Also note that the BDM debugger does not perform any checks to ensure a target is connected before executing a command.

The BDM debugger can be entered either from D-BugS08's command line (see the BDMDB command description) or from the startup menu (option 4) if communication cannot be established with a target. When entering the BDM debugger from the D-BugS08 command line, the timing of the BDM driver routines is the same as the timing established by D-BugS08. If the BDM debugger is entered by selecting option four of the startup menu, the timing of the BDM driver routines can be establish by using the BDM debugger SYNC command.

The following list summarizes the BDM debugger command set. Each command's function and command line syntax are described in detail.

- ACKDI - Disable ACK Handshake protocol.
- ACKEN - Enable ACK Handshake protocol.
- A - Read or write the CPU A accumulator.
- BKGD - Place target in active background mode.
- CCR - Read or write the CPU Condition Code Register
- EXIT - Exit the BDM debugger and (re)enter D-BugS08.
- G - Exit active background, begin target execution at the current PC.
- HX - Read or write the target HX index register.
- PC - Read or write the target Program Counter.
- RB - Read a byte from target memory.
- RBR - Read BDC Breakpoint Register.
- RBS - Read Byte With Status.
- RESET - Reset target in special single chip mode.
- RL - Read Last, read byte from previously read memory location with status
- RNX - Pre-increment target HX-index register by 1 and read the byte pointed to by HX.
- RNXS - Pre-increment target HX-index register by 1 and read the byte pointed to by HX, With Status.
- RS - Read BDC Status/Control register.
- SP - Read or write the target Stack pointer.
- SYNC - Measure target BDC speed and configure BDM drivers.
- T - Execute a single instruction and return to active background.

- TG - Enable instruction tagging and begin target execution at the current PC.
- WB - Write a byte to target memory.
- WBR - Write Breakpoint Register.
- WBS - Write byte with status.
- WC - Write BDC Status/Control Register.
- WNX - Pre-increment target HX-index register by 1 and write the byte pointed to by HX.
- WNXS - Pre-increment target HX-index register by 2 and write the byte pointed to by HX, With Status.

ACKDI - Disable ACK Handshake protocol

Command Line Format

ACKDI

Parameter Description

None.

BDM Command Name/Type

ACK_DISABLE ($D6) / Non-intrusive

Command Description

The ACKDI command disables the ACK handshake protocol if it was previously enabled. An ACK handshake pulse is not issued at the completion of the command.

Example

```
?ACKDI
?
```

ACKEN - Enable ACK Handshake protocol

Command Line Format

ACKEN

Parameter Description

None.

BDM Command Name/Type

ACK_ENABLE ($D5) / Non-intrusive

Command Description

The ACKEN command enables the ACK handshake protocol if it was previously disabled. An ACK handshake pulse is issued at the completion of the command.

Example

```
?ACKEN
?
```

A - Read or write the A-accumulator

Command Line Format

A [<Data8>]

Parameter Description

<Data8> An 18-bit hexadecimal number representing the new value of the A-accumulator.

BDM Command Name/Type

READ_A ($68) or WRITE_A ($48) / Active Background

Command Description

The A command is used to read or write the value of the target's A-accumulator. Entering the A command without the <Data8> parameter causes the current value of the A-accumulator to be displayed by sending the READ_A BDC command to the target. Supplying the optional <Data8> parameter results in the value of the target's A-accumulator being modified by sending the WRITE_A BDC command.

Example

```
?A
A: 12
?A 56
?A
A: 56
?
```

BKGD - Place target in active background mode

Command Line Format

BKGD

Parameter Description

None.

BDM Command Name/Type

BACKGROUND ($90) / Non-intrusive

Command Description

The BKGD command causes the target device to enter active background if firmware is enabled (ENBDM = 1). An ACK handshake pulse is issued when the part enters active background mode if the ACK protocol has been previously enabled.

Example

```
?BKGD
?
```

CCR - Read or write the Condition Code Register

Command Line Format

CCR [<Data8>]

Parameter Description

<Data8> An 8-bit hexadecimal number representing the new value of the Condition Code Register.

BDM Command Name/Type

READ_CCR ($69) or WRITE_CCR ($49) / Active Background

Command Description

The CCR command is used to read or write the value of the target's Condition Code Register. Entering the CCR command without the <Data8> parameter causes the current value of the Condition Code Register to be displayed by sending the READ_CCR BDC command to the target. Supplying the optional <Data8> parameter results in the value of the target's Condition Code Register being modified by sending the WRITE_CCR BDC command.

Example

```
?CCR
CCR: 68
?CCR 69
?CCR
CCR: 69
?
```

EXIT - Exit the BDM debugger and (re)enter D-BugS08

Command Line Format

EXIT

Parameter Description

None.

BDM Command Name/Type

N/A

Command Description

The EXIT command does not correspond to one of the BDM hardware or software commands, instead it is used to leave the BDM debugger and return to D-BugS08.

Example

```
?EXIT
S>
```

G - Exit active background, begin target execution at the current PC

Command Line Format

G

Parameter Description

None.

BDM Command Name/Type

GO ($08) / Active Background

Command Description

The G command causes the target device to leave active background and begin program execution at the instruction pointed to by the current value of the Program Counter. An ACK handshake pulse is issued when the part leaves active background mode if the ACK protocol has been previously enabled.

Example

```
?G
?
```

HX - Read or write the HX index register

Command Line Format

HX [<Data16>]

Parameter Description

<Data16> A 16-bit hexadecimal number representing the new value of the HX index register.

BDM Command Name/Type

READ_HX ($6C) or WRITE_HX ($4C) / Active Background

Command Description

The HX command is used to read or write the value of the target's HX index register. Entering the command without the <Data16> parameter causes the current value of the HX index register to be displayed by sending the READ_HX BDM command to the target. Supplying the optional <Data16> parameter results in the value of the target's HX index register being modified by sending the WRITE_HX BDM command.

Example

```
?HX
HX: C05A
?HX 5678
?HX
HX: 5678
?
```

PC - Read or write the Program Counter

Command Line Format

PC [<Data16>]

Parameter Description

<Data16> A 16-bit hexadecimal number representing the new value of the Program Counter.

BDM Command Name/Type

READ_PC ($6B) or WRITE_PC ($4B) / Active Background

Command Description

The PC command is used to read or write the value of the target's Program Counter. Entering the PC command without the <Data16> parameter causes the current value of the Program Counter to be displayed by sending the READ_PC BDM command to the target. Supplying the optional <Data16> parameter results in the modification of the target's Program Counter by sending the WRITE_PC BDM command.

Example

```
?PC
PC: C13A
?PC f000
?PC
PC: F000
?
```

RB - Read a byte from target memory

Command Line Format

RB <Address16>

Parameter Description

<Address16> A 16-bit hexadecimal number representing the address at which to read a byte of data.

BDM Command Name/Type

READ_BYTE ($E0) / Non-intrusive

Command Description

The RB command is used to read a byte of data from the target's memory. Because this is a Non-intrusive command, it may be executed even if the target is not in active background mode. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory.

Example

```
?rb c000
C000: CF
?
```

RBR - Read BDC Breakpoint Register

Command Line Format

RBR

Parameter Description

None.

BDM Command Name/Type

READ_BKPT ($E2) / Non-intrusive

Command Description

The RBR command is used to read the contents of the BDC breakpoint register. Because this is a Non-intrusive command, it may be executed even if the target is not in active background mode. For enhanced BDM modules, if handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory.

Example

```
?rbr
Breakpoint: 180A
?
```

RBS - Read a byte from target memory and return BDC status Register

Command Line Format

RBS <Address16>

Parameter Description

<Address16> A 16-bit hexadecimal number representing the address at which to read a byte of data.

BDM Command Name/Type

READ_BYTE_WS ($E1) / Non-intrusive

Command Description

The RBS command is used to read a byte of data from the target's memory. In addition to reading the byte of data from the target address location, the contents of the BDC Status/Control register is returned. Because this is a Non-intrusive command, it may be executed even if the target is not in active background mode. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory.

Example

```
?rbs 180a
180A: 00  Status: 80
?
```

RESET - Reset target in special single chip mode

Command Line Format

RESET

Parameter Description

None.

BDM Command Name/Type

N/A

Command Description

The RESET command does not correspond to one of the BDM Non-intrusive or Active background commands, instead it is used to reset the target MCU, placing it in Special Single-chip mode.

Example

```
?reset
?
```

RL - Read Last, read byte from previously read memory location with status

Command Line Format

RL

Parameter Description

None.

BDM Command Name/Type

READ_LAST ($E8) / Non-intrusive

Command Description

The RL command is used to read a byte of data from the same address as the location of the previous read. Note that this includes READ_NEXT and READ_NEXT_WS commands. In addition to reading the byte of data from the target location, the contents of the BDC Status/Control register is returned. Because this is a Non-intrusive command, it may be executed even if the target is not in active background mode. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory.

Example

```
?rb 180a
180A: 00
?rl
Data: 00  Status: 80
?
```

RNX - Pre-increment HX-index register by 1, read the byte pointed to by HX

Command Line Format

RNX

Parameter Description

None.

BDM Command Name/Type

READ_NEXT ($70) / Active Background

Command Description

The RNX command reads a byte of data from target memory. Before the read is performed, the target's HX index register is incremented by one. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory.

Example

```
?rnx
[+X]: C9
?
```

RNXS -Pre-increment HX-index register by 1, read the byte pointed to by HX & return Status

Command Line Format

RNXS

Parameter Description

None.

BDM Command Name/Type

READ_NEXT_WS ($71) / Active Background

Command Description

The RNXS command reads a byte of data from target memory. Before the read is performed, the target's HX index register is incremented by one. In addition to reading the byte of data from the target address location, the contents of the BDC Status/Control register is returned. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory.

Example

```
?hx
HX: 1084
?rnxs
[+X]: C6  Status: C8
?
```

RS - Read the BDC Status/Control register

Command Line Format

RS

Parameter Description

None.

BDM Command Name/Type

READ_STATUS ($E4) / Non-intrusive

Command Description

The RS command reads the BDC Status/Control register.

Example

```
?rs
Status: C8
?
```

SP - Read or write the Stack Pointer

Command Line Format

SP [<Data16>]

Parameter Description

<Data16> A 16-bit hexadecimal number representing the new value of the Stack Pointer.

BDM Command Name/Type

READ_SP ($6F) or WRITE_SP ($4F) / Active Background

Command Description

The SP command is used to read or write the value of the target's Stack Pointer. Entering the SP command without the <Data16> parameter causes the current value of the Stack Pointer to be displayed by sending the READ_SP BDM command to the target. Supplying the optional <Data16> parameter results in the modification of the target's Stack Pointer by sending the WRITE_SP BDM command.

Example

```
?SP
SP: 3FDC
?SP 4000
?SP
SP: 4000
?
```

SYNC - Measure BDC clock speed and reinitialize the BDM drivers

Command Line Format

SYNC

Parameter Description

None.

BDM Command Name/Type

N/A / Non-intrusive

Command Description

The SYNC command does not correspond to an actual BDC serial command, instead it uses the SYNC protocol to measure the operating frequency of the BDC module. After obtaining the BDC module operating frequency, the BDM software drivers are initialized using the measured frequency. Note that if the CLKSW bit in the BDC Status/Control register is set, the BDC module's input clock frequency is the same as the target's bus clock. Otherwise, the BDC module will be clocked by the 8.0 MHz nominal Self Clock mode clock.

Example

```
?rs
Status: 88
?sync
BDC Speed: 20000 KHz
?wc 80
?rs
Status: FF
?sync
BDC Speed: 7679 KHz
?rs
Status: 80
?
```

T - Execute a single instruction at the current PC, return to active background

Command Line Format

T

Parameter Description

None.

BDM Command Name/Type

TRACE1 ($10) / Active Background

Command Description

Execute one instruction at the current PC and return to active background mode. An ACK handshake pulse is issued when the part returns to active background mode if the ACK protocol has been previously enabled.

Example

```
?t
?
```

TG - Enable instruction tagging, begin execution at the current PC

Command Line Format

TG

Parameter Description

None.

BDM Command Name/Type

TAGGO ($18) / Active Background

Command Description

The TG command causes the target device to, enable instruction tagging, leave active background and begin program execution at the instruction pointed to by the current value of the Program Counter. Note that because the S08 devices do not have external instruction tagging capability, the TAGGO BDC command operates the same as the GO command. An ACK handshake pulse is issued when the part returns to active background mode if the ACK protocol has been previously enabled.

Example

```
?TG
?
```

WB - Write a byte to target memory

Command Line Format

WB <Address16> <Data8>

Parameter Description

<Address16> A 16-bit hexadecimal number representing the address at which to write a byte of data.

<Data8> An 8-bit hexadecimal number representing the data to be written to <Address16>

BDM Command Name/Type

WRITE_BYTE ($C0) / Non-intrusive

Command Description

The WB command is used to write a byte of data to the target's memory. Because this is a Non-intrusive command, it may be executed even if the target is not in active background mode. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been written to memory, indicating that the next BDM command may be sent.

Example

```
?wb 1000 16
?
```

WBR - Write a word to the BDC breakpoint register

Command Line Format

WBR <Data16>

Parameter Description

<Data16> An 16-bit hexadecimal number representing the data to be written to the BDC breakpoint register.

BDM Command Name/Type

WRITE_BKPT ($C2) / Non-intrusive

Command Description

The WBR command is used to write a word of data to the BDC's breakpoint register.

Example

```
?wbr 1087
?
```

WBS - Write a byte to target memory with status

Command Line Format

WBS <Address16> <Data8>

Parameter Description

<Address16> A 16-bit hexadecimal number representing the address at which to write a byte of data.

<Data8> An 8-bit hexadecimal number representing the data to be written to <Address16>

BDM Command Name/Type

WRITE_BYTE ($C1) / Non-intrusive

Command Description

The WB command is used to write a byte of data to the target's memory. In addition to writing the byte of data to the target address, the contents of the BDC Status/Control register is returned. Because this is a Non-intrusive command, it may be executed even if the target is not in active background mode. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been written to memory, indicating that the next BDM command may be sent.

Example

```
?wbs 80 55
Status: FF
?sync
BDC Speed: 7660 KHz
?wbs 80 55
Status: 00
?
```

WC - Write the BDC Status/Control Register

Command Line Format

WC <Data8>

Parameter Description

<Data8> An 8-bit hexadecimal number representing the data to be written to the BDC Status/Control register

BDM Command Name/Type

WRITE_CONTROL ($C4) / Non-intrusive

Command Description

The WC command is used to write a byte to the BDC Status/Control register. Because this is a Non-intrusive command, it may be executed even if the target is not in active background mode.

Example

```
?wc C8
?
```

WNX - Pre-increment HX-index register by 1, write the byte pointed to by HX

Command Line Format

WNX <Data8>

Parameter Description

<Data8> An 8-bit hexadecimal number representing the data to be written to the location pointed to by HX.

BDM Command Name/Type

WRITE_NEXT ($50) / Active Background

Command Description

The WNX command writes a byte of data from target memory. Before the write is performed, the target's HX index register is incremented by one. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory.

Example

```
?wnx a5
?
```

WNXS - Pre-increment HX-index register by 1, write the byte pointed to by HX, return status

Command Line Format

WNXS <Data8>

Parameter Description

<Data8> An 8-bit hexadecimal number representing the data to be written to the location pointed to by HX.

BDM Command Name/Type

WRITE_NEXT_WS ($51) / Active Background

Command Description

The WNXS command writes a byte of data from target memory and returns the value of the BDC Status/Control register. Before the write is performed, the target's HX index register is incremented by one. If handshaking has been enabled, an ACK handshake pulse is issued when the data has been read from memory, indicating the status register may be read.

Example

```
?wnx a5
?
```