

AN2104

Using Background Debug Mode for the M68HC12 Family

By Timothy J. Airaudi
Applications Engineering, Microcontroller Division
Austin, Texas

Introduction

This application note describes the basic operation of the background debug mode (BDM) and some of its applications, as it relates to Motorola's M68HC12 Family of microcontrollers (MCU). Examples of in-circuit programming of internal FLASH memory and in-circuit debugging, using P&E Microcomputer Systems' BDM interface cable and its software, are also contained in this document.

The BDM's main purpose is to allow debugging of the actual microcontroller being used in the user's target application. This takes the place of hardware such as an in-circuit emulator, which uses external components to attempt to duplicate operation of the MCU from outside of the target application.

Instead of having this external hardware, and a variety of potential problems, the debug logic is built into the MCU's on-chip integration module. This differs from other systems that have the debugging logic located in the central processor unit (CPU). Not having the debugging logic in the CPU allows for reading and writing of memory locations, while the CPU is executing user code, with no degradation in real-time operation. This is an example of the BDM being enabled but not active.

Application Note

When the BDM is active, it takes over control of the microprocessor, which allows for debugging, etc.

Other examples of what the BDM can be used for, besides debugging, vary from programming EPROM, EEPROM, and FLASH (internal and external) to performing calibration on a target application (in manufacturing and in the field) to transferring collected and stored information to another system.

Theory of Operation

Because software packages, such as P&E Microcomputer Systems' Windows Development Package (PKG12), take care of the operation of the BDM, this discussion does not go into great detail. For more in-depth information on this subject, refer to the documents referenced in [Technical Resources](#) at the end of this document.

The operation of the BDM system requires a host PC with software, a BDM interface POD or BDM interface, and the user's target application. See [Figure 1](#). The host PC is connected to the POD with a DB-25 parallel cable from the PC's parallel port. The POD is then connected to the target application via a custom 6-pin BDM connector and cable. See [Figure 2](#).

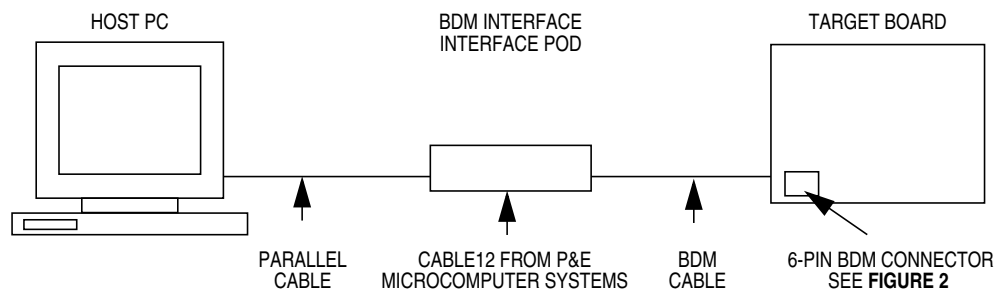


Figure 1. BDM System

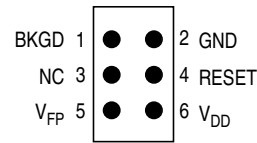


Figure 2. BDM Tool Connector

To communicate with the BDM on the part, two pins are used: BKGD and GND. This method of serial interface is used to both send and receive data. A special communications protocol is used that resynchronizes at the beginning of each bit. By doing this, a greater frequency tolerance for synchronization is allowed.

All bits are started with a falling edge signal that is initiated by the external host. After the MCU sees this falling edge, it waits nine E-clock cycles and then samples the level on the BKGD pin. The data is transferred MSB (most significant bit) first at the rate of 16 E-clock cycles per bit. The E-clock is defined as the SYSCLK divided by two.

The two types of BDM commands are:

- Hardware
- Firmware

When using hardware commands, the BDM is enabled, but not active, and the user's code is running. See [Table 1](#). These commands allow all internal and external memory, which is accessible to the microcontroller, to be read or written. This also includes on-chip I/O (input/output) and control registers.

The control logic watches the bus for any free bus cycles that it can use to execute the hardware command. By using the free bus cycles, the CPU is not disturbed. If, however, a free cycle is not found within a specified time, it will use a bus cycle, which temporarily freezes the CPU.

Table 1. BDM Hardware Commands

| Command | Opcode (Hex) | Data | Description |
|--------------------------------|--------------|-----------------------------------|--|
| BACKGROUND | 90 | None | Enter background mode (if firmware enabled). |
| READ_BD_BYTE | E4 | 16-bit address 16-bit data out | Read from memory with BDM in map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte. |
| STATUS ⁽¹⁾ | E4 | FF01, 0000 0000 (out) | READ_BD_BYTE \$FF01. Running user code. (BGND instruction is not allowed.) |
| | | FF01, 1000 0000 (out) | READ_BD_BYTE \$FF01. BGND instruction is allowed. |
| | | FF01, 1100 0000 (out) | READ_BD_BYTE \$FF01. Background mode active (waiting for single wire serial command). |
| READ_BD_WORD | EC | 16-bit address 16-bit data out | Read from memory with BDM in map (may steal cycles if external access) must be aligned access. |
| READ_BYTE | E0 | 16-bit address 16-bit data out | Read from memory with BDM out of map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte. |
| READ_WORD | E8 | 16-bit address 16-bit data out | Read from memory with BDM out of map (may steal cycles if external access) must be aligned access. |
| WRITE_BD_BYTE | C4 | 16-bit address 16-bit data in | Write to memory with BDM in map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte. |
| ENABLE_FIRMWARE ⁽²⁾ | C4 | FF01, 1xxx xxxx (in) | Write byte \$FF01, set the ENBDM bit. This allows execution of commands which are implemented in firmware. Typically, read STATUS, OR in the MSB, write the result back to STATUS. |
| WRITE_BD_WORD | CC | 16-bit address 16-bit data in | Write to memory with BDM in map (may steal cycles if external access) must be aligned access. |
| WRITE_BYTE | C0 | 16-bit address 16-bit data in | Write to memory with BDM out of map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte. |
| WRITE_WORD | C8 | 16-bit address 16-bit data in | Write to memory with BDM out of map (may steal cycles if external access) must be aligned access. |

1. STATUS command is a specific case of the READ_BD_BYTE command.

2. ENABLE_FIRMWARE is a specific case of the WRITE_BD_BYTE command.

To execute firmware commands, the user must have the BDM enabled and active. See [Table 2](#). When the BDM is active, it has control of the CPU, which executes code out of the BDM ROM.

Table 2. BDM Firmware Commands

| Command | Opcode (Hex) | Data | Description |
|------------|--------------|-----------------|---|
| READ_NEXT | 62 | 16-bit data out | $X = X + 2$; Read next word pointed to by X |
| READ_PC | 63 | 16-bit data out | Read program counter |
| READ_D | 64 | 16-bit data out | Read D accumulator |
| READ_X | 65 | 16-bit data out | Read X index register |
| READ_Y | 66 | 16-bit data out | Read Y index register |
| READ_SP | 67 | 16-bit data out | Read stack pointer |
| WRITE_NEXT | 42 | 16-bit data in | $X = X + 2$; Write next word pointed to by X |
| WRITE_PC | 43 | 16-bit data in | Write program counter |
| WRITE_D | 44 | 16-bit data in | Write D accumulator |
| WRITE_X | 45 | 16-bit data in | Write X index register |
| WRITE_Y | 46 | 16-bit data in | Write Y index register |
| WRITE_SP | 47 | 16-bit data in | Write stack pointer |
| GO | 08 | None | Go to user program |
| TRACE1 | 10 | None | Execute one user instruction then return to BDM |
| TAGGO | 18 | None | Enable tagging and go to user program |

Application Note

BDM Registers

Seven BDM registers are mapped into addresses \$FF00–\$FF06. See [Table 3](#).

NOTE: Remember that the BDM firmware ROM and registers contain different data than the normal memory mapped locations for these addresses.

Table 3. BDM Registers

| Address | Register | Mnemonic |
|---------------|--------------------------|-------------|
| \$FF00 | BDM instruction register | INSTRUCTION |
| \$FF01 | BDM status register | STATUS |
| \$FF02–\$FF03 | BDM shift register | SHIFTER |
| \$FF04–\$FF05 | BDM address register | ADDRESS |
| \$FF06 | BDM CCR holding register | CCRSV |

Only two registers are discussed here:

- BDM status register (STATUS)
- BDM CCR (condition code register) holding register (CCRSV)

The BDM status register can be read at any time, but must not be written to during BDM operation. See [Figure 3](#) for a description of the bits.

Address: \$FF01

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|-------------------------|-------|--------|-------|-----|-------|---|---|-------|
| Read: | ENBDM | EDMACT | ENTAG | SDV | TRACE | 0 | 0 | 0 |
| Write: | | | | | | | | |
| Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Single-Chip Peripheral: | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3. BDM Status Register (STATUS)

This register can be read or written by BDM commands or firmware.

ENBDM — Enable BDM Bit (permit active background debug mode)
 0 = BDM cannot be made active (hardware commands still allowed).
 1 = BDM can be made active to allow firmware commands.

BDMACT — Background Mode Active Status Bit

0 = BDM not active

1 = BDM active and waiting for serial commands

ENTAG — Instruction Tagging Enable Bit

Set by the TAGGO instruction and cleared when BDM is entered

0 = Tagging not enabled or BDM active

1 = Tagging active (BDM cannot process serial commands while tagging is active.)

SDV — Shifter Data Valid Bit

Shows that valid data is in the serial interface shift register. Used by firmware-based instructions.

0 = No valid data

1 = Valid data

TRACE

Asserted by the TRACE1 instruction

The second register of interest is the BDM CCR holding register. This register contains the value of the CPU's condition code register (CCR) from the user's program upon entering the BDM. See [Figure 4](#).

Address: \$FF06

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|--------|-------|------|------|------|------|------|------|-------|
| Read: | CCR7 | CCR6 | CCR5 | CCR4 | CCR3 | CCR2 | CCR1 | CCR0 |
| Write: | | | | | | | | |
| Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 4. BDM CCR Holding Register (CCRSV)

Operation of Active BDM

Here is a brief description of what transpires when going into the active BDM:

- When the CPU gets the command to go into the BDM, the user's return address is stored in a temporary register.
- Next, the BDM ROM is turned on and the CPU fetches a vector that points to the beginning of the BDM firmware program.

- Next the BDM firmware saves the contents of the user's D register in another temporary register and then saves the user's CCR register in the CCRSAV register.
- The BDM firmware then checks the ENBDM bit in the STATUS register to see if it will be allowed to go into the active BDM. If it is, the BDM firmware enters a software loop and waits for a valid firmware command in which to execute. The user's program counter (PC), stack pointer (SP), and X and Y registers are not changed by the BDM firmware, so the user doesn't need to save or stack these values.

During exit from the BDM, the user's register values are restored and a value is stored in the BDM STATUS register. Then a jump command is executed to resume execution of the user's program.

M68HC12 Operating Modes

The two basic modes of operation (see [Table 4](#)) for the M68HC12 Family are:

- Normal modes — Provide protection for control registers from being accidentally modified
- Special modes — Allow access to these control registers for system development and special testing

If any of the normal operating modes are entered (BKGD high), the BDM is available, but must be enabled and/or made active.

If the special single-chip mode is selected (BKGD, MODA, and MODB all low), the BDM comes up enabled and active.

[Table 4](#) also shows that the states of the BKGD, MODA, and MODB pins determine a specific mode where the port A and port B pins are configured for different functions.

Table 4. Mode Selection

| BKGD | MODB | MODA | Mode | Port A | Port B |
|------|------|------|------------------------------------|-------------------------|---------------------|
| 0 | 0 | 0 | Special single chip | General-purpose I/O | General-purpose I/O |
| 0 | 0 | 1 | Special expanded narrow | ADDR[15:8] DATA[7:0] | ADDR[7:0] |
| 0 | 1 | 0 | Special peripheral | ADDR DATA | ADDR DATA |
| 0 | 1 | 1 | Special expanded wide | ADDR DATA | ADDR DATA |
| 1 | 0 | 0 | Normal single chip | General-purpose I/O | General-purpose I/O |
| 1 | 0 | 1 | Normal expanded narrow | ADDR[15:8] DATA[7:0] | ADDR[7:0] |
| 1 | 1 | 0 | Reserved (forced to peripheral) | — | — |
| 1 | 1 | 1 | Normal expanded wide | ADDR DATA | ADDR DATA |

These examples deal with the levels on the BKGD, MODA, and MODB pins during a reset to determine which mode the part will come up in. The user can also change the mode of operation by writing to the mode register after the part is powered up. See [Figure 5](#).

The MODE register can be read at any time. However, writing to this register presents some restrictions. First, if the part comes up in the normal mode, it can be changed only to another normal mode. This change can be done only once.

The special mode does not have this limitation, since the values of the MODA and MODB pins can be changed as many times as desired as long as the part remains in special mode.

Next, coming up in the special mode, the part can change to the normal mode, but must write to the SMODN bit in this register two times, as the first write is ignored.

Application Note

Address: \$000B

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|--------|-------|------|------|------|------|--------|---|-------|
| Read: | SMODN | MODB | MODA | ESTR | IVIS | EBSWAI | 0 | EME |
| Write: | | | | | | | | |

Reset states:

| | | | | | | | | |
|--------------------------|---|---|---|---|---|---|---|---|
| Normal expanded narrow: | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Normal expanded wide: | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Special expanded narrow: | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| Special expanded wide: | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| Peripheral: | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| Normal single-chip: | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Special single-chip: | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Figure 5. Mode Register (MODE)

Operating Mode and Background Debug Mode Hints

These hints will help steer the user away from the most commonly made mistakes.

- The states of the MODA and MODB pins, upon power-up, determine how the port A and port B pins will be configured (see [Table 4](#)).
- The BKGD pin is used for two purposes:
 - It determines, upon reset, which operating mode the part will enter, normal or special (see [Table 4](#)).
 - Then it is used as the serial communication pin for the BDM.
- Once the part is operating in a mode, the mode can be changed by writing to the mode register. The limitations to this are listed in [Figure 5](#).
- When in normal operating mode, special modes cannot be accessed.

- When in normal operating mode, another normal operating mode can be accessed, but this can be done only once.
- To change to the normal operating mode, when the part is in special operating mode, a 1 (one) must be written twice to the SMODN bit in the mode register.
- When the part comes up in special single-chip mode, the BDM is enabled and active.
- When the part comes up in special single-chip mode, it accesses the BDM ROM, not the normal memory mapped locations at \$FF00–\$FFFF.
- To perform hardware commands, the BDM does not need to be active (see [Table 1](#)).
- To perform firmware commands, the BDM must be enabled and active (see [Table 2](#)).
- The BDM does not operate in stop mode.

Background Debug Mode Application Examples

Two BDM application examples are given here in a step-by-step format.

In-Circuit Programming of Internal FLASH

This application example of the BDM explains how to perform in-circuit programming of the internal FLASH memory of an MC68HC912B32 using P&E Microcomputer Systems' Cable12 POD and software (see [Figure 1](#)). The target board for this example is the M68EVB912B32 evaluation board.

Follow these steps in order:

1. Load P&E's PKG12Z software.
2. Connect a parallel cable from the host PC to the Cable12 POD.
3. Connect the 6-pin BDM cable from the POD to the evaluation board making sure that pin 1 of the cable is connected to pin 1 of the POD and target. On the evaluation board, make sure that jumpers W3 and W4 are in the EVB positions and jumper W7 is in the V_{DD} position.
4. Apply +5 Vdc to P5 of the evaluation board and +12 Vdc to W8.
5. Launch P&E's WinIDE.
6. Open P&E's sample code named SCI.
7. Assemble/compile this file. See [Figure 6](#).
8. Launch the programmer. If the correction assistant window opens, select the correct parallel port being used. Defaults should work for the other options in this window. See [Figure 7](#).
9. Select the 9b32_32k.12p programming algorithm.
10. Input \$8000 for the base address when prompted.
11. Move jumper W7, on the evaluation board, to the V_{PP} position.
12. Select **Erase Module**.
13. Ensure that the SCI.s19 file is in the S-record in the configuration window. If not, select **Specify S record** and select this file.
14. Select **Program Module**.
15. After programming is complete, move jumper W7 to the V_{DD} position. Do not leave the programming voltage on the FLASH.
16. The SCI.s19 file has now been erased and programmed into the FLASH of the MC68HC912B32 using the BDM. Select **Verify Module** to verify that this programming is correct. The code also can be viewed by selecting **Show Module** at address \$8000.

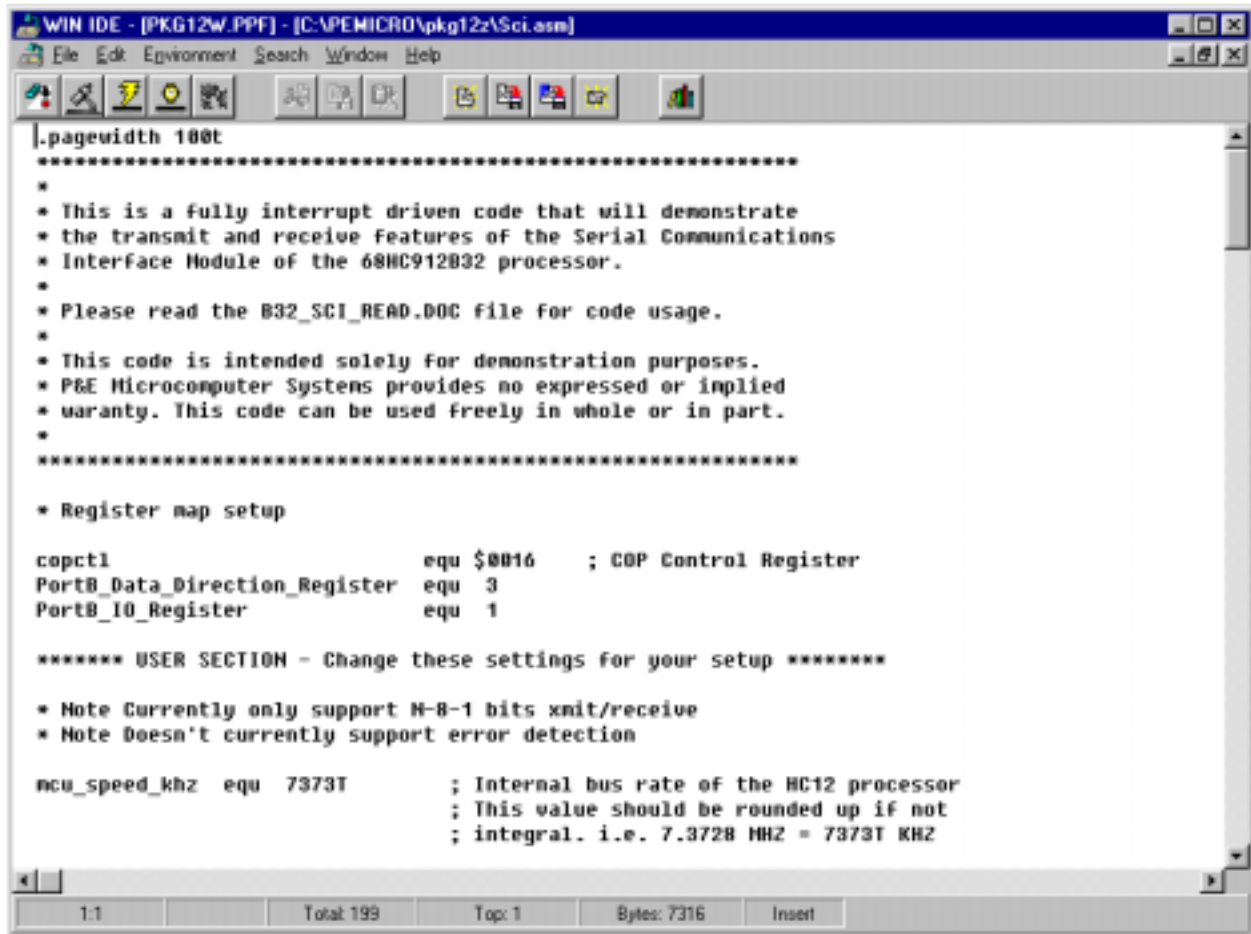


Figure 6. P&E's WinIDE Window

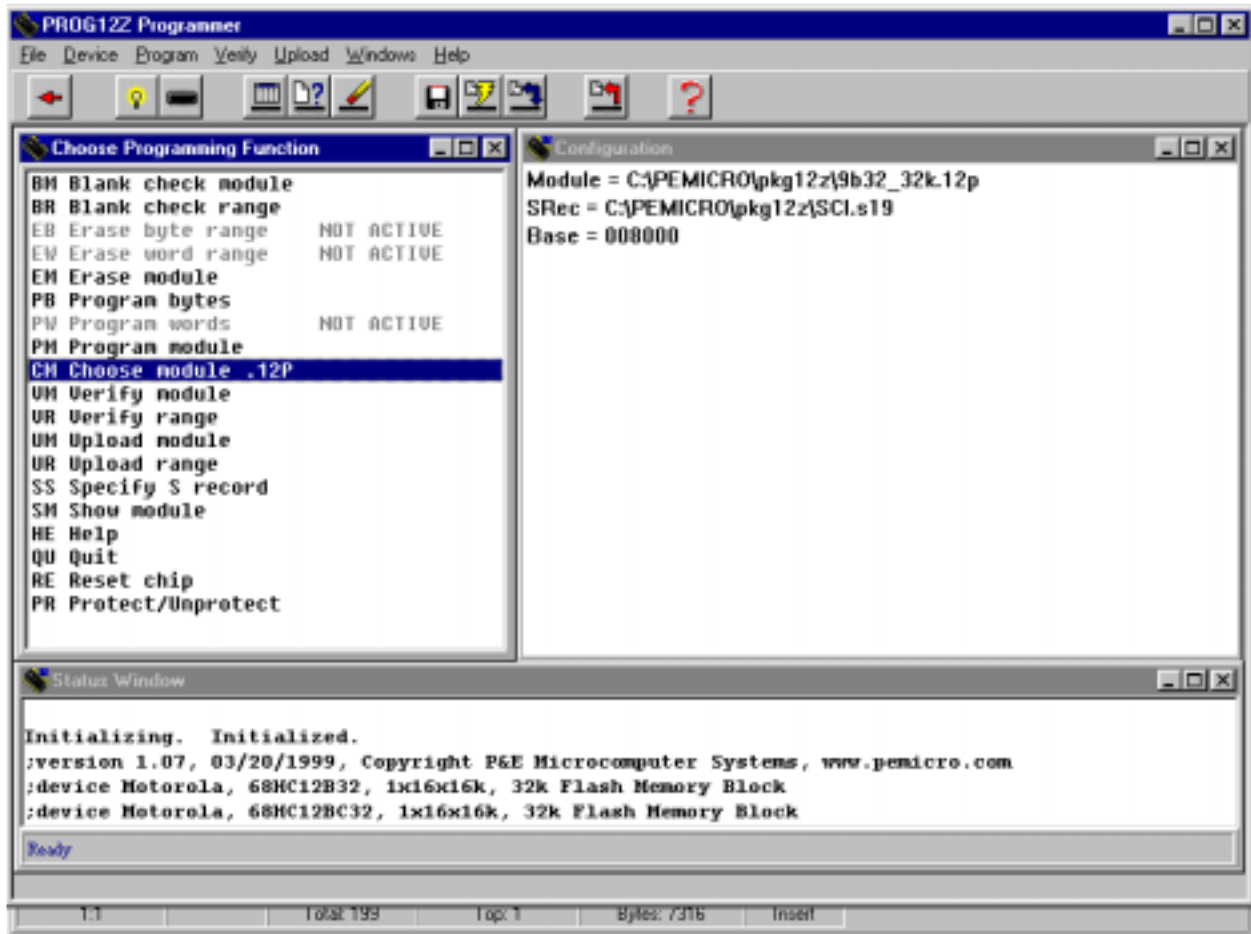


Figure 7. P&E’s Programmer Window

In-Circuit Debugging

This application example of the BDM explains how to perform in-circuit debugging of an MC68HC912B32 using P&E Microcomputer Systems’ Cable12 POD and software (see [Figure 1](#)). The target board for this example will be the M68EVB912B32 evaluation board.

Follow these steps in order:

1. Load P&E’s PKG12Z software.
2. Connect a parallel cable from the host PC to the Cable12 POD.
3. Connect the 6-pin BDM cable from the POD to the evaluation board making sure that pin 1 of the cable is connected to pin 1 of the POD and target. On the evaluation board, make sure that

jumpers W3 and W4 are in the EVB positions and jumper W7 is in the V_{DD} position.

4. Apply +5 Vdc to P5 of the evaluation board.
5. Launch P&E's WinIDE.
6. Open P&E's sample code named SCI.
7. Assemble/compile this file. See **Figure 6**. At this point, ensure that the FLASH is programmed as in the previous application example in **In-Circuit Programming of Internal FLASH**.
8. Launch the debugger. If the correction assistant window opens, select the correct parallel port being used. Defaults should work for the other options in this window. See **Figure 8**.
9. Verify that the correct S19 is loaded in the debugger by selecting the **File** drop down menu and selecting **Load S19 File** and the SCI.S19 file.
10. In the **Execute** drop down menu, select **Reset Processor**.
11. From this point, the code can be debugged by selecting **Single step**, **Multiple step**, or **Go**.

Breakpoints also can be set by selecting the line of code chosen for a breakpoint, clicking the right mouse button, and selecting **Toggle Breakpoint at Cursor**.

Application Note

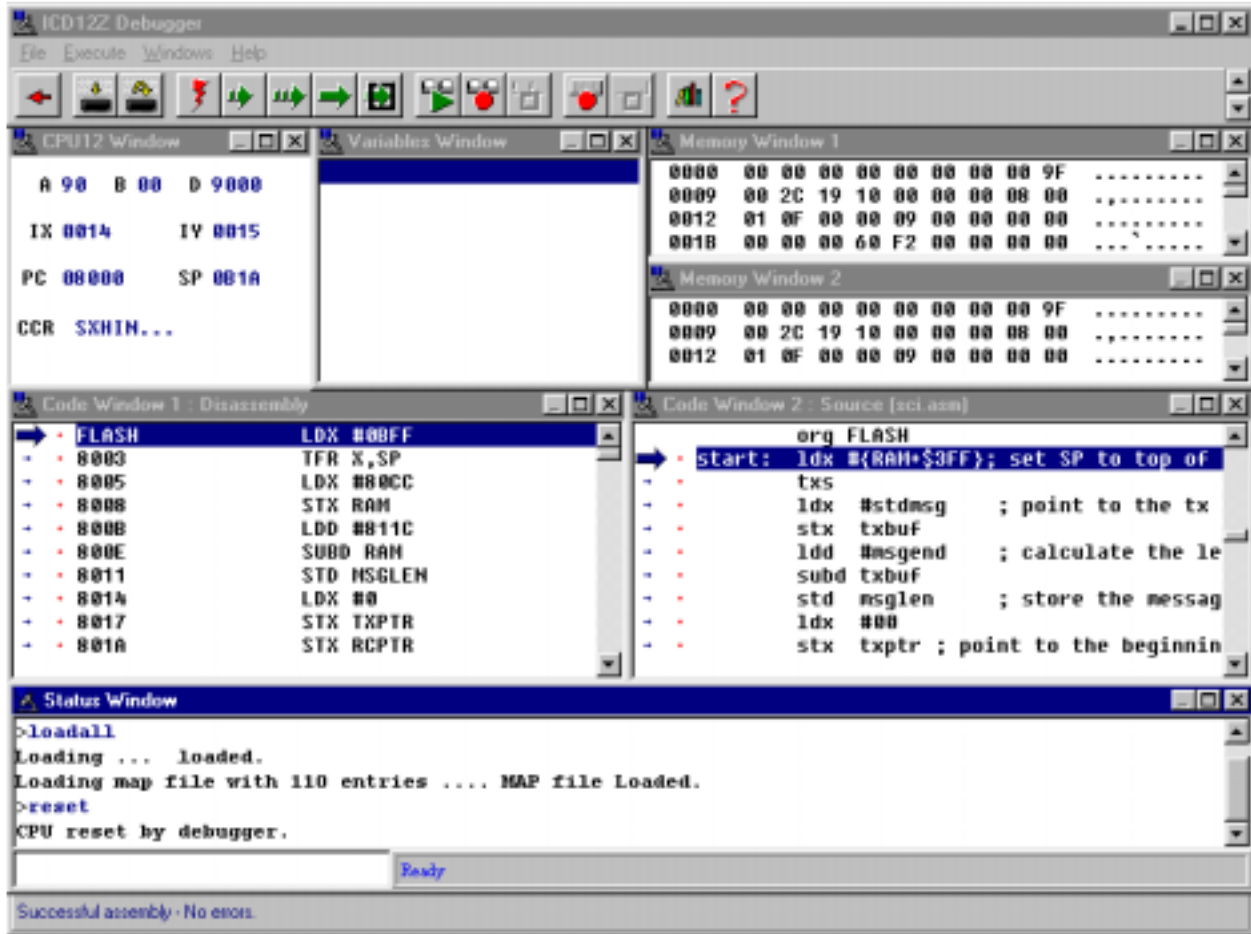


Figure 8. P&E's Debugger Window

Summary

This application note gives an overview of the BDM as it relates to Motorola's M68HC12 Family of MCUs. By providing the appropriate connections for the BDM in the user's application, and using a BDM interface POD with software, it is easy to debug code, erase, or program the FLASH in the target application.

Technical Resources

- *Software and Hardware Engineering: Motorola M68HC12* by Fredrick M. and James M. Sibigtroth
- *CPU12 Reference Manual*, document order number CPU12RM/AD
- *M68HC12B Family Advance Information*, Motorola document order number M68HC12B/D
- *MC68HC812A4 Advance Information*, Motorola document order number MC68HC812A4/D
- *MC68HC912D60 Advance Information*, Motorola document order number MC68HC912D60/D
- *MC68HC912DG128 Advance Information*, Motorola document order number MC68HC912DG128/D





Application Note

How to Reach Us:

Home Page:

www.freescale.com

E-mail:

support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

