CME-12B/BC

Development Board for Motorola 68HC12B32 and 68HC12BC32 Microcontrollers

xiom
anufacturing
™

2813 Industrial Ln. • Garland, TX 75041 • (972) 926-9303 FAX (972) 926-6063
email: Gary@axman.com • web: http://www.axman.com

CONTENTS

GETTING STARTED

The Axiom CME-12B/BC single board computer is a fully assembled, fully functional development system for the Motorola 68HC12B32 and BC32 microcontrollers, complete with wall plug power supply and serial cable. Support software for this development board is provided for Windows 95 and NT operating systems and DOS.

Follow the steps in this section to get started quickly and verify everything is working correctly.

Installing the Software

1. Insert the Axiom 68HC12 support disk in your PC. If the setup program does not start, run the file called "SETUP.EXE" on the disk.

2. Follow the instructions on screen to install the support software onto your PC.

3. The flash programming utilities require you to specify the board you are using. You should select either “CME-12B32” or "CME-12B/BC" for this board.

Board Startup

Follow these steps to connect and power on the board. This assumes you're using the provided AXIDE or AX12 terminal program (installed in the previous section) or a similar communications terminal program on your PC. If you're using a different terminal program than the one provided, set it's parameters to 9600 baud, N,8,1.

1. Make certain the CONFIG SWITCH is set as follows:

1	2	3	4	**5**	6	**7**	**8**
ON	ON	ON	ON	**OFF**	ON	**OFF**	**OFF**

2. Connect one end of the supplied 9-pin serial cable to a free COM port on your PC. Connect the other end of the cable to the COM1 port on the CME-12B/BC board.

3. Apply power to the board by plugging in the power adapter that came with the system.

4. If everything is working properly, you should see a message to “PRESS KEY TO START MONITOR…” in your terminal window. Press the ENTER key and you should see:

```
D-Bug12 v2.0.2
Copyright 1996 - 1997 Motorola Semiconductor
For Commands type "Help"

> _
```

5. Your board is now ready to use! If you do not see this message prompt, or if the text is garbage, see the **TROUBLESHOOTING** section at the end of this manual.

Support Software

There are many useful programs and documents on the included HC12 support disk that can make developing projects on the CME-12B/BC easier. You should browse the disk and copy anything you want to your hard drive. See the README.TXT file for a description of what is included.

The flash programming utilities communicate with the board via its COM1 port and the supplied cable. These programs also include a simple Terminal for interfacing with other programs running on the CME-12B/BC, such as the D-Bug12 and information from your own programs that send output to the serial port.

Also on the disk is a free assembler, example source code, and other tools to get you started.

Software Development

Software development on the CME12B32 can be performed using either the DBUG12 monitor utility installed in EPROM (sockets U6/U7), the DBUG12 monitor programmed into the internal Flash EPROM or a Background Debug Module (BDM) connected to the BDM-IN connector. Any of these tools can be used to assist in creating and debugging your program stored in either internal RAM ($800-$C00) or external RAM (U4/U5 see **Memory Map**).

After satisfactory operation running under a debugger, your program can be written to the Internal Flash Memory by relocating its start address to $8000 and programming it using either the included programming utilities or a BDM. Your program will then run automatically whenever the board is powered on or RESET is applied.

Option jumpers and switches on the board allow for easy transition from one memory type to another and restoring an operating monitor.

TUTORIAL

This section was written to help you get started developing software with the CME-12B/BC board. Be sure to read the rest of this manual as well as the documentation on the disk if you need further information.

The following sections take you through the complete development cycle of a simple "hello world" program, which sends the string "Hello World" to the serial port.

Creating source code

You can write source code for the CME-12B/BC board using any language that compiles to Motorola 68HC12 instructions. Included on the software disk is a free Assembler and also a freeware C compiler and Basic compiler.

You can write your source code using any ASCII text editor. You can use the free EDIT or NOTEPAD programs that come with your computer. A better code editor can be downloaded at: http://www.winedit.com. Once your source code is written and saved to a file, you can assemble or compile it to a Motorola S-Record (hex) format. This type of output file usually has a .MOT, .HEX or .S19 file extension and is in a format that can be read by the programming utilities to be programmed into the CME-12B/BC board.

It's important to understand your development board's use of Memory and Addressing when writing source code so you can locate your code at valid addresses. For example, when in debug mode, you should put your program CODE in External RAM. In assembly language, you do this with ORG statements in your source code. Any lines following an ORG statement will begin at that ORG location, which is the first number following the word ORG, for example: **`ORG $2000`**.

You must start your DATA (or variables) in a RAM location unused by your program, for example: ORG $1000. When finished debugging, you must change these ORG statements so that your program is moved to a valid EEPROM area - somewhere after hex 8000. Do this by putting an ORG $8000 in front of your Program CODE. Data may remain where it is or be moved down to internal RAM starting at ORG $800. You must also program the STACK register somewhere at the top of your available RAM, for example hex BFF. Do this with this instruction as the first instruction in your program code: LDS #$0BFF.

A look at the example programs on the disk can make all of this clearer. If you're using a compiler instead of an assembler, consult the compiler documentation for methods used to locate your code and data.

Source code created to run under the D-bug12 monitor environment will be slightly different than code written for stand-alone operation. The monitor contains interrupt and RESET vectors that your code must provide when it's no longer running under the monitor. See the **Programming Flash EEPROM** section for more information on this.

Assembling source code

An example program called “HELLO.ASM” is provided under the \EXAMPLE directory.

You can assemble your source code using command line tools under a DOS prompt by typing:

```
AS12 HELLO.ASM –LHELLO
```

Most compilers and assemblers allow many command line options so using a MAKE utility or batch file is recommended if you use this method. Run AS12 without any arguments to see all the options, or see the AS12.TXT file on the disk.

The programming utilities provided with this board also contain a simple interface to the free assembler. Use it by selecting "Build" or “Assembler” from the menu. This will prompt you for the file to be assembled.

DO NOT use long path names (> 8 characters). The free assembler is an old DOS tool that does not recognize them.

If there are no errors in your source code, 2 output files will be created:

HELLO.S19	a Motorola S-Record file that can be programmed into memory
HELLO.LST	a common listing file which shows the relationship between source and output

The listing file is especially helpful to look at when debugging your program. If your program has errors, they will be displayed and no output will be generated, otherwise the listing file will be displayed.

If you prefer a windows integrated programming environment – try the Motorola MCU-EZ tools. Refer to the MCU-EZ documentation on the disk for more information.

Also, a port for the free GNU C compiler and tools for the HC12 is in the works and may be finished as you read this. Check www.axman.com or :

http://home.worldnet.fr/~stcarrez/m68hc11_port.html

Running your application

After creating an S-Record file you can "upload" it to the development board for a test run. The provided example “HELLO.ASM” was created to run from RAM so you can use the D-Bug12 Monitor to test it without programming it into EEPROM.

If you haven’t done so already, verify that the CME-12B/BC board is connected and operating properly by following the steps under “GETTING STARTED” until you see the D-Bug12 prompt, then follow these steps to run your program:

1. Press and release the RESET button on the CME-12B/BC board. You should see the PRESS ANY KEY message. Hit the return key ↵ to get the monitor prompt.

2. Type **`LOAD`** ↵
   This will prepare D-Bug12 to receive a program.

3. Select Upload or Send and when prompted for a file name select your assembled program file in s-record format that was created in the previous section called: **`HELLO.S19`**
   Your program will be sent to the board thru the serial port.

4. When finished loading you will see the > prompt again.
   Type **`CALL 1000`** ↵
   This tells D-Bug12 to execute the subroutine at address $1000, which is the start of our test program.

5. If everything is working properly you should see the message “Hello World” echoed back to your terminal screen then, since we return at then end of our program, a line containing the internal register status displayed by D-Bug12 and a message prompt.

6. If you do not get this message, try going thru this tutorial once more, then if still no go, see the **TROUBLESHOOTING** section in this manual

You can modify the hello program to display other strings or do anything you want. The procedures for assembling your code, uploading it to the board and executing it remain the same. D-Bug12 has many powerful features such as breakpoints, assembly/disassembly, memory dump and modify and program trace. Type HELP at the D-Bug12 prompt for a listing of commands or consult the D-Bug12 documentation on the disk for more information.

Programming Flash EEPROM

You can program your application into EEPROM so it executes automatically when you apply power to the board as follows:

1. Make a backup copy of HELLO.ASM then use a text editor to modify it. Change the start of the program to **$8000** which is the beginning of the EEPROM. Do this by changing the **"ORG $1000" to "ORG $8000"**.

2. Remove the comment **;** character from before the following line to initialize the stack pointer which is necessary when running outside of D-Bug12:

```
        LDS     #$C00   ; initialize the stack pointer
```

3. Add a comment **;** character to the beginning of the first RTS statement, which will cause the program to end gracefully with an endless loop:

```
;       RTS     ; return (use this only if called, from monitor
        bra     ENDPROG ; endless loop
```

4. Remove the comment **;** character from before the following 2 lines at the end, to set the reset vector to go to the beginning of the program (the label START) when powered on:

```
org     $fffe           reset vector
        fdb     START
```

5. Re-Assemble HELLO.ASM as described in the "Assembling Source Code" section.

6. Select "CME12BC" under AxIDE.

7. Select the "Program" option and when prompted for a file name, enter the new HELLO.S19 file then select [OK].

8. Set the CONFIG SWITCH positions 1,2,3,4 and 6 to ON.

9. Press the RESET button on the board then select Continue or hit ENTER. A new utilities menu should be displayed. If you have trouble here, see the TROUBLESHOOTING section.

10. Set the CONFIG SWITCH position 5 ON. The red VPP light should come on.

11. When prompted to "Erase" choose Yes.

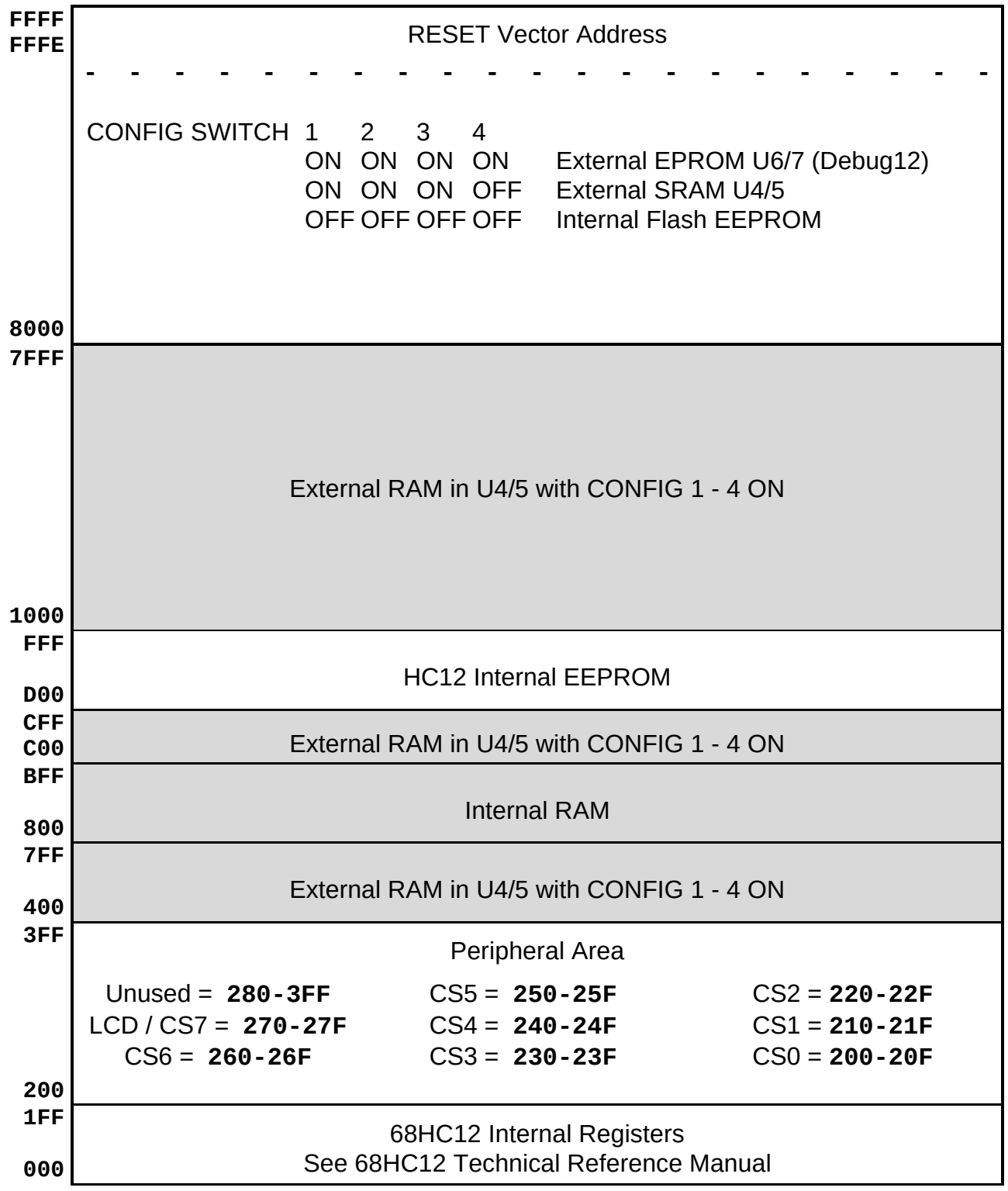
12. When finished programming, set the CONFIG SWITCH positions 1,2,3,4 and 5 OFF. The VPP light should turn off.

13. Cycle power or press RESET on the board. Your new program should start automatically and the "Hello World" prompt should be displayed in the terminal window.

To return to the D-Bug12 monitor program, set the CONFIG SWITCH positions 1,2,3 and 4 back ON then press RESET.

MEMORY MAP

Following is the default memory map for this development board. Consult the HC12 technical reference manual on the support disk for details of the internal memory map for the processor.



CONFIG SWITCH

The CME-12B/BC board is shipped from the manufacturer with the following default CONFIG SWITCH settings:

1	2	3	4	**5**	6	**7**	**8**
ON	ON	ON	ON	**OFF**	ON	**OFF**	**OFF**

The 8 position CONFIG SWITCH provides an easy method of configuring the CME-12B/BC32 board operation. Following are the configuration switch descriptions and HC12 I/O port usage:

CONFIG SWITCH	OPERATION when in ON position	HC12 I/O PORT USED
1	MODE A selection (see Mode chart below)	MODA / PE5
2	MODE B selection (see Mode chart below)	MODB / PE6
3	EXT –External Memory enable (1)	N/A
4	MON –Monitor Memory enable (2)	N/A
5	VPP - Flash VPP voltage enable	N/A
6	RXD - Serial Port RXD input enable	PS0 / RXD
7	PC0 –CAN Port RXCAN enable	PC0
8	PC1 –CAN Port TX enable	PC2

(1) Enables memory bus operation for access to board memory. Expanded bus must be on for proper operation.

(2) Enables monitor EPROM's in memory map at 0x8000 – FFFF hex if CONFIG SWITCH position 3 is also on. When in off position memory space is SRAM for BDM use.

MODE CHART

Single Chip Mode	A and B = OFF
Expanded Wide Mode	A and B = ON

MEM-SEL JUMPERS

Memory sockets U4/5 are shipped with 32K byte RAM devices. U6/7 are shipped with 32K byte EPROM devices programmed with the D-bug12 monitor.

1,3,4	IN: enables RAM or EEPROM, 32K devices to U6/7
1,3	IN: enables RAM or EEPROM, 8K devices in U6/7
2,4	IN: enables EPROM, 8K or 32K devices in U6/7 (Default)

PORTS AND CONNECTORS

LCD_PORT

The LCD_PORT interface is connected to the data bus and memory mapped to locations 270 – 27F hex assigned to CS7. For the standard display, address 270 is the Command register, address 271 is the Data register.

The interface supports all OPTREX™ DMC series displays in 8 bit bus mode with up to 80 characters and provides the most common pinout for a dual row rear mounted display connector. Power, ground, and Vee are also available at the LCDPORT-1 connector.

+5V	2	1	GND
A0	4	3	LCD-Vee
LCD1	6	5	/RW
D9	8	7	D8
D11	10	9	D10
D13	12	11	D12
D15	14	13	D14

Command Register: **$270**
Data Register: **$271**

LCD-Vee is supplied by U13 and is adjusted by the R18 Potentiometer (adjustable resistor).

See the file **KEYLCD12.ASM** for an example program using this LCD connector.

LCD3	2	1	LCD2
	4	3	LCD4

Additional lines can be used as enables for larger panels and are mapped as:

LCD2 = $274 & $275
LCD3 = $278 & $279
LCD4 = $27C & $27D

KEYPAD

1	PS4
2	PS5
3	PS6
4	PS7
5	PP4
6	PP5
7	PP6
8	PP7

The KEYPAD connector is a passive 8-pin connector that can be used to connect a 4 x 4 matrix (16 key) keypad device. The connector is mapped to the HC12 I/O ports S and P. This interface is implemented as a software keyscan. Pins PS4-7 are used as column drivers which are active low outputs. Pins PP4-7 are used for row input and provide an idle hi condition with internal pull-ups to provide active key detection under software control.

See the HC12 Technical Reference Manual for a full description of these pins and the file **KEYLCD12.ASM** for an example program using this connector.

MCU_PORT

The **MCU_PORT** provides access to the peripheral features and I/O lines of the HC12 as follows:

A14	**1**	**2**	A15
D0	**3**	**4**	D1
D2	**5**	**6**	D3
D4	**7**	**8**	D5
D6	**9**	**10**	D7
/XIRQ	**11**	**12**	/DBE
VFP	**13**	**14**	/LSTRB
VRH	**15**	**16**	VRL
+5V	**17**	**18**	GND
PAD0	**19**	**20**	PAD1
PAD2	**21**	**22**	PAD3
PAD4	**23**	**24**	PAD5
PAD6	**25**	**26**	PAD7
PS0 / RXD	**27**	**28**	PS1 / TXD
PS2	**29**	**30**	PS3
PS4	**31**	**32**	PS5
PS6	**33**	**34**	PS7
PC6	**35**	**36**	PC5
PC4	**37**	**38**	PC3
PC2	**39**	**40**	PC1 / TXCAN
PC0 / RXCAN	**41**	**42**	PP7
PP6	**43**	**44**	PP5
PP4	**45**	**46**	PP3
PP2	**47**	**48**	PP1
PP0	**49**	**50**	PT1
PT0	**51**	**52**	PT3
PT2	**53**	**54**	PT5
PT4	**55**	**56**	PT7
PT6	**57**	**58**	GND
+5V	**59**	**60**	GND

D0 – D7 Low Byte of the Data Bus in Wide Expanded Mode. Port B in Single Chip Mode.

/XIRQ HC12 XIRQ interrupt input .

VFP Programming voltage, 12v, when VPP_EN jumper is on.

/LSTRB HC12 LSTRB (PE3) output indicates 8 bit bus access. Should be enabled in software for bus use.

PP0 – PP7 HC12 Port P I/O or PWM port. PP3-7 also used by the KEYPAD Port.

PT0 – PT7 HC12 Port T I/O or Timer port.

VRH / VRL HC12 A/D Converter Reference Pins. See A/D Reference Section.

PAD0 – PAD7 HC12 Port AD is an input port or the A/D Converter inputs.

PS0 – PS7 HC12 Port S I/O or Serial Port lines. PS4-7 also used by the KEYPAD Port.

RXD / TXD Serial Port (SCI) receive and transmit pins.

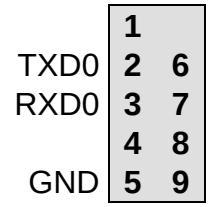
PC0 – PC7 HC12 PDLC I/O or CAN lines.

CAN_PORT

1	GND
2	CAN-H
3	CAN-L
4	+5V

The CAN_PORT connector provides an interface to the MSCAN12 on the microcontroller. See the MC68HC912BC32 data sheet for information on using this peripheral.

COM1 SERIAL PORT



The **COM-1** port has a Female DB9 connector that interfaces to the HC12 internal SCI0 serial port. It uses a simple 2 wire asynchronous serial interface.

Pins 1, 4, and 6 are connected for default handshake standards.
Pins 7 and 8 are connected for default handshake standards.

Handshake pins can be easily isolated and connected to I/O ports if necessary.

J2

1	T2IN
2	R2OUT
3	R2IN
4	T2OUT

The J2 connector (near COM1) contains spare RS232 translator inputs and outputs. It can be used for implementing hardware handshaking on COM1 if necessary.

BUS_PORT

The BUS_PORT supports off-board memory devices as follows:

GND	**1**	**2**	D11
D10	**3**	**4**	D12
D9	**5**	**6**	D13
D8	**7**	**8**	D14
A0	**9**	**10**	D15
A1	**11**	**12**	A2
A10	**13**	**14**	A3
/ OE	**15**	**16**	A4
A11	**17**	**18**	A5
A9	**19**	**20**	A6
A8	**21**	**22**	A7
A12	**23**	**24**	A13
/ WE	**25**	**26**	CS0
CS1	**27**	**28**	CS2
CS3	**29**	**30**	CS4
CS5	**31**	**32**	IRQ
+5V	**33**	**34**	/P-SEL
/RW	**35**	**36**	CS6
E	**37**	**38**	CS7
GND	**39**	**40**	/ RESET

D8 - D15 High Byte Data Bus in Wide Expanded Mode and Peripheral 8 bit data bus. Port A in Single Chip Mode.

A0 – A15 Memory Addresses 0 to 15.

/OE Memory Output Enable signal, Active Low. Valid with ECLK and R/W high.

CS0 – CS7 Peripheral chip selects, 16 bytes each located at $200 - $27F hex, 8 bit access (narrow bus).

/WE Memory Write Enable signal, Active Low. Valid with ECLK high and R/W low.

IRQ HC12 IRQ (PE1) Interrupt Input.

/RW HC12 Read/Write (PE2) control signal.

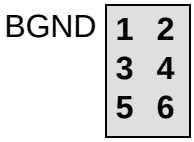
E HC12 ECLK (PE4) bus clock signal. Stretch should be enabled in software.

/P-SEL Selects Peripheral area, register following space, 8 bits wide.

/RESET HC12 active low RESET signal.

BDM-IN

The BDM-IN port is a 6 pin header compatible in pinout with the Motorola Background Debug Mode (BDM) Pod. This allows the connection of a background debugger for software development, programming and debugging in real-time, since the BDM control logic does not reside in the CPU.



See the HC12 Technical Reference Manual for complete documentation of the BDM.

A Background Debug Module is available from the manufacturer.

The **BDM-OUT** port is provided on the board for Motorola MC68HC912B32 EVU Board compatibility. Consult the EVU board documentation for usage information.

EVU Compatibility Jumpers

The EVB-H3 and EVB-H4 jumpers provide backward compatibility with the Flash Options on the Motorola MC68HC912B32 EVU Board. These can be installed by the user and the Debug12.s19 file programmed into the Internal Flash EPROM for EVU similar operation of the development board. Normally this is not necessary due to the Utilities provided in the UTL12 EPROM's.

TROUBLESHOOTING

The CME-12B/BC board is fully tested and operational before shipping. If it fails to function properly, inspect the board for obvious physical damage first. Ensure that all IC devices in sockets are properly seated. Verify the communications setup as described under GETTING STARTED and see the **Tips and Suggestions** sections following for more information.

The most common problems are improperly configured communications parameters, and attempting to use the wrong COM port.

1. Verify that your communications port is working by substituting a known good serial device or by doing a loop back diagnostic.

2. Verify the jumpers on the board are installed correctly.

3. Verify the power source. You should measure approximately 9 volts between the GND and +9V test point pads on the board.

4. If no voltage is found, verify the wall plug connections to 115VAC outlet and the power connector.

5. Disconnect all external connections to the board except for COM1 to the PC and the wall plug.

6. Make sure that the RESET line is not being held low.
   Check for this by measuring the RESET pin on P4 for +5V.

7. Verify the presence of an 8MHz sine wave on the crystal, or 2MHz E clock signal if possible.

Tips and Suggestions

Following are a number of tips, suggestions and answers to common questions that will solve many problems users have with the CME-12B/BC development system. You can download the latest software from the Support section of our web page at:

www.axman.com

Utilities

- If you're trying to program memory or start the HC12 Utilities, make sure all jumpers and CONFIG SWITCH settings are correct.
- Be certain that the data cable you're using is bi-directional and is connected securely to both the PC and the board. Also, make sure you are using the correct serial port.
- Make sure the correct power is supplied to the board. You should only use a 9 volt, 300 mA adapter or power supply. If you're using a power strip, make sure it is turned on.
- Make sure you load your code to an address space that actually exists. See the Memory Map if you're not sure.
- Make sure you're not over-writing memory used by the monitor program.
- If you're running in a multi-tasking environment (such as Windows™) close all programs in the background to be certain no serial conflict occurs.

Code Execution

- Make sure the CONFIG SWITCH is set for the proper mode. If executing from the BDM, you should turn switches 1-4 OFF. If debugging from internal flash EEPROM, disable any reset macro's.
- If you're using D-Bug12 breakpoints may not be acknowledged if you use the CALL command. You should use one of the GO command instead.
- Check the HC12 reset vector located at FFFEh - FFFFh. These 2 bytes contain the address where execution will begin when the unit is powered on.
- When running your code stand-alone, you must initialize ALL peripherals used by the micro, including the Stack, Serial Port, etc.
- You must either reset the COP watchdog timer in the main loop of your code or disable it when not running under MBug or BDM mode. The micro enables this by default and if you don't handle it your code will reset every couple of ms.

ImageCraft C

- Your make or build should create a .MAP file. Some versions change this to a .MP file. At the top of this file should be a label __START. This is where you should CALL or GO to when debugging in D-Bug12.

TABLES

TABLE 1. LCD Command Codes

Command codes are used for LCD setup and control of character and cursor position. All command codes are written to LCD panel address $B5F0. The BUSY flag (bit 7) should be tested before any command updates to verify that any previous command is completed. A read of the command address $B5F0 will return the BUSY flag status and the current display character location address.

Command	Code	Delay
Clear Display, Cursor to Home	**$01**	1.65ms
Cursor to Home	**$02**	1.65ms
Entry Mode:		
Cursor Decrement, Shift off	**$04**	40us
Cursor Decrement, Shift on	**$05**	40us
Cursor Increment, Shift off	**$06**	40us
Cursor Increment, Shift on	**$07**	40us
Display Control:		
Display, Cursor, and Cursor Blink off	**$08**	40us
Display on, Cursor and Cursor Blink off	**$0C**	40us
Display and Cursor on, Cursor Blink off	**$0E**	40us
Display, Cursor, and Cursor Blink on	**$0F**	40us
Cursor / Display Shift: (nondestructive move)		
Cursor shift left	**$10**	40us
Cursor shift right	**$14**	40us
Display shift left	**$18**	40us
Display shift right	**$1C**	40us
Display Function (default 2x40 size)	**$3C**	40us
Character Generator Ram Address set	**$40-$7F**	40us
Display Ram Address set	**$80- $FF**	40us

TABLE 2. LCD Character Codes

$20	Space	$2D	-	$3A	:	$47	G	$54	T	$61	a	$6E	n	$7B	{
$21	!	$2E	.	$3B	;	$48	H	$55	U	$62	b	$6F	o	$7C	|
$22	“	$2F	/	$3C	{	$49	I	$56	V	$63	c	$70	p	$7D	}
$23	#	$30	0	$3D	=	$4A	J	$57	W	$64	d	$71	q	$7E	>
$24	$	$31	1	$3E	}	$4B	K	$58	X	$65	e	$72	r	$7F	<
$25	%	$32	2	$3F	?	$4C	L	$59	Y	$66	f	$73	s		
$26	&	$33	3	$40	Time	$4D	M	$5A	Z	$67	g	$74	t		
$27	‘	$34	4	$41	A	$4E	N	$5B	[	$68	h	$75	u		
$28	(	$35	5	$42	B	$4F	O	$5C	Yen	$69	i	$76	v		
$29	)	$36	6	$43	C	$50	P	$5D	]	$6A	j	$77	w		
$2A	*	$37	7	$44	D	$51	Q	$5E	^	$6B	k	$78	x		
$2B	+	$38	8	$45	E	$52	R	$5F	_	$6C	l	$79	y		
$2C	,	$39	9	$46	F	$53	S	$60	`	$6D	m	$7A	z		

TABLE 3. D-Bug12 Monitor Commands

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