

# CME11E9-EVBU

Development Board

---



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

---

# CONTENTS

<b>GETTING STARTED</b> .....	<b>3</b>
Installing the Software.....	3
Board Startup.....	3
Support Software .....	4
Software Development.....	4
<b>TUTORIAL</b> .....	<b>5</b>
Creating source code.....	5
Assembling source code.....	6
Running your application .....	7
Programming External EEPROM.....	8
Programming On-Chip PROM.....	9
Single-Chip Applications.....	10
<b>MEMORY</b> .....	<b>11</b>
ADDRESS DECODING.....	11
MEMORY MAP .....	12
<b>HARDWARE</b> .....	<b>13</b>
PORTS AND CONNECTORS.....	14
SERIAL PORT.....	14
LCD_PORT .....	14
SS: KEYPAD .....	15
MCU_PORT .....	15
A/D REFERENCE .....	15
BUS_PORT .....	16
VPP Connector.....	16
PWR Terminal Block .....	16
JUMPERS .....	16
Mode Select Jumpers.....	16
Memory Selection Jumpers .....	17
Programming Jumpers .....	18
TRACE / PROG Jumper.....	18
SYNC Jumper .....	18
MEM-EN Jumper.....	18
<b>TROUBLESHOOTING</b> .....	<b>19</b>
<b>TABLES</b> .....	<b>22</b>
TABLE 1. LCD Command Codes.....	22
TABLE 2. LCD Character Codes .....	22
TABLE 3. Buffalo Monitor Commands .....	23
TABLE 4. Buffalo Monitor Interrupt Jump Table .....	23

# GETTING STARTED

The Axiom CME11E9-EVBU single board computer is a fully assembled, fully functional development system for the Motorola 68HC11 Micro controllers, complete with wall plug power supply and serial cable. Follow the steps in this section to get started quickly and verify everything is working correctly.

Support software for this development board is provided for both DOS, Windows 3.1 as well as 32-bit Windows 95 and greater operating systems.

## Installing the Software

1. Insert the Axiom 68HC11 support CD in your PC. If the setup program does not start automatically, run the file called "SETUP.EXE" on the CD.
2. Follow the instructions on screen to install the AxIDE software onto your hard drive then run the AxIDE program.
3. Select the PC serial port you're using to connect to the board.
4. Select your development board " CME11E9-EVBU" from the drop-down menu bar just below the main menu.

## Board Startup

Follow these steps to connect and power on the board. This assumes you're using the provided utility program (described in the previous section) or a similar communications terminal program on your PC, and that all the jumpers are in their default (from the manufacturer) positions. If you're using a different terminal program than the one provided, set it's parameters to 9600 baud, N,8,1.

1. 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 COM port on the CME11E9 board.
2. Apply power to the board by plugging in the wall plug power supply that came with the system.
3. If everything is working properly, you should see a Buffalo Monitor prompt similar to that below in the Terminal window. Press the ENTER key and you should see a prompt which is the > character.

```
BUFFALO 3.4 (ext) - Bit User Fast Friendly Aid to Logical
Operation
>_
```

4. Your board is now ready to use!

If you do not see the buffalo message prompt like that above, or if the text is garbage, see the **TROUBLESHOOTING** section at the end of this manual.

## Support Software

There are many useful programs on the included CD that can make developing projects on the CME11E9-EVBU easier. You can also download the latest software free from our web site at: <http://www.axman.com>.

The main programming interface to the CME11E9 board is the AxIDE program for 32-bit Windows. This program communicates with the board via its COM port and includes a Terminal window for interfacing with other programs running on the CME11E9, such as the Buffalo Monitor or the Basic11 interpreter. It is also useful for displaying information from your own programs that send output to the serial port.

In addition to the terminal interface, these programs also provide an easy to use programming and configuration interface to the board.

Also on the CD is a free Assembler, C compiler and example source code to help get you started.

## Software Development

Software development on the CME11E9 is performed using the Buffalo Monitor utility to test and debug your program that is stored in RAM on U5. During this debug phase your program should be located just above the internal register block, for example \$2000 (see the **Memory Map** section for details).

After satisfactory operation your program can be written to the EEPROM in U7 by relocating the program to start at address \$E000 then programming the relocated code into U7 using the utility software. After setting the appropriate jumpers, your program will start automatically when the board is powered on.

It is also possible to write your program to the On-Chip PROM of 68HC711 devices installed in U1 by relocating the program to start at address \$D000 then programming the internal chip memory on the 68HC711 micro-controller using the utility software. You can then test it on the board or remove the 68HC711 device and install it in your product. Note that a 12 Volt VPP supply is necessary to do this.

# TUTORIAL

This section was written to help you get started with the specifics of the CME11E9 software development process. Be sure to read the rest of this manual for further information. Also, you should see the 68HC11 reference manual and other documentation on the CD.

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 CME11E9 board using any language that compiles to Motorola 68HC11 instructions. Included on the CD is a free Assembler and also a freeware C compiler. Inexpensive or free compilers are also available for Basic and Forth languages. See the [www.axman.com](http://www.axman.com) web page for more information.

You should write your source code using a standard ASCII text editor. Many powerful code editors are available or you can use the free EDIT or NOTEPAD programs that come with your computer. 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 CME11E9 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 \$1040. When finished debugging, you must change these ORG statements so that your program is moved to a valid EEPROM area - somewhere after hex E000. Do this by putting an ORG \$E000 in front of your Program CODE. Data may remain where it is or be moved down to internal RAM starting at ORG \$0000. You must also program the STACK register somewhere at the top of your available RAM, for example hex 1FF. Do this with this instruction as the first instruction in your program code: LDS #\$01FF.

A look at the example programs on the CD 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 buffalo monitor environment will be slightly different than code written for stand-alone operation. The buffalo monitor contains interrupt and RESET vectors for example, your code must provide these when it's no longer running under the monitor. See the **Programming External 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:

```
AS11 HELLO.ASM -l cre s >HELLO.LST
```

Most compilers and assemblers allow many command line options so using a MAKE utility or batch file is recommended if you use this method.

The AxIDE utility software provided with this board contains a simple interface to the free assembler. Use it by selecting "Build" 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:

<b>HELLO.S19</b>	a Motorola S-Record file that can be programmed into memory
<b>HELLO.LST</b>	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.

# Running your application

After creating an S-Record file you can "upload" it to the development board for a test run. Since this version of HELLO.ASM was created to run from RAM, you can use the Buffalo Monitor to test your code without programming it.

If you haven't done so already, verify that the CME11E9-EVBU board is connected and operating properly by following the steps under "GETTING STARTED" until you see the buffalo prompt, then follow these steps to run your program:

1. Press and release the RESET button on the CME11E9 board. You should see the Buffalo Monitor message. Hit the return key ↵ to get the monitor prompt.
2. Type `LOAD T` ↵  
This will prepare buffalo to receive a program.
3. Press the Upload button to send a text file to the board. When prompted for a file name, select the file that was just 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 2400` ↵  
This tells buffalo to execute the subroutine at address \$2400, which is the start of our test program.
5. If everything is working properly you should see the message "Hello World" on your terminal screen then, since we return at the end of our program, a line containing the internal register status displayed by buffalo and the buffalo 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. Buffalo has many powerful features such as breakpoints, assembly/disassembly, memory dump and modify and program trace. Type HELP at the buffalo prompt for a listing of commands or consult the buffalo documentation file on the CD for more information.

# Programming External EEPROM

When finished with program development you can program your application into EEPROM so it executes automatically when you apply power to the board. The following procedure does this:

1. Make a backup copy of the HELLO.ASM source code.
2. Use a text editor to modify HELLO.ASM. Change the start of the program to `$E000` which is the beginning of the EEPROM in U7. This can be done easily in the file since there is a line: `ORG $E000` already there you can add by removing the comment `*` character from the beginning of it.
3. Remove the comment `*` character from before the following lines also:

```
*      LDS  #$23FF          set the stack pointer
*eloop      nop            endless loop
*      bra  eloop
*
*      org  $FFFE          set the reset vector
*      fdb  START
```

4. This will initialize the stack pointer which is necessary when running outside of buffalo but should not be done while running under buffalo since it must handle the stack; cause the program to NOT try to "return" at the end and lastly, program the HC11 reset vector to go to the beginning of the program when powered on.
5. Re-Assemble HELLO.ASM as described in the "Assembling Source Code" section.
6. Start AxIDE and select Configure and follow the onscreen instructions. Make sure ROMON is disabled and if not configure it to be so.
7. Select Program and make sure External EEPROM is selected.
8. At the file prompt enter your newly assembled HELLO.S19 file.
9. Click OK then follow the onscreen instructions, installing the programming jumpers then power off/on or press the RESET switch.
10. When finished programming, remove the jumpers as instructed by the program.
11. Return to the AxIDE terminal window and cycle power or press RESET on the board. Your new program should start automatically.

To return to the buffalo monitor program, select Configure again and this time enable ROMON. The buffalo monitor on chip will execute the next time the board is powered on.

# Programming On-Chip PROM

If a 68HC711 micro-controller is installed in U1, you can program it using this development board and the utility software provided. You can then remove the micro-controller and install it in your product if desired.

Unlike the external EEPROM in U7 however, this type of memory cannot be erased or overwritten by this development board. It may be possible to erase the device using an external UV eraser so that it can be re-programmed.

If your application programmed into external EEPROM and runs ok when power is applied, then it should run ok out of U1. Follow these steps to program it:

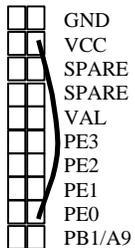
1. Change the start address of your program to `$D000` which is the beginning of the On-Chip PROM in a 68HC711 device.
2. Change the Reset Vector in your code to `$D000` then re-assemble it.
3. Apply a 12V DC programming voltage to the VPP connector.
4. Install a jumper in the PROG position (1-2).
5. Start the programming interface (AX11E for DOS or AXIDE for windows).
6. Select "Configure Processor" or [Configure] and follow the onscreen instructions. Make sure ROMON is enabled and if not configure it to be so.
7. Select "Program Internal Memory". If using the windows utility select [Program] and make sure On-Chip EPROM is selected.
8. At the file prompt enter your newly assembled S19 file.
9. Click OK then follow the onscreen instructions.
10. When finished programming, remove the jumpers as instructed by the program.
11. **IMPORTANT:** Be sure to remove the PROG jumper and the VPP programming voltage at this time.

Your new program should be programmed into the micro-controller and will start automatically when power or RESET is applied.

# Single-Chip Applications

The 68HC11 is in Single-Chip mode with the EVBU boards' MODA jumper ON and MODB jumper OFF. Single-Chip mode will make available all I/O ports including ports B and C for your use.

You can program a small (512 byte) program into the 68HC11 internal EEPROM to be executed in Single-Chip mode. Internal EEPROM is located from B600 - B7FF hex, which is where your program must be located. See the Memory Map for more information.



To configure the CME11E9-EVBU board to start program execution at address B600 whenever power or RESET is applied, you must connect a wire between VCC and PE0, which can be accessed from the EVBU I/O Port (P4). This will cause the Buffalo Monitor on-chip to jump to your program after reset.

In Single-Chip mode you do not have external RAM available, so use internal RAM to test your code first under the Buffalo Monitor. Locate your code starting at address 100 hex for example to debug it from internal RAM.

When you're ready to program your application into the Internal EEPROM, change the starting location of your code to address B600 hex. This should be the address of the first instruction to be executed, not data or a subroutine. Also, make sure you add the stack pointer initialization, as described in the previous "Programming External EEPROM" section. You can use internal RAM for this also, 1FF for example (`LDS       #$1FF`).

Re-Assemble or compile your code and Start the programming interface (AX11E for DOS or AXIDE for windows). Choose "Program HC11 EEPROM" and follow the onscreen instructions.

When finished programming, your application should start if the VCC to PE0 wire is installed. You can re-program the internal EEPROM as often as you like.

To return to the on-chip Buffalo Monitor remove the VCC to PE0 wire and reset or re-apply power to the board.

**CAUTION:** When operating in Single-Chip mode, the memory control signals assume different functions. If the PORTCL handshaking features (STRA and STRB) are used, memory devices located in U5, U6 and U7 should be removed to prevent I/O line contention.

# MEMORY

## ADDRESS DECODING

Address decoding is accomplished using a GAL16V8 programmable logic device. Address lines A<8:15>, AS (address strobe), R/W (read/write), and E (clock) are processed to provide the memory control signals as shown below by default. Custom configurations, differing from that shown below, are also possible. Contact the factory for assistance in redefining the memory map if required.

- OE** Output enable to U5, U6, and U7
- WR** Write enable to U5 direct, and to U6 and U7 through jumpers JP6 and WRITE\_EN respectively.
- M1** Chip select to U5 active from 0 to 8K (with mirrored mapping) or 0 to 32k depending on the status of JP3. See U5 JP3 selection for more information.
- M2** Chip select to U6 active for the 24k between 8000 and CFFF, with the exception of B580 through B7FF inclusive. B580 to B5FF is used to generate the peripheral chip selects.
- M3** Chip select to U7 active for the 8k between E000 and FFFF. Note that U7 is mapped to the same location as the internal ROM of the HC11.
- P** Peripheral Access CS0 - CS7. B580 through B5FF.

All of these signals except P are active low. P is active high. Signal line M2 is also connected to the BUS\_PORT expansion connector allowing M2 to work in conjunction with the CS and Address lines to implement off board, page banked memory. When M2 is used in this manner, U6 must be removed from the board.

U5 is intended to be either an 8k or a 32k RAM. U6 can accommodate RAM, EEPROM, or ROM. U7 is to be used primarily for ROM but it can also accommodate EEPROM. The ROMON bit in the configuration word must be OFF for U7 to be in the memory map. Jumpers JP3 through JP9 and WRITE\_EN determine how U5, U6, and U7 are used. See the **Memory Selection Jumpers** section for details.

Peripheral Access 'P' is used in conjunction with A<4:6>, and AS to generate CS<0:7>. Each of these eight chip selects controls sixteen bytes in the memory map from B580 through B5FF. CS<7:0> are brought out to the BUS\_PORT where they can be used to control peripherals external to the development board. See the Memory Map on the next page for further clarification.

# MEMORY MAP

The following memory map is for a 68HC11E9 as shipped in this development board. Other 68HC11 devices in the A and E series may also be used with this board. These optional devices differ in the amount of internal RAM, ROM and EEPROM available and the factory default value of the CONFIG register. Consult the technical reference for the specific device you are using for additional information.

FFFF	RESET Vector Address
FFFE	
	-----
	Memory Socket U7 (8K device) if ROMON disabled
	or
E000	-----
	68HC711E9 Internal PROM (12K) in U1 if ROMON enabled
D000	
CFFF	Program or Data Memory EEPROM or RAM in U6 (not installed)
B800	
B7FF	HC11 Internal EEPROM in U1 Program or Data
B600	
B5FF	Peripheral Area CS0 - CS7
	CS7 = B5F2-B5FF      CS5 = B5D0-B5DF      CS2 = B5A0-B5AF
	LCD = B5F0-B5F1      CS4 = B5C0-B5CF      CS1 = B590-B59F
	CS6 = B5E0-B5EF      CS3 = B5B0-B5BF      CS0 = B580-B58F
B580	
B57F	Program or Data Memory EEPROM or RAM in U6 (not installed)
8000	
7FFF	Data Memory RAM in U5
1040	
103F	68HC11 Internal Registers See 68HC11 Technical Data Manual
1000	
0FFF	Data Memory RAM in U5
0200	
01FF	
0000	68HC11 Internal RAM in U1 - (42-FF reserved by Buffalo Monitor)

# HARDWARE

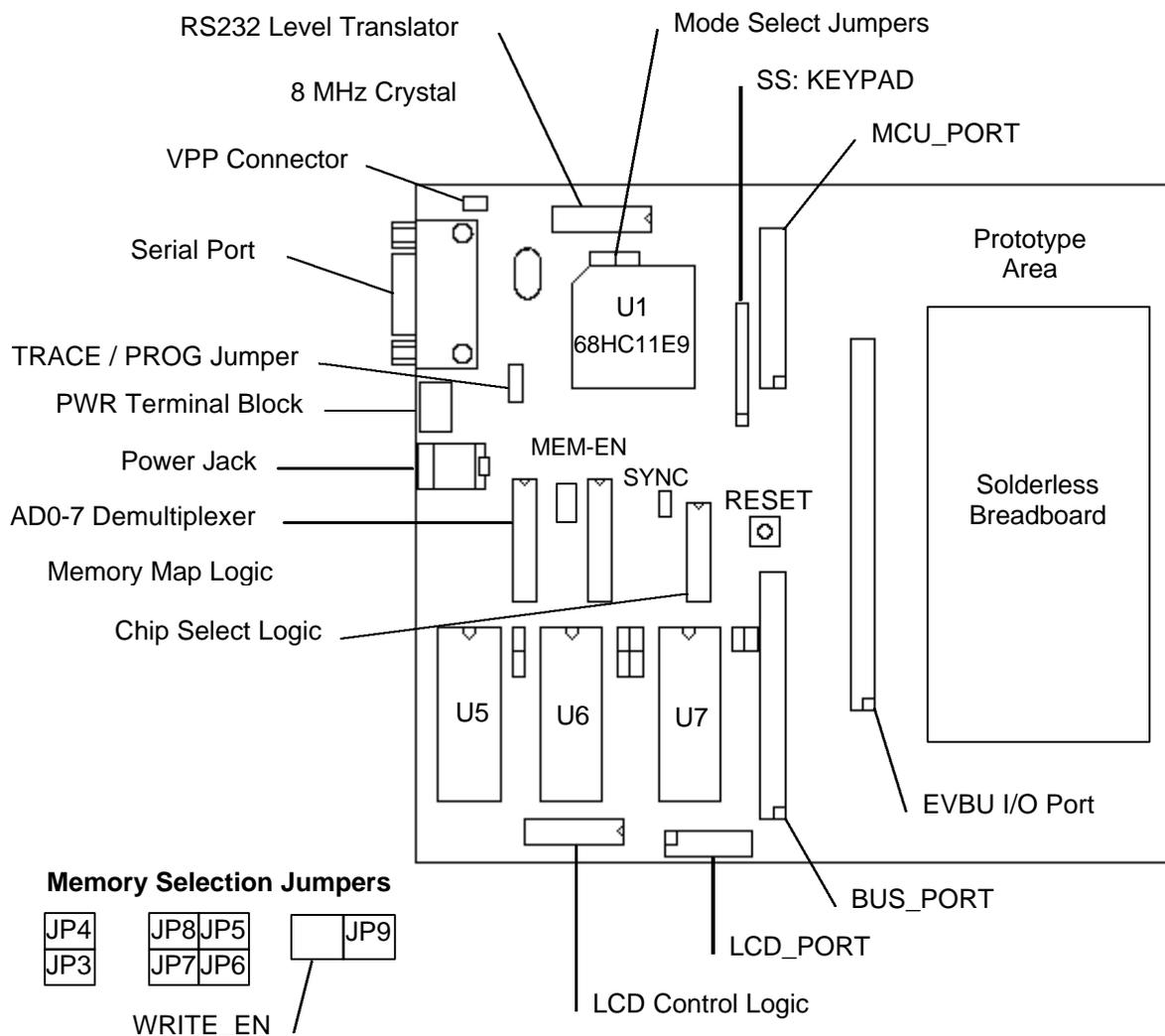
The CME11E9-EVBU board is shipped from the manufacturer with the following default jumper settings:

SYNC = ON	JP3 = ON	JP6 = ON	JP9 = OFF
TRACE/PROG = OFF	JP4 = ON	JP7 = ON	MODA = OFF
WRITE_EN = ON	JP5 = OFF	JP8 = OFF	MODB = OFF
MEM-EN = ON (where available)			

Memory socket U5 is shipped with a 32K byte RAM device. U7 is shipped with an 8K byte EEPROM flash memory device. U6 is normally shipped with no device installed.

Also, the 68HC11 chip configuration word is set as follows:

EEON On Chip EEPROM	ENABLED
ROMON On Chip ROM	ENABLED
NOCOP Watchdog System	DISABLED
NOSEC Security	DISABLED



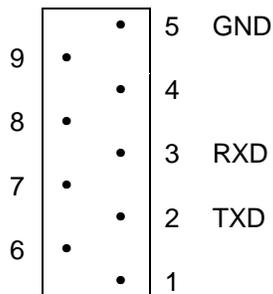
# PORTS AND CONNECTORS

## SERIAL PORT

The onboard serial port COM1 is a simple three wire asynchronous serial interface with hard wired Clear to Send (CTS) and Data Terminal Ready (DTR). It is driven by the HC11 internal SCI port using I/O pins PD0 and PD1. These two logic level signals are coupled through an RS232 level shifter to the COM1 connector.

COM1 is the default serial interface for the Buffalo Monitor and programming software.

### COM1 DB9S Style Connector



- Cut-Away jumpers between following pins:  
4 1 and 6 (DTR/DSR/DCD)  
7 8 (RTS/CTS)
- COM1 is set to connect directly to a PC serial port with a straight thru type of cable (supplied).

## LCD\_PORT

The LCD Display interface is connected to the data bus and memory mapped to locations \$B5F0 through \$B5F3. Addresses \$B5F0 and \$B5F1 are the Command and Data registers respectively. The LCD interface supports all OPTREX™ DMC series displays up to 80 characters and provides the most common pinout. Power, ground and Vee are also available at the LCD\_PORT connector.

A list of valid command codes is provided in the Tables section at the back of this manual. Example programs using this LCD device are provided on the CD, see the files:

<b>B5F0</b>	Command Register
<b>B5F1</b>	Data Register

+5V	<b>2</b>	<b>1</b>	GND
A0	<b>4</b>	<b>3</b>	Vee
/LCDCS	<b>6</b>	<b>5</b>	/RW
D1	<b>8</b>	<b>7</b>	D0
D3	<b>10</b>	<b>9</b>	D2
D5	<b>12</b>	<b>11</b>	D4
D7	<b>14</b>	<b>13</b>	D6

KLCD-E.C  
KEYLCD-E.C  
KEYLCD-E.ASM

## SS: KEYPAD

The SS: KEYPAD connector is an ten pin connector that implements 4 bits of Port D and 4 bits of Port E as a simple serial or keypad interface. This interface provides connection to the SPI port on Port D as a simple serial interface or may be implemented as a software key scan for a passive keypad.

This board is manufactured with no connectors installed in pins 1 and 2. This is to prevent confusion when connecting an 8-pin keypad. Pins 1 and 2 are available however for SPI port applications. When connecting the keypad, align the RED stripe closest to pin 1. Example programs using this keypad are provided on the CD. See the files:

KLCD-E.C  
KEYLCD-E.C  
KEYLCD-E.ASM

1	2	3	4	5	6	7	8	9	10
									
+5	Gnd	D2	D3	D4	D5	E0	E1	E2	E3

## MCU\_PORT

The 68HC11E9 is configured for expanded / multiplexed mode. It uses Port B and Port C for address and data buss to external memory and memory mapped I/O devices. This leaves CPU Port D, Port A, and Port E to provide all other parallel I/O from the controller. MCU port lines are mixed as input only, output only, and some are input or output. All MCU port lines serve dual functions with internal CPU peripherals such as the timer subsystem and port A, the A/D converter on port E, and the SPI or SCI on port D.

The MCU\_PORT connector is a dual row 13 pin Berg-style connector ( 26 pins total ) which is configured as shown.

PD0 and PD1 are used by the SCI to implement COM1. PD<2:5> are used by the HC11 SPI to implement the SS: KEYPAD port. These port D lines can also be used for parallel I/O, but then they will not be available for COM1 or the SS: KEYPAD port. Use caution when assigning port D lines to functions other than COM1 and the SS: KEYPAD port.

PD0/RXD0	1 2	PD1/TXD0
PD2/SI	3 4	PD3/SO
PD4/SCLK	5 6	PD5/SEL0
PA0	7 8	PA1
PA2	9 10	PA3
PA4	11 12	PA5
PA6	13 14	PA7
PE7	15 16	PE3
PE6	17 18	PE2
PE5	19 20	PE1
PE4	21 22	PE0
VRL	23 24	VRH
Gnd.	25 26	+5V

## A/D REFERENCE

The VRH and VRL lines from the HC11 are connected to +5v through R3 and to ground through R2 respectively. These two resistors are located next to the MCU\_PORT header. The resistors are identified on the silk screen by their reference designators. The appropriate resistor(s) need to be removed in order to apply an external reference to the VRH and/or VRL inputs.

## BUS\_PORT

The BUS\_PORT supports off-board devices. Power (+5V), Ground, Address lines, Data lines, and Control lines are brought out to this 40 pin connector. Pin assignments are shown here.

### VPP Connector

The VPP + and - pins are used to apply a 12V DC programming voltage for the programming of the 68HC711 On Chip EPROM.

### PWR Terminal Block

These 3 connection points allow for alternate power input (such as a battery or power supply) as well as output voltage supply.

<b>+V</b>	+7 to 16V DC unregulated Input or Output
<b>GND</b>	Ground
<b>+5V</b>	+5V Input or Output

Gnd	<b>1</b>	<b>2</b>	D3
D2	<b>3</b>	<b>4</b>	D4
D1	<b>5</b>	<b>6</b>	D5
D0	<b>7</b>	<b>8</b>	D6
A0	<b>9</b>	<b>10</b>	D7
A1	<b>11</b>	<b>12</b>	A2
A10	<b>13</b>	<b>14</b>	A3
/OE	<b>15</b>	<b>16</b>	A4
A11	<b>17</b>	<b>18</b>	A5
A9	<b>19</b>	<b>20</b>	A6
A8	<b>21</b>	<b>22</b>	A7
A12	<b>23</b>	<b>24</b>	A13
/WR	<b>25</b>	<b>26</b>	/CS0
/CS1	<b>27</b>	<b>28</b>	/CS2
/CS3	<b>29</b>	<b>30</b>	/CS4
/CS5	<b>31</b>	<b>32</b>	/IRQ
+5V	<b>33</b>	<b>34</b>	M2
R/W	<b>35</b>	<b>36</b>	CS6
ECLK	<b>37</b>	<b>38</b>	CS7
Gnd	<b>39</b>	<b>40</b>	/RESET

**BUS\_PORT**

## JUMPERS

### Mode Select Jumpers

The MODA and MODB pins of the HC11 are pulled high by two 10k resistors. This is the normal EXPANDED MODE configuration. A jumper on MODA will take MODA to ground. A jumper on MODB will take MODB to ground. These two jumpers allow selection of any of the following modes of operation:

MODA	MODB	Mode of Operation
ON	ON	Special Bootstrap / Programming
OFF	ON	Special Test
ON	OFF	Normal Single Chip
OFF	OFF	Normal Expanded (default)

## Memory Selection Jumpers

The factory setting for the jumpers should be correct for the memory devices that came with your board. If you add or modify the type or size of memory, you must change the following jumpers accordingly. All memory selection jumpers are two-pin jumpers and are installed vertically.

### JP3 - U5 Device Configuration

Must be **ON** if a 32K device is installed in U5. If an 8K device is installed in U5 then:

**OFF** configures 8K from 0000 - 1FFF and mirrored at 2000 - 3FFF, 4000 - 5FFF, and 6000 - 7FFF. This position will allow CPU internal Ram and I/O ports to segment the 8K address space.

**ON** configures 8K from 2000 - 3FFF hex and mirrored at 6000 - 7FFF. This is the recommended position for Buffalo Monitor and Small C operation. No segmentation occurs.

### JP4 - JP6 - U6 Device Configuration

JP6 write protects the device in U6 when OFF.

	<b>JP4</b>	<b>JP5</b>	<b>JP7</b>	<b>JP6</b>
8K RAM or EEPROM	OFF	OFF	OFF	ON
8K EPROM	OFF	OFF	OFF	OFF
32K EPROM	OFF	ON	ON	OFF
32K RAM or EEPROM	ON	OFF	ON	ON

### JP8, JP9, WRITE\_EN - U7 Device Configuration

WRITE\_EN write protects the device in U7 when OFF.

	<b>JP8</b>	<b>JP9</b>	<b>WRITE_EN</b>
8K EEPROM	OFF	OFF	ON
8K EPROM	OFF	OFF	OFF
32K EPROM	OFF	ON	OFF
32K EEPROM	ON	OFF	ON

## *Programming Jumpers*

Programming the CME11E9-EVBU external EEPROM memory is performed by installing MODA and MODB jumpers and the WRITE\_EN jumper. The utility software provided can be used to program the EEPROM. See the **Tutorial** section in this manual for more information.

After programming, remove the WRITE\_EN jumper before applying RESET or removing power from the board to guarantee retention of the new program.

### *TRACE / PROG Jumper*

The Buffalo Monitor Trace and Single Step functions can be enabled by installing the TRACE jumper (position 2-3 of the 3 position jumper). Use caution when installing this jumper that no other connections are made to the XIRQ or Port A3 I/O pins of the 68HC11E9.

The PROG position (1-2) will connect VPP+ to the XIRQ port line for programming the internal EPROM of a 68HC711 device installed in U1. **CAUTION** should be used so that the PROG jumper is only installed (and VPP applied) during this programming operation.

### *SYNC Jumper*

This jumper changes the timing of the peripheral chip selects C0-7. When open (default) the chip selects will be active for the whole bus cycle to the connected peripheral. This position provides chip select setup and hold time required by some peripheral devices.

If this jumper is installed, the chip selects will be valid whenever data on the data bus is valid. This allows simple latches to operate from a chip select to add input and output lines.

### *MEM-EN Jumper*

This jumper is not available on all boards.

It should be installed for Expanded Mode operation and for programming the external memory devices. The jumper should be removed for true Single-Chip Mode operation to prevent memory data bus conflicts with Port C if the Port C strobes are used.

# TROUBLESHOOTING

The CME11E9-EVBU 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.

Please check off these steps and list any others you have performed before calling so we can better help you.

# Tips and Suggestions

Following are a number of tips, suggestions and answers to common questions that will solve most problems users have with the CME11E9 development system. You can download the latest software from the Support section of our web page at:

[www.axman.com](http://www.axman.com)

## *Programming Utilities*

- If you're trying to start a program in external EEPROM (U7), make sure jumpers MODA and MODB are NOT installed and ROMON is disabled in the Configuration word.
- 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, 200mA adapter or power supply. If you're using a power strip, make sure it is turned on.
- If the configuration file loads (the first 100 bytes or so), but you get a time-out error when the program section begins to download, make sure the HC11 is internally configured correctly by selecting Configure Processor from the main menu.
- ROMON must be set to off (disabled) to enable external EEPROM (U7) and must be on (enabled) to enable the 68HC11 Internal EPROM in U1. During program development the EEON should be enabled. The other bits should be disabled.
- Make sure you load your code to an address space that actually exists. See the Memory Map.
- If you are running in a multi-tasking environment (such as Windows) close all programs in the background to be certain no serial conflict occurs.
- If programming is slow, run the batch file PAGE.BAT in the ax11e directory then see if programming is faster. If programming doesn't work following this, return to normal operation by running the batch file BYTE.BAT.
- If the Assembler or Small C compiler menu options do not work properly on your system, you can modify their operation by editing the files DO\_SC.BAT (for the C compiler) and DO\_ASM.BAT (for the assembler). These are the batch files that are run when their menu items are selected. Try putting a PAUSE statement at the end of the batch files. This will halt the batch file before returning to AX11E so you can see any error messages they may be giving you.
- You can reset all AX11E configuration options to their original state by deleting the file named AX11.CFG. This file will be re-created the next time you run AX11E.

## *Code Execution*

- Make sure ALL jumpers are set correctly according to your board's configuration. Read the hardware manual section on jumpers carefully if you're not sure.
- Always remember to remove the Programming Jumpers after programming the code memory.
- If you programmed your code into external EEPROM memory and it doesn't run, check the HC11 reset vector, located at \$FFFE - \$FFFF. These 2 bytes contain the reset vector address where execution will begin when the unit is powered on or reset. The default vector is E000, which is the beginning of the 8k program address space.
- Verify that all peripheral devices are initialized properly in your code. Failure to initialize the serial port, for example, is a common problem. This is usually the case when the code works under buffalo but not after programming.

## *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 buffalo.

## *SMALLC Compiler*

- If you're programming to ROM, delete the first line of the S19 record generated, since SMALL.C adds data there.
- Make sure you put the CODE, DATA, and STACK at the correct locations per your memory configuration. DATA may need to be at 0000 if you're burning a ROM, and STACK should have plenty of room.
- Don't forget to use the -R option if you're compiling for ROM.
- If Internal Registers have been re-mapped (default at 0x1000) be sure to change the #define REG\_BASE accordingly.
- If you're having trouble with the shift operations (>> or <<) this is probably because the small c compiler uses a rotate thru carry instruction here. Try using inline assembly code to accomplish the shift.
- You may have trouble returning any values larger than 1 byte from functions (i.e. char functions). You will have to use global integer values instead.
- Small C does not support structures or unions.
- Small C is a free "un-supported" public domain compiler. If your application is complex, consider purchasing a commercial quality 68HC11 C compiler.

# 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. Buffalo Monitor Commands

ASM [<address>]	Assembler/Disassembler
BF <addr1> <addr2> <data>	Block fill memory with data
BR [-] [<address>]...	Breakpoint set
BULK	Bulk erase EEPROM
BULKALL	Bulk erase EEPROM + CONFIG register
CALL [<address>]	Execute subroutine
G [<address>]	Execute program
HELP	Display monitor commands
LOAD T	Download S-records via terminal port
MD [<addr1> [<addr2>]]	Dump memory to terminal
MM [<address>]	Memory modify
MOVE <addr1> <addr2> [,dest>]	Move memory to new location
P	Proceed/continue from breakpoint
RM[p,y,x,a,b,c,s,]	Register modify
T [<n>]	Trace \$1-\$FF instructions (TRACE Jumper = ON)
VERIFY T	Compare memory to download data via terminal

- Address and data values should be given in Hex notation, for example: asm 2bf0
- See the buffalo monitor users guide on the CD for more information.

## TABLE 4. Buffalo Monitor Interrupt Jump Table

\$00C4-\$00C6	Serial communications Interface (SCI)
\$00C7-\$00C9	Serial Peripheral Interface (SPI)
\$00CA-\$00CC	Pulse Accumulator Input Edge
\$00CD-\$00CF	Pulse Accumulator Overflow
\$00D0-\$00D2	Timer Overflow
\$00D3-\$00D5	Timer Output Compare 5
\$00D6-\$00D8	Timer Output Compare 4
\$00D9-\$00DB	Timer Output Compare 3
\$00DC-\$00DE	Timer Output Compare 2
\$00DF-\$00E1	Timer Output Compare 1
\$00E2-\$00E4	Timer Input Capture 3
\$00E5-\$00E7	Timer Input Capture 2
\$00E8-\$00EA	Timer Input Capture 1
\$00EB-\$00ED	Real Time Interrupt
\$00EE-\$00FO	IRQ
\$00F1-\$00F3	XIRQ
\$00F4-\$00F6	Software Interrupt (SWI)
\$00F7-\$00F9	Illegal Opcode
\$00FA-\$00FC	Computer Operating Properly (COP)
\$00FD-\$00FF	Clock Monitor

See the Buffalo manual for help using these vectors in your program.