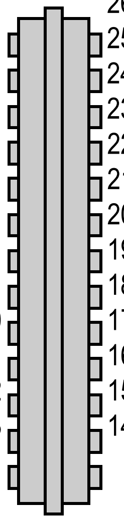
APPENDIX A
Pinouts for SB Connector Option
MicroCore-11

COMPONENT SIDE

PA7	1	26	PD2 (MISO)
PA6	2	25	PD3 (MOSI)
PA5	3	24	PD4 (SCK)
PA4	4	23	PD5 (SS*)
PA3	5	22	+5VDC
PA2	6	21	IRQ*
PA1	7	20	XIRQ*
PA0	8	19	VRL
VRH	9	18	GROUND
PE0	10	17	PE4
PE1	11	16	PE5
PE2	12	15	PE6
PE3	13	14	PE7



SOLDER SIDE

TOP VIEW

Note: * indicates active low

1 INTRODUCTION

1.1 Congratulations...

You are now the proud owner of **MicroCore-11**! It s one of the smallest 68HC11 microcontroller modules around, and we re sure you ll find it useful in your application! Your questions and comments are always welcome. We provide friendly, knowledgeable technical support by telephone, fax, and e-mail to all our customers. As well, we have a comprehensive website with a Tech Support page, an Applications page, and a Resources page featuring new information, software, and links to other useful sites on the Internet. Visit the Tech Support page for instructions on joining our techart-micros email forum for receiving news, updates, and networking with other customers using ourproducts. See back cover for how to contact Technological Arts.

1.2 What is MicroCore-11™?

MicroCore-11 is an evaluation and application tool for Motorola s popular MC68HC11 microcontroller. It is unique among evaluation boards in that it is designed to plug vertically into any standard solderless breadboard. It is a fully functional, standalone implementation of an expanded-mode 68HC11 configuration, and can be plugged, just like a chip, into your breadboard. Then you simply wire up the desired application circuits and download your appropriate code into the micro, to rapidly evaluate and develop your application ideas.

1.3 Product Description

MicroCore-11 comes with on-board EEPROM program memory (8K or 32K, depending on version) and a built-in RS232 interface. These two features mean that **MicroCore-11** s memory is directly loadable via your PC serial port for quick and easy programming. The 68HC11 s handy Bootstrap mode makes separate programming hardware unnecessary. MicroCore-11 modules will work with any A-series and E-series 68HC11 chips that are packaged in a 52-pin PLCC.

1.4 Description of Product Configurations

MicroCore-11 Starter Packages include a 68HC11E0 MCU, with 512 bytes RAM on-chip. Operating in expanded-chip mode, two of the 68HC11 input/output ports form the address and data buses, and are thus unavailable for user applications. External memory of 32K RAM (lower half of memory map), and 32K EEPROM (upper half) is provided. In configurations which have only 8K EEPROM, the 8K block appears redundantly, at each 8K boundary in the upper half of the memory map.

MicroCore-11 satisfies the needs of users who want lots of memory, but only require a modest number of I/O lines. It is also suitable for datalogging applications, where a large chunk of RAM is required for storing samples or events. This configuration is also well-suited to applications requiring large calibration tables, or a lot of text messages (an alphanumeric LCD module, for example).

A Turbo version of MicroCore-11 is also offered, which uses a 9.8304MHz crystal and a faster MCU (68HC11E0CFN3). Besides giving a 21% speed increase, this crystal frequency was chosen because it supports standard baudrates up to 38400 (vs. 9600 running at 8MHz).

1.5 Communications

An RS-232-compatible 3-wire serial interface port (RX, TX, and Ground) is built into MicroCore-11, allowing communication with a PC, or any other device which has an RS-232 serial port. A jumper is provided on the board (W4) to disconnect the RS232 interface chip s power supply, for applications not requiring RS232, and where power consumption is to be kept to an absolute minimum.

1.6 MicroCore-11 Versus Other Evaluation Boards

There are many evaluation and development systems available. Most of them try to be universal in their application, containing every imaginable kind of support circuitry on-board, or providing a large area for prototyping by soldering or wire-wrapping parts on the board. The designer

NOTES:

- G. J. Lipovski - 1988, Prentice-Hall / ISBN 0-13-810557-X 025
- 68HC11-specific textbook with examples & problems

The M68HC11 Microcontroller: Applications in control, instrumentation and Communication
- Michael Kheir - Prentice Hall

tried to second-guess what you might want to do, and as a result you pay more and have less flexibility. We chose instead, to take a modular approach. With **MicroCore-11**, all available I/O lines and control signals are brought out to a standard 26-pin interface connector. With several different connector options available, you can use the module in whatever way best suits your needs. With the solderless breadboard header, you can treat the module like a big chip, and plug it right into your breadboard. Forget soldering or wire-wrapping get started developing your application right away. Your prototyping space is virtually unlimited, using solderless breadboards! When you ve got a design working and you re ready to move to something more permanent, a full range of accessories give you the ability to easily build fully customized, compact applications at low cost, with a full range of accesories including backplanes, prototyping cards, motor-drivers, and other application-specific cards.

Using MicroCore-11 with Solderless Breadboards

The standard connector option installed on the Starter Package board is the SB style, designed to plug into a solderless breadboard. If your board has a different connector option, you can still use it with a solderless breadboard, by way of a 26-pin solderless breadboard adapter. See the **MicroCore-11** Accessories page on our website, or contact us, for further information on these adapters. For contact information, refer to the back cover of this manual.

CAUTION!
Never insert your module into or remove it from a live breadboard. Make sure the power is OFF !

1) Any breadboard will do; however, you will find that the kind made with a softer, more pliable plastic (such as nylon) will be easier to use and more durable. Avoid excessive removal and insertion of your board, to extend the life of your breadboard.

2) When plugging **MicroCore-11** into your breadboard, keep it

vertical and press gently but firmly, rocking the module back and forth slightly, until the pins are seated in the sockets. Use the same side-to-side gentle rocking motion, while pulling gently upward, to remove the board.

3) Plug **MicroCore-11** into the middle area of your breadboard strip to allow maximum access on each end to all the signals. If possible, place an additional breadboard section in parallel on each side of **MicroCore-11** for easier wiring of your circuits. *(HELPFUL HINT: If you are using the Analog inputs, make sure to wire your analog circuits as close to these pins as possible, to keep noise levels down.)*

4) Choose a convention for wiring your power distribution buses. A logical approach is to make the inside bus logic 5V, and the outside buses GROUND. Never supply external power via J1 if you are supplying 5VDC via the breadboard connector pins. However, always connect the breadboard GROUND to **MicroCore-11** GROUND.

5) If you are using voltages other than 5V, make sure to keep these well away from **MicroCore-11** pins and tie-strips, to avoid accidental shorts which may damage the module.

3 TUTORIAL

Note that this manual is not meant to provide an exhaustive study of the 68HC11 family, but rather to help you get started using **MicroCore-11** microcontroller boards as a learning and application development tool for 68HC11, whether you re a beginner or an expert. If you are a beginner, you will benefit from additional material listed in the Reference section of this manual, and links provided on the Resource page of our website (see back cover for URL). Be sure to check the Beginner section on our Tech Support page.

CAUTION!
Never insert MicroCore-11 into or remove MicroCore-11 from a live breadboard. Make sure the power is OFF !

Manual (M68FCASS/AD1)

Other Books

The 68HC11 Microcontroller
- Joseph D. Greenfield (at R.I.T.) - ISBN 0-03-051588-2
- 1992, Saunders College Publishing, (Harcourt Brace Jovanovich)
Data Acquisition & Process Control w/ the M68HC11 Microcontroller
- Frederick Driscoll, Robert Coughlin, Robert Villanucci of Wentworth Institute of Technology.
- 1994, Macmillan Publishing Company / ISBN 0-02-33055-X
- example applications of interfaces to various sensors.
Design with Microcontrollers
- John B. Peatman (professor at Georgia Tech) / ISBN 0-07-049238-7
- This book is on a more advanced level. Uses both the 68hc11 and Intel 8096 as example systems.
Embedded Systems Programming in C and Assembler
- John Forrest Brown / ISBN 0-442-01817-7
- Van Nostrand Reinhold, 1994 - 304 pages, $49.95
- covers Motorola and Intel processors
Microcomputer Engineering
- Gene H. Miller / ISBN 0-13-584475-4
- 1993, Prentice Hall, Englewood Cliffs, NJ 07632
- Explains basics. Many clear, concise examples.
Microcontroller Technology, The 68HC11
- Peter Spasov / ISBN 0-13-583568-2 - Prentice Hall
Microcontrollers: Architecture, Implementation, & Programming
- Kenneth Hintz and Daniel Tabak - ISBN 0-07-028977-8
- 1992, McGraw-Hill Inc.
Mobile Robots: Inspiration to Implementation
- Joseph L. Jones and Anita M. Flynn - Very hands-on book.
- Focuses on every detail involved in design & construction of "Rug Warrior", based on 68HC11A1, using Interactive C compiler.
Programming Microcontrollers in C
- Ted Van Sickle - ISBN 1-878707-14-0
- 1994, HighText Publications - 394 pages, $29.95
- thorough tutorial on C programming, covers aspects of C programming specific to embedded systems
MC68HC11: An Introduction
- Han-Way Huang - West Publishing / ISBN 0-314-06735-3
Single- and Multiple-Chip Microcomputer Interfacing

HW Echo Mode should be set to Normal.

Leave the Config Reg. setting at 0 (ie. don t change) unless you want the bootloader to change the value of the Config. register. (Remember, the new setting only takes effect following a Reset of the HC11).

After exiting Bootstrap Options, click on Bootstrap Download, and select the s-record file you wish to download. Press the RESET button and slide the Write Protect switch (SW3) to WRITE. Then click OK. When finished, slide SW3 back to PROT, SW2 to RUN, and press RESET.

5.0 SOURCES

Computer Bulletin Board Systems

Motorola Freeware BBS
(512) 891-3733 Austin,TX
(619) 279-3907 San Diego, CA
Dunfield Development Systems BBS: (613) 256-6289

Internet Resources

Technological Arts: Check here often for new product info, new utilities, tips, applications information, and resources on the web. http://www.technologicalarts.com

e-mail address: support@technologicalarts.com

Motorola Freeware:	http://freeware.aus.sps.mot.com/freeweb/
University of Alberta:	ftp://ftp.ee.ualberta.ca

Publications

Motorola Fax-on-Demand: (602) 244-6609 or 800-774-1848
Motorola Semiconductor Literature Distribution Center
P.O. Box 20912, Phoenix, AZ 85036 1-800-441-2447
Motorola Microcontroller Development Tools Directory (MCUDEVTLDIR/D)
M68HC11 Reference Manual (M68HC11RM/AD)
M68HC11 E-series Technical Data book (MC68HC11E/D)
MCU Toolbox (MCUTLBX/D)
Motorola Freeware PC-Compatible 8-Bit Cross-Assemblers User s

3.1 Getting Started

Important! Be sure to browse the README.TXT file on your Starter Package disk for information on what s on the disk and how to use it. Use any text editor or browser you like, such as Notepad or DOS edit, and follow the instructions for copying the disk contents onto your hard drive.

MicroCore-11 has a demonstration program already programmed into the EEPROM when you receive it. This is a useful program for testing your communications setup and monitoring & controlling the various I/O lines of the micro.

You can power the module in one of two ways:

1) supply power via the external power connector; just connect a DC voltage greater than 5.6 Volts to the external power connector (J1) on **MicroCore-11**. Red is positive, and black is negative (ground). ***CAUTION! Make sure you have the polarity correct!***

or, 2) supply regulated 5VDC via the appropriate pins on the 26-pin connector (H1). See Appendix A for the pinout diagram of **MicroCore-11**. ***CAUTION! Double-check your connections before applying power!***

To use the demo program, connect the supplied serial cable between **MicroCore-11** and a serial port on your PC. (With some PCs, you will need a 9-pin to 25-pin adapter.) Run a terminal program (such as ProCommPlus, or the Windows Terminal program) on your PC, in your terminal program, set the baud rate to 2400 (38400 if you re using a Turbo MicroCore-11), parity to NONE, # DATA BITS = 8, and #STOP BITS = 1. With SW2 set to RUN, press the **MicroCore-11** RESET button. LED D1 will blink twice, indicating the demo programming is running. Hit <ENTER> on your keyboard. A menu of commands will appear on your terminal window s screen, followed by a command prompt ? symbol. Each command is activated by a single keystroke. A sample screen showing results of each command is shown in Figure 3.1. Typing a command not listed will cause the menu to be re-displayed.

The A and D commands in the demo program allow you to examine the states of PORTA and PORTD. Try putting switches on some of these input port lines. Use a 10K or higher pullup resistor on one side and connect the other side of the switch to ground. Note that PA0-PA2 are inputs, PA3 and PA7 are programmable as input or output, and default to inputs. PA4-PA6 are outputs only. In the demo program, PA6 is used as a tone output for a speaker. It is also connected to LED D1, to provide a visual output. You can drive a small piezo speaker directly by hooking one end to PA6 through a 330-Ohm resistor, and the other end to ground. When you press RESET, or type S when the demo program is running, you will hear two beeps from the speaker (or the LED will flash twice).

Typing a digit between **4** and **6** causes the output state of the corresponding PORTA line to be toggled (eg. typing **5** causes PA5 to flip to a high if it was low previously, or a low if it was high previously). This allows you to activate LEDs (when driving LEDs directly from an output port, limit the current to a maximum of 10mA with 330-Ohm current limiting resistors on each LED); or drive relays, solenoids, or motors (with appropriate driver circuits). Typing **R** causes the values of all 8 analog-to-digital converter (ANALOG) channels of PORTE to be continuously updated on the screen (near Real-time updates) The display will continue to be updated until a key is pressed. The display will continue to be updated until a key is pressed. Analog channels (ANALOG0-ANALOG7) can read voltages between 0 and 5 Volts. Try putting a 10K-Ohm (or higher) pot across the VRL and VRH pins (pins 19 and 9), and connect the wiper to an ANALOG input through a 1K-Ohm current limiting resistor; then change the pot setting, monitoring the ANALOG values on the screen. Unused ANALOG channels should be grounded to VRL through a minimum 1K-Ohm resistor. These inputs are not internally protected from electrostatic discharge (ESD) as the other input port lines are. *(HELPFUL HINT: Grounding multiple adjacent analog inputs is easy by plugging a bussed resistor SIP in your breadboard and jumpering the SIP common pin to VRL.)*

3.2 Writing Your First Program

If you are already experienced with the 68HC11 family of microcontrollers, you can skip this section. However, you may find the

working in C within a DOS environment. If you have purchased the Windows version, however, the following section provides some basic hints and guidelines for setting it up and using it with the MicroCore-11.

The LINKER is the part of a compiler package that marries the software with the particular hardware configuration being used. Therefore, you need to setup the LINKER before using ICC11. Find the Linker setup tab by pulling down the OPTIONS menu to COMPILER. In the Linker, text refers to code, and should be the starting address of EEPROM (or wherever you plan to put the code); data is for variables, and should be the start of RAM; stack should be the end of RAM, since it builds downward . There are some exceptions: if you have a large amount of RAM, you would probably start the data section following the last address of on-chip registers, and start the stack just below the internal register block.

If you have ICC11 V4.5 or higher, you can use the SetupWizard feature of the Options|Compiler|Linker pulldown menu. For MicroCore-11(8K) select Generic 8K Upper ROM, 32K lower RAM . For MicroCore-11(32K) select Generic 32K Upper ROM, 32K lower RAM .

Every C program that you write requires the file vectors.c to be included (either explicitly at the end of the program, or as the last file in the Project file, if you are using the ProjectBuilder feature of ICC11). You should examine and compile the program hello.c , in the Examples directory of ICC11. This program has the essentials set up for you. The only time you would not need vectors.c is if you had a resident monitor that was controlling the loading and execution of a user program (such as the Buffalo monitor from Motorola).

To compile and download hello.c, select Compile to Executable in the pulldown menu. ICC11 will compile, assemble, and link, creating an s-record file called **hello.s19**. This is the file you will download to your board.

Make sure Bootstrap Download Mode is checked in the TERMINAL pulldown menu. Then activate the terminal window, and click on Bootstrap Options.

Select EXTERNAL EEPROM as the Bootloader Programming setting.

4.27 Exploring Further with PCBUG11. For information about PCBUG11 commands and their syntax, enter:

help

at the PCBUG11 prompt. For complete details on PCBUG11, refer to the PCBUG11 Manual. To access the entire manual with your web browser, follow the PCBUG11 link provided on the RESOURCE page of our website (see back cover for our URL).

4.3 Using SBASIC

Here are some example configurations and the approprate /c and /v compiler options used to specify the starting addresses for code (EEPROM, usually) and variables (RAM), where infile is your SBASIC program filename, and outfile is the name you want the target assembly language file to be called.

MicroCore-11 with 8K EEPROM:

```
sbasic infile /ce000 /v1040 /s0fff >outfile.asc
```

MicroCore-11 with 32K EEPROM:

```
sbasic infile /c8000 /v1040 /s0fff >outfile.asc
```

The resulting file outfile.asc is in assembler source code, and can be viewed and edited if you wish. When ready, assemble the file to produce an s-record file suitable for downloading to your board. Use the assembler provided with SBASIC, as follows:

```
asmhc11 outfile
```

and it will produce the file **outfile.s19**.

4.4 Using ImageCraft's ICC11 Windows C Compiler

ImageCraft offers a good low-cost ANSI C compiler, which is quite popular in the 68HC11 community. A Freeware DOS version is included on your Starter Package disk, and is intended for those who are quite familiar

```
         MICROCORE-11 DEMO PROGRAM MENU

         A  => SHOW PORT A STATUS
         C  => CLEAR PORT A OUTPUTS
         D  => SHOW PORT D STATUS
         R  => SHOW REAL-TIME ANALOG VALUES
         S  => BEEP SPEAKER (CONNECT TO PA6)
          ?
        <A>
         PORTA=000
        <D>
         PORTD=001
        <R>
        AD0=000  AD1=000  AD2=000  AD3=000  AD4=100  AD5=100
AD6=099 AD7=100
        <S>
         >>> BEEP! <<<
```

Figure 3.1 MicroCore-11 Demo Software Screen

suggestions here useful to familiarize yourself with the essentials of getting a program to work the first time.

As mentioned in the previous section, a demo program resides in your microcontroller s EEPROM when you receive it. This demo program is written in Motorola s Freeware AS11 cross-assembler syntax, and is intended to provide you with an easy way to verify your hardware setup (ie. power supply, serial connection, PC software, etc.). It also provides you with an excellent starting point for developing your own program. Rather than starting from scratch, you can make a copy of the demo source file and remove and add features, to transform it into what you need.

Many people approach programming by spending hours or even days writing a program from scratch, then assembling it and downloading it. Then they cross their fingers and reset the board, praying everything will work. About 99% of the time, their hopes are dashed, as the board does something completely different than they expected, or worse it appears to do nothing! At that point, they either give up, or purchase expensive diagnostic equipment, such as logic analyzers and in-circuit emulators to begin the long hard road of diagnosing and correcting their software and/or hardware mistakes.

A much more sensible and rewarding approach is to start with something that works, and then add new features incrementally. The modular design of MicroCore-11 gives you that starting point-- hardware that works, and software that works. Now, if you build on that incrementally, each diagnostic step is small and manageable. And it will probably end up taking a lot less time, and costing a lot less money.

A powerful asset for program development is the serial communications interface (SCI). The SCI gives you a window on what s going on inside the microcontroller. Simple diagnostic messages, placed at strategic points in your evolving program, will be invaluable in debugging your software and hardware.

If you look at the demo source code (mc11demo.asm), you will see that it consists of an initialization section, one main loop, and several subroutines and interrupt routines. The main subroutine is called ProcessCommand. Its primary function is to interpret a single-character command that is received via the serial port, and perform the appropriate action for that command. You can easily extend the list of commands within the ProcessCommand subroutine by adding code to recognize and process other single-character commands you define. Then it is simply a matter of writing those subroutines to perform the functions you wish to implement. This software structure gives you a way to independently test new functions you are trying to implement.

For example, let s suppose you are implementing an autonomous vehicle, using a miniature toy car. You will need to read position and collision sensors, and control the speed and directrion of one or more motors. First write a basic motor-speed control routine, and assign some commands to test it. You could assign F for faster (increase the motor speed), and S for slower (to decrease motor speed). Then implement a pulse-width

CONFIG register contents. There is one caveat, however. The new value of the CONFIG register will not take effect until following a RESET of the HC11. Therefore, PCBUG11 will return a BAD MEMORY error when you try to modify the location. This is because the Memory Set and Memory Modify commands automatically attempt to verify the change they just made. In the case of the CONFIG register, reading location $103f returns the contents of a register which reflects the value of the CONFIG register at the time of RESET. It is a read-only register. The actual EEPROM cell implementing the CONFIG register, is a Write-only location, from the user s point of view.

4.26 Using PCBUG11 s Terminal Window. PCBUG11 has a basic communications terminal which is useful for testing and debugging your software. To use it, set up the necessary communications parameters such as baud rate, parity, etc. by entering:

control

A list of parameters is displayed. Use the Page Down and Page Up keys to move to the parameter that needs changing. Use the Up and Down arrow keys to make the changes, or enter new values via the keyboard. When you re done, press the ESC key to return to the PCBUG11 prompt. Now enter:

term

to open the terminal window. If you have the Demo program loaded in your board, you can use the terminal window to interact with it. First, make sure you have set up the baud rate, etc., as required for the demo program. Then, switch your board to RUN, and press RESET. Pressing ENTER will cause the menu of the demo program to be displayed. Select the menu commands, as you desire. When you have finished using the demo program, you can return to PCBUG11 by hitting ESC. To re-establish communication between PCBUG11 and the HC11, reset your board in BOOT mode, and enter:

baud 9600
restart

To run your program, reset the board in RUN mode. To use PCBUG11 again, reset your board in BOOT mode, and type:

restart

Again, switch on access to external memory with:

ms 103c e5

4.23 Manipulating Memory.
To change the value stored in a RAM, external EEPROM, or register location (eg. clear location $2000), simply use the Memory Set command, as follows:

ms 2000 00

You can view locations, and modify them as you require, by using the Memory Modify command, as follows:

mm 2000

In this example, PCBUG11 will display the current value of location $2000. If you wish to change it, type the new value, and press ENTER. If not, just press ENTER. The next memory location will be displayed. To end the Memory Modify mode, press ESC on your keyboard.

4.24 Programming the CONFIG Register. The CONFIG register ($103f), allows the user to enable or disble such things as the EEPROM, ROM (or EPROM), and COP (watchdog timer). The CONFIG register is implemented in EEPROM, so you need to specify this in PCBUG11 before you can modify it.
Enter the following command:

eeprom 103f

If PCBUG11 returns a message saying Erase-before-write disabled, enter the following command to enable automatic erase:

eeprom erase enable

Then make sure the PTCON bit in the Block Protect Register is cleared:

ms 1035 0f

Now you can use Memory Modify or Memory Set to change the

modulation scheme in your motor speed routine, decreasing the pulse-width by some increment every time the S key is pressed, and incrementing pulse-width by the same amount every time the F key is pressed. Of course you will want to put tests for minimum and maximum pulse-widths before you decrement or increment, so that the motor doesn t suddenly jump from stopped to full speed when the
pulse-width value wraps around.

Note: If you plan to incorporate all or part of the demo program in your own code, and you re using a different cross-assembler than AS11, you may have to revise the syntax.

3.21 A Very Simple Program If you are a beginner, the size of the demo program may look overwhelming, and you may wonder what is really required just to do something very simple. In reality, you can throw away almost everything, ending up with about a half dozen lines of code, to produce a very simple program. _Rule #1:_ *Every program requires a Reset Vector (located at $fffe and $ffff), since the contents of these locations is loaded into the Program Counter immediately after reset.* Therefore, the Reset Vector should point *to the beginning of your code.*

Below is an example of a very basic program, which just turns on the LED on PORTA (PA6), and then waits in an infinite loop. (It was written for an 8K EEPROM space, so if your board has 32K, you can change the org to $8000, but it s not necessary)

```
        org     $e000   ;start at top 8K EEPROM

begin:  ldaa    #$40    ;write a logic high to PA6
        staa    $1000   ; (and logic low to all other portA bits)
        bra     *       ;branch forever (until reset occurs)

        org     $fffe   ;define the reset vector to point to
        fdb     begin   ;  the beginning of your code
```

Now let s take it one more step: blink the LED on and off. To make the blinking slow enough to perceive with the human eye, we will write a subroutine to create a half-second delay.

<u>Rule #2:</u> *If you are using interrupts, or incorporating any subroutines in your program, you must initialize the stack pointer at the beginning of your program.*

```
        org     $e000   ;start of top 8K EEPROM

begin:  lds     #$fff   ;initialize the stack pointer
loop:   ldaa    #$40    ;write a logic high to PA6
        staa    $1000   ; (and a logic low to all other portA bits)
        bsr     Delay   ;do a time delay
        clr     $1000   ;make all portA bits logic low
        bsr     Delay
        bra     loop    ;do it all again

Delay:
        ldy     #$ffff
D1:     dey             ;pad delay loop with extra cycles
        iny             ;  for total of 15 cycles
        dey             ;this gives approx. 1/2 second at 8MHz
        bne     D1
        rts

        org     $fffe   ;define the reset vector to point to
        fdb     begin   ;  the beginning of your code
```

3.3 Downloading Your Code

Once you have assembled your code with no errors, you can download the resulting s-record file (***filename.s19***) to your board using the appropriate DOS batch file provided or via the Windows application **MicroLoad**. Connect the supplied serial cable (handshaking signals are

in BOOT mode):

pcbug11 -e port=2

for a Turbo board, enter:

pcbug11 -e baud=9600 port=2

(if you re using COM1, omit the port=2 parameter).

If you prefer to use hexadecimal numbers in your PCBUG11 session (instead of the default decimal notation), enter the following command at the PCBUG11 prompt:

control base hex

Enable access to external memory by changing the HPRIO register:

ms 103c e5

4.22 Loading an S-record File into external EEPROM. First tell PCBUG11 the range of addresses that contain EEPROM, as follows:

eeprom f800 ffff

This is done strictly to provide some delay after each byte is written. A single byte in 32K EEPROM takes up to 10 ms to write, while bytes in an 8K EEPROM device require up to 5 ms per byte.

Then disable Erase-before-Write, since external EEPROM does this automatically:

eeprom erase disable

Slide the Write Protect switch (SW3) to WRITE, and then you re ready to load your s-record file. Suppose you have generated a file called myprog.s19, use the following command to load it into EEPROM:

loads myprog

If it is in a different directory, (eg. c:\myfiles) you should specify the directory path. For example:

loads \myfiles\myprog

To prevent unwanted modification of EEPROM locations, slide SW3 back to PROT when you are finished. To verify that your file myprog.s19 was actually written to EEPROM, use the Verify command:

verf \myfiles\myprog

xload2.bat (for COM2). For example, to download your S-record file called **myprog.s19** to MicroCore-11, via COM2, at the DOS prompt, you would enter:

xload2 myprog

4.2 Using PCBUG11 with MicroCore-11

Motorola s PCBUG11 is a flexible, powerful, and easy-to-use program that runs on a PC. It allows you to examine and modify on-chip memory (RAM, EEPROM, and EPROM), load HC11 code, debug, trace, erase EEPROM, disassemble blocks of code, assemble line-by-line, and even includes a basic terminal program. What s more, it works with all varieties of HC11 micros, and is fully compatible with all configurations of MicroCore-11. PCBUG11 is included on your Starter Package disk, and is also available on the Resource page of our website. It is a self-extracting archive, so just copy it to the directory on your harddrive where you want it to reside, and type pcbug342 at the DOS prompt to extract all the files.

A Windows 95/NT version of PCBUG11 recently become available on the internet, and can be downloaded via a link on the RESOURCES page of our website.

4.21 Running PCBUG11. Always RESET your module in BOOT mode before starting PCBUG11. It is important to make sure there are no background tasks running on you PC, such as faxmodem drivers, networks, etc. PCBUG11 needs to access the serial port chip directly, and does not co-exist very happily with other programs. Whenever possible, start PCBUG11 directly from DOS (not a DOS shell). If you can t get it to work at all, visit Motorola s freeware website, and look for Application Snapshots dealing with PCBUG11 for more tips on making it work.

It is also strongly recommended that you get the Adobe format version of the Motorola s PCBUG11 User Manual. Look for the link on our website RESOURCES page.

Type the following command at the DOS prompt (after RESET

jumpered inside the cable for use with DOS) between connector J2 on your module and COM1 or COM2 of your PC. (You can use a different COM port, but if you re downloading via DOS, you will need to edit the batchfile to reflect the COM port number you use).

3.31 Downloading via DOS. Use **xload1.bat** (for COM1) or **xload2.bat** (for COM2). if you re using the Turbo version of MicroCore-11, use **xload91.bat** or **xload92.bat** instead. If required, use a text editor to modify these batch files to suit your needs. On a Windows95 system, you may have to add \dev\ to every reference to the com port path (eg. copy %1.s19 \dev\com1).

You may find using the batchfile more convenient if you put it into your working directory, along with the assembler or compiler you are using, and the source code you are writing. Make sure to put the associated boot file in the directory as well (**xload.bin** is required by **xload1.bat** and **xload2.bat**).

Always reset the board in BOOT mode and slide the WRITE PROT switch (SW3) to WRITE just before you download. After downloading, switch SW3 back to PROT, and switch SW2 to RUN mode. Press RESET to start your code running.

Note: avoid powering up the board with the WRITE PROT switch in the WRITE position. If you do so, one or more external EEPROM locations may get inadvertently corrupted. If this happens, your code may not work as intended, and you ll have to repeat the above procedure to download it again.

3.32 Downloading via Windows. You can use the supplied MicroLoad for Windows application to send your .s19 file to your board.

Other Windows applications you can use are: HCLOAD, a shareware program for W9x/NT available via our RESOURCES webpage, and ICC11 for Windows (ImageCraft). See section 4.2 for details on using the ICC11 bootloader.

3.4 MicroCore-11 Memory Map

Note that internal MCU memory blocks such as RAM, registers, and EEPROM take priority over any external devices mapped into their address space. In some situations, you may wish to move the internal RAM and/or Register Block to a different 4K address boundary. To do this, change the value of the INIT register in the first few lines of your code. Refer to the 68HC11 Programming Reference Guide for information on the INIT registers. Take care when relocating RAM, when it is to be used for the stack. This is especially important if you re using ImageCraft s ICC11 C compiler, since it jumps to your startup code by executing a JSR instruction. If your startup code moves internal RAM, then the subsequent RTS instruction will not find the return address on the stack (it won t even find the stack). The solution to this is to define the stack in external RAM ($0fff, typically).

3.5 About the On-board Voltage Regulator

MicroCore-11 uses a low-dropout LM2931Z-5 regulator in a TO-92 package, capable of dissipating about 500 mW at room temperature. It has some nice features, such as a very low quiescent current, and will work with an input voltage down to 5 Volts (or below), making it quite well-suited to battery operation. It is also designed to withstand reverse polarity and, if unused, does not present a load to an external regulated 5-Volt supply applied via the 26-pin header H1. One drawback, however, is that it can become unstable and start to oscillate at low temperatures, especially if the input voltage source is connected to J1 via long wires. In the former circumstance, the on-board 10uF tantalum capacitor can be replaced with a higher value (47uF or 100uF). To compensate for long lead-in wires, add capacitance of 100uF or so at, or close to, the J1 connector.

4 REFERENCE

4.1 How EEPROM is Programmed

MicroCore-11 uses parallel EEPROM for program storage, and

is in-system programmable , using the MCU in special bootstrap mode to erase and load the code via any serial port. This means you can erase and re-program your code right in-circuit, without the need for special programming boards and UV erasers. This approach results in a low-cost, easy-to-use configuration suitable for education and new product development. Also, there is nothing proprietary about the design configuration and the parts used. The MCU is a standard Motorola part--you can buy it from Technological Arts or any Motorola supplier to use in your application, without being concerned about availability and licensing.

Bootstrap Mode.
All 68HC11 MCUs contain an internal bootstrap loader program in ROM. This program runs whenever the chip is powered up or reset with the mode select pins configured for SPECIAL BOOTSTRAP mode (see Motorola 68HC11 Reference Manual for details). This configuration can be selected on MicroCore-11 by placing the BOOT/RUN switch in the BOOT position. When the RESET button is pressed (or when a power-on event occurs), the MCU executes the internal bootstrap program (bootloader ROM source code listings can be found in Appendix B of the M68HC11 Reference Manual). It first initializes the serial port (SCI), and waits in a loop checking its receive buffer until it gets a value of $FF (BREAK control character). It uses that character to determine the incoming baudrate, and adjusts its BAUD register accordingly. It then places each byte it receives in RAM, storing them sequentially, starting at address 0 and continuing until a timeout delay has elapsed with no characters arriving at the serial port. At this point, the internal bootstrap program loads the Program Counter with 0, causing the CPU to begin executing the program just loaded into RAM. Thus, the program that you have just downloaded begins executing. This feature of the 68HC11 allows an EEPROM erase/write routine to be placed in RAM and executed automatically. Routines for programming external EEPROM/RAM (in expanded mode) from a Motorola s-record file are included on your MicroCore-11 Starter Package disk. The source code file is called **xload.asm**.

To program the EEPROM, use **xload1.bat** (for COM1) or