

MC68HC11 Bootstrap Mode

Prepared by: **Jim Sibigtroth**
Mike Rhoades
John Langan

INTRODUCTION

M68HC11 MCUs have a bootstrap mode that allows a user-defined program to be loaded into the internal random access memory (RAM) by way of the serial communications interface (SCI); the M68HC11 then executes this loaded program. The loaded program can do anything a normal user program can do as well as anything a factory test program can do because protected control bits are accessible in bootstrap mode. Although the bootstrap mode is a single-chip mode of operation, expanded mode resources are accessible because the mode control bits can be changed while operating in the bootstrap mode.

This application note explains the operation and application of the M68HC11 bootstrap mode. Although the basic concepts associated with this mode are quite simple, the more subtle implications of these functions require careful consideration. Useful applications of this mode are overlooked due to an incomplete understanding of the bootstrap mode. Also, common problems associated with the bootstrap mode could be avoided by a more complete understanding of its operation and implications.

Topics included in this application note are as follows:

- Basic operation of the M68HC11 bootstrap mode
- General discussion of bootstrap mode uses
- Detailed explanation of on-chip bootstrap logic
- Detailed explanation of bootstrap firmware
- Bootstrap firmware vs. EEPROM security
- Incorporating the bootstrap mode into a system
- Driving bootstrap mode from another M68HC11
- Driving bootstrap mode from a personal computer
- Common bootstrap mode problems
- Variations for specific versions of M68HC11
- Commented listings for selected M68HC11 bootstrap ROMs

BASIC BOOTSTRAP MODE

This section describes only basic functions of the bootstrap mode. Other functions of the bootstrap mode are described in detail in the remainder of this application note.

When an M68HC11 is reset in bootstrap mode, the reset vector is fetched from a small internal read-only memory (ROM) called the bootstrap ROM or (boot ROM). The firmware program in this boot ROM then controls the bootloading pro-

cess. First, the on-chip SCI is initialized. The first character received (\$FF) determines which of two possible baud rates should be used for the remaining characters in the download operation. Next, a binary program is received by the SCI system and is stored in RAM. Finally, a jump instruction is executed to pass control from the bootloader firmware to the user's loaded program. Bootstrap mode is useful both at the component level and after the MCU has been embedded into a finished user system.

At the component level, Motorola uses the bootstrap mode to control a monitored burn-in program for the on-chip electrically erasable programmable read-only memory (EEPROM). Units to be tested are loaded into special circuit boards that each hold fifty MCUs. These boards are then placed in burn-in ovens. Driver boards outside the ovens download an EEPROM exercise and diagnostic program to all fifty MCUs in parallel. The MCUs under test independently exercise their internal EEPROM and monitor programming and erase operations. This technique could be utilized by an end user to load program information into the EPROM or EEPROM of an M68HC11 before it is installed into an end product. As in the burn-in setup, many M68HC11s can be gang programmed in parallel. This technique can also be used to program the EPROM of finished products after final assembly.

Motorola also uses bootstrap mode for programming target devices on the M68HC11EVM Evaluation Modules. Because bootstrap mode is a privileged mode like special test, the EEPROM-based configuration register (CONFIG) can be programmed using bootstrap mode on the EVM.

The greatest benefits from bootstrap mode are realized by designing the finished system so that bootstrap mode can be used *after* final assembly. The finished system need not be a single-chip mode application for the bootstrap mode to be useful because the expansion bus can be enabled after resetting the MCU in bootstrap mode. Allowing this capability requires almost no hardware or design cost and the addition of this capability is invisible in the end product until it is needed.

The ability to control the embedded processor through downloaded programs is achieved without the disassembly and chip-swapping usually associated with such control. This mode provides an easy way to load non-volatile memories such as EEPROM with calibration tables or to program the application firmware into a one-time programmable (OTP) MCU after final assembly.

Another powerful use of bootstrap mode in a finished assembly is for final test. Short programs can be downloaded to check parts of the system, including components and circuitry external to the embedded MCU. If any problems appear during product development, diagnostic programs can be downloaded to find the problems, and corrected routines can be downloaded and checked before incorporating them into the main application program.



Bootstrap mode can also be used to interactively calibrate critical analog sensors. Since this calibration is done in the final assembled system, it can compensate for any errors in discrete interface circuitry and cabling between the sensor and the analog inputs to the MCU. Note that this calibration routine is a downloaded program that does not take up space in the normal application program.

BOOTSTRAP MODE LOGIC

In the MC68HC11 very little logic is dedicated to the bootstrap mode: Thus, this mode adds almost no extra cost to the MCU system. The biggest piece of circuitry for bootstrap mode is the small boot ROM. This ROM is 192 bytes in the original MC68HC11A8, but some of the newest members of the M68HC11 Family have as much as 448 bytes to accommodate added features. Normally, this boot ROM is present in the memory map only when the MCU is reset in the bootstrap mode to prevent interference with the user's normal memory space. The enable for this ROM is controlled by the read boot ROM (RBOOT) control bit in the highest priority interrupt (HPRIO) register. The RBOOT bit can be written by software whenever the MCU is in special test or special bootstrap modes; when the MCU is in normal modes, RBOOT reverts to zero and becomes a read-only bit. All other logic in the MCU would be present whether or not there was a bootstrap mode.

Figure 1 shows the composite memory map of the MC68HC711E9 in its four basic modes of operation, including bootstrap mode. The active mode is determined by the mode A (MDA) and special mode (SMOD) control bits in the HPRI0 control register. These control bits are in turn controlled by the state of the mode A (MODA) and mode B (MODB) pins during reset. Table 1 shows the relationship between the state of these pins during reset, the selected mode, and the state of the MDA, SMOD, and RBOOT control bits. Refer to the composite memory map and Table 1 for the following discussion.

The MDA control bit is determined by the state of the MODA pin as the MCU leaves reset. MDA selects between single-chip and expanded operating modes. When MDA is zero, a single-chip mode is selected, either normal single chip or special bootstrap mode. When MDA is one, an expanded mode is selected, either normal expanded mode or special test mode.

The SMOD control bit is determined by the inverted state of the MODB pin as the MCU leaves reset. SMOD controls whether a normal mode or a special mode is selected. When SMOD is zero, one of the two normal modes is selected, either normal single-chip or normal expanded mode. When SMOD is one, one of the two special modes is selected, either special bootstrap mode or special test mode. When either special mode is in effect (SMOD = 1), certain privileges are in effect —

i.e., the ability to write to the mode control bits and fetching the reset and interrupt vectors from \$BFxx rather than \$FFxx.

The alternate vector locations are achieved by simply driving address bit A14 low during all vector fetches if SMOD = 1. For special test mode, the alternate vector locations assure that the reset vector can be fetched from external memory space so the test system can control MCU operation. In special bootstrap mode, the small boot ROM is enabled in the memory map by RBOOT = 1 so the reset vector will be fetched from this ROM and the boot loader firmware will control MCU operation.

RBOOT is reset to one in bootstrap mode to enable the small boot ROM. In the other three modes, RBOOT is reset to zero to keep the boot ROM out of the memory map. While in special test mode, SMOD = 1; which allows the RBOOT control bit to be written to one by software to enable the boot ROM for testing purposes.

BOOT ROM FIRMWARE

The main program in the boot ROM is the boot loader, which is automatically executed as a result of resetting the MCU in bootstrap mode. Some newer versions of the M68HC11 Family have additional utility programs that can be called from a downloaded program. One utility is available to program EPROM or OTP versions of the M68HC11. A second utility allows the contents of memory locations to be uploaded to a host computer. In the MC68HC711K4 boot ROM, a section of code is used by Motorola for stress testing the on-chip EEPROM. These test and utility programs are similar to self-test ROM programs in other MCUs except that the boot ROM does not use valuable space in the normal memory map.

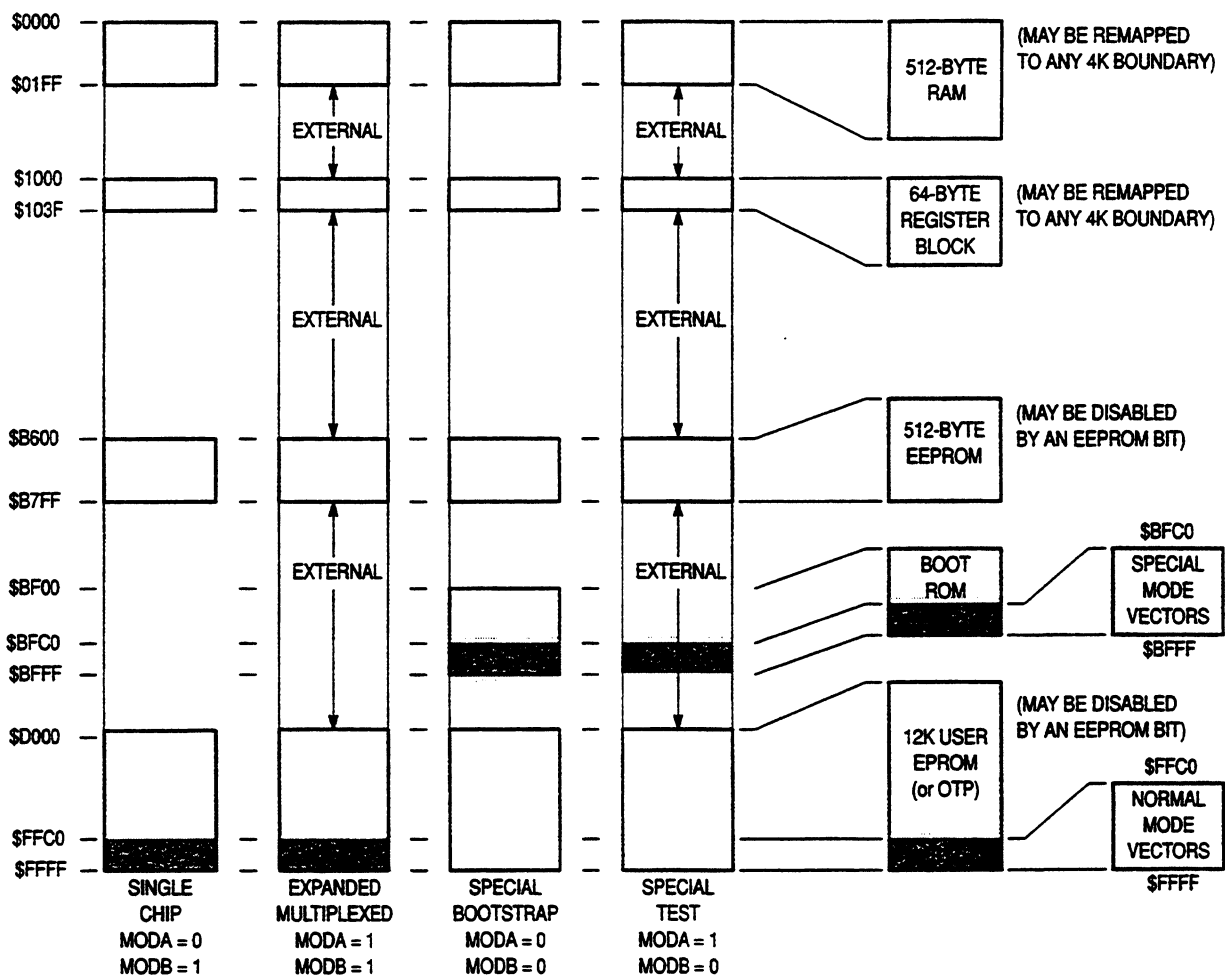
Bootstrap firmware is also involved in an optional EEPROM security function on some versions of the M68HC11. This EEPROM security feature prevents a software pirate from seeing what is in the on-chip EEPROM. The secured state is invoked by programming the no security (NOSEC) EEPROM bit in the CONFIG register. Once this NOSEC bit is programmed to zero, the MCU will ignore the mode A pin and always come out of reset in normal single-chip mode or special bootstrap mode, depending on the state of the mode B pin. Normal single-chip mode is the usual way a secured part would be used. Special bootstrap mode is used to disengage the security function (only after the contents of EEPROM and RAM have been erased). Refer to the M68HC11RM/AD, *M68HC11 Reference Manual* for additional information on the security mode and complete listings of the boot ROMs that support the EEPROM security functions.

AUTOMATIC SELECTION OF BAUD RATE

The boot loader program in the MC68HC711E9 accommodates either of two baud rates. The higher of these baud rates (7812 baud at a 2-MHz E-clock rate) is used in systems that operate from a binary frequency crystal such as 2²³ Hz (8.389 MHz). At this crystal frequency the baud rate is 8192 baud which was used extensively in automotive applications based on the MC6801 MCU. The second baud rate available to the M68HC11 boot loader is 1200 baud at a 2-MHz E-clock rate. Some of the newest versions of the M68HC11 accommodate other baud rates using the same differentiation technique explained here. Refer to the reference numbers in square brackets in Figure 2 during the following explanation.

Table 1. Mode Selection Summary

Input Pins		Mode Selected	Control Bits in HPRI0		
MODB	MODA		RBOOT	SMOD	MDA
1	0	Normal Single Chip	0	0	0
1	1	Normal Expanded	0	0	1
0	0	Special Bootstrap	1	1	0
0	1	Special Test	0	1	1



NOTE: Software can change some aspects of the memory map after reset.

Figure 1. MC68HC711E9 Composite Memory Map

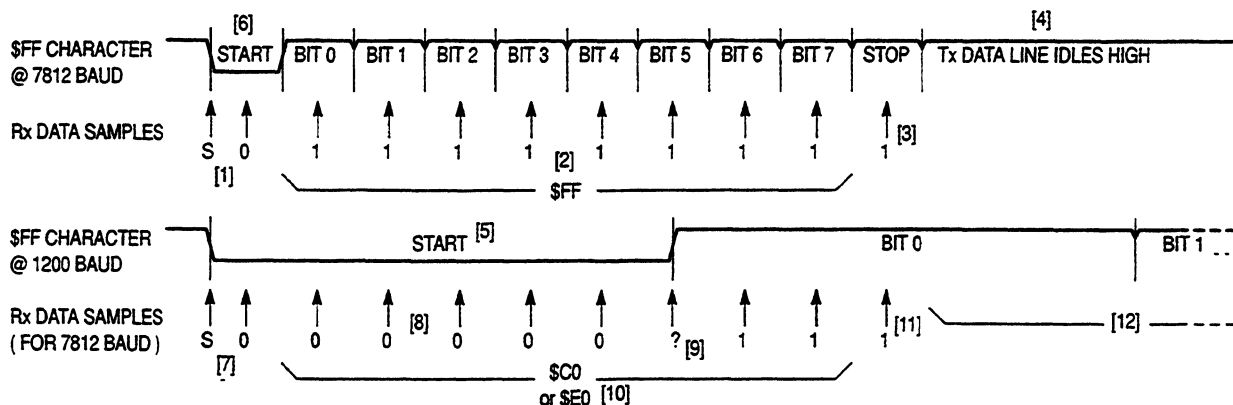


Figure 2. Automatic Detection of Baud Rate

Figure 2 shows how the bootloader program differentiates between the default baud rate (7812 baud at a 2-MHz E-clock rate) and the alternate baud rate (1200 baud at a 2-MHz E-clock rate). The host computer sends an initial \$FF character, which is used by the bootloader to determine the baud rate that will be used for the downloading operation. The top half of Figure 2 shows normal reception of \$FF. Receive data samples at [1] detect the falling edge of the start bit and then verify the start bit by taking a sample at the center of the start bit time. Samples are then taken at the middle of each bit time [2] to reconstruct the value of the received character (all ones in this case). A sample is then taken at the middle of the stop bit time as a framing check (a one is expected) [3]. Unless another character immediately follows this \$FF character, the receive data line will idle in the high state as shown at [4].

The bottom half of Figure 2 shows how the receiver will incorrectly receive the \$FF character that is sent from the host at 1200 baud. Because the receiver is set to 7812 baud, the receive data samples are taken at the same times as in the upper half of Figure 2. The start bit at 1200 baud [5] is 6.5 times as long as the start bit at 7812 baud [6].

Samples taken at [7] detect the falling edge of the start bit and verify it is a logic zero. Samples taken at the middle of what the receiver thinks are the first five bit times [8] detect logic zeros. The sample taken at the middle of what the receiver thinks is bit 5 [9] may detect either a zero or a one because the receive data has a rising transition at about this time. The samples for bits 6 and 7 detect ones, causing the receiver to think the received character was \$C0 or \$E0 [10] at 7812 baud instead of the \$FF which was sent at 1200 baud. The stop bit sample detects a one as expected [11], but this detection is actually in the middle of bit 0 of the 1200 baud \$FF character. The SCI receiver is not confused by the rest of the 1200 baud \$FF character because the receive data line is high [12] just as it would be for the idle condition. If a character other than \$FF is sent as the first character, an SCI receive error could result.

MAIN BOOTLOADER PROGRAM

Figure 3 is a flowchart of the main bootloader program in the MC68HC711E9. This bootloader demonstrates the most important features of the bootloaders used on all M68HC11 Family members. For complete listings of other M68HC11 versions refer to Listings 3–8 at the end of this application note, and

appendix B of the M68HC11RM/AD, *M68HC11 Reference Manual*.

The reset vector in the boot ROM points to the start [1] of this program. The initialization block [2] establishes starting conditions and sets up the SCI and port D. The stack pointer is set because there are push and pull instructions in the bootloader program. The X index register is pointed at the start of the register block (\$1000) so indexed addressing can be used. Indexed addressing takes one less byte of ROM space than extended instructions, and bit manipulation instructions are not available in extended addressing forms. The port D wire-OR mode (DWOM) bit in the serial peripheral interface control register (SPCR) is set to configure port D for wired-OR operation to minimize potential conflicts with external systems that use the PD1/TxD pin as an input. The baud rate for the SCI is initially set to 7812 baud at a 2-MHz E-clock rate but can automatically switch to 1200 baud based on the first character received. The SCI receiver and transmitter are enabled. The receiver is required by the bootloading process, and the transmitter is used to transmit data back to the host computer for optional verification. The last item in the initialization is to set an intercharacter delay constant used to terminate the download when the host computer stops sending data to the MC68HC711E9. This delay constant is stored in the timer output compare 1 (TOC1) register, but the on-chip timer is not used in the bootloader program. This example illustrates the extreme measures used in the bootloader firmware to minimize memory usage. However such measures are not usually considered good programming technique because they are misleading to someone trying to understand the program.

After initialization, a break character is transmitted [3] by the SCI. By connecting the TxD pin to the RxD pin (with a pullup because of port D wired-OR mode), this break will be received as a \$00 character and cause an immediate jump [4] to the start of the on-chip EEPROM (\$B600 in the MC68HC711E9). This feature is useful to pass control to a program in EEPROM essentially from reset. Refer to **COMMON BOOTSTRAP MODE PROBLEMS** before using this feature.

If the first character is received as \$FF, the baud rate is assumed to be the default rate (7812 baud at a 2-MHz E-clock rate). If \$FF was sent at 1200 baud by the host, the SCI will receive the character as \$E0 or \$C0 because of the baud rate mismatch, and the bootloader will switch to 1200 baud [5] for the rest of the download operation. When the baud rate is switched to 1200 baud, the delay constant used to monitor the

intercharacter delay must also be changed to reflect the new character time.

At [6], the Y index register is initialized to \$0000 to point to the start of on-chip RAM. The index register Y is used to keep track of where the next received data byte will be stored in RAM. The main loop for loading begins at [7].

The number of data bytes in the downloaded program can be any number between zero and 512 bytes (the size of on-chip RAM). This procedure is called 'variable-length download' and is accomplished by ending the download sequence when an idle time of at least four character times occurs after the last character to be downloaded. In M68HC11 Family members which have 256 bytes of RAM, the download length is fixed at exactly 256 bytes plus the leading \$FF character.

The intercharacter delay counter is started [8] by loading the delay constant from TOC1 into the X index register. The 19-E-cycle wait loop is executed repeatedly until either a character is received [9] or the allowed intercharacter delay time expires [10]. For 7812 baud, the delay constant is 10,241 E cycles (539 X 19 E cycles per loop). Four character times at 7812 baud is 10,240 E cycles (baud prescale of 4 X baud divider of 4 X 16 internal SCI clocks/bit time X 10 bit times/character X 4 character times). The delay from reset to the initial \$FF character is not critical since the delay counter is not started until after the first character (\$FF) is received.

To terminate the bootloading sequence and jump to the start of RAM without downloading any data to the on-chip RAM, simply send \$FF and nothing else. This feature is similar to the jump to EEPROM at [4] except the \$FF causes a jump to the start of RAM. This procedure requires that the RAM has been loaded with a valid program since it would make no sense to jump to a location in uninitialized memory.

After receiving a character, the downloaded byte is stored in RAM [11]. The data is transmitted back to the host [12] as an indication that the download is progressing normally. At [13], the RAM pointer is incremented to the next RAM address. If the RAM pointer has not passed the end of RAM, the main download loop (from [7] to [14]) is repeated.

When all data has been downloaded, the bootloader goes to [16] because of an intercharacter delay timeout [10] or because the entire 512-byte RAM has been filled [15]. At [16], the X and Y index registers are set up for calling the PROGRAM utility routine, which saves the user from having to do this in a downloaded program. The PROGRAM utility is fully explained in EPROM PROGRAMMING UTILITY. The final step of the bootloader program is to jump to the start of RAM [17], which starts the user's downloaded program.

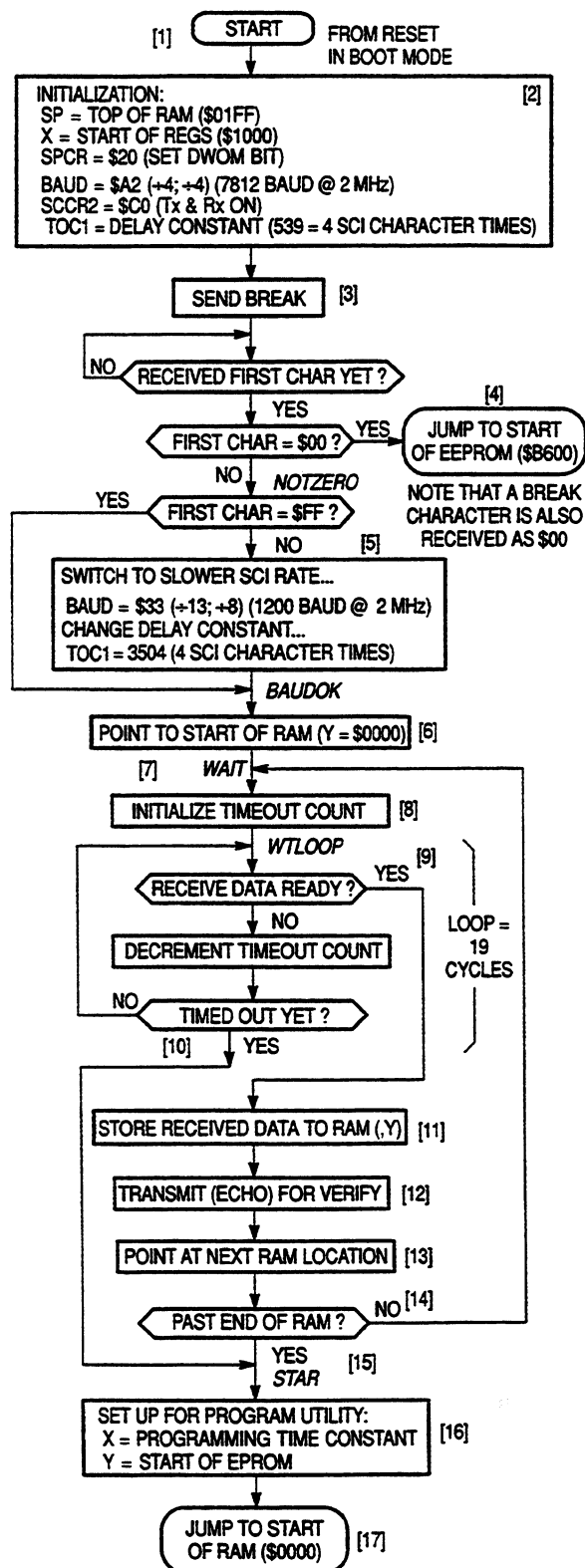


Figure 3. MC68HC711E9 Bootloader Flowchart

UPLOAD UTILITY

The UPLOAD utility subroutine transfers data from the MCU to a host computer system over the SCI serial data link. Note that M68HC11 versions that support EEPROM security do not include this utility. Verification of EPROM contents is one example of how the UPLOAD utility could be used. Before calling this program, the Y index register is loaded (by user firmware) with the address of the first data byte to be uploaded. If a baud rate other than the current SCI baud rate is to be used for the upload process, the user's firmware must also write to the BAUD register. The UPLOAD program sends successive bytes of data out the SCI transmitter until a reset is issued (the upload loop is infinite). For a complete commented listing of the UPLOAD utility, refer to Listings at the back of this application note.

EPROM PROGRAMMING UTILITY

The EPROM programming utility is one way of programming data into the internal EPROM of the MC68HC711E9 MCU. An external 12-V programming power supply is required to program on-chip EPROM. The simplest way to use this utility program is to bootload a three-byte program consisting of a single jump instruction to the start of the PROGRAM utility program (\$BF00). The bootloader program sets the X and Y index registers to default values before jumping to the downloaded program (see [16] at the bottom of Figure 3). When the host computer sees the \$FF character, data to be programmed into the EPROM is sent, starting with the character for location \$D000. After the last byte to be programmed is sent to the MC68HC711E9 and the corresponding verification data is returned to the host, the programming operation is terminated by resetting the MCU.

The number of bytes to be programmed, the first address to be programmed, and the programming time can be controlled by the user if values other than the default values are desired.

To understand the detailed operation of the EPROM programming utility, refer to Figure 4 during the following discussion. Figure 4 is composed of three interrelated parts. The upper-left portion shows the flowchart of the PROGRAM utility running in the boot ROM of the MCU. The upper-right portion shows the flowchart for the user-supplied driver program running in the host computer. The lower portion of Figure 4 is a timing sequence showing the relationship of operations between the MCU and the host computer. Reference numbers in the flowcharts in the upper half of Figure 4 have matching numbers in the lower half to help the reader relate the three parts of the figure.

The shaded area [1] refers to the software and hardware latency in the MCU leading to the transmission of a character (in this case, the \$FF). The shaded area [2] refers to a similar latency in the host computer (in this case, leading to the transmission of the first data character to the MCU).

The overall operation begins when the MCU sends the first character (\$FF) to the host computer, indicating that it is ready for the first data character. The host computer sends the first data byte [3] and enters its main loop. The second data character is sent [4], and the host then waits [5] for the first verify byte to come back from the MCU.

After the MCU sends \$FF [8], it enters the WAIT1 loop [9] and waits for the first data character from the host. When this character is received [10] the MCU programs it into the address pointed to by the Y index register. When the programming time delay is over, the MCU reads the programmed data, transmits it to the host for verification [11], and returns to the top of the WAIT1 loop to wait for the next data character [12]. Because the host previously sent the second data character, it is already waiting in the SCI receiver of the MCU. Steps [13], [14], and [15] correspond to the second pass through the WAIT1 loop.

Back in the host, the first verify character is received, and the third data character is sent [6]. The host then waits for the second verify character [7] to come back from the MCU. The sequence continues as long as the host continues to send data to the MCU. Since the WAIT1 loop in the PROGRAM utility is an indefinite loop, reset is used to end the process in the MCU after the host has finished sending data to be programmed.

ALLOWING FOR BOOTSTRAP MODE

Since bootstrap mode requires very few connections to the MCU, it is easy to design systems that accommodate the bootstrap mode. Bootstrap mode is useful for diagnosing or repairing systems that have failed due to changes in the CONFIG register or failures of the expansion address/data buses, (rendering programs in external memory useless). Bootstrap mode can also be used to load information into the EPROM or EEPROM of an M68HC11 after final assembly of a module. Bootstrap mode is also useful for performing system checks and calibration routines. The following paragraphs explain system requirements for use of bootstrap mode in a product.

MODE SELECT PINS: It must be possible to force the MODA and MODB pins to logic zero, which implies that these two pins should be pulled up to V_{DD} through resistors rather than being tied directly to V_{DD}. If mode pins are connected directly to V_{DD} it is not possible to force a mode other than the one the MCU is hard wired for. It is also good practice to use pulldown resistors to V_{SS} rather than connecting mode pins directly to V_{SS} because it is sometimes a useful debug aid to attempt reset in modes other than the one the system was primarily designed for. Physically, this requirement sometimes calls for the addition of a test point or a wire connected to one or both mode pins. Mode selection only uses the mode pins while **RESET** is active.

RESET: It must be possible to initiate a reset while the mode select pins are held low. In systems where there is no provision for manual reset, it is usually possible to generate a reset by turning power off and back on.

RxD PIN: It must be possible to drive the PD0/RxD pin with serial data from a host computer (or another MCU). In many systems, this pin is already used for SCI communications; thus no changes are required.

In systems where the PD0/RxD pin is normally used as a general-purpose output, a serial signal from the host can be connected to the pin without resulting in output driver conflicts. It may be important to consider what the existing logic will do with the SCI serial data instead of the signals that would have been produced by the PD0 pin. In systems where the PD0 pin is normally used as a general-purpose input, the driver circuit

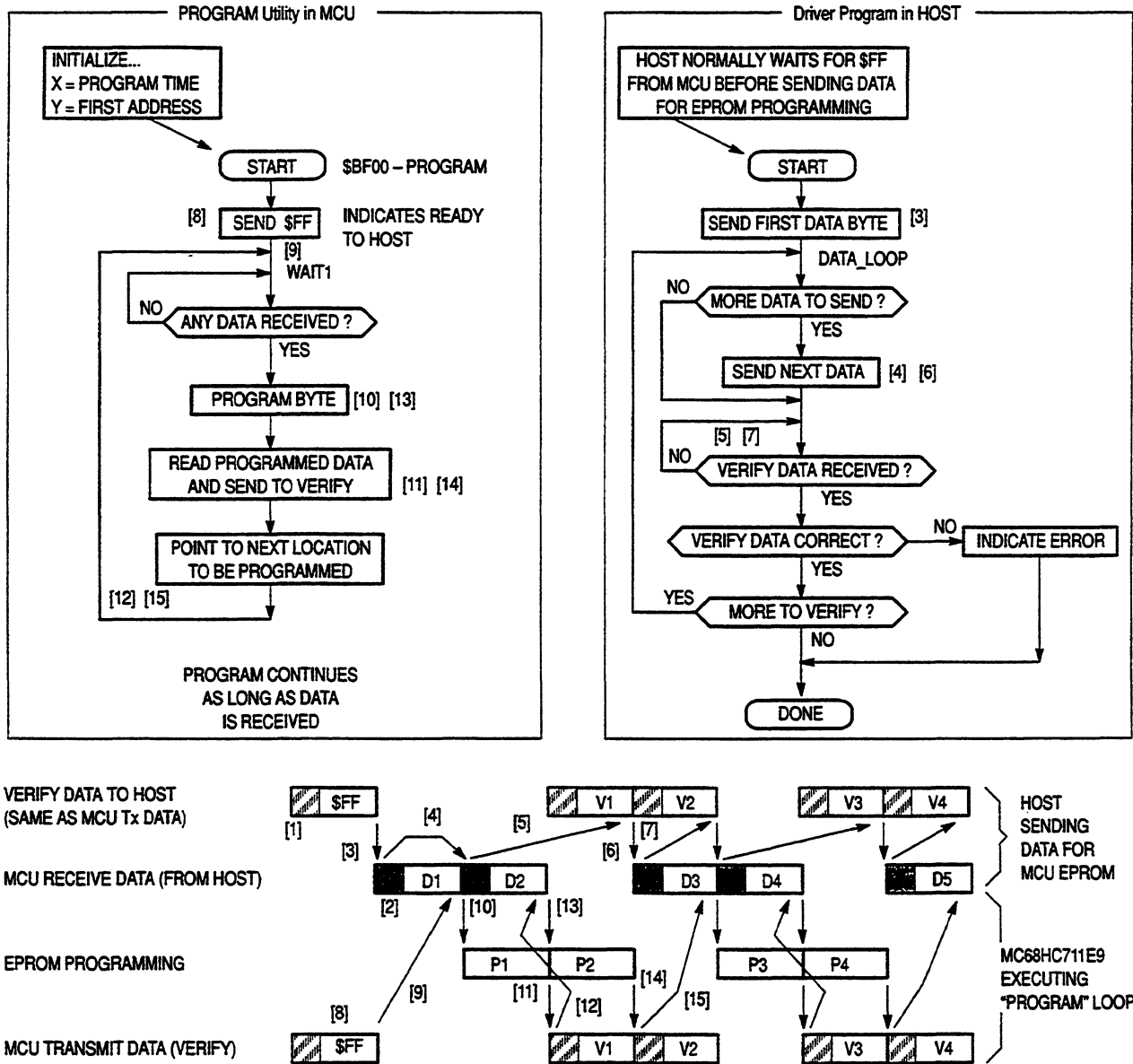


Figure 4. Host and MCU Activity during EPROM PROGRAM Utility

that drives the PD0 pin must be designed so that the serial data can override this driver, or the driver must be disconnected during the bootstrap download. A simple series resistor between the driver and the PD0 pin solves this problem as shown in Figure 5. The serial data from the host computer can then be connected to the PD0/RxD pin, and the series resistor will prevent direct conflict between the host driver and the normal PD0 driver.

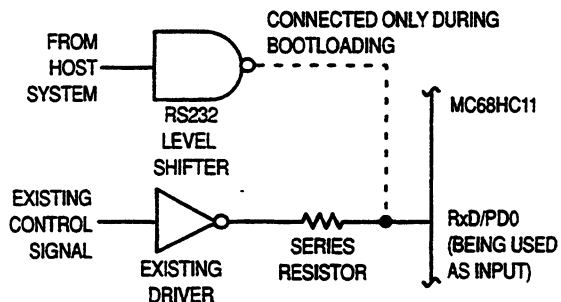


Figure 5. Preventing Driver Conflict

TxD PIN: The bootloader program uses the PD1/TxD pin to send verification data back to the host computer. To minimize the possibility of conflicts with circuitry connected to this pin, port D is configured for wire-OR mode by the bootloader program during initialization. Since the wire-OR configuration prevents the pin from driving active high levels, a pullup resistor to V_{DD} is needed if the TxD signal is used.

In systems where the PD1/TxD pin is normally used as a general-purpose output, there are no output driver conflicts. It may be important to consider what the existing logic will do with the SCI serial data instead of the signals that would have been produced by the PD1 pin.

In systems where the PD1 pin is normally used as a general-purpose input, the driver circuit that drives the PD1 pin must be designed so that the PD1/TxD pin driver in the MCU can override this driver. A simple series resistor between the driver and the PD1 pin can solve this problem. The TxD pin can then be configured as an output, and the series resistor will prevent direct conflict between the internal TxD driver and the external driver connected to PD1 through the series resistor.

OTHER: The bootloader firmware sets the DWOM control bit, which configures all port D pins for wire-OR operation. During the bootloading process, all port D pins except the PD1/TxD pin are configured as high-impedance inputs. Any port D pin that is normally used as an output should have a pullup resistor so it does not float during the bootloading process.

DRIVING BOOT MODE FROM ANOTHER M68HC11

A second M68HC11 system can easily act as the host to drive bootstrap loading of an M68HC11 MCU. This method is used to examine and program nonvolatile memories in target M68HC11s in Motorola EVMs. The following hardware and software example will demonstrate this and other bootstrap mode features.

The schematic in Figure 6 shows the circuitry for a simple EPROM duplicator for the MC68HC711E9. The circuitry is built in the wire-wrap area of an M68HC11EVBU Evaluation

Board to simplify construction. The schematic shows only the important portions of the EVBU circuitry to avoid confusion. To see the complete EVBU schematic, refer to the M68HC11EVBU/D, *M68HC11EVBU Universal Evaluation Board User's Manual*.

The default configuration of the EVBU must be changed to make the appropriate connections to the circuitry in the wire-wrap area and to configure the master MCU for bootstrap mode. A fabricated jumper must be installed at J6 to connect the XTAL output of the master MCU to the wire-wrap connector P5, which has been wired to the EXTAL input of the target MCU. Cut traces that short across J8 and J9 must be cut on the solder side of the printed circuit board to disconnect the normal SCI connections to the RS232 level translator (U4) of the EVBU. The J8 and J9 connections can easily be restored at a later time by installing fabricated jumpers on the component side of the board. A fabricated jumper must be installed across J3 to configure the master MCU for bootstrap mode.

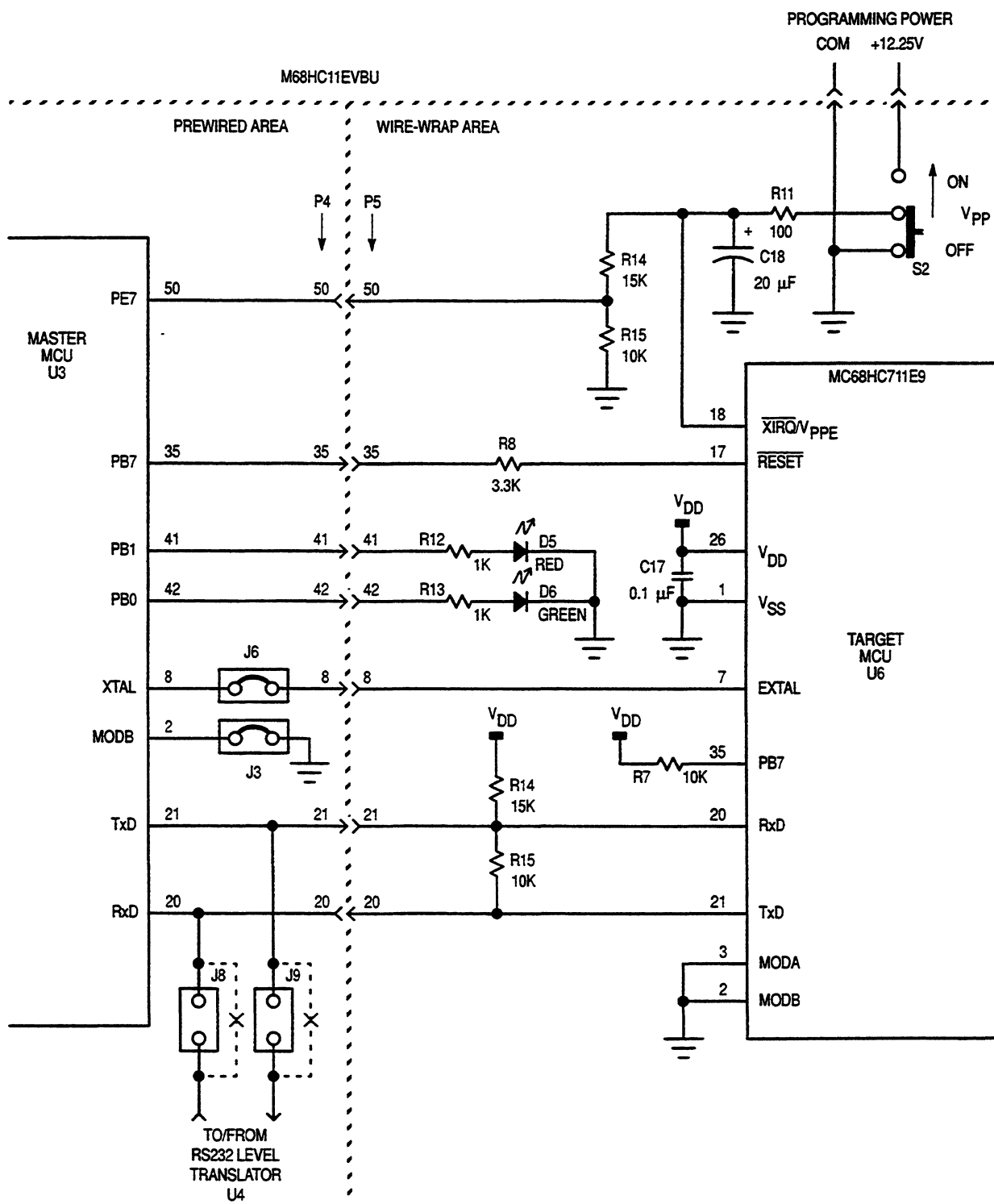
One MC68HC711E9 is first programmed by other means with a desired 12K-byte program in its EPROM and a small duplicator program in its EEPROM. Alternately, the ROM program in an MC68HC11E9 can be copied into the EPROM of a target MC68HC711E9 by programming only the duplicator program into the EEPROM of the master MC68HC11E9. The master MCU is installed in the EVBU at socket U3. A blank MC68HC711E9 to be programmed is placed in the socket in the wire-wrap area of the EVBU (U6).

With the V_{pp} power switch off, power is applied to the EVBU system. As power is applied to the EVBU, the master MCU (U3) comes out of reset in bootstrap mode. Target MCU (U6) is held in reset by the PB7 output of master MCU (U3). The PB7 output of U3 is forced to zero when U3 is reset. The master MCU will later release the reset signal to the target MCU under software control. The RxD and TxD pins of the target MCU (U6) are high-impedance inputs while U6 is in reset so they will not affect the TxD and RxD signals of the master MCU (U3) while U3 is coming out of reset. Since the target MCU is being held in reset with MODA and MODB at zero, it is configured for the EPROM emulation mode, and PB7 is the output enable signal for the EPROM data I/O pins. Pullup resistor R7 causes the port D pins including RxD and TxD, to remain in the high-impedance state so they do not interfere with the RxD and TxD pins of the master MCU as it comes out of reset.

As U3 leaves reset, its mode pins select bootstrap mode so the bootloader firmware begins executing. A break is sent out the TxD pin of U3. Pullup resistor R10 and resistor R9 cause the break character to be seen at the RxD pin of U3. The bootloader performs a jump to the start of EEPROM in the master MCU (U3) and starts executing the duplicator program. This sequence demonstrates how to use bootstrap mode to pass control to the start of EEPROM after reset.

The complete listing for the duplicator program in the EEPROM of the master MCU is provided in Listing 1.

The duplicator program in EEPROM clears the DWOM control bit to change port D (thus, TxD) of U3 to normal driven outputs. This configuration will prevent interference due to R9 when TxD from the target MCU (U6) becomes active. Series resistor R9 demonstrates how TxD of U3 can drive RxD of U3 and later TxD of U6 can drive RxD of U3 without a destructive conflict between the TxD output buffers.



NOTE: Only the most important portions of EVBU circuitry are shown.

Figure 6. MCU to MCU EPROM Duplicator Schematic

As the target MCU (U6) leaves reset, its mode pins select bootstrap mode so the bootloader firmware begins executing. A break is sent out the TxD pin of U6. At this time, the TxD pin of U3 is at a driven high so R9 acts as a pullup resistor for TxD of the target MCU (U6). The break character sent from U6 is received by U3 so the duplicator program that is running in the EEPROM of the master MCU knows that the target MCU is ready to accept a bootloaded program.

The master MCU sends a leading \$FF character to set the baud rate in the target MCU. Next, the master MCU passes a three-instruction program to the target MCU and pauses so the bootstrap program in the target MCU will stop the loading process and jump to the start of the downloaded program. This sequence demonstrates the variable-length download feature of the MC68HC711E9 bootloader.

The short program downloaded to the target MCU clears the DWOM bit to change its TxD pin to a normal driven CMOS output and jumps to the EPROM programming utility in the bootstrap ROM of the target MCU.

Note that the small downloaded program did not have to set up the SCI or initialize any parameters for the EPROM programming process. The bootstrap software that ran prior to the loaded program left the SCI turned on and configured in a way that was compatible with the SCI in the master MCU (the duplicator program in the master MCU also did not have to set up the SCI for the same reason). The programming time and starting address for EPROM programming in the target MCU were also set to default values by the bootloader software before jumping to the start of the downloaded program.

Before the EPROM in the target MCU can be programmed, the Vpp power supply must be available at the \overline{XIRQ}/V_{PP} pin of the target MCU. The duplicator program running in the master MCU monitors this voltage (for presence or absence — not level) at PE7 through resistor divider R14 - R15. The PE7 input was chosen because the internal circuitry for port E pins can tolerate voltages slightly higher than V_{DD} ; therefore resistors R14 and R15 are less critical. No data to be programmed is passed to the target MCU until the master MCU senses that Vpp has been stable for about 200 ms.

When Vpp is ready, the master MCU turns on the red LED and begins passing data to the target MCU. **EPROM PROGRAMMING UTILITY** explains the activity as data is sent from the master MCU to the target MCU and programmed into the EPROM of the target. The master MCU in the EVBU corresponds to the HOST in the programming utility description, and the "PROGRAM utility in MCU" is running in the bootstrap ROM of the target MCU.

Each byte of data sent to the target is programmed and then the programmed location is read and sent back to the master for verification. If any byte fails, the red and green LEDs are turned off, and the programming operation is aborted. If the entire 12K bytes are programmed and verified successfully, the red LED is turned off, and the green LED is turned on to indicate success. The programming of all 12K bytes takes about 30 sec.

After a programming operation, the Vpp switch (S2) should be turned off before the EVBU power is turned off.

```

1 *****
2 * 68HC711E9 Duplicator Program for AN1060
3 *****
4
5 *****
6 * Equates - All reg addrs except INIT are 2-digit
7 * for direct addressing
8 *****
9 103D      INIT      EQU    $103D          RAM, Reg mapping
10 0028      SPCR      EQU    $28           DWOM in bit-5
11 0004      PORTB     EQU    $04           Red LED = bit-1, Grn = bit-0
12 *
13 *          Reset of prog socket = bit-7
14 0080      RESET     EQU    %10000000
15 0002      RED       EQU    %00000010
16 0001      GREEN     EQU    %00000001
17 000A      PORTE     EQU    $0A           Vpp Sense in bit-7, 1=ON
18 002E      SCSR      EQU    $2E           SCI status register
19 * TDRE, TC, RDRF, IDLE; OR, NF, FE, -
20 0080      TDRE      EQU    %10000000
21 0020      RDRF      EQU    %00100000
22 002F      SCDR      EQU    $2F           SCI data register
23 BF00      PROGRAM   EQU    $BF00        EPROM prog utility in boot ROM
24 D000      EPSTRT    EQU    $D000        Starting address of EPROM
25
26 B600      ORG        $B600              Start of EEPROM
27
28 *****
29 *
30 B600 7F103D BEGIN      CLR      INIT      Moves Registers to $0000-3F
31 B603 8604          LDAA     #$04          Pattern for DWOM off, no SPI
32 B605 9728          STAA     SPCR          Turns off DWOM in EVBU MCU
33 B607 8680          LDAA     #RESET
34 B609 9704          STAA     PORTB        Release reset to target MCU
35 B60B 132E20FC WT4BRK BRCLR    SCSR RDRF WT4BRK Loop till char received
36 B60F 86FF          LDAA     #$FF          Leading char for bootstrap ...
37 B611 972F          STAA     SCDR          to target MCU
38 B613 CEB675        LDX      #BLPROG      Point at program for target
39 B616 8D53          BSR      SEND1        Bootstrap to target
40 B618 8CB67D        CPX      #ENDBPR      Past end ?
41 B61B 26F9          BNE      BLLOOP       Continue till all sent
42
43 *****
44 * Delay for about 4 char times to allow boot related
45 * SCI communications to finish before clearing
46 * Rx related flags
47 B61D CE06A7        LDX      #1703        # of 6 cyc loops
48 B620 09          DLYLP      DEX          [3]
49 B621 26FD          BNE      DLYLP        [3] Total loop time = 6 cyc
50 B623 962E          LDAA     SCSR          Read status (RDRF will be set)
51 B625 962F          LDAA     SCDR          Read SCI data reg to clear RDRF
52
53 *****
54 * Now wait for character from target to indicate it's ready for
55 * data to be programmed into EPROM
56 B627 132E20FC WT4FF  BRCLR    SCSR RDRF WT4FF Wait for RDRF
57 B62B 962F          LDAA     SCSR          Clear RDRF, don't need data
58 B62D CED000        LDX      #EPSTRT      Point at start of EPROM
59
60 * Handle turn-on of Vpp
61 B630 18CE523D WT4VPP LDY      #21053      Delay counter (about 200ms)
62 B634 150402        BCLR     PORTB RED    Turn off RED LED
63 B637 960A          DLYLP2     LDAA     PORTE [3] Wait for Vpp to be ON
64 B639 2AF5          BPL      WT4VPP [3] Vpp sense is on port E MSB
65 B63B 140402        BSET     PORTB RED    [6] Turn on RED LEDdd
66 B63E 1809          DEY          [4]
67 B640 26F5          BNE      DLYLP2 [3] Total loop time = 19 cyc
68
69 * Vpp has been stable for 200ms
70
71 B642 18CED000        LDY      #EPSTRT      X=Tx pointer, Y=verify pointer
72 B646 8D23          BSR      SEND1        Send first data to target
73 B648 8C0000        CPX      #0           X points at $0000 after last
74 B64B 2702          BEQ      VERF          Skip send if no more
75 B64D 8D1C          BSR      SEND1        Send another data char
76 B64F 132E20FC VERF  BRCLR    SCSR RDRF VERF Wait for Rx ready
77 B653 962F          LDAA     SCDR          Get char and clr RDRF
78 B655 18A100        CMPA     0,Y          Does char verify ?
79 B658 2705          BEQ      VERFOK        Skip error if OK
80 B65A 150403        BCLR     PORTB (RED+GREEN) Turn off LEDs
81 B65D 2007          BRA      DUNPRG        Done (programming failed)
82
83 B65F          VERFOK      INY          Advance verify pointer
84 B65F 1808          BNE      DATALP        Continue till all done
85 B661 26E5

```

Listing 1. MCU to MCU Duplicator Program

Sheet 2 of 2

```

80 B663
81 B663 140401          BSET   PORTB GREEN      Grn LED ON
82 B666
83 B666 150482  DUNPRG   BCLR   PORTB (RESET+RED) Red OFF, apply reset
84 B669 20FE      BRA    *                    Done so just hang
85 B66B
86
87      *****
88      * Subroutine to get & send an SCI char. Also
89      * advances pointer (X).
90      *****
90 B66B A600  SEND1      LDAA    0,X              Get a character
91 B66D 132E80FC TRDYLP  BRCLR   SCSR TDRE TRDYLP Wait for TDRE
92 B671 972F      STAA    SCDR                    Send character
93 B673 08        INX                      Advance pointer
94 B674 39        RTS                        ** Return **
95
96      *****
97      * Program to be bootloaded to target '711E9
98      *****
99 B675 8604  BLPROG     LDAA    #$04              Pattern for DWOM off, no SPI
100 B677 B71028 STAA     $1028              Turns off DWOM in target MCU
101
102      * NOTE: Can't use direct addressing in target MCU because
103      *       regs are located at $1000.
103 B67A 7EBF00  JMP      PROGRAM              Jumps to EPROM prog routine
104 B67D        ENDBPR    EQU      *

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BEGIN	B600	*00029		
BLLOOP	B616	*00038	00040	
BLPROG	B675	*00099	0037	
DATALP	B648	*00068	00079	
DLYLP	B620	*00046	00047	
DLYLP2	B637	*00059	00063	
DUNPRG	B666	*00083	00076	
ENDBPR	B67D	*00104	00039	
EPSTRT	D000	*00023	00055	00066
GREEN	0001	*00015	00075	00081
INIT	103D	*00009	00029	
PORTB	0004	*00011	00033	00058 00061 00075 00081 00083
PORTE	000A	*00016	00059	
PROGRAM	BF00	*00022	00103	
RDRF	0020	*00020	00034	00053 00071
RED	0002	*00014	00058	00061 00075 00083
RESET	0080	*00013	00032	00083
SCDR	002F	*00021	00036	00049 00054 00072 00092
SCSR	002E	*00017	00034	00048 00053 00071 00091
SEND1	B66B	*00090	00038	00067 00070
SPCR	0028	*00010	00031	
TDRE	0080	*00019	00091	
TRDYLP	B66D	*00091	00091	
VERF	B64F	*00071	00069	00071
VERFOK	B65F	*00078	00074	
WT4BRK	B60B	*00034	00034	
WT4FF	B627	*00053	00053	
WT4VPP	B630	*00057	00060	

Errors: None

Labels: 28

Last Program Address: \$B67C

Last Storage Address: \$0000

Program Bytes: \$007D 125

Storage Bytes: \$0000 0

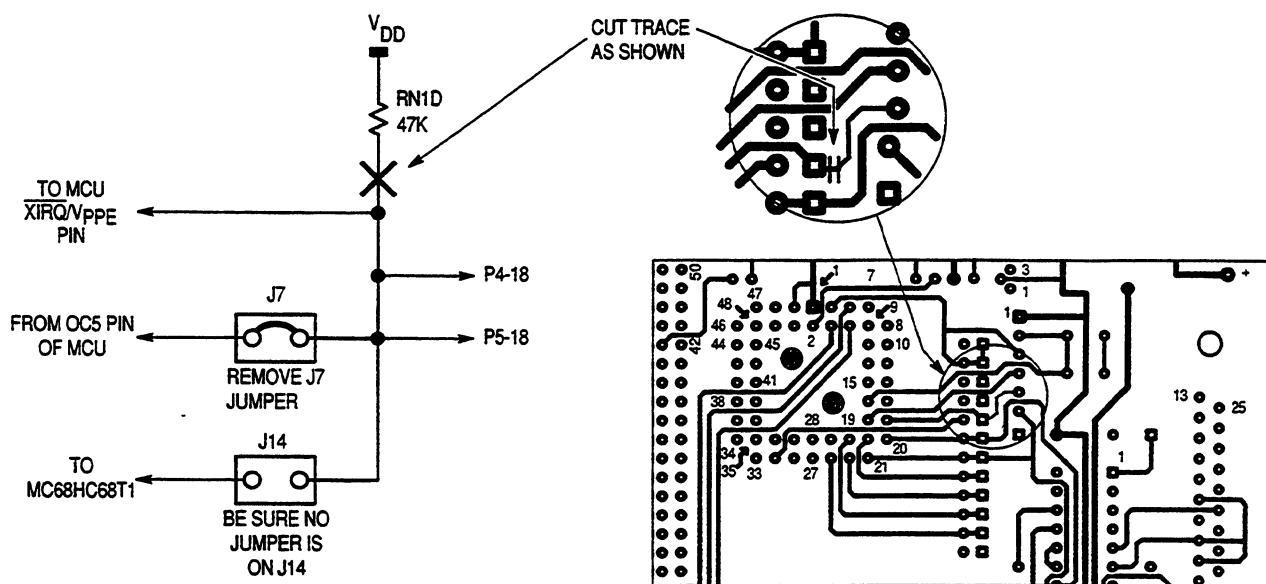


Figure 7. Isolating EVBU $\overline{\text{XIRQ}}$ Pin

DRIVING BOOT MODE FROM A PERSONAL COMPUTER

In this example, a personal computer is used as the host to drive the bootloader of an MC68HC711E9. An M68HC11EVBU is used for the target MC68HC711E9. A large program is transferred from the personal computer into the EPROM of the target MC68HC711E9.

HARDWARE: Figure 7 shows a small modification to the EVBU to accommodate the 12-V (nominal) EPROM programming voltage. The $\overline{\text{XIRQ}}$ pin is connected to a pullup resistor, two jumpers, and the 60-pin connectors, P4 and P5. The object of the modification is to isolate the $\overline{\text{XIRQ}}$ pin and then connect it to the programming power supply. Carefully cut the trace on the solder side of the EVBU as indicated in Figure 7. This disconnects the pullup resistor RN1D from $\overline{\text{XIRQ}}$ but leaves P4-18, P5-18, and jumpers J7 and J14 connected so the EVBU can still be used for other purposes after programming is done. Remove any fabricated jumpers from J7 and J14. The EVBU normally has a jumper at J7 to support the trace function.

Figure 8 shows a small circuit that is added to the wire-wrap area of the EVBU. The three-terminal jumper allows the $\overline{\text{XIRQ}}$ line to be connected to either the programming power supply or to a substitute pullup resistor for $\overline{\text{XIRQ}}$. The 100-ohm resistor is a current limiter to protect the 12-V input of the MCU. The resistor and LED connected to P5 pin 9 (port C bit 0) is an optional indicator that lights when programming is complete.

SOFTWARE: BASIC was chosen as the programming language due to its readability and availability in parallel versions on both the IBM^{TM1} PC and the Macintosh^{TM2}. The program demonstrates several programming techniques for use with an M68HC11 and is not necessarily intended to be a finished, commercial program. For example, there is very little error checking, and the user interface is very elementary. A complete listing of the BASIC program is included in Listing 2 with moderate comments. The following paragraphs include a more detailed discussion of the program as it pertains to communicating with and programming the target

MC68HC711E9. Lines 25–45 initialize and define the variables and array used in the program. Changes to this section would allow for other programs to be downloaded.

1. IBM is a trademark of International Business Machines.
2. Macintosh is a trademark of Apple Computers, Inc.

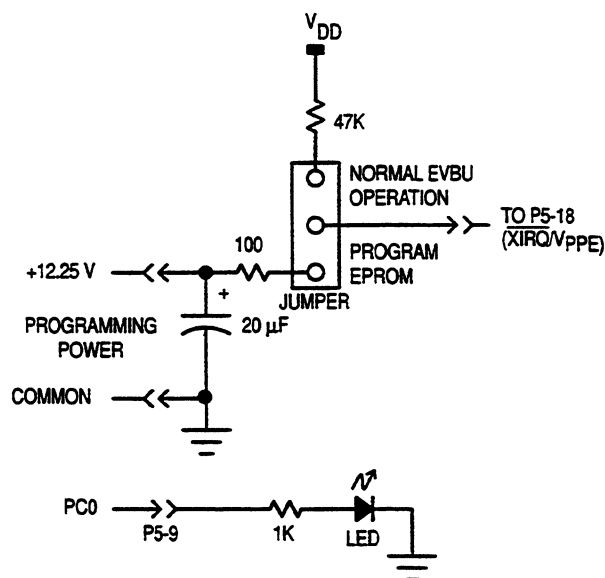


Figure 8. PC to MCU Programming Circuit

Lines 50–95 read in the small bootloader from DATA statements at the end of the listing. The source code for this bootloader is presented in the DATA statements. The bootloaded code makes port C bit 0 low, initializes the X and Y registers for use by the EPROM programming utility routine contained in the boot ROM, and then jumps to that routine. The hexadecimal values read in from the DATA statements are converted to binary values by a subroutine. The binary values are then saved as one string (BOOTCODE\$).

The next long section of code (lines 97–1250) reads in the S-records from an external disk file (in this case, BUF34.S19), converts them to integer, and saves them in an array. The techniques used in this section show how to convert ASCII S-records to binary form that can be sent (bootloaded) to an M68HC11.

This S-record translator only looks for the S1 records that contain the actual object code. All other S-record types are ignored.

When an S1 record is found (line 1000–1024), the next two characters form the hex byte giving the number of hex bytes to follow. This byte is converted to integer by the same subroutine that converted the bootloaded code from the DATA statements. This BYTECOUNT is adjusted by subtracting 3, which accounts for the address and checksum bytes and leaves just the number of object-code bytes in the record.

Starting at line 1100, the two-byte (four-character) starting address is converted to decimal. This address is the starting address for the object-code bytes to follow. An index into the CODE% array is formed by subtracting the base address initialized at the start of the program from the starting address for this S-record.

A FOR-NEXT loop starting at line 1130 converts the object-code bytes to decimal and saves them in the CODE% array. When all the object-code bytes have been converted from the current S-record, the program loops back to find the next S1 record.

A problem arose with the BASIC programming technique used. The draft versions of this program tried saving the object-code bytes directly as binary in a string array. This caused “Out of Memory” or “Out of String Space” errors on both a 2M Macintosh and a 640K PC. The solution was to make the array an integer array and perform the integer-to-binary conversion on each byte as it is sent to the target part.

The one compromise made to accommodate both Macintosh and PC versions of BASIC is in lines 1500 and 1505. Use line 1500 and comment out line 1505 if the program is to be run on a Macintosh and, conversely, use line 1505 and comment out line 1500 if a PC is used.

After the COM port is opened, the code to be bootloaded is modified by adding the \$FF to the start of the string. \$FF synchronizes the bootloader in the MC68HC711E9 to 1200 baud. The entire string is simply sent to the COM port by PRINTing the string. This is possible since the string is actually queued in BASIC’s COM buffer, and the operating system takes care of sending the bytes out one at a time. The M68HC11 echoes the data received for verification. No automatic verification is provided, though the data is printed to the screen for manual verification.

Once the MCU has received this bootloaded code, the bootloader automatically jumps to it. The small bootloaded program in turn includes a jump to the EPROM programming routine in the boot ROM.

Refer to the previous explanation of the **EPROM PROGRAMMING UTILITY** for the following discussion. The host system sends the first byte to be programmed through the COM port to the SCI of the MCU. The SCI port on the MCU buffers one byte while receiving another byte, increasing the throughput of the EPROM programming operation by sending the second byte while the first is being programmed.

When the first byte has been programmed, the MCU reads the EPROM location and sends the result back to the host system. The host then compares what was actually programmed to what was originally sent. A message indicating which byte is being verified is displayed in the lower half of the screen. If there is an error, it is displayed at the top of the screen.

As soon as the first byte is verified, the third byte is sent. In the meantime, the MCU has already started programming the second byte. This process of verifying and queueing a byte continues until the host finishes sending data. If the programming is completely successful, no error messages will have been displayed at the top of the screen. Subroutines follow the end of the program to handle some of the repetitive tasks. These routines are short, and the commenting in the source code should be sufficient explanation.

MODIFICATIONS: This example programmed version 3.4 of the BUFFALO monitor into the EPROM of an MC68HC711E9; the changes to the BASIC program to download some other program are minor. The necessary changes are as follows:

1. In line 30, the length of the program to be downloaded must be assigned to the variable “CODESIZE%”.
2. Also in line 30, the starting address of the program is assigned to the variable “ADRSTART”.
3. In line 9570, the start address of the program is stored in the third and fourth items in that DATA statement in hexadecimal.
4. If any changes are made to the number of bytes in the boot code in the DATA statements in lines 9500–9580, then the new count must be set in the variable “BOOTCOUNT” in line 25.

OPERATION: Configure the EVBU for boot mode operation by putting a jumper at J3. Ensure that the trace command jumper at J7 is not installed because this would connect the 12-V programming voltage to the OC5 output of the MCU.

Connect the EVBU to its DC power supply. When it is time to program the MCU EPROM, turn on the 12-V programming power supply to the new circuitry in the wire-wrap area.

Connect the EVBU serial port to the appropriate serial port on the host system. For the Macintosh, this is the modem port with a modem cable. For the MS-DOS computer, it is connected to COM1 with a “straight through” or modem cable. Power up the host system and start the BASIC program. If the program has not been compiled, this is accomplished from within the appropriate BASIC compiler or interpreter. Power up the EVBU.

Answer the prompt for filename with either a [RETURN] to accept the default shown or by typing in a new filename and pressing [RETURN].

The program will inform the user that it is working on converting the file from S-records to binary. This process will take from 30 sec to a few minutes, depending on the computer.

A prompt reading, "Comm port open?" will appear at the end of the file conversion. This is the last chance to ensure that everything is properly configured on the EVBU. Pressing [RETURN] will send the bootcode to the target MC68HC711E9. The program then informs the user that the bootload code is being sent to the target, and the results of the echoing of this code are displayed on the screen.

Another prompt reading "Programming is ready to begin. Are you?" will appear. Turn on the 12-V programming power supply and press [RETURN] to start the actual programming of the target EPROM.

A count of the byte being verified will be continually updated on the screen as the programming progresses. Any failures will be flagged as they occur.

When programming is complete, a message will be displayed as well as a prompt requesting you to press [RETURN] to quit.

Turn off the 12-V programming power supply before turning off 5 V to the EVBU.

Listing 2. BASIC Program for Personal Computer

Sheet 1 of 3

```

1  '*****
2  '*'
3  '*   E9BUF.BAS - A PROGRAM TO DEMONSTRATE THE USE OF THE BOOT MODE
4  '*               ON THE HC11 BY PROGRAMMING AN MC68HC711E9 WITH
5  '*               BUFFALO 3.4
6  '*'
7  '*               REQUIRES THAT THE S-RECORDS FOR BUFFALO (BUF34.S19)
8  '*               BE AVAILABLE IN THE SAME DIRECTORY OR FOLDER
9  '*'
10 '*'               THIS PROGRAM HAS BEEN RUN BOTH ON A MS-DOS COMPUTER
11 '*'               USING QUICKBASIC 4.5 AND ON A MACINTOSH USING
12 '*'               QUICKBASIC 1.0.
13 '*'
14 '*'
15 '*****
25 HS = "0123456789ABCDEF"      'STRING TO USE FOR HEX CONVERSIONS
30 DEFINT B, I: CODESIZE% = 8192: ADDRSTART= 57344!
35 BOOTCOUNT = 25              'NUMBER OF BYTES IN BOOT CODE
40 DIM CODE$(CODESIZE%)        'BUFFALO 3.4 IS 8K BYTES LONG
45 BOOTCODE$ = ""              'INITIALIZE BOOTCODE$ TO NULL
49 REM ***** READ IN AND SAVE THE CODE TO BE BOOT LOADED *****
50 FOR I = 1 TO BOOTCOUNT      '# OF BYTES IN BOOT CODE
55 READ QS
60 AS = MID$(QS, 1, 1)
65 GOSUB 7000                   'CONVERTS HEX DIGIT TO DECIMAL
70 TEMP = 16 * X                'HANG ON TO UPPER DIGIT
75 AS = MID$(QS, 2, 1)
80 GOSUB 7000
85 TEMP = TEMP + X
90 BOOTCODE$ = BOOTCODE$ + CHR$(TEMP)      'BUILD BOOT CODE
95 NEXT I
96 REM ***** S-RECORD CONVERSION STARTS HERE *****
97 FILNAM$="BUF34.S19"          'DEFAULT FILE NAME FOR S-RECORDS
100 CLS
105 PRINT "Filename.ext of S-record file to be downloaded ("&FILNAM$&") ";
107 INPUT QS
110 IF QS<>" " THEN FILNAM$=QS
120 OPEN FILNAM$ FOR INPUT AS #1
130 PRINT : PRINT "Converting "&FILNAM$& " to binary..."
999 REM ***** SCANS FOR 'S1' RECORDS *****
1000 GOSUB 6000                 'GET 1 CHARACTER FROM INPUT FILE
1010 IF FLAG THEN 1250          'FLAG IS EOF FLAG FROM SUBROUTINE
1020 IF AS <> "S" THEN 1000
1022 GOSUB 6000
1024 IF AS <> "1" THEN 1000
1029 REM ***** S1 RECORD FOUND, NEXT 2 HEX DIGITS ARE THE BYTE COUNT *****
1030 GOSUB 6000
1040 GOSUB 7000                 'RETURNS DECIMAL IN X
1050 BYTECOUNT = 16 * X        'ADJUST FOR HIGH NIBBLE
1060 GOSUB 6000
1070 GOSUB 7000
1080 BYTECOUNT = BYTECOUNT + X 'ADD LOW NIBBLE
1090 BYTECOUNT = BYTECOUNT - 3 'ADJUST FOR ADDRESS + CHECKSUM
1099 REM ***** NEXT 4 HEX DIGITS BECOME THE STARTING ADDRESS FOR THE DATA *****
1100 GOSUB 6000                 'GET FIRST NIBBLE OF ADDRESS
1102 GOSUB 7000                 'CONVERT TO DECIMAL

```

```

1104 ADDRESS= 4096 * X
1106 GOSUB 6000 'GET NEXT NIBBLE
1108 GOSUB 7000
1110 ADDRESS= ADDRESS+ 256 * X
1112 GOSUB 6000
1114 GOSUB 7000
1116 ADDRESS= ADDRESS+ 16 * X
1118 GOSUB 6000
1120 GOSUB 7000
1122 ADDRESS= ADDRESS+ X
1124 ARRAYCNT = ADDRESS-ADRSTART 'INDEX INTO ARRAY
1129 REM ***** CONVERT THE DATA DIGITS TO BINARY AND SAVE IN THE ARRAY *****
1130 FOR I = 1 TO BYTECOUNT
1140 GOSUB 6000
1150 GOSUB 7000
1160 Y = 16 * X 'SAVE UPPER NIBBLE OF BYTE
1170 GOSUB 6000
1180 GOSUB 7000
1190 Y = Y + X 'ADD LOWER NIBBLE
1200 CODE%(ARRAYCNT) = Y 'SAVE BYTE IN ARRAY
1210 ARRAYCNT = ARRAYCNT + 1 'INCREMENT ARRAY INDEX
1220 NEXT I
1230 GOTO 1000
1250 CLOSE 1
1499 REM ***** DUMP BOOTLOAD CODE TO PART *****
1500 'OPEN "R",#2,"COM1:1200,N,8,1" 'Macintosh COM statement
1505 OPEN "COM1:1200,N,8,1,CD0,CS0,DS0,RS" FOR RANDOM AS #2 'DOS COM statement
1510 INPUT "Comm port open"; Q$
1512 WHILE LOC(2) > 0 'FLUSH INPUT BUFFER
1513 GOSUB 8020
1514 WEND
1515 PRINT : PRINT "Sending bootload code to target part..."
1520 A$ = CHR$(255) + BOOTCODE$ 'ADD HEX FF TO SET BAUD RATE ON TARGET HC11
1530 GOSUB 6500
1540 PRINT
1550 FOR I = 1 TO BOOTCOUNT '# OF BYTES IN BOOT CODE BEING ECHOED
1560 GOSUB 8000
1564 K=ASC(B$):GOSUB 8500
1565 PRINT "Character #"; I; " received = "; HX$
1570 NEXT I
1590 PRINT "Programming is ready to begin.": INPUT "Are you ready"; Q$
1595 CLS
1597 WHILE LOC(2) > 0 'FLUSH INPUT BUFFER
1598 GOSUB 8020
1599 WEND
1600 XMT = 0: RCV = 0 'POINTERS TO XMIT AND RECEIVE BYTES
1610 A$ = CHR$(CODE%(XMT))
1620 GOSUB 6500 'SEND FIRST BYTE
1625 FOR I = 1 TO CODESIZE% - 1 'ZERO BASED ARRAY 0 -> CODESIZE-1
1630 A$ = CHR$(CODE%(I)) 'SEND SECOND BYTE TO GET ONE IN QUEUE
1635 GOSUB 6500 'SEND IT
1640 GOSUB 8000 'GET BYTE FOR VERIFICATION
1650 RCV = I - 1
1660 LOCATE 10,1:PRINT "Verifying byte #"; I; " "
1664 IF CHR$(CODE%(RCV)) = B$ THEN 1670
1665 K=CODE%(RCV):GOSUB 8500
1666 LOCATE 1,1:PRINT "Byte #"; I; " ", " - Sent "; HX$;
1668 K=ASC(B$):GOSUB 8500
1669 PRINT " Received "; HX$;
1670 NEXT I
1680 GOSUB 8000 'GET BYTE FOR VERIFICATION
1690 RCV = CODESIZE% - 1
1700 LOCATE 10,1:PRINT "Verifying byte #"; CODESIZE%; " "
1710 IF CHR$(CODE%(RCV)) = B$ THEN 1720
1713 K=CODE%(RCV):GOSUB 8500
1714 LOCATE 1,1:PRINT "Byte #"; CODESIZE%; " ", " - Sent "; HX$;
1715 K=ASC(B$):GOSUB 8500
1716 PRINT " Received "; HX$;
1720 LOCATE 8, 1: PRINT : PRINT "Done!!!!"
4900 CLOSE
4910 INPUT "Press [RETURN] to quit...", Q$
5000 END

```



```

5900 '*****
5910 '* SUBROUTINE TO READ IN ONE BYTE FROM A DISK FILE
5930 '* RETURNS BYTE IN A$
5940 '*****
6000 FLAG = 0
6010 IF EOF(1) THEN FLAG = 1: RETURN
6020 A$ = INPUT$(1, #1)
6030 RETURN
6490 '*****
6492 '* SUBROUTINE TO SEND THE STRING IN A$ OUT TO THE DEVICE
6494 '* OPENED AS FILE #2.
6496 '*****
6500 PRINT #2, A$;
6510 RETURN
6590 '*****
6594 '* SUBROUTINE THAT CONVERTS THE HEX DIGIT IN A$ TO AN INTEGER
6596 '*****
7000 X = INSTR(H$, A$)
7010 IF X = 0 THEN FLAG = 1
7020 X = X - 1
7030 RETURN
7990 '*****
7992 '* SUBROUTINE TO READ IN ONE BYTE THROUGH THE COMM PORT OPENED
7994 '* AS FILE #2. WAITS INDEFINITELY FOR THE BYTE TO BE
7996 '* RECEIVED. SUBROUTINE WILL BE ABORTED BY ANY
7998 '* KEYBOARD INPUT. RETURNS BYTE IN B$. USES Q$.
7999 '*****
8000 WHILE LOC(2) = 0 'WAIT FOR COMM PORT INPUT
8005 Q$ = INKEY$: IF Q$ <> "" THEN 4900 'IF ANY KEY PRESSED, THEN ABORT
8010 WEND
8020 B$ = INPUT$(1, #2)
8030 RETURN
8490 '*****
8491 '* DECIMAL TO HEX CONVERSION
8492 '* INPUT: K - INTEGER TO BE CONVERTED
8493 '* OUTPUT: HX$ - TWO CHARACTER STRING WITH HEX CONVERSION
8494 '*****
8500 IF K > 255 THEN HX$="Too big":GOTO 8530
8510 HX$=MID$(H$,K16+1,1) 'UPPER NIBBLE
8520 HX$=HX$+MID$(H$, (K MOD 16)+1,1) 'LOWER NIBBLE
8530 RETURN
9499 '***** BOOT CODE *****
9500 DATA 86, 23 'LDAA #$23
9510 DATA B7, 10, 02 'STAA OPT2 make port C wire or
9520 DATA 86, FE 'LDAA #$FE
9530 DATA B7, 10, 03 'STAA PORTC light 1 LED on port C bit 0
9540 DATA C6, FF 'LDAB #$FF
9550 DATA F7, 10, 07 'STAB DDRC make port C outputs
9560 DATA CE, 0F, A0 'LDX #4000 2msec at 2MHz
9570 DATA 18, CE, E0, 00 'LDY #$E000 Start of BUFFALO 3.4
9580 DATA 7E, BF, 00 'JMP $BF00 EPROM routine start address
9590 '*****

```

COMMON BOOTSTRAP MODE PROBLEMS

It is not unusual for a user to encounter problems with bootstrap mode because it is new to many users. By knowing some of the common difficulties, the user can avoid them or at least recognize and quickly correct them.

Reset conditions vs. conditions as bootloaded program starts.

It is common to confuse the reset state of systems and control bits with the state of these systems and control bits when a bootloaded program in RAM starts. Between these times, the bootloader program is executed, which changes the states of some systems and control bits.

- The SCI system is initialized and turned on (Rx and Tx).
- The SCI system has control of the PD0 and PD1 pins.
- Port D outputs are configured for wire-OR operation.

- The stack pointer is initialized to the top of RAM.
- Time has passed (two or more SCI character times).
- Timer has advanced from its reset count value.

Users also forget that bootstrap mode is a special mode; thus privileged control bits are accessible, and write protection for some registers is not in effect. The bootstrap ROM is in the memory map. The DISR bit in the TEST1 control register is set, which disables resets from the COP and clock monitor systems.

Since bootstrap is a special mode, these conditions can be changed by software. The bus can even be switched from single-chip mode to expanded mode to gain access to external memories and peripherals.

Connecting RxD to V_{SS} does not cause the SCI to receive a break — To force an immediate jump to the start of EEPROM, the bootstrap firmware looks for the first received

Table 2. Summary of Boot-ROM-Related Features

MCU Part #	BOOT ROM Revision (@\$BFD1)	Mask Set I.D. (@\$BFD2,3)	MCU Type I.D. (@\$BFD4,5)	Security	Download Length	JMP on BRK or \$00 ¹	JMP to RAM ²	Default RAM Location	EPROM ³ PROGRAM Utility	UPLOAD ⁴ Utility	Notes
MC68HC11A0	—	—	Mask Set #	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11A1	—	—	Mask Set #	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11A8	—	—	Mask Set #	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68SEC11A8	—	—	Mask Set #	Yes	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11D3	\$00	Mask Set #	\$11D3	—	0–192	\$F000–ROM	—	\$0040–FF	—	—	6
MC68HC711D3	\$42 (B)	\$0000	\$71D3	—	0–192	\$F000–EPROM	—	\$0040–FF	Yes	Yes	6
MC68HC811E2	—	\$0000	\$E2E2	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68SEC811E2	—	—	\$E25C	Yes	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11E0	—	Mask Set #	\$E9E9	—	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68HC11E1	—	Mask Set #	\$E9E9	—	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68HC11E9	—	Mask Set #	\$E9E9	—	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68SEC11E9	—	Mask Set #	\$E95C	Yes	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68HC711E9	\$41 (A)	\$0000	\$71E9	—	0–512	\$B600	—	\$0000–1FF	Yes	Yes	6
MC68HC11F1	\$42 (B)	\$0000	\$F1F1	—	0–1024	\$FE00	—	\$0000–3FF	—	—	6, 8
MC68HC11K4	\$30 (0)	Mask Set #	\$044B	—	0–768	\$0D80	—	\$0080–37F	—	—	6, 8
MC68HC711K4	\$42 (B)	\$0000	\$744B	—	0–768	\$0D80	—	\$0080–37F	Yes	Yes	6, 8

NOTES:

1. By sending \$00 or a break as the first SCI character after reset in bootstrap mode, a jump (JMP) is executed to the address in this table rather than doing a download. Unless otherwise noted, this address is the start of EEPROM. Tying RxD to TxD and using a pullup resistor from TxD to VDD will cause the SCI to see a break as the first received character.
2. If \$55 is received as the first character after reset in bootstrap mode, a jump (JMP) is executed to the start of on-chip RAM rather than doing a download. This \$55 character must be sent at the default baud rate (7812 baud @ E = 2MHz).
- For devices with variable-length download, the same effect can be achieved by sending \$FF and no other SCI characters. After four SCI character times, the download terminates, and a jump (JMP) to the start of RAM is executed.
- The jump to RAM feature is only useful if the RAM was previously loaded with a meaningful program.
3. A callable utility subroutine is included in the bootstrap ROM of the indicated versions to program bytes of on-chip EPROM with data received via the SCI.
4. A callable utility subroutine is included in the bootstrap ROM of the indicated versions to upload contents of on-chip memory to a host computer via the SCI.
5. The complete listing for this bootstrap ROM may be found in the M68HC11RM/AD, M68HC11 Reference Manual.
6. The complete listing for this bootstrap ROM is included in this application note.
7. Due to the extra program space needed for EEPROM security on this device, there are no pseudo-vectors for SCI, SPI, PAIF, PAOVF, TOF, OC5F, or OC4F interrupts.
8. This bootloader extends the automatic software detection of baud rates to include 9600 baud at 2-MHz E-clock rate.

character to be \$00 (or break). The data reception logic in the SCI looks for a one-to-zero transition on the RxD pin to synchronize to the beginning of a receive character. If the RxD pin is tied to ground, no one-to-zero transition occurs. The SCI transmitter sends a break character when the bootloader firmware starts, and this break character can be fed back to the RxD pin to cause the jump to EEPROM. Since TxD is configured as an open-drain output, a pullup resistor is required.

An \$FF character is required before data is loaded into RAM — The initial character (usually \$FF) that sets the download baud rate is often forgotten.

Original M68HC11 versions required exactly 256 bytes to be downloaded to RAM — Even users that know about the 256 bytes of download data sometimes forget the initial \$FF that makes the total number of bytes required for the entire download operation equal to 256 + 1 or 257 bytes.

Variable-length download — When on-chip RAM surpassed 256 bytes, the time required to serially load this many characters became more significant. The variable-length download feature allows shorter programs to be loaded without sacrificing compatibility with earlier fixed-length download versions of the bootloader. The end of a download is indicated by an idle RxD line for at least four character times. If a personal computer is being used to send the download data to the MCU, there can be problems keeping characters close enough together to avoid tripping the end-of-download detect mechanism. Using 1200 as the baud rate rather than the faster default rate may help this problem.

Assemblers often produce S-record encoded programs which must be converted to binary before bootloading them to the MCU. The process of reading S-record data from a file and translating it to binary can be slow, depending on the personal computer and the programming language used for the translation. One strategy that can be used to overcome this problem is to translate the file into binary and store it into a RAM array before starting the download process. Data can then be read and downloaded without the translation or file-read delays.

The end-of-download mechanism goes into effect when the initial \$FF is received to set the baud rate. Any amount of time may pass between reset and when the \$FF is sent to start the download process.

EPROM/OTP versions of M68HC11 have an EPROM emulation mode — The conditions that configure the MCU for EPROM emulation mode are essentially the same as those for resetting the MCU in bootstrap mode. While $\overline{\text{RESET}}$ is low and mode select pins are configured for bootstrap mode (low), the MCU is configured for EPROM emulation mode.

The port pins that are used for EPROM data I/O lines may be inputs or outputs, depending on the pin that is emulating the

EPROM output enable pin ($\overline{\text{OE}}$). To make these data pins appear as high-impedance inputs as they would on a non-EPROM part in reset, connect the PB7/($\overline{\text{OE}}$) pin to a pull-up resistor.

Bootloading a program to perform a ROM checksum — The bootloader ROM must be turned off before performing the checksum program. To remove the boot ROM from the memory map, clear the RBOOT bit in the HPRIO register. This is normally a write-protected bit that is zero, but in bootstrap mode it is reset to one and can be written. If the boot ROM is not disabled, the checksum routine will read the contents of the boot ROM rather than the user's mask ROM or EPROM at the same addresses.

Inherent delays caused by double buffering of SCI data — This problem is troublesome in cases where one MCU is bootloading to another MCU.

Because of transmitter double buffering, there may be one character in the serial shifter as a new character is written into the transmit data register. In cases such as downloading in which this two-character pipeline is kept full, a two-character time delay occurs between when a character is written to the transmit data register and when that character finishes transmitting. A little more than one more character time delay occurs between the target MCU receiving the character and echoing it back. If the master MCU waits for the echo of each downloaded character before sending the next one, the download process takes about twice as long as it would if transmission is treated as a separate process or if verify data is ignored.

BOOT ROM VARIATIONS

Different versions of the M68HC11 have different versions of the bootstrap ROM program. Table 2 summarizes the features of the boot ROMs in 16 members of the M68HC11 Family.

The boot ROMs for the MC68HC11F1, the MC68HC711K4, and the MC68HC11K4 allow additional choices of baud rates for bootloader communications. For the three new baud rates, the first character used to determine the baud rate is not \$FF as it was in earlier M68HC11s. The intercharacter delay that terminates the variable-length download is also different for these new baud rates. Table 3 shows the synchronization characters, delay times, and baud rates as they relate to E-clock frequency.

COMMENTED BOOT ROM LISTINGS

Listings 3-8 contain complete commented listings of the boot ROM programs in six specific versions of the M68HC11. Other versions can be found in appendix B of the M68HC11RM/AD, *M68HC11 Reference Manual*.

Table 3. Bootloader Baud Rates

Sync Character	Timeout Delay	Baud Rates at E-clock =					
		2 MHz	2.1 MHz	3 MHz	3.15 MHz	4 MHz	4.2 MHz
\$FF	4 Characters	7812	8192	11,718	12,288	15,624	16,838
\$FF	4 Characters	1200	1260	1800	1890	2400	2520
\$F0	4.9 Characters	9600	10,080	14,400	15,120	19,200	20,160
\$FD	17.3 Characters	5208	5461	7812	8192	10,416	10,922
\$FD	13 Characters	3906	4096	5859	6144	7812	8192

```

1      *****
2      * BOOTLOADER FIRMWARE FOR 68HC711E9 - 21 Aug 89
3      *****
4      * Features of this bootloader are...
5      *
6      * Auto baud select between 7812.5 and 1200 (8 MHz)
7      * 0 - 512 byte variable length download
8      * Jump to EEPROM at $B600 if 1st download byte = $00
9      * PROGRAM - Utility subroutine to program EPROM
10     * UPLOAD - Utility subroutine to dump memory to host
11     * Mask I.D. at $BFD4 = $71E9
12     *****
13     * Revision A -
14     *
15     * Fixed bug in PROGRAM routine where the first byte
16     * programmed into the EPROM was not transmitted for
17     * verify.
18     * Also added to PROGRAM routine a skip of bytes
19     * which were already programmed to the value desired.
20     *
21     * This new version allows variable length download
22     * by quitting reception of characters when an idle
23     * of at least four character times occurs
24     *
25     *****
26
27     * EQUATES FOR USE WITH INDEX OFFSET = $1000
28     *
29     0008      PORTD          EQU      $08
30     000E      TCNT          EQU      $0E
31     0016      TOC1          EQU      $16
32     0023      TFLG1         EQU      $23
33     * BIT EQUATES FOR TFLG1
34     0080      OC1F          EQU      $80
35     *
36     0028      SPCR          EQU      $28          (FOR DWOM BIT)
37     002B      BAUD          EQU      $2B
38     002D      SCCR2         EQU      $2D
39     002E      SCSR          EQU      $2E
40     002F      SCDAT         EQU      $2F
41     003B      PPROG         EQU      $3B
42     * BIT EQUATES FOR PPROG
43     0020      ELAT          EQU      $20
44     0001      EPGM          EQU      $01
45     *
46
47     * MEMORY CONFIGURATION EQUATES
48     *
49     B600      EEPMSTR        EQU      $B600      Start of EEPROM
50     B7FF      EEPMEND        EQU      $B7FF      End of EEPROM
51     *
52     D000      EPRMSTR        EQU      $D000      Start of EPROM
53     FFFF      EPRMEND        EQU      $FFFF      End of EPROM
54     *
55     0000      RAMSTR         EQU      $0000
56     01FF      RAMEND         EQU      $01FF
57
58     * DELAY CONSTANTS
59     *
60     0DB0      DELAYS         EQU      3504      Delay at slow baud
61     021B      DELAYF         EQU      539       Delay at fast baud
62     *
63     1068      PROGDEL        EQU      4200      2 ms programming delay
64     *                               At 2.1 MHz
65

```

```

66 *****
67 BF00          ORG      $BF00
68 *****
69
70 * Next two instructions provide a predictable place
71 * to call PROGRAM and UPLOAD even if the routines
72 * change size in future versions.
73 *
74 BF00  7EBF13  PROGRAM      JMP      PRGROUT      EPROM programming utility
75 BF03          UPLOAD      EQU      *            Upload utility
76
77 *****
78 * UPLOAD - Utility subroutine to send data from
79 * inside the MCU to the host via the SCI interface.
80 * Prior to calling UPLOAD set baud rate, turn on SCI
81 * and set Y=first address to upload.
82 * Bootloader leaves baud set, SCI enabled, and
83 * Y pointing at EPROM start ($D000) so these default
84 * values do not have to be changed typically.
85 * Consecutive locations are sent via SCI in an
86 * infinite loop. Reset stops the upload process.
87 *****
88 BF03  CE1000          LDX      #$1000          Point to internal registers
89 BF06  18A600  UPLOOP    LDAA     0,Y          Read byte
90 BF09  1F2E80FC      BRCLR    SCSR,X $80 *    Wait for TDRE
91 BF0D  A72F          STAA     SCDAT,X        Send it
92 BF0F  1808          INY
93 BF11  20F3          BRA      UPLOOP        Next...
94
95 *****
96 * PROGRAM - Utility subroutine to program EPROM.
97 * Prior to calling PROGRAM set baud rate, turn on SCI
98 * set X=2ms prog delay constant, and set Y=first
99 * address to program. SP must point to RAM.
100 * Bootloader leaves baud set, SCI enabled, X=4200
101 * and Y pointing at EPROM start ($D000) so these
102 * default values don't have to be changed typically.
103 * Delay constant in X should be equivalent to 2 ms
104 * at 2.1 MHz X=4200; at 1 MHz X=2000.
105 * An external voltage source is required for EPROM
106 * programming.
107 * This routine uses 2 bytes of stack space
108 * Routine does not return. Reset to exit.
109 *****
110 BF13          PRGROUT    EQU      *
111 BF13  3C          PSHX          Save program delay constant
112 BF14  CE1000      LDX      #$1000        Point to internal registers
113 BF17
114 * Send $FF to indicate ready for program data
115
116 BF17  1F2E80FC      BRCLR    SCSR,X $80 *    Wait for TDRE
117 BF1B  86FF          LDAA     #$FF
118 BF1D  A72F          STAA     SCDAT,X
119
120 BF1F          WAIT1     EQU      *
121 BF1F  1F2E20FC      BRCLR    SCSR,X $20 *    Wait for RDRF
122 BF23  E62F          LDAB     SCDAT,X        Get received byte
123 BF25  18E100      CMPB     $0,Y          See if already programmed
124 BF28  271D          BEQ      DONEIT        If so, skip prog cycle
125 BF2A  8620          LDAA     #ELAT        Put EPROM in prog mode
126 BF2C  A73B          STAA     PPROG,X
127 BF2E  18E700      STAB     0,Y          Write the data
128 BF31  8621          LDAA     #ELAT+EPGM
129 BF33  A73B          STAA     PPROG,X        Turn on prog voltage
130 BF35  32          PULA          Pull delay constant
131 BF36  33          PULB          into D-reg
132 BF37  37          PSHB          But also keep delay
133 BF38  36          PSHA          keep delay on stack
134 BF39  E30E          ADDD     TCNT,X        Delay const + present TCNT
135 BF3B  ED16          STD      TOC1,X        Schedule OC1 (2ms delay)
136 BF3D  8680          LDAA     #OC1F
137 BF3F  A723          STAA     TFLG1,X        Clear any previous flag
138
139 BF41  1F2380FC      BRCLR    TFLG1,X OC1F *    Wait for delay to expire
140 BF45  6F3B          CLR      PPROG,X        Turn off prog voltage
141
142 BF47          *
          DONEIT      EQU      *

```

```

143 BF47 1F2E80FC      BRCLR   SCSR,X $80 *      Wait for TDRE
144 BF4B 18A600      LDAA    $0,Y              Read from EPROM and...
145 BF4E A72F        STAA    SCDAT,X          Xmit for verify
146 BF50 1808        INY                      Point at next location
147 BF52 20CB        BRA     WAIT1           Back to top for next
148                                     * Loops indefinitely as long as more data sent.
149
150 *****
151 * Main bootloader starts here
152 *****
153 * RESET vector points to here
154
155 BF54      BEGIN      EQU      *
156 BF54      8E01FF      LDS      #RAMEND      Initialize stack pntr
157 BF57      CE1000      LDX      #$1000      Point at internal regs
158 BF5A      1C2820      BSET     SPCR,X $20      Select port D wire-OR mode
159 BF5D      CCA20C      LDD      #$A20C      BAUD in A, SCCR2 in B
160 BF60      A72B        STAA     BAUD,X      SCPx = ÷4, SCRx = ÷4
161                                     * Writing 1 to MSB of BAUD resets count chain
162 BF62      E72D        STAB     SCCR2,X      Rx and Tx Enabled
163 BF64      CC021B      LDD      #DELAYF      Delay for fast baud rate
164 BF67      ED16        STD      TOC1,X      Set as default delay
165
166                                     * Send BREAK to signal ready for download
167 BF69      1C2D01      BSET     SCCR2,X $01      Set send break bit
168 BF6C      1E0801FC    BRSET    PORTD,X $01 *      Wait for RxD pin to go low
169 BF70      1D2D01      BCLR     SCCR2,X $01      Clear send break bit
170 BF73
171 BF73      1F2E20FC    BRCLR     SCSR,X $20 *      Wait for RDRF
172 BF77      A62F        LDAA     SCDAT,X      Read data
173                                     * Data will be $00 if BREAK OR $00 received
174 BF79      2603        BNE      NOTZERO      Bypass JMP if not 0
175 BF7B      7EB600      JMP      EEPROMSTR      Jump to EEPROM if it was 0
176 BF7E      NOTZERO     EQU      *
177 BF7E      81FF        CMPA     #$FF          $FF will be seen as $FF
178 BF80      2708        BEQ      BAUDOK        If baud was correct
179                                     * Or else change to ÷104 (÷13 & ÷8) 1200 @ 2MHZ
180 BF82      1C2B33      BSET     BAUD,X $33      Works because $22 -> $33
181 BF85      CC0DB0      LDD      #DELAYS      And switch to slower...
182 BF88      ED16        STD      TOC1,X      delay constant
183 BF8A      BAUDOK      EQU      *
184 BF8A      18CE0000    LDY      #RAMSTR      Point at start of RAM
185
186 BF8E      WAIT        EQU      *
187 BF8E      EC16        LDD      TOC1,X      Move delay constant to D
188 BF90      WTLOOP      EQU      *
189 BF90      1E2E2007    BRSET    SCSR,X $20 NEWONE      Exit loop if RDRF set
190 BF94      8F          XGDX          Swap delay count to X
191 BF95      09          DEX           Decrement count
192 BF96      8F          XGDX          Swap back to D
193 BF97      26F7        BNE      WTLOOP      Loop if not timed out
194 BF99      200F        BRA      STAR       Quit download on timeout
195
196 BF9B      NEWONE      EQU      *
197 BF9B      A62F        LDAA     SCDAT,X      Get received data
198 BF9D      18A700      STAA     $00,Y      Store to next RAM location
199 BFA0      A72F        STAA     SCDAT,X      Transmit it for handshake
200 BFA2      1808        INY                      Point at next RAM location
201 BFA4      188C0200    CPY      #RAMEND+1      See if past end
202 BFA8      26E4        BNE      WAIT       If not, Get another
203
204 BFAA      STAR        EQU      *
205 BFAA      CE1068      LDX      #PROGDEL      Init X with programming delay
206 BFAD      18CED000    LDY      #EPRMSTR      Init Y with EPROM start addr
207 BFB1      7E0000      JMP      RAMSTR      ** EXIT to start of RAM **
208 BFB4
209 *****
210 * Block fill unused bytes with zeros
211
212 BFB4      000000000000      BSZ      $BFD1-*
213      000000000000
214      000000000000
215      000000000000
216      000000000000
217
218 *****
219 * Boot ROM revision level in ASCII
220 * (ORG $BFD1)
221 FCC      "A"

```

```

218 *****
219 * Mask set I.D. ($0000 FOR EPROM PARTS)
220 * (ORG $BFD2)
221 BFD2 0000 FDB $0000
222 *****
223 * '711E9 I.D. - Can be used to determine MCU type
224 * (ORG $BFD4)
225 BFD4 71E9 FDB $71E9
226 *****
227 * VECTORS - point to RAM for pseudo-vector JUMPS
228
229
230 BFD6 00C4 FDB $100-60 SCI
231 BFD8 00C7 FDB $100-57 SPI
232 BFDA 00CA FDB $100-54 PULSE ACCUM INPUT EDGE
233 BFDC 00CD FDB $100-51 PULSE ACCUM OVERFLOW
234 BFDE 00D0 FDB $100-48 TIMER OVERFLOW
235 BFE0 00D3 FDB $100-45 TIMER OUTPUT COMPARE 5
236 BFE2 00D6 FDB $100-42 TIMER OUTPUT COMPARE 4
237 BFE4 00D9 FDB $100-39 TIMER OUTPUT COMPARE 3
238 BFE6 00DC FDB $100-36 TIMER OUTPUT COMPARE 2
239 BFE8 00DF FDB $100-33 TIMER OUTPUT COMPARE 1
240 BFEA 00E2 FDB $100-30 TIMER INPUT CAPTURE 3
241 BFEC 00E5 FDB $100-27 TIMER INPUT CAPTURE 2
242 BFEE 00E8 FDB $100-24 TIMER INPUT CAPTURE 1
243 BFF0 00EB FDB $100-21 REAL TIME INT
244 BFF2 00EE FDB $100-18 IRQ
245 BFF4 00F1 FDB $100-15 XIRQ
246 BFF6 00F4 FDB $100-12 SWI
247 BFF8 00F7 FDB $100-9 ILLEGAL OP-CODE
248 BFFA 00FA FDB $100-6 COP FAIL
249 BFFC 00FD FDB $100-3 CLOCK MONITOR
250 BFFE BF54 FDB BEGIN RESET
251 C000 END

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUD	002B	*00037	00160	00180
BAUDOK	BF8A	*00183	00178	
BEGIN	BF54	*00155	00250	
DELAYF	021B	*00061	00163	
DELAYS	0DB0	*00060	00181	
DONEIT	BF47	*00142	00124	
EEPMEND	B7FF	*00050		
EEPMSTR	B600	*00049	00175	
ELAT	0020	*00043	00125	00128
EPGM	0001	*00044	00128	
EPRMEND	FFFF	*00053		
EPRMSTR	D000	*00052	00206	
NEWONE	BF9B	*00196	00189	
NOTZERO	BF7E	*00176	00174	
OC1F	0080	*00034	00136	00139
PORTD	0008	*00029	00168	
PPROG	003B	*00041	00126	00129 00140
PRGROUT	BF13	*00110	00074	
PROGDEL	1068	*00063	00205	
PROGRAM	BF00	*00074		
RAMEND	01FF	*00056	00156	00201
RAMSTR	0000	*00055	00184	00207
SCCR2	002D	*00038	00162	00167 00169
SCDAT	002F	*00040	00091	00118 00122 00145 00172 00197 00199
SCSR	002E	*00039	00090	00116 00121 00143 00171 00189
SPCR	0028	*00036	00158	
STAR	BFAA	*00204	00194	
TCNT	000E	*00030	00134	
TFLG1	0023	*00032	00137	00139
TOC1	0016	*00031	00135	00164 00182 00187
UPLOAD	BF03	*00075		
UPLOOP	BF06	*00089	00093	
WAIT	BF8E	*00186	00202	
WAIT1	BF1F	*00120	00147	
WTLOOP	BF90	*00188	00193	

Errors: None

Labels: 35

Last Program Address: \$BFFF

Last Storage Address: \$0000

Program Bytes: \$0100 256

Storage Bytes: \$0000 0

```

1      *****
2      * BOOTLOADER FIRMWARE FOR MC68HC11D3 - 13 Apr 89
3      *****
4      * Features of this bootloader are...
5      *
6      * Auto baud select between 7812 and 1200 (E = 2 MHz).
7      * 0 - 192 byte variable length download:
8      *   reception of characters quits when an idle of at
9      *   least four character times occurs.
10     * Jump to EPROM at $F000 if first download byte = $00.
11     * PROGRAM - Utility subroutine to program EPROM.
12     * UPLOAD - Utility subroutine to dump memory to host.
13     * Part I.D. at $BFD4 is $71D3.
14     *****
15
16     * Equates (registers in direct space)
17     *
18     0008     PORTD      EQU     $08
19     0009     DDRD      EQU     $09
20     000E     TCNT      EQU     $0E
21     0016     TOC1      EQU     $16
22     0023     TFLG1     EQU     $23
23     * Bit equates for TFLG1
24     0080     OC1F      EQU     $80
25     *
26     0028     SPCR      EQU     $28             (For DWOM bit)
27     002B     BAUD      EQU     $2B
28     002C     SCCR1     EQU     $2C
29     002D     SCCR2     EQU     $2D
30     002E     SCSR      EQU     $2E
31     002F     SCDAT     EQU     $2F
32     003B     PPROG     EQU     $3B
33     * Bit equates for PPROG
34     0020     LAT       EQU     $20
35     0001     EPGM      EQU     $01
36     *
37     003E     TEST1     EQU     $3E
38     003F     CONFIG    EQU     $3F
39     *
40
41     * Memory configuration equates
42     *
43     F000     ROMSTR     EQU     $F000             Start of ROM
44     FFFF     ROMEND     EQU     $FFFF             End of ROM
45     *
46     0040     RAMSTR     EQU     $0040             Start of RAM
47     00FF     RAMEND     EQU     $00FF             End of RAM
48
49     * Delay constants
50     *
51     0DB0     DELAYS     EQU     3504             Delay at slow baud
52     021B     DELAYF     EQU     539             Delay at fast baud
53     *
54
55     *****
56     BF40     ORG        $BF40
57     *****
58     * Main bootloader starts here
59     *****
60     * RESET vector points to here
61
62     BF40     BEGIN      EQU     *
63     BF40     8E00FF     LDS     #RAMEND             Initialize stack pntr
64     BF43     142820     BSET    SPCR $20             Select port D wire-OR mode
65     BF46     CCA20C     LDD     #A20C             Baud in A, SCCR2 in B
66     BF49     972B       STAA    BAUD             SCPx = /4, SCRx = /4
67
68     BF4B     D72D       * Writing 1 to MSB of BAUD resets count chain
69     BF4D     CC021B     STAB    SCCR2             Rx and Tx enabled
70     BF50     DD16       LDD     #DELAYF           Delay for fast baud rate
71                                     STD     TOC1           Set as default delay
72
73     BF52     142D01     * Send BREAK to signal ready for download
74     BF55     120801FC   BSET    SCCR2 $01         Set send break bit
75     BF59     152D01     BRESET PORTD $01 *       Wait for RxD pin to go low
76                                     BCLR    SCCR2 $01         Clear send break bit
77
78     BF5C     132E20FC   BRCLR   SCSR $20 *       Wait for RDRF
79     BF60     962F       LDAA    SCDAT             Read data
80     BF62     2603       * Data will be $00 if BREAK or $00 received
81                                     BNE     NOTZERO           Bypass jump if not $00

```



```

81 BF64 7EF000      JMP      ROMSTR      Jump to ROM if it was $00
82 BF67            EQU      *
83 BF67 81FF        NOTZERO    EQU      *
84 BF69 2708        CMPA      #$FF      $FF will be seen as $FF...
85                BEQ      BAUDOK      if baud was correct
86 BF6B 142B33      * Or else change to /104 (/13 & /8) 1200 @ 2MHz
87 BF6E CC0DB0      BSET      BAUD $33    Works because $22 -> $33
88 BF71 DD16        LDD      #DELAYS      And switch to slower...
89 BF73            STD      TOC1      delay constant
90 BF73 18CE0040    BAUDOK      EQU      *
91                LDY      #RAMSTR      Point to start of RAM
92 BF77            WAIT          EQU      *
93 BF77 DE16        LDX      TOC1      Move delay constant to X
94 BF79            WTLOOP        EQU      *
95 BF79 122E2009    BRSET     SCSR $20 NEWONE Exit loop if RDRF set
96 BF7D 09          DEX          Decrement count
97 BF7E 01          NOP          Kill...
98 BF7F 01          NOP          ...seven cycles.....
99 BF80 2100        BRN      *+2      ..to match original program
100 BF82 26F5       BNE      WTLOOP    Loop if not timed out
101 BF84 200F       BRA      STAR      Quit download on timeout
102
103 BF86            NEWONE       EQU      *
104 BF86 962F       LDAA      SCDAT      Get received data
105 BF88 18A700     STAA      $00,Y      Store to next RAM location
106 BF8B 972F       STAA      SCDAT      Transmit it for handshake
107 BF8D 1808       INY          Point to next RAM location
108 BF8F 188C0100   CPY      #RAMEND+1    See if past end
109 BF93 26E2       BNE      WAIT      If not, get another
110
111 BF95            STAR          EQU      *
112 BF95 7E0040     JMP      RAMSTR      ** Exit to start of RAM **
113 *****
114 * Block fill unused bytes with zero
115
116 BF98 000000000000 BSZ      $BFD1-*
117 000000000000
118 000000000000
119 000000000000
120 000000000000
121 000000000000
122 000000000000
123 000000000000
124 000000000000
125 000000000000
126 000000
127
128 *****
129 * Boot ROM revision level in ASCII
130 * (ORG $BFD1)
131 FCB 0
132 *****
133 * Mask set I.D. -
134 * (ORG $BFD2)
135 FDB $0000
136 *****
137 * 11D3 I.D. - can be used to determine MCU type
138 * (ORG $BFD4)
139 FDB $11D3
140 *****
141 * VECTORS - point to RAM for pseudo-vector JUMPs
142
143 BFD6 00C4        FDB      $100-60      SCI
144 BFD8 00C7        FDB      $100-57      SPI
145 BFDA 00CA        FDB      $100-54      PULSE ACCUM INPUT EDGE
146 BFDC 00CD        FDB      $100-51      PULSE ACCUM OVERFLOW
147 BFDE 00D0        FDB      $100-48      TIMER OVERFLOW
148 BFE0 00D3        FDB      $100-45      TIMER OUTPUT COMPARE 5
149 BFE2 00D6        FDB      $100-42      TIMER OUTPUT COMPARE 4
150 BFE4 00D9        FDB      $100-39      TIMER OUTPUT COMPARE 3
151 BFE6 00DC        FDB      $100-36      TIMER OUTPUT COMPARE 2
152 BFE8 00DF        FDB      $100-33      TIMER OUTPUT COMPARE 1
153 BFEA 00E2        FDB      $100-30      TIMER INPUT CAPTURE 3
154 BFEC 00E5        FDB      $100-27      TIMER INPUT CAPTURE 2
155 BFEE 00E8        FDB      $100-24      TIMER INPUT CAPTURE 1
156 BFF0 00EB        FDB      $100-21      REAL TIME INT
157 BFF2 00EE        FDB      $100-18      IRQ
158 BFF4 00F1        FDB      $100-15      XIRQ
159 BFF6 00F4        FDB      $100-12      SWI
160 BFF8 00F7        FDB      $100-9       ILLEGAL OP-CODE
161 BFFA 00FA        FDB      $100-6       COP FAIL

```

Listing 4. MC68HC11D3 Bootloader ROM

Sheet 3 of 3

```

152 BFFC 00FD          FDB    $100-3      CLOCK MONITOR
153 BFFE BF40          FDB    BEGIN        RESET
154 C000              END

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUD	002B	*00027	00066	00086
BAUDOK	BF73	*00089	00084	
BEGIN	BF40	*00062	00153	
CONFIG	003F	*00038		
DDRD	0009	*00019		
DELAYF	021B	*00052	00069	
DELAYS	0DB0	*00051	00087	
EPGM	0001	*00035		
LAT	0020	*00034		
NEWONE	BF86	*00103	00095	
NOTZERO	BF67	*00082	00080	
OC1F	0080	*00024		
PORTD	0008	*00018	00074	
PProg	003B	*00032		
RAMEND	00FF	*00047	00063	00108
RAMSTR	0040	*00046	00090	00112
ROMEND	FFFF	*00044		
ROMSTR	F000	*00043	00081	
SCCR1	002C	*00028		
SCCR2	002D	*00029	00068	00073 00075
SCDAT	002F	*00031	00078	00104 00106
SCSR	002E	*00030	00077	00095
SPCR	0028	*00026	00064	
STAR	BF95	*00111	00101	
TCNT	000E	*00020		
TEST1	003E	*00037		
TFLG1	0023	*00022		
TOC1	0016	*00021	00070	00088 00093
WAIT	BF77	*00092	00109	
WTLOOP	BF79	*00094	00100	

Errors: None

Labels: 30

Last Program Address: \$BFFF

Last Storage Address: \$0000

Program Bytes: \$00C0 192

Storage Bytes: \$0000 0

```

1 *****
2 * BOOTLOADER FIRMWARE FOR MC68HC711D3 - 28 Aug 90
3 *****
4 * Features of this bootloader are...
5 *
6 * Auto baud select between 7812 and 1200 (E = 2 MHz).
7 * 0 - 192 byte variable length download:
8 *   reception of characters quits when an idle of at
9 *   least four character times occurs.
10 * Jump to EPROM at $F000 if first download byte = $00.
11 * PROGRAM - Utility subroutine to program EPROM.
12 * UPLOAD - Utility subroutine to dump memory to host.
13 * Part I.D. at $BFD4 is $71D3.
14 *****
15 * Revision B -
16 *
17 * Changed program delay to 2 mSec at E = 2 MHz.
18 *****
19 * Revision A -
20 *
21 * Fixed bug in PROGRAM routine where the first byte
22 * programmed into the EPROM was not transmitted for
23 * verify.
24 * Also added to PROGRAM routine a skip of bytes
25 * which were already programmed to the value desired.
26 *****
27
28 * Equates (registers in direct space)
29 *
30 0008 PORTD EQU $08
31 0009 DDRD EQU $09
32 000E TCNT EQU $0E
33 0016 TOC1 EQU $16
34 0023 TFLG1 EQU $23
35 * Bit equates for TFLG1
36 0080 OC1F EQU $80
37 *
38 0028 SPCR EQU $28 (For DWOM bit)
39 002B BAUD EQU $2B
40 002C SCCR1 EQU $2C
41 002D SCCR2 EQU $2D
42 002E SCSR EQU $2E
43 002F SCDAT EQU $2F
44 003B PPROG EQU $3B
45 * Bit equates for PPROG
46 0020 LAT EQU $20
47 0001 EPGM EQU $01
48 *
49 003E TEST1 EQU $3E
50 003F CONFIG EQU $3F
51 *
52
53 * Memory configuration equates
54 *
55 F000 EPRMSTR EQU $F000 Start of EPROM
56 FFFF EPRMEND EQU $FFFF End of EPROM
57 *
58 0040 RAMSTR EQU $0040 Start of RAM
59 00FF RAMEND EQU $00FF End of RAM
60
61 * Delay constants
62 *
63 0DB0 DELAYS EQU 3504 Delay at slow baud
64 021B DELAYF EQU 539 Delay at fast baud
65 *
66 1068 PROGDEL EQU 4200 2 mSec programming delay
67 *
68
69 *****
70 BF00 ORG $BF00
71 *****
72
73 * Next two instructions provide a predictable place
74 * to call PROGRAM and UPLOAD even if the routines
75 * change size in future versions.
76 *
77 BF00 7EBF10 PROGRAM JMP PRGROUT EPROM programming utility
78 BF03 UPLOAD EQU * Upload utility
79

```

```

80      *****
81      * UPLOAD - Utility subroutine to send data from
82      * inside the MCU to the host via the SCI interface.
83      * Prior to calling UPLOAD set baud rate, turn on SCI
84      * and set Y=first address to upload.
85      * Bootloader leaves baud set, SCI enabled, and
86      * Y pointing at EPROM start ($F000) so these default
87      * values do not have to be changed typically.
88      * Consecutive locations are sent via SCI in an
89      * infinite loop. Reset stops the upload process.
90      *****
91      BF03      UPLOOP      EQU      *
92      BF03 18A600      LDAA      0,Y      Read byte
93      BF06 132E80FC      BRCLR     SCSR $80 *      Wait for TDRE
94      BF0A 972F      STAA      SCDAT      Send it
95      BF0C 1808      INY
96      BF0E 20F3      BRA      UPLOOP      Next....
97
98      *****
99      * PROGRAM - Utility subroutine to program EPROM.
100     * Prior to calling PROGRAM set baud rate, turn on SCI
101     * set X=2ms prog delay constant, and set Y=first
102     * address to program. SP must point to RAM.
103     * Bootloader leaves baud set, SCI enabled, X=4200
104     * and Y pointing at EPROM start ($F000) so these default
105     * values do not have to be changed typically.
106     * Delay constant in X should be equivalent to 2 ms
107     * at 2.1 MHz X=4200; at 1 MHz X=2000.
108     * An external voltage source is required for EPROM
109     * programming.
110     * This routine uses 2 bytes of stack space.
111     * Routine does not return. Reset to exit.
112     *****
113     BF10      PRGROUT      EQU      *
114     * Send $FF to indicate ready for program data
115     BF10 132E80FC      BRCLR     SCSR $80 *      Wait for TDRE
116     BF14 86FF      LDAA      #$FF
117     BF16 972F      STAA      SCDAT
118
119     BF18      WAIT1      EQU      *
120     BF18 132E20FC      BRCLR     SCSR $20 *      Wait for RDRF
121     BF1C D62F      LDAB      SCDAT      Get received byte
122     BF1E 18E100      CMPB      $0,Y      See if already programmed
123     BF21 271D      BEQ      DONEIT      If so, skip prog cycle
124     BF23 8620      LDAA      #LAT      Put EPROM in prog mode
125     BF25 973B      STAA      PPROG
126     BF27 18E700      STAB      0,Y      Write data
127     BF2A 8621      LDAA      #LAT+EPGM
128     BF2C 973B      STAA      PPROG      Turn on prog voltage
129     BF2E 3C      PSHX      Save delay on stack
130     BF2F 8F      XGDX      Put delay into D-reg
131     BF30 38      PULX      Save delay in X
132     BF31 D30E      ADDD      TCNT      Delay const + present TCNT
133     BF33 DD16      STD      TOC1      Schedule OC1 (prog delay)
134     BF35 8680      LDAA      #OC1F
135     BF37 9723      STAA      TFLG1      Clear any previous flag
136
137     BF39 132380FC      BRCLR     TFLG1 OC1F *      Wait for delay to expire
138     BF3D 7F003B      CLR      PPROG      Turn off prog voltage
139     BF40      DONEIT      EQU      *
140     BF40 132E80FC      BRCLR     SCSR $80 *      Wait for TDRE
141     BF44 18A600      LDAA      $0,Y      Read from EPROM and...
142     BF47 972F      STAA      SCDAT      Xmit for verify
143     BF49 1808      INY      Point to next location
144     BF4B 20CB      BRA      WAIT1      Back to top for next
145     * Loops indefinitely as long as more data sent.
146
147     *****
148     * Main bootloader starts here
149     *****
150     * RESET vector points to here
151
152     BF4D      BEGIN      EQU      *
153     BF4D 8E00FF      LDS      #RAMEND      Initialize stack pntr
154     BF50 142820      BSET      SPCR $20      Select port D wire-OR mode
155     BF53 CCA20C      LDD      #$A20C      Baud in A, SCCR2 in B
156     BF56 972B      STAA      BAUD      SCPx = /4, SCRx = /4
157     * Writing 1 to MSB of BAUD resets count chain
158     BF58 D72D      STAB      SCCR2      Rx and Tx enabled
159     BF5A CC021B      LDD      #DELAYF      Delay for fast baud rate

```

```

160 BF5D DD16          STD      TOC1          Set as default delay
161
162          * Send BREAK to signal ready for download
163 BF5F 142D01        BSET     SCCR2 $01      Set send break bit
164 BF62 120801FC      BRSET    PORTD $01 *    Wait for RxD pin to go low
165 BF66 152D01        BCLR     SCCR2 $01      Clear send break bit
166
167 BF69 132E20FC      BRCLR    SCSR $20 *    Wait for RDRF
168 BF6D 962F          LDAA     SCDAT          Read data
169          * Data will be $00 if BREAK or $00 received
170 BF6F 2603          BNE      NOTZERO        Bypass jump if not $00
171 BF71 7EF000        JMP      EPRMSTR          Jump to EEPROM if $00
172 BF74              NOTZERO      EQU      *
173 BF74 81FF          CMPA     #$FF           $FF will be seen as $FF...
174 BF76 2708          BEQ      BAUDOK         if baud was correct
175          * Or else change to /104 (/13 & /8) 1200 @ 2MHz
176 BF78 142B33        BSET     BAUD $33      Works because $22 -> $33
177 BF7B CC0DB0        LDD      #DELAYS       And switch to slower...
178 BF7E DD16          STD      TOC1          delay constant
179 BF80              BAUDOK      EQU      *
180 BF80 18CE0040      LDY      #RAMSTR       Point to start of RAM
181
182 BF84              WAIT        EQU      *
183 BF84 DE16          LDX      TOC1          Move delay constant to X
184 BF86              WTLOOP      EQU      *
185 BF86 122E2009      BRSET    SCSR $20 NEWONE  Exit loop if RDRF set
186 BF8A 09            DEX              Decrement count
187 BF8B 01            NOP              Kill...
188 BF8C 01            NOP              ...seven cycles....
189 BF8D 2100          BRN      *+2          ..to match original program
190 BF8F 26F5          BNE      WTLOOP       Loop if not timed out
191 BF91 200F          BRA      STAR         Quit download on timeout
192
193 BF93              NEWONE      EQU      *
194 BF93 962F          LDAA     SCDAT          Get received data
195 BF95 18A700        STAA     $00,Y         Store to next RAM location
196 BF98 972F          STAA     SCDAT          Transmit it for handshake
197 BF9A 1808          INY              Point to next RAM location
198 BF9C 188C0100      CPY      #RAMEND+1    See if past end
199 BFA0 26E2          BNE      WAIT         If not, get another
200
201 BFA2              STAR        EQU      *
202 BFA2 CE1068        LDX      #PROGDEL       Init X with program delay
203 BFA5 18CEF000      LDY      #EPRMSTR      Init Y with EPROM start addr
204 BFA9 7E0040        JMP      RAMSTR       ** Exit to start of RAM **
205          *****
206          * Block fill unused bytes with zero
207
208 BFAC 000000000000    BSZ      $BFD1-*
209          000000000000
210          000000000000
211          000000000000
212          000000000000
213          00
214
215          *****
216          * Boot ROM revision level in ASCII
217          * (ORG      $BFD1)
218          FCC      "B"
219          *****
220          * Mask set I.D. ($0000 for EPROM parts)
221          * (ORG      $BFD2)
222          FDB      $0000
223          *****
224          * 711D3 I.D. - can be used to determine MCU type
225          * (ORG      $BFD4)
226          FDB      $71D3
227          *****
228          * VECTORS - point to RAM for pseudo-vector JUMPs
229
230 BFD6 00C4          FDB      $100-60        SCI
231 BFD8 00C7          FDB      $100-57        SPI
232 BFDA 00CA          FDB      $100-54        PULSE ACCUM INPUT EDGE
233 BFDC 00CD          FDB      $100-51        PULSE ACCUM OVERFLOW
234 BFDE 00D0          FDB      $100-48        TIMER OVERFLOW
235 BFE0 00D3          FDB      $100-45        TIMER OUTPUT COMPARE 5
236 BFE2 00D6          FDB      $100-42        TIMER OUTPUT COMPARE 4
237 BFE4 00D9          FDB      $100-39        TIMER OUTPUT COMPARE 3
238 BFE6 00DC          FDB      $100-36        TIMER OUTPUT COMPARE 2

```

Listing 5. MC68HC711D3 Bootloader ROM

Sheet 4 of 4

234 BFE8 00DF	FDB	\$100-33	TIMER OUTPUT COMPARE 1
235 BFEA 00E2	FDB	\$100-30	TIMER INPUT CAPTURE 3
236 BFEC 00E5	FDB	\$100-27	TIMER INPUT CAPTURE 2
237 BFEE 00E8	FDB	\$100-24	TIMER INPUT CAPTURE 1
238 BFF0 00EB	FDB	\$100-21	REAL TIME INT
239 BFF2 00EE	FDB	\$100-18	IRQ
240 BFF4 00F1	FDB	\$100-15	XIRQ
241 BFF6 00F4	FDB	\$100-12	SWI
242 BFF8 00F7	FDB	\$100-9	ILLEGAL OP-CODE
243 BFFA 00FA	FDB	\$100-6	COP FAIL
244 BFFC 00FD	FDB	\$100-3	CLOCK MONITOR
245 BFFE BF4D	FDB	BEGIN	RESET
246 C000	END		

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUD	002B	*00039	00156	00176
BAUDOK	BF80	*00179	00174	
BEGIN	BF4D	*00152	00245	
CONFIG	003F	*00050		
DDRD	0009	*00031		
DELAYF	021B	*00064	00159	
DELAYS	0DB0	*00063	00177	
DONEIT	BF40	*00139	00123	
EPGM	0001	*00047	00127	
EPRMEND	FFFF	*00056		
EPRMSTR	F000	*00055	00171	00203
LAT	0020	*00046	00124	00127
NEWONE	BF93	*00193	00185	
NOTZERO	BF74	*00172	00170	
OC1F	0080	*00036	00134	00137
PORTD	0008	*00030	00164	
PProg	003B	*00044	00125	00128 00138
PRGROUT	BF10	*00113	00077	
PROGDEL	1068	*00066	00202	
PROGRAM	BF00	*00077		
RAMEND	00FF	*00059	00153	00198
RAMSTR	0040	*00058	00180	00204
SCCR1	002C	*00040		
SCCR2	002D	*00041	00158	00163 00165
SCDAT	002F	*00043	00094	00117 00121 00142 00168 00194 00196
SCSR	002E	*00042	00093	00115 00120 00140 00167 00185
SPCR	0028	*00038	00154	
STAR	BFA2	*00201	00191	
TCNT	000E	*00032	00132	
TEST1	003E	*00049		
TFLG1	0023	*00034	00135	00137
TOC1	0016	*00033	00133	00160 00178 00183
UPLOAD	BF03	*00078		
UPLOOP	BF03	*00091	00096	
WAIT	BF84	*00182	00199	
WAIT1	BF18	*00119	00144	
WTLOOP	BF86	*00184	00190	

Errors: None

Labels: 37

Last Program Address: \$BFFF

Last Storage Address: \$0000

Program Bytes: \$0100 256

Storage Bytes: \$0000 0

```

1      *****
2      * BOOTLOADER FIRMWARE FOR MC68HC11F1 - 04 May 90
3      *****
4      * Features of this bootloader are...
5      *
6      * Auto baud select between 7812, 1200, 9600, 5208
7      * and 3906 (E = 2 MHz).
8      * 0 - 1024 byte variable length download:
9      *   reception of characters quits when an idle of at
10     *   least four character times occurs. (Note: at 9600
11     *   baud rate this is almost five bit times and at
12     *   5208 and 3906 rates the timeout is even longer).
13     * Jump to EEPROM at $FE00 if first download byte = $00.
14     * Part I.D. at $BFD4 is $F1F1.
15     *****
16     * Revision B -
17     *
18     * Added new baud rates: 5208, 3906.
19     *****
20     * Revision A -
21     *
22     * Added new baud rate: 9600.
23     *****
24
25     * Equates (use with index offset = $1000)
26     *
27     PORTD      EQU    $08
28     DDRD       EQU    $09
29     TOC1       EQU    $16
30     SPCR       EQU    $28                (for DWOM bit)
31     BAUD       EQU    $2B
32     SCCR1      EQU    $2C
33     SCCR2      EQU    $2D
34     SCSR       EQU    $2E
35     SCDAT      EQU    $2F
36     PPROG      EQU    $3B
37     TEST1      EQU    $3E
38     CONFIG     EQU    $3F
39
40     * Memory configuration equates
41     *
42     FE00      EEPSTR      EQU    $FE00        Start of EEPROM
43     FFFF      EEPEND      EQU    $FFFF        End of EEPROM
44     *
45     0000      RAMSTR      EQU    $0000        Start of RAM
46     03FF      RAMEND      EQU    $03FF        End of RAM
47
48     * Delay constants
49     *
50     0DB0      DELAYS      EQU    3504        Delay at slow baud rate
51     021B      DELAYF      EQU    539         Delay at fast baud rates
52     *
53     *****
54     BF00      ORG        $BF00
55     *****
56     * Main bootloader starts here
57     *****
58     * RESET vector points to here
59     BEGIN     EQU        *
60     BF00 8E03FF    LDS      #RAMEND        Initialize stack pntr
61     BF03 CE1000    LDX      #$1000        X points to registers
62     BF06 1C2820    BSET     SPCR,X $20     Select port D wire-OR mode
63     BF09 CCB00C    LDD      #B00C         Baud in A, SCCR2 in B
64     BF0C A72B      STAA     BAUD,X         SCPx = /13, SCRx = /1
65
66     * Writing 1 to MSB of BAUD resets count chain
67     BF0E E72D      STAB     SCCR2,X        Rx and Tx enabled
68     BF10 CC021B    LDD      #DELAYF        Delay for fast baud rates
69     BF13 ED16      STD      TOC1,X         Set as default delay
70
71     * Send BREAK to signal start of download
72     BF15 1C2D01    BSET     SCCR2,X $01     Set send break bit
73     BF18 1E0801FC  BRSET    PORTD,X $01 *   Wait for RxD pin to go low
74     BF1C 1D2D01    BCLR     SCCR2,X $01     Clear send break bit
75
76     BF1F 1F2E20FC  BRCLR    SCSR,X $20 *   Wait for RDRF
77     BF23 A62F      LDAA     SCDAT,X        Read data

```

* Data will be \$00 if BREAK or \$00 received

```

78 BF25 2603          BNE     NOTZERO          Bypass jump if not $00
79 BF27 7EFE00        JMP     EEPSTR          Jump to EEPROM if it was $00
80 BF2A              EQU     *
81          * Check div by 13 (9600 baud at 2 MHz)
82 BF2A 81F0          CMPA    #$F0          $F0 will be seen as $F0...
83 BF2C 271D          BEQ     BAUDOK          if baud was correct
84          * Check div by 104 (1200 baud at 2 MHz)
85 BF2E C633          LDAB    #$33          Initialize B for this rate
86 BF30 8180          CMPA    #$80          $FF will be seen as $80...
87 BF32 2710          BEQ     SLOBAUD          if baud was correct
88          * Check div by 32 (3906 baud at 2 MHz)
89          * (equals: 8192 baud at 4.2 MHz)
90 BF34 C605          LDAB    #$05          Initialize B for this rate
91 BF36 8520          BITA    #$20          $FD shows as bit 5 clear...
92 BF38 270A          BEQ     SLOBAUD          if baud was correct
93          * Change to div by 16 (7812 baud at 2 MHz)
94          * (equals: 8192 baud at 2.1 MHz)
95 BF3A C622          LDAB    #$22          Initialize B for this rate
96 BF3C E72B          STAB    BAUD,X
97 BF3E 8508          BITA    #$08          $FF shows as bit 3 set...
98 BF40 2609          BNE     BAUDOK          if baud was correct
99          * Change to div by 24 (5208 baud at 2 MHz)
100          * (equals: 8192 BAUD at 3.15 MHz)
101 BF42 C613          LDAB    #$13          By default
102
103 BF44              SLOBAUD          EQU     *
104 BF44 E72B          STAB    BAUD,X          Store baud rate
105 BF46 CC0DB0        LDD     #DELAYS          Switch to slower...
106 BF49 ED16          STD     TOC1,X          delay constant
107 BF4B              BAUDOK          EQU     *
108 BF4B 18CE0000      LDY     #RAMSTR          Point to start of RAM
109
110 BF4F              WAIT          EQU     *
111 BF4F EC16          LDD     TOC1,X          Move delay constant to D
112 BF51              WTLOOP        EQU     *
113 BF51 1E2E2007      BRSET   SCSR,X $20 NEWONE Exit loop if RDRF set
114 BF55 8F            XGDX          Swap delay count to X
115 BF56 09            DEX           Decrement count
116 BF57 8F            XGDX          Swap back to D
117 BF58 26F7          BNE     WTLOOP      Loop if not timed out
118 BF5A 200F          BRA      STAR        Quit download on timeout
119
120 BF5C              NEWONE        EQU     *
121 BF5C A62F          LDAA    SCDAT,X          Get received data
122 BF5E 18A700        STAA    $00,Y          Store to next RAM location
123 BF61 A72F          STAA    SCDAT,X          Transmit it for handshake
124 BF63 1808          INY           Point to next RAM location
125 BF65 18C0400       CPY     #RAMEND+1      See if past end
126 BF69 26E4          BNE     WAIT        If not, get another
127
128 BF6B              STAR          EQU     *
129 BF6B 7E0000        JMP     RAMSTR          ** Exit to start of RAM **
130          *****
131          * Block fill unused bytes with zero
132
133 BF6E 000000000000  BSZ     $BFD1-*
134          000000000000
135          000000000000
136          000000000000
137          000000000000
138          000000000000
139          000000000000
140          000000000000
141          000000000000
142          000000000000
143          000000000000
144          000000000000
145          000000
146
147          *****
148          * Boot ROM revision level in ASCII
149          * (ORG $BFD1)
150          FCC "B"
151          *****
152          * Mask set I.D. - ($0000 for ROMless parts)
153          * (ORG $BFD2)
154          FDB $0000
155          *****
156          * 11F1 I.D. - can be used to determine MCU type
157          * (ORG $BFD4)
158          FDB $F1F1
159          *****

```



```

148          * VECTORS - point to RAM for pseudo-vector JUMPs
149
150 BFD6 00C4          FDB      $100-60          SCI
151 BFD8 00C7          FDB      $100-57          SPI
152 BFDA 00CA          FDB      $100-54          PULSE ACCUM INPUT EDGE
153 BFDC 00CD          FDB      $100-51          PULSE ACCUM OVERFLOW
154 BFDE 00D0          FDB      $100-48          TIMER OVERFLOW
155 BFE0 00D3          FDB      $100-45          TIMER OUTPUT COMPARE 5
156 BFE2 00D6          FDB      $100-42          TIMER OUTPUT COMPARE 4
157 BFE4 00D9          FDB      $100-39          TIMER OUTPUT COMPARE 3
158 BFE6 00DC          FDB      $100-36          TIMER OUTPUT COMPARE 2
159 BFE8 00DF          FDB      $100-33          TIMER OUTPUT COMPARE 1
160 BFEA 00E2          FDB      $100-30          TIMER INPUT CAPTURE 3
161 BFEC 00E5          FDB      $100-27          TIMER INPUT CAPTURE 2
162 BFEE 00E8          FDB      $100-24          TIMER INPUT CAPTURE 1
163 BFF0 00EB          FDB      $100-21          REAL TIME INT
164 BFF2 00EE          FDB      $100-18          IRQ
165 BFF4 00F1          FDB      $100-15          XIRQ
166 BFF6 00F4          FDB      $100-12          SWI
167 BFF8 00F7          FDB      $100-9           ILLEGAL OP-CODE
168 BFFA 00FA          FDB      $100-6           COP FAIL
169 BFFC 00FD          FDB      $100-3           CLOCK MONITOR
170 BFFE BF00          FDB      BEGIN          RESET
171 C000          END

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUD	002B	*00031	00064	00096 00104
BAUDOK	BF4B	*00107	00083	00098
BEGIN	BF00	*00059	00170	
CONFIG	003F	*00038		
DDRD	0009	*00028		
DELAYF	021B	*00051	00067	
DELAYS	0DB0	*00050	00105	
EEPEND	FFFF	*00043		
EEPSTR	FE00	*00042	00079	
NEWONE	BF5C	*00120	00113	
NOTZERO	BF2A	*00080	00078	
PORTD	0008	*00027	00072	
PPROG	003B	*00036		
RAMEND	03FF	*00046	00060	00125
RAMSTR	0000	*00045	00108	00129
SCCR1	002C	*00032		
SCCR2	002D	*00033	00066	00071 00073
SCDAT	002F	*00035	00076	00121 00123
SCSR	002E	*00034	00075	00113
SLOBAUD	BF44	*00103	00087	00092
SPCR	0028	*00030	00062	
STAR	BF6B	*00128	00118	
TEST1	003E	*00037		
TOC1	0016	*00029	00068	00106 00111
WAIT	BF4F	*00110	00126	
WTLOOP	BF51	*00112	00117	

```

Errors: None
Labels: 26
Last Program Address: $BFFF
Last Storage Address: $0000
Program Bytes: $0100 256
Storage Bytes: $0000 0

```

```

1      *****
2      * BOOTLOADER FIRMWARE FOR MC68HC11K4 - 18 Jul 90
3      *****
4      * Features of this bootloader are...
5      *
6      * Auto baud select between 7812, 1200, 9600, 5208
7      * and 3906 (E = 2 MHz).
8      * 0 - 768 byte variable length download:
9      * reception of characters quits when an idle of at
10     * least four character times occurs. (Note: at 9600
11     * baud rate this is almost five bit times and at
12     * 5208 and 3906 rates the timeout is even longer).
13     * Jump to EEPROM at $0D80 if first download byte = $00.
14     * PROGRAM - Utility subroutine to program EPROM.
15     * UPLOAD - Utility subroutine to dump memory to host.
16     * Part I.D. at $BFD4 is $044B.
17     *****
18
19     * Equates (registers in direct space)
20     *
21     PORTB      EQU      $04
22     PORTF      EQU      $05
23     PORTD      EQU      $08
24     DDRD       EQU      $09
25     *
26     TCNT       EQU      $0E
27     TOC1       EQU      $16
28     TFLG1      EQU      $23
29     * Bit equates for TFLG1
30     OCLF       EQU      $80
31     *
32     EPROG      EQU      $2B
33     * Bit equates for EPROG
34     ELAT       EQU      $20
35     EPGM       EQU      $01
36     *
37     PPROG      EQU      $3B
38     TEST1      EQU      $3E
39     CONFIG     EQU      $3F
40     *
41     SCBD       EQU      $70
42     SCCR1      EQU      $72
43     SCCR2      EQU      $73
44     SCSR1      EQU      $74
45     SCSR2      EQU      $75
46     SCDRH      EQU      $76
47     SCDRL      EQU      $77
48
49     * Memory configuration equates
50     *
51     0D80      EEPMSTR      EQU      $0D80      Start of EEPROM
52     0FFF      EEPMEND      EQU      $0FFF      End of EEPROM
53     *
54     2000      ROMSTR       EQU      $2000      Start of ROM
55     7FFF      ROMEND      EQU      $7FFF      End of ROM
56     *
57     0080      RAMSTR       EQU      $0080      Start of RAM
58     037F      RAMEND      EQU      $037F      End of RAM
59
60     * Delay constants
61     *
62     15AB      DELAYS       EQU      5547      Delay at slow baud rate
63     0356      DELAYF      EQU      854        Delay at fast baud rates
64     *
65     1068      PROGDEL      EQU      4200      2 mSec programming delay
66     *
67     *
68     BE40      CYCLCOD      EQU      $BE40      EPROM cycling code (TEST)
69     *
70     *****
71     BF00      ORG          $BF00
72     *****
73     * Main bootloader starts here
74     *****
75     * RESET vector points to here
76     BEGIN     EQU          *
77     BF00      8E037F      LDS          #RAMEND      Initialize stack pntr
78     *****
79     * Special jump for EEPROM Cycling routine
80     * (This is intended for factory test only)

```

```

81      * If ports B and F both have %1001 0110 on them ...
82 BF03 CC9696      LDD      #$9696
83 BF06 1A9304      CPD      PORTB      Port F follows port B
84 BF09 2603        BNE      CONTINU
85      * ... then execute the cycling code
86 BF0B 7EBE40      JMP      CYCLCOD
87 BF0E             CONTINU      EQU      *
88
89 BF0E CC001A      LDD      #$001A      Initialize baud for...
90 BF11 DD70        STD      SCBD      9600 baud at 2 MHz
91 BF13 CC400C      LDD      #$400C      Put SCI in wire-OR mode...
92 BF16 DD72        STD      SCCR1      Enable Xmtr and Rcvr
93 BF18 CC0356      LDD      #DELAYF    Delay for fast baud rates
94 BF1B DD16        STD      TOC1      Set as default delay
95
96 BF1D 147301      * Send BREAK to signal ready for download
97 BF20 120801FC    BSET      SCCR2 $01      Set send break bit
98 BF24 157301      BRSET     PORTD $01 *      Wait for RxD pin to go low
99                  BCLR      SCCR2 $01      Clear send break bit
100 BF27 137420FC   BRCLR     SCSR1 $20 *      Wait for RDRF
101 BF2B 9677        LDAA      SCDRL      Read data
102
103 BF2D 2603        BNE      NOTZERO      Bypass jump if not $00
104 BF2F 7E0D80      JMP      EEPMSTR      Jump to EEPROM if $00
105 BF32             NOTZERO      EQU      *
106      * Check div by 26 (9600 baud at 2 MHz)
107 BF32 81F0        CMPA      #$F0      $F0 will be seen as $F0...
108 BF34 271D        BEQ      BAUDOK      if baud was correct
109
110 BF36 C6D0        * Check div by 208 (1200 baud at 2 MHz)
111 BF38 8180        LDAB      #$D0      Initialize B for this rate
112 BF3A 2710        CMPA      #$80      $FF will be seen as $80...
113                  BEQ      SLOBAUD      if baud was correct
114
115 BF3C C640        * Check div by 64 (3906 baud at 2 MHz)
116 BF3E 8520        * (equals: 8192 baud at 4.2 MHz)
117 BF40 270A        LDAB      #$40      Initialize B for this rate
118                  BITA      #$20      $FD shows as bit 5 clear...
119                  BEQ      SLOBAUD      if baud was correct
120
121 BF42 C620        * Change to div by 32 (7812 baud at 2 MHz)
122 BF44 D771        * (equals: 8192 baud at 2.1 MHz)
123 BF46 8508        LDAB      #$20      Initialize B for this rate
124 BF48 2609        STAB      SCBD+1
125                  BITA      #$08      $FF shows as bit 3 set...
126 BF4A C630        BNE      BAUDOK      if baud was correct
127
128 BF4C             * Change to div by 48 (5208 baud at 2 MHz)
129 BF4C D771        * (equals: 8192 BAUD at 3.15 MHz)
130 BF4E CC15AB      LDAB      #$30      By default
131 BF51 DD16        SLOBAUD      EQU      *
132 BF53             SLOBAUD      STAB      SCBD+1      Store baudrate
133 BF53 18CE0080    LDD      #DELAYS    Switch to slower...
134                  STD      TOC1      delay constant
135 BF57             BAUDOK      EQU      *
136 BF57 DE16        BAUDOK      LDY      #RAMSTR      Point to start of RAM
137 BF59             WAIT      EQU      *
138 BF59 12742005    WAIT      LDX      TOC1      Move delay constant to X
139 BF5D 09          WTLOOP      EQU      *
140 BF5E 26F9        WTLOOP      BRSET     SCSR1 $20 NEWONE      Exit loop if RDRF set
141 BF60 200F        WTLOOP      DEX      Decrement count
142                  BNE      WTLOOP      Loop if not timed out
143                  BRA      STAR      Quit download on timeout
144
145 BF62             NEWONE      EQU      *
146 BF62 9677        NEWONE      LDAA      SCDRL      Get received data
147 BF64 18A700      NEWONE      STAA      $00,Y      Store to next RAM location
148 BF67 9777        NEWONE      STAA      SCDRL      Transmit it for handshake
149 BF69 1808        NEWONE      INY      Point to next RAM location
150 BF6B 188C0380    NEWONE      CPY      #RAMEND+1      See if past end
151 BF6F 26E6        NEWONE      BNE      WAIT      If not, get another
152
153 BF71             STAR      EQU      *
154 BF71 7E0080      STAR      JMP      RAMSTR      ** Exit to start of RAM **
155      *****
156      * Block fill unused bytes with zero

```

```

156 BF74 000000000000          BSZ    $BFD1-*
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000
157
158
159      * Boot ROM revision level in ASCII
160      *      (ORG      $BFD1)
161 BFD1 30      FCC      "0"
162
163      * Mask set I.D. - set with user's ROM code mask layer
164      *      (ORG      $BFD2)
165 BFD2 0000      FDB      $0000      Reserve 2 bytes
166
167      * 11K4 I.D. - can be used to determine MCU type
168      * (note: $4B = K in ASCII)
169      *      (ORG      $BFD4)
170 BFD4 044B      FDB      $044B
171
172      * VECTORS - point to RAM for pseudo-vector JUMPs
173
174 BFD6 00C4      FDB      $100-60      SCI
175 BFD8 00C7      FDB      $100-57      SPI
176 BFDA 00CA      FDB      $100-54      PULSE ACCUM INPUT EDGE
177 BFDC 00CD      FDB      $100-51      PULSE ACCUM OVERFLOW
178 BFDE 00D0      FDB      $100-48      TIMER OVERFLOW
179 BFE0 00D3      FDB      $100-45      TIMER OUTPUT COMPARE 5
180 BFE2 00D6      FDB      $100-42      TIMER OUTPUT COMPARE 4
181 BFE4 00D9      FDB      $100-39      TIMER OUTPUT COMPARE 3
182 BFE6 00DC      FDB      $100-36      TIMER OUTPUT COMPARE 2
183 BFE8 00DF      FDB      $100-33      TIMER OUTPUT COMPARE 1
184 BFEA 00E2      FDB      $100-30      TIMER INPUT CAPTURE 3
185 BFEC 00E5      FDB      $100-27      TIMER INPUT CAPTURE 2
186 BFEE 00E8      FDB      $100-24      TIMER INPUT CAPTURE 1
187 BFF0 00EB      FDB      $100-21      REAL TIME INT
188 BFF2 00EE      FDB      $100-18      IRQ
189 BFF4 00F1      FDB      $100-15      XIRQ
190 BFF6 00F4      FDB      $100-12      SWI
191 BFF8 00F7      FDB      $100-9      ILLEGAL OP-CODE
192 BFFA 00FA      FDB      $100-6      COP FAIL
193 BFFC 00FD      FDB      $100-3      CLOCK MONITOR
194 BFFE BF00      FDB      BEGIN      RESET
195 C000      END

```

Symbol Table:

Symbol Name	Value	Def. #	Line Number	Cross Reference
BAUDOK	BF53	*00132	00108	00123
BEGIN	BF00	*00076	00194	
CONFIG	003F	*00039		
CONTINU	BF0E	*00087	00084	
CYCLCOD	BE40	*00068	00086	
DDRD	0009	*00024		
DELAYF	0356	*00063	00093	
DELAYS	15AB	*00062	00130	
EEPMEND	0FFF	*00052		
EEPMSTR	0D80	*00051	00104	
ELAT	0020	*00034		
EPGM	0001	*00035		
EPROG	002B	*00032		
NEWONE	BF62	*00143	00138	
NOTZERO	BF32	*00105	00103	
OC1F	0080	*00030		
PORTB	0004	*00021	00083	
PORTD	0008	*00023	00097	
PORTF	0005	*00022		
PPROG	003B	*00037		
PROGDEL	1068	*00065		
RAMEND	037F	*00058	00077	00148
RAMSTR	0080	*00057	00133	00152
ROMEND	7FFF	*00055		
ROMSTR	2000	*00054		
SCBD	0070	*00041	00090	00121 00129
SCCR1	0072	*00042	00092	
SCCR2	0073	*00043	00096	00098
SCDRH	0076	*00046		
SCDRL	0077	*00047	00101	00144 00146
SCSR1	0074	*00044	00100	00138
SCSR2	0075	*00045		
SLOBAUD	BF4C	*00128	00112	00117
STAR	BF71	*00151	00141	
TCNT	000E	*00026		
TEST1	003E	*00038		
TFLG1	0023	*00028		
TOC1	0016	*00027	00094	00131 00136
WAIT	BF57	*00135	00149	
WTLOOP	BF59	*00137	00140	

Errors: None

Labels: 40

Last Program Address: \$BFFF

Last Storage Address: \$0000

Program Bytes: \$0100 256

Storage Bytes: \$0000 0

```

1      *****
2      * BOOTLOADER FIRMWARE FOR MC68HC711K4 - 25 Apr 90
3      *****
4      * Features of this bootloader are...
5      *
6      * Auto baud select between 7812, 1200, 9600, 5208
7      *   and 3906 (E = 2 MHz).
8      * 0 - 768 byte variable length download:
9      *   reception of characters quits when an idle of at
10     *   least four character times occurs. (Note: at 9600
11     *   baud rate this is almost five bit times and at
12     *   5208 and 3906 rates the timeout is even longer).
13     * Jump to EEPROM at $0D80 if first download byte = $00.
14     * PROGRAM - Utility subroutine to program EPROM.
15     * UPLOAD - Utility subroutine to dump memory to host.
16     * Part I.D. at $BFD4 is $744B.
17     *****
18     * Revision B -
19     *
20     * Added new baud rates: 5208, 3906.
21     *****
22     * Revision A -
23     *
24     * Added new baud rate: 9600.
25     *****
26
27     * Equates (registers in direct space)
28     *
29 0004    PORTB      EQU    $04
30 0005    PORTF      EQU    $05
31 0008    PORTD      EQU    $08
32 0009    DDRD       EQU    $09
33     *
34 000E    TCNT       EQU    $0E
35 0016    TOC1       EQU    $16
36 0023    TFLG1      EQU    $23
37     * Bit equates for TFLG1
38 0080    OC1F       EQU    $80
39     *
40 002B    EPROG      EQU    $2B
41     * Bit equates for EPROG
42 0020    ELAT       EQU    $20
43 0001    EPGM       EQU    $01
44     *
45 003B    PPROG      EQU    $3B
46 003E    TEST1     EQU    $3E
47 003F    CONFIG     EQU    $3F
48     *
49 0070    SCBD       EQU    $70
50 0072    SCCR1      EQU    $72
51 0073    SCCR2      EQU    $73
52 0074    SCSR1      EQU    $74
53 0075    SCSR2      EQU    $75
54 0076    SCDRH      EQU    $76
55 0077    SCDRL      EQU    $77
56
57     * Memory configuration equates
58     *
59 0D80    EEPMSTR     EQU    $0D80    Start of EEPROM
60 0FFF    EEPMEND     EQU    $0FFF    End of EEPROM
61     *
62 2000    EPRMSTR     EQU    $2000    Start of EPROM
63 7FFF    EPRMEND     EQU    $7FFF    End of EPROM
64     *
65 0080    RAMSTR      EQU    $0080    Start of RAM
66 037F    RAMEND      EQU    $037F    End of RAM
67
68     * Delay constants
69     *
70 15AB    DELAYS      EQU    5547    Delay at slow baud rate
71 0356    DELAYF      EQU    854     Delay at fast baud rates
72     *
73 1068    PROGDEL     EQU    4200    2 mSec programming delay
74     *                                     at 2.1MHz
75     *
76 BE40    CYCLCOD     EQU    $BE40    EPROM cycling code (TEST)
77
78     *****
79 BF00    ORG          $BF00
80
81     *****

```

```

82      * Next two instructions provide a predictable place
83      * to call PROGRAM and UPLOAD even if the routines
84      * change size in future versions.
85      *
86      BF00 7EBF1D  PROGRAM      JMP      PRGROUT      EPROM programming utility
87      BF03          UPLOAD      EQU      *          Upload utility
88
89      *****
90      * UPLOAD - Utility subroutine to send data from
91      * inside the MCU to the host via the SCI interface.
92      * Prior to calling UPLOAD set baud rate, turn on SCI
93      * and set Y=first address to upload.
94      * Bootloader leaves baud set, SCI enabled.
95      * Consecutive locations are sent via SCI in an
96      * infinite loop. Reset stops the upload process.
97      *****
98      BF03 8D0D          BSR      INIT          Initialization subroutine
99
100     BF05          UPLOOP      EQU      *
101     BF05 18A600          LDAA     0,Y          Read byte
102     BF08 137480FC        BRCLR   SCSR1 $80 *  Wait for TDRE
103     BF0C 9777          STAA     SCDRL        Send it
104     BF0E 1808          INY
105     BF10 20F3          BRA      UPLOOP        Next....
106
107     *****
108     * Initialization subroutine - Forces EPROM to be
109     * enabled at $2000 so it is not overlapped by the
110     * BOOTLOADER firmware.
111     * User's address in index Y is adjusted to point to
112     * EPROM in this space instead of $A000.
113     *****
114     BF12          INIT      EQU      *
115     BF12 860F          LDAA     #$0F          EPROM is turned on
116     BF14 973F          STAA     CONFIG        at address $2000
117     BF16 188F          XGDY
118     BF18 847F          ANDA     #$7F          Get user's address
119     BF1A 188F          XGDY          Clear bit 15 of address
120     BF1C 39          RTS          Return adjusted address
121
122     *****
123     * PROGRAM - Utility subroutine to program EPROM.
124     * Prior to calling PROGRAM set baud rate, turn on SCI
125     * set X=2ms prog delay constant, and set Y=first
126     * address to program. SP must point to RAM.
127     * Bootloader leaves baud set, and SCI enabled so these
128     * default values do not have to be changed typically.
129     * Delay constant in X should be equivalent to 2 ms
130     * at 2.1 MHz X=4200; at 1 MHz X=2000, at 4MHz X=8000.
131     * An external voltage source is required for EPROM
132     * programming.
133     * This routine uses 2 bytes of stack space.
134     * Routine does not return. Reset to exit.
135     *****
136     BF1D          PRGROUT   EQU      *
137     BF1D 8DF3          BSR      INIT
138     * Send $FF to indicate ready for program data
139     BF1F 137480FC        BRCLR   SCSR1 $80 *  Wait for TDRE
140     BF23 86FF          LDAA     #$FF
141     BF25 9777          STAA     SCDRL
142     * WAIT FOR A BYTE
143     BF27          WAIT1     EQU      *
144     BF27 137420FC        BRCLR   SCSR1 $20 *  Wait for RDRF
145     BF2B D677          LDAB     SCDRL        Get received byte
146     BF2D 18E100        CMPB     $0,Y        See if already programmed
147     BF30 271D          BEQ      DONEIT      If so, skip prog cycle
148     BF32 8620          LDAA     #ELAT      Put EPROM in prog mode
149     BF34 972B          STAA     EPROG
150     BF36 18E700        STAB     0,Y        Write data
151     BF39 8621          LDAA     #ELAT+EPGM
152     BF3B 972B          STAA     EPROG      Turn on prog voltage
153     BF3D 3C          PSHX          Save delay on stack
154     BF3E 32          PULA          Put delay into D-reg
155     BF3F 33          PULB
156     BF40 D30E          ADDD     TCNT        Delay const + present TCNT
157     BF42 DD16          STD      TOC1        Schedule OC1 (prog delay)
158     BF44 8680          LDAA     #OC1F
159     BF46 9723          STAA     TFLG1        Clear any previous flag
160
161     BF48 132380FC        BRCLR   TFLG1 OC1F *  Wait for delay to expire

```

```

162 BF4C 7F002B      CLR      EPROG      Turn off prog voltage
163 BF4F             EQU      *
164 BF4F 137480FC    DONEIT  BRCLR     SCSR1 $80 *      Wait for TDRE
165 BF53 18A600      LDAA     $0,Y      Read from EPROM and...
166 BF56 9777        STAA     SCDRL     Xmit for verify
167 BF58 1808        INY             Point to next location
168 BF5A 20CB        BRA      WAIT1     Back to top for next
169                  * Loops indefinitely as long as more data sent.
170
171                  *****
172                  * Main bootloader starts here
173                  *****
174                  * RESET vector points to here
175 BF5C             BEGIN    EQU      *
176 BF5C 8E037F      LDS      #RAMEND      Initialize stack pntr
177                  *****
178                  * Special jump for EEPROM Cycling routine
179                  * (This is intended for factory test only)
180                  * If ports B and F both have $1001 0110 on them ...
181 BF5F CC9696      LDD      #$9696
182 BF62 1A9304      CPD      PORTB      Port F follows port B
183 BF65 2603        BNE      CONTINU
184                  * ... then execute the cycling code
185 BF67 7EBE40      JMP      CYCLCOD
186 BF6A             CONTINU EQU      *
187
188 BF6A CC001A      LDD      #$001A      Initialize baud for...
189 BF6D DD70        STD      SCBD      9600 baud at 2 MHz
190 BF6F CC400C      LDD      #$400C      Put SCI in wire-OR mode...
191 BF72 DD72        STD      SCCR1     Enable Xmtr and Rcvr
192 BF74 CC0356      LDD      #DELAYF     Delay for fast baud rates
193 BF77 DD16        STD      TOC1      Set as default delay
194                  * Send BREAK to signal ready for download
195 BF79 147301      BSET     SCCR2 $01      Set send break bit
196 BF7C 120801FC    BRSET    PORTD $01 *      Wait for RxD pin to go low
197 BF80 157301      BCLR     SCCR2 $01      Clear send break bit
198
199 BF83 137420FC    BRCLR     SCSR1 $20 *      Wait for RDRF
200 BF87 9677        LDAA     SCDRL     Read data
201                  * Data will be $00 if BREAK or $00 received
202 BF89 2603        BNE      NOTZERO     Bypass jump if not $00
203 BF8B 7E0D80      JMP      EEPROMSTR      Jump to EEPROM if $00
204 BF8E             NOTZERO EQU      *
205                  * Check div by 26 (9600 baud at 2 MHz)
206 BF8E 81F0        CMPA     #$F0      $F0 will be seen as $F0...
207 BF90 271D        BEQ      BAUDOK      if baud was correct
208                  * Check div by 208 (1200 baud at 2 MHz)
209 BF92 C6D0        LDAB     #$D0      Initialize B for this rate
210 BF94 8180        CMPA     #$80      $FF will be seen as $80...
211 BF96 2710        BEQ      SLOBAUD     if baud was correct
212                  * Check div by 64 (3906 baud at 2 MHz)
213                  * (equals: 8192 baud at 4.2 MHz)
214 BF98 C640        LDAB     #$40      Initialize B for this rate
215 BF9A 8520        BITA     #$20      $FD shows as bit 5 clear...
216 BF9C 270A        BEQ      SLOBAUD     if baud was correct
217                  * Change to div by 32 (7812 baud at 2 MHz)
218                  * (equals: 8192 baud at 2.1 MHz)
219 BF9E C620        LDAB     #$20      Initialize B for this rate
220 BFA0 D771        STAB     SCBD+1
221 BFA2 8508        BITA     #$08      $FF shows as bit 3 set...
222 BFA4 2609        BNE      BAUDOK      if baud was correct
223                  * Change to div by 48 (5208 baud at 2 MHz)
224                  * (equals: 8192 BAUD at 3.15 MHz)
225 BFA6 C630        LDAB     #$30      By default
226
227 BFA8             SLOBAUD EQU      *
228 BFA8 D771        STAB     SCBD+1      Store baudrate
229 BFAA CC15AB      LDD      #DELAYS     Switch to slower...
230 BFAD DD16        STD      TOC1      delay constant
231 BFAF             BAUDOK  EQU      *
232 BFAF 18CE0080    LDY      #RAMSTR      Point to start of RAM
233
234 BFB3             WAIT    EQU      *
235 BFB3 DE16        LDX      TOC1      Move delay constant to X
236 BFB5             WTLOOP EQU      *
237 BFB5 12742005    BRSET    SCSR1 $20 NEWONE Exit loop if RDRF set
238 BFB9 09          DEX             Decrement count
239 BFBA 26F9        BNE      WTLOOP     Loop if not timed out
240 BFBC 200F        BRA      STAR      Quit download on timeout
241

```



```

242 BFBE          NEWONE      EQU      *
243 BFBE 9677      LDAA      SCDRL      Get received data
244 BFC0 18A700    STAA      $00,Y      Store to next RAM location
245 BFC3 9777      STAA      SCDRL      Transmit it for handshake
246 BFC5 1808      INY          Point to next RAM location
247 BFC7 188C0380 CPY      #RAMEND+1    See if past end
248 BFCB 26E6      BNE      WAIT      If not, get another
249
250 BFCD          STAR        EQU      *
251 BFCD 7E0080     JMP      RAMSTR      ** Exit to start of RAM **
252          *****
253          * Block fill unused bytes with zero
254
255 BFD0 00          BSZ      $BFD1-*
256
257          *****
258          * Boot ROM revision level in ASCII
259          * (ORG      $BFD1)
260 BFD1 42          FCC      "B"
261          *****
262          * Mask set I.D. ($0000 for EPROM parts)
263          * (ORG      $BFD2)
264 BFD2 0000       FDB      $0000
265          *****
266          * 711K4 I.D. - can be used to determine MCU type
267          * (note: $4B = K in ASCII)
268          * (ORG      $BFD4)
269 BFD4 744B       FDB      $744B
270          *****
271          * VECTORS - point to RAM for pseudo-vector JUMPs
272
273 BFD6 00C4       FDB      $100-60      SCI
274 BFD8 00C7       FDB      $100-57      SPI
275 BFDA 00CA       FDB      $100-54      PULSE ACCUM INPUT EDGE
276 BFDC 00CD       FDB      $100-51      PULSE ACCUM OVERFLOW
277 BFDE 00D0       FDB      $100-48      TIMER OVERFLOW
278 BFE0 00D3       FDB      $100-45      TIMER OUTPUT COMPARE 5
279 BFE2 00D6       FDB      $100-42      TIMER OUTPUT COMPARE 4
280 BFE4 00D9       FDB      $100-39      TIMER OUTPUT COMPARE 3
281 BFE6 00DC       FDB      $100-36      TIMER OUTPUT COMPARE 2
282 BFE8 00DF       FDB      $100-33      TIMER OUTPUT COMPARE 1
283 BFEA 00E2       FDB      $100-30      TIMER INPUT CAPTURE 3
284 BFEC 00E5       FDB      $100-27      TIMER INPUT CAPTURE 2
285 BFEE 00E8       FDB      $100-24      TIMER INPUT CAPTURE 1
286 BFF0 00EB       FDB      $100-21      REAL TIME INT
287 BFF2 00EE       FDB      $100-18      IRQ
288 BFF4 00F1       FDB      $100-15      XIRQ
289 BFF6 00F4       FDB      $100-12      SWI
290 BFF8 00F7       FDB      $100-9       ILLEGAL OP-CODE
291 BFFA 00FA       FDB      $100-6       COP FAIL
292 BFFC 00FD       FDB      $100-3       CLOCK MONITOR
293 BFFE BF5C       FDB      BEGIN      RESET
294 C000          END

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUDOK	BFAF	*00231	00207	00222
BEGIN	BF5C	*00175	00293	
CONFIG	003F	*00047	00116	
CONTINU	BF6A	*00186	00183	
CYCLCOD	BE40	*00076	00185	
DDRD	0009	*00032		
DELAYF	0356	*00071	00192	
DELAYS	15AB	*00070	00229	
DONEIT	BF4F	*00163	00147	
EEPMEND	0FFF	*00060		
EEPMSTR	0D80	*00059	00203	
ELAT	0020	*00042	00148	00151
EPGM	0001	*00043	00151	
EPRMEND	7FFF	*00063		
EPRMSTR	2000	*00062		
EPROG	002B	*00040	00149	00152 00162
INIT	BF12	*00114	00098	00137
NEWONE	BFBE	*00242	00237	
NOTZERO	BF8E	*00204	00202	
OC1F	0080	*00038	00158	00161
PORTB	0004	*00029	00182	
PORTD	0008	*00031	00196	
PORTF	0005	*00030		
PProg	003B	*00045		
PRGROUT	BF1D	*00136	00086	
PROGDEL	1068	*00073		
PROGRAM	BF00	*00086		
RAMEND	037F	*00066	00176	00247
RAMSTR	0080	*00065	00232	00251
SCBD	0070	*00049	00189	00220 00228
SCCR1	0072	*00050	00191	
SCCR2	0073	*00051	00195	00197
SCDRH	0076	*00054		
SCDRL	0077	*00055	00103	00141 00145 00166 00200 00243 00245
SCSR1	0074	*00052	00102	00139 00144 00164 00199 00237
SCSR2	0075	*00053		
SLOBAUD	BFA8	*00227	00211	00216
STAR	BFCF	*00250	00240	
TCNT	000E	*00034	00156	
TEST1	003E	*00046		
TFLG1	0023	*00036	00159	00161
TOC1	0016	*00035	00157	00193 00230 00235
UPLOAD	BF03	*00087		
UPLOOP	BF05	*00100	00105	
WAIT	BFB3	*00234	00248	
WAIT1	BF27	*00143	00168	
WTLOOP	BFB5	*00236	00239	

Errors: None

Labels: 47


Last Program Address: SBFFF

Last Storage Address: \$0000

Program Bytes: \$0100 256

Storage Bytes: \$0000 0

This page intentionally left blank.

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

Literature Distribution Centers:

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.

ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.



MOTOROLA

AN1060/D

