Welcome to E-XFL.COM

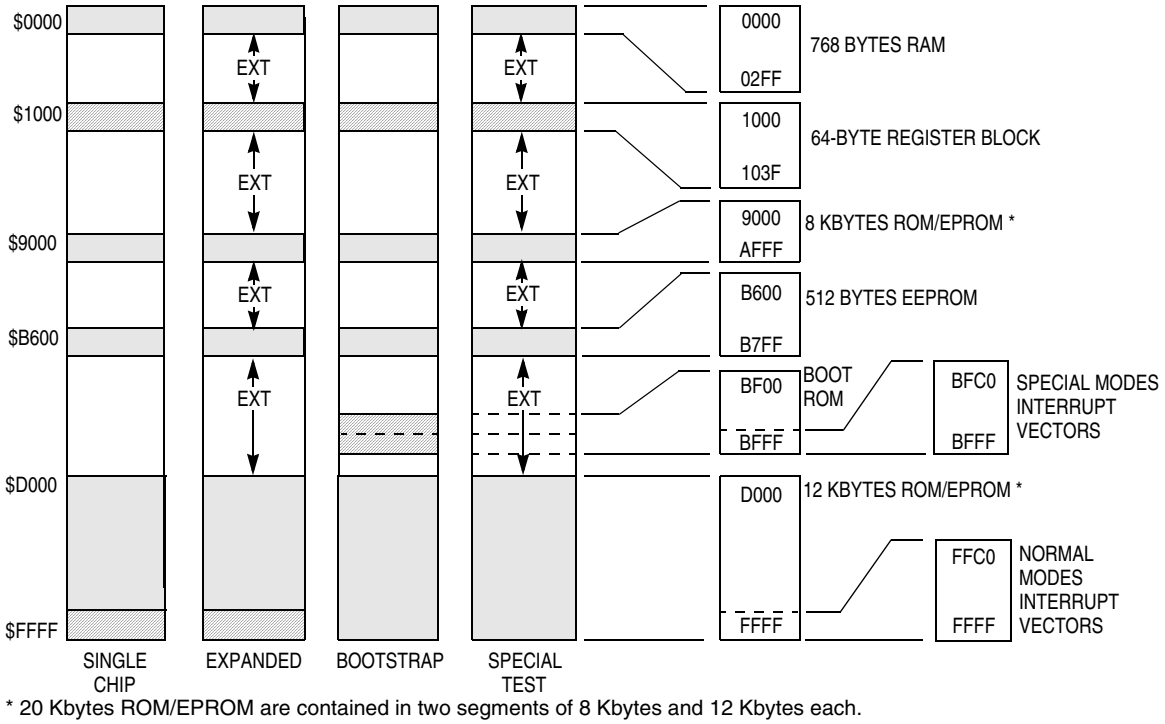
What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	-
Core Size	-
Speed	-
Connectivity	-
Peripherals	-
Number of I/O	-
Program Memory Size	-
Program Memory Type	-
EEPROM Size	-
RAM Size	-
Voltage - Supply (Vcc/Vdd)	-
Data Converters	-
Oscillator Type	-
Operating Temperature	-
Mounting Type	-
Package / Case	-
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68hc11e0cfne3

Chapter 5
Resets and Interrupts



* 20 Kbytes ROM/EPROM are contained in two segments of 8 Kbytes and 12 Kbytes each.

Figure 2-5. Memory Map for MC68HC(7)11E20

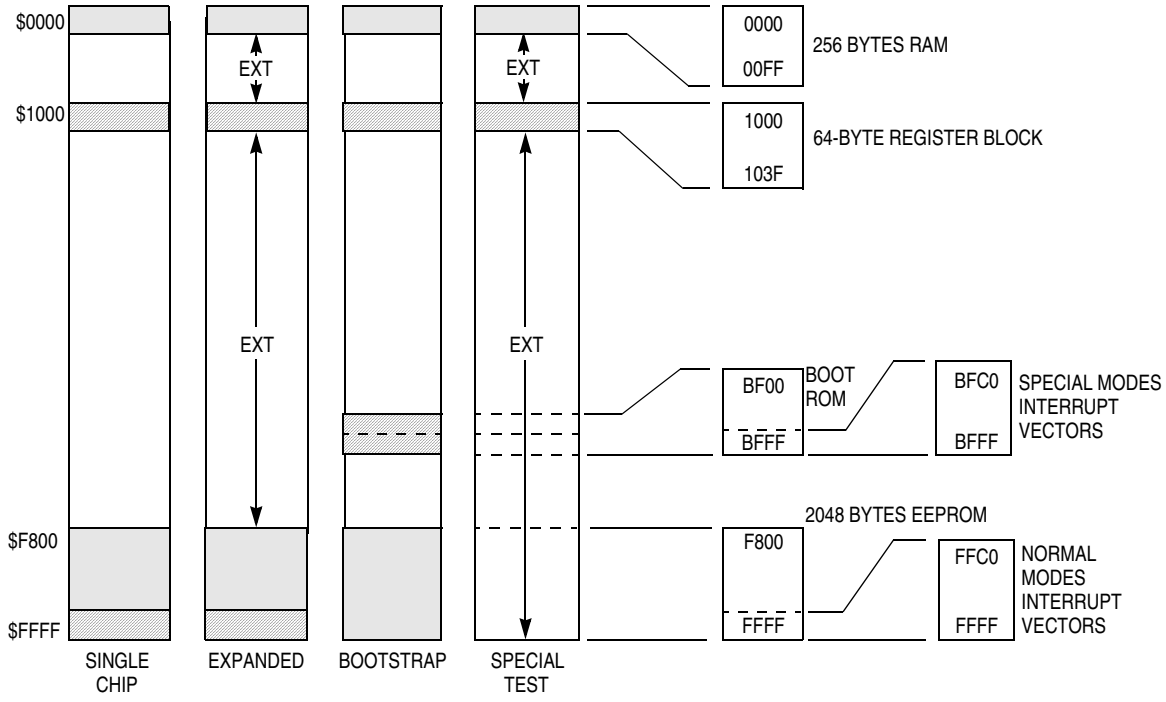


Figure 2-6. Memory Map for MC68HC811E2

3.4 Conversion Process

The A/D conversion sequence begins one E-clock cycle after a write to the A/D control/status register, ADCTL. The bits in ADCTL select the channel and the mode of conversion.

An input voltage equal to V_{RL} converts to $00 and an input voltage equal to V_{RH} converts to $FF (full scale), with no overflow indication. For ratiometric conversions of this type, the source of each analog input should use V_{RH} as the supply voltage and be referenced to V_{RL}.

3.5 Channel Assignments

The multiplexer allows the A/D converter to select one of 16 analog signals. Eight of these channels correspond to port E input lines to the MCU, four of the channels are internal reference points or test functions, and four channels are reserved. Refer to Table 3-1.

Table 3-1. Converter Channel Assignments

Channel Number	Channel Signal	Result in ADRx if MULT = 1
1	AN0	ADR1
2	AN1	ADR2
3	AN2	ADR3
4	AN3	ADR4
5	AN4	ADR1
6	AN5	ADR2
7	AN6	ADR3
8	AN7	ADR4
9 – 12	Reserved	—
13	V_{RH}(1)	ADR1
14	V_{RL}(1)	ADR2
15	$(V_{RH})/2$(1)	ADR3
16	Reserved(1)	ADR4

1. Used for factory testing

3.6 Single-Channel Operation

The two types of single-channel operation are:

1. When SCAN = 0, the single selected channel is converted four consecutive times. The first result is stored in A/D result register 1 (ADR1), and the fourth result is stored in ADR4. After the fourth conversion is complete, all conversion activity is halted until a new conversion command is written to the ADCTL register.
2. When SCAN = 1, conversions continue to be performed on the selected channel with the fifth conversion being stored in register ADR1 (overwriting the first conversion result), the sixth conversion overwriting ADR2, and so on.

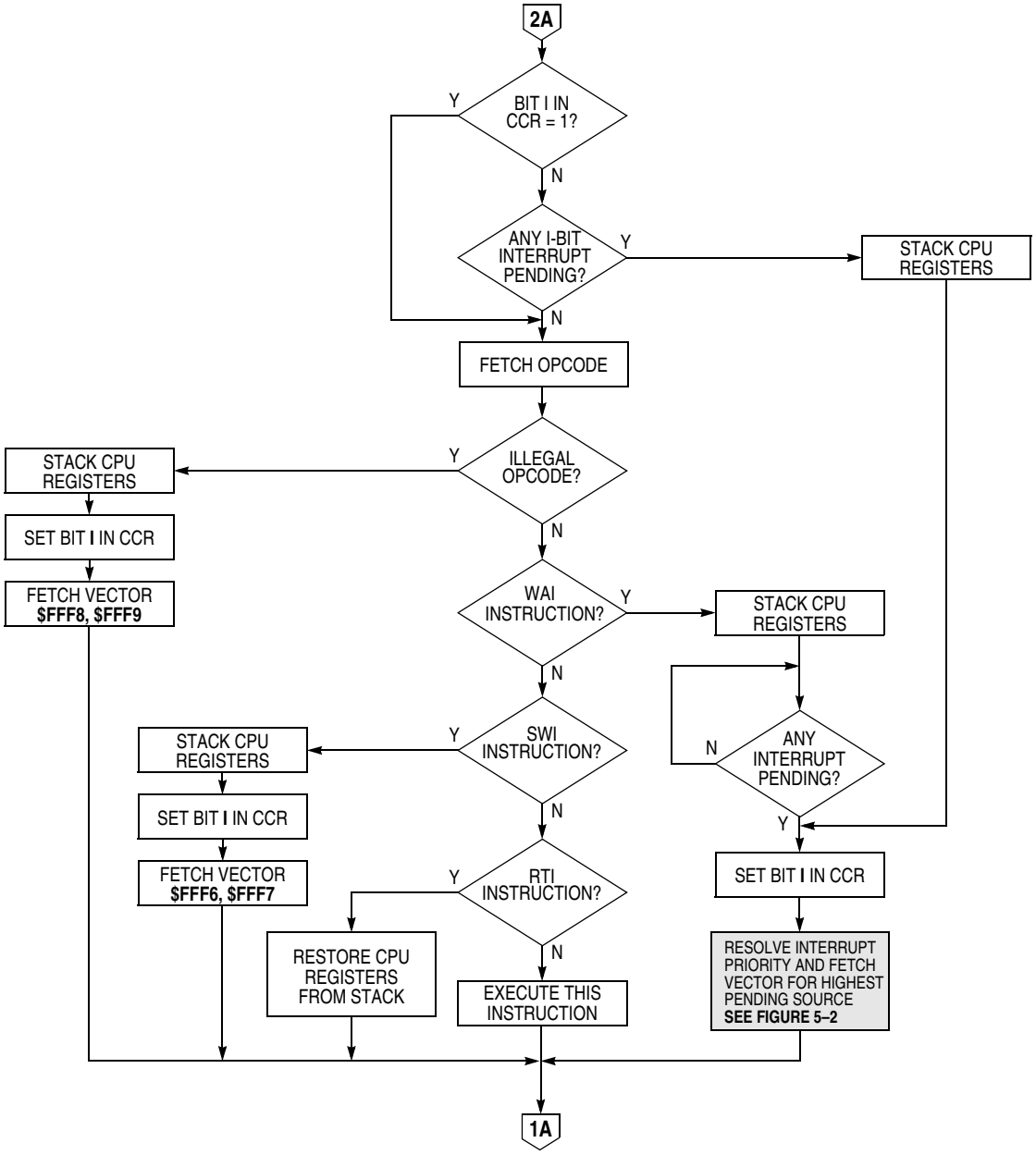


Figure 5-5. Processing Flow Out of Reset (Sheet 2 of 2)

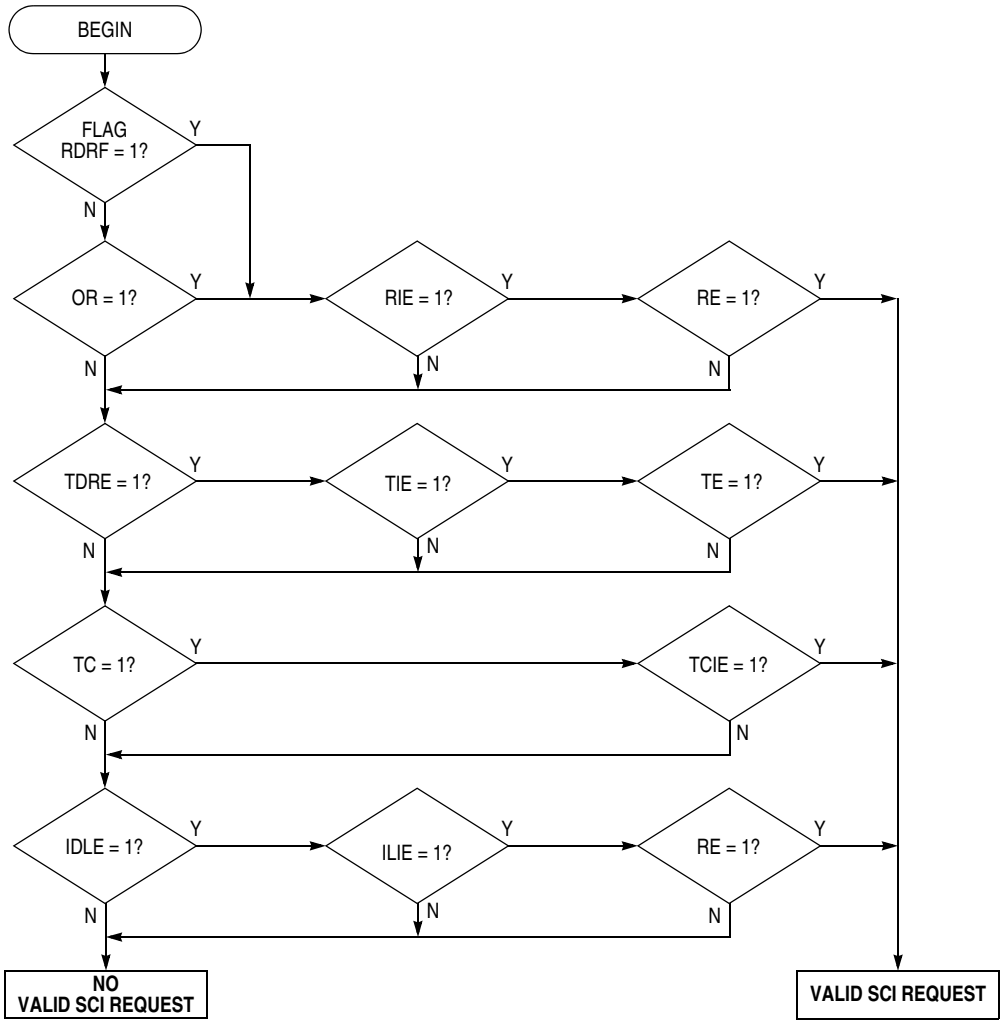


Figure 5-7. Interrupt Source Resolution Within SCI

5.6.2 Stop Mode

Executing the STOP instruction while the S bit in the CCR is equal to 0 places the MCU in stop mode. If the S bit is not 0, the stop opcode is treated as a no-op (NOP). Stop mode offers minimum power consumption because all clocks, including the crystal oscillator, are stopped while in this mode. To exit stop and resume normal processing, a logic low level must be applied to one of the external interrupts ($\overline{\text{IRQ}}$ or $\overline{\text{XIRQ}}$) or to the $\overline{\text{RESET}}$ pin. A pending edge-triggered $\overline{\text{IRQ}}$ can also bring the CPU out of stop.

Because all clocks are stopped in this mode, all internal peripheral functions also stop. The data in the internal RAM is retained as long as V_{DD} power is maintained. The CPU state and I/O pin levels are static and are unchanged by stop. Therefore, when an interrupt comes to restart the system, the MCU resumes processing as if there were no interruption. If reset is used to restart the system, a normal reset sequence results in which all I/O pins and functions are also restored to their initial states.

To use the $\overline{\text{IRQ}}$ pin as a means of recovering from stop, the I bit in the CCR must be clear ($\overline{\text{IRQ}}$ not masked). The $\overline{\text{XIRQ}}$ pin can be used to wake up the MCU from stop regardless of the state of the X bit in the CCR, although the recovery sequence depends on the state of the X bit. If X is set to 0 ($\overline{\text{XIRQ}}$ not

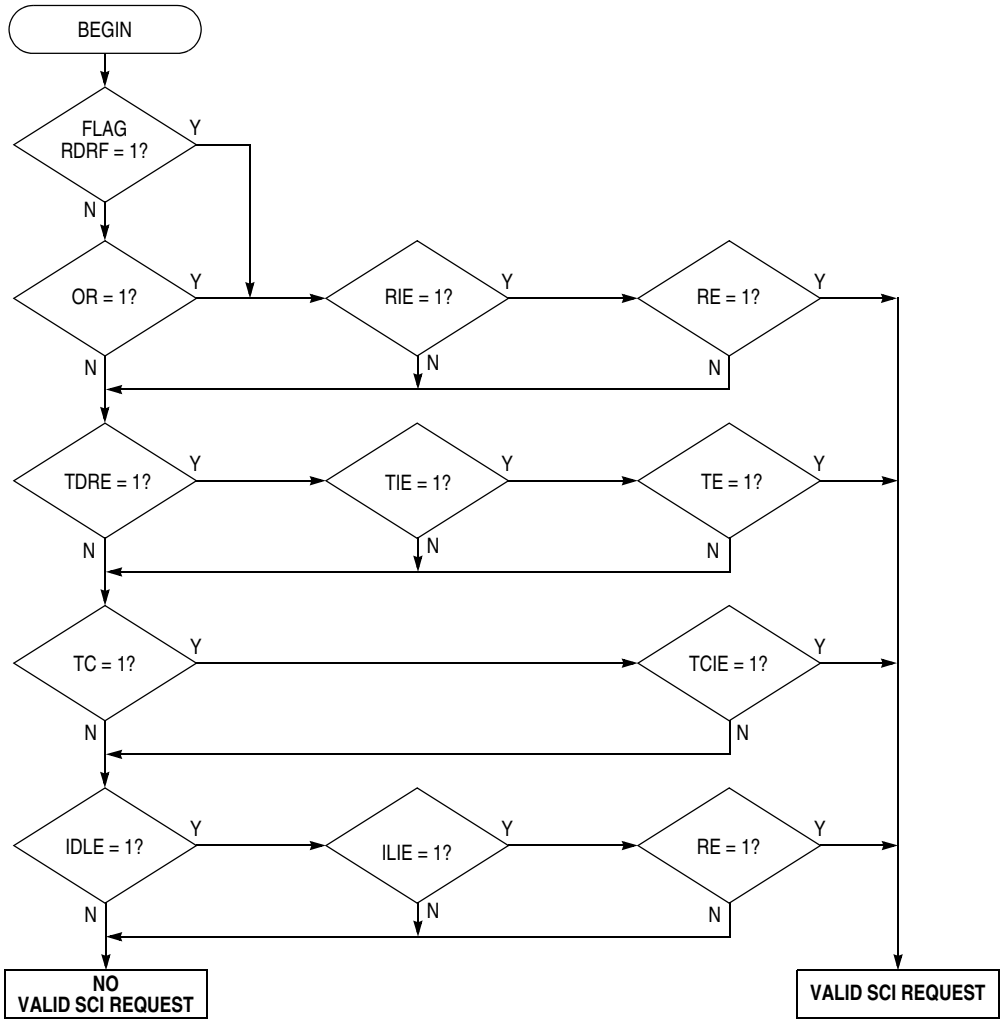


Figure 7-10. Interrupt Source Resolution Within SCI

8.5.3 Serial Clock

SCK, an input to a slave device, is generated by the master device and synchronizes data movement in and out of the device through the MOSI and MISO lines. Master and slave devices are capable of exchanging a byte of information during a sequence of eight clock cycles.

Four possible timing relationships can be chosen by using control bits CPOL and CPHA in the serial peripheral control register (SPCR). Both master and slave devices must operate with the same timing. The SPI clock rate select bits, SPR[1:0], in the SPCR of the master device, select the clock rate. In a slave device, SPR[1:0] have no effect on the operation of the SPI.

8.5.4 Slave Select

The slave select ($\overline{SS}$) input of a slave device must be externally asserted before a master device can exchange data with the slave device. $\overline{SS}$ must be low before data transactions and must stay low for the duration of the transaction.

The $\overline{SS}$ line of the master must be held high. If it goes low, a mode fault error flag (MODF) is set in the serial peripheral status register (SPSR). To disable the mode fault circuit, write a 1 in bit 5 of the port D data direction register. This sets the $\overline{SS}$ pin to act as a general-purpose output rather than the dedicated input to the slave select circuit, thus inhibiting the mode fault flag. The other three lines are dedicated to the SPI whenever the serial peripheral interface is on.

The state of the master and slave CPHA bits affects the operation of $\overline{SS}$. CPHA settings should be identical for master and slave. When CPHA = 0, the shift clock is the OR of $\overline{SS}$ with SCK. In this clock phase mode, $\overline{SS}$ must go high between successive characters in an SPI message. When CPHA = 1, $\overline{SS}$ can be left low between successive SPI characters. In cases where there is only one SPI slave MCU, its $\overline{SS}$ line can be tied to V_{SS} as long as only CPHA = 1 clock mode is used.

8.6 SPI System Errors

Two system errors can be detected by the SPI system. The first type of error arises in a multiple-master system when more than one SPI device simultaneously tries to be a master. This error is called a mode fault. The second type of error, write collision, indicates that an attempt was made to write data to the SPDR while a transfer was in progress.

When the SPI system is configured as a master and the $\overline{SS}$ input line goes to active low, a mode fault error has occurred — usually because two devices have attempted to act as master at the same time. In cases where more than one device is concurrently configured as a master, there is a chance of contention between two pin drivers. For push-pull CMOS drivers, this contention can cause permanent damage. The mode fault mechanism attempts to protect the device by disabling the drivers. The MSTR control bit in the SPCR and all four DDRD control bits associated with the SPI are cleared and an interrupt is generated subject to masking by the SPIE control bit and the I bit in the CCR.

Other precautions may need to be taken to prevent driver damage. If two devices are made masters at the same time, mode fault does not help protect either one unless one of them selects the other as slave. The amount of damage possible depends on the length of time both devices attempt to act as master.

A write collision error occurs if the SPDR is written while a transfer is in progress. Because the SPDR is not double buffered in the transmit direction, writes to SPDR cause data to be written directly into the SPI shift register. Because this write corrupts any transfer in progress, a write collision error is generated. The transfer continues undisturbed, and the write data that caused the error is not written to the shifter.

9.3.3 Timer Input Capture 4/Output Compare 5 Register

Use TI4/O5 as either an input capture register or an output compare register, depending on the function chosen for the PA3 pin. To enable it as an input capture pin, set the I4/O5 bit in the pulse accumulator control register (PACTL) to logic level 1. To use it as an output compare register, set the I4/O5 bit to a logic level 0. Refer to 9.7 Pulse Accumulator.

Register name: Timer Input Capture 4/Output Compare 5 (High) Address: $101E

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reset:	1	1	1	1	1	1	1	1

Register name: Timer Input Capture 4/Output Compare 5 (Low) Address: $101F

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset:	1	1	1	1	1	1	1	1

Figure 9-7. Timer Input Capture 4/Output Compare 5 Register Pair (TI4/O5)

9.4 Output Compare

Use the output compare (OC) function to program an action to occur at a specific time — when the 16-bit counter reaches a specified value. For each of the five output compare functions, there is a separate 16-bit compare register and a dedicated 16-bit comparator. The value in the compare register is compared to the value of the free-running counter on every bus cycle. When the compare register matches the counter value, an output compare status flag is set. The flag can be used to initiate the automatic actions for that output compare function.

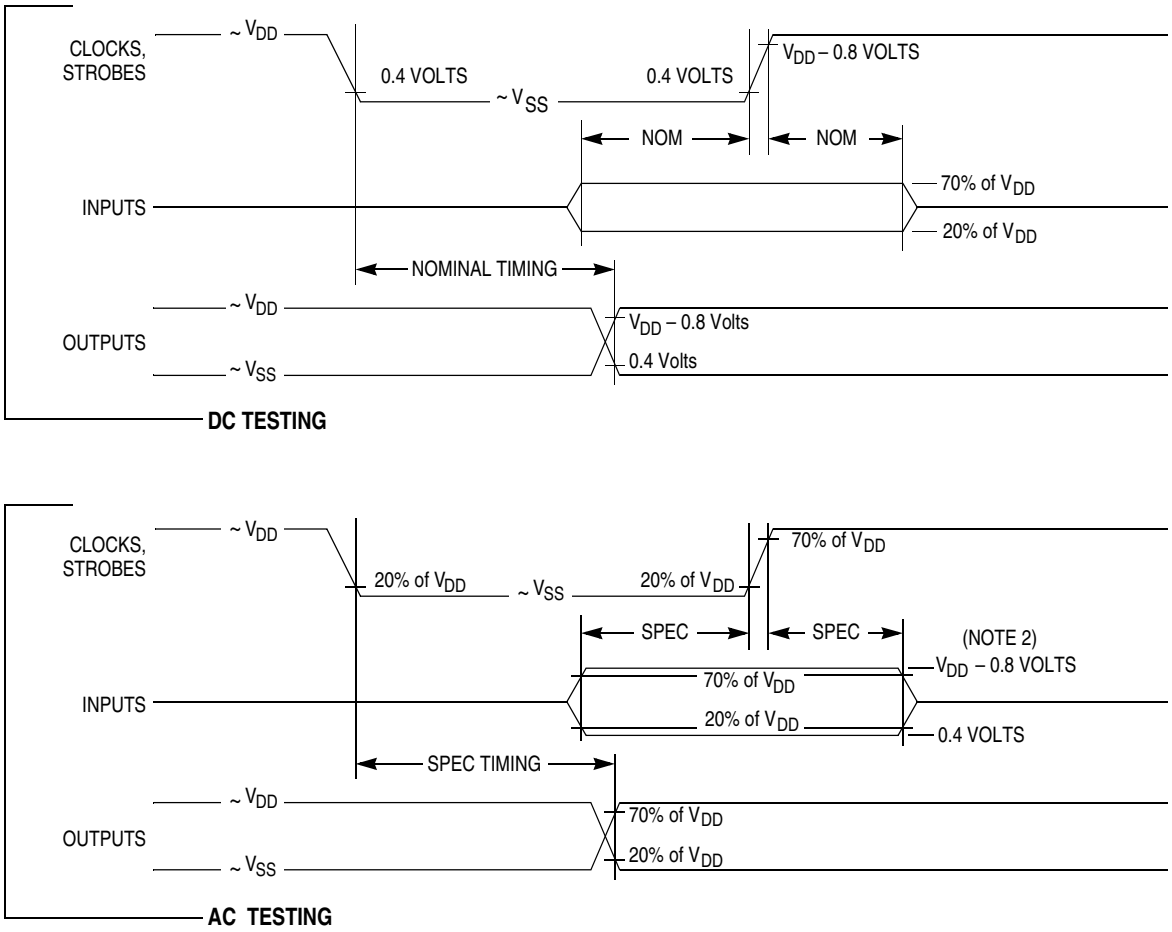
To produce a pulse of a specific duration, write a value to the output compare register that represents the time the leading edge of the pulse is to occur. The output compare circuit is configured to set the appropriate output either high or low, depending on the polarity of the pulse being produced. After a match occurs, the output compare register is reprogrammed to change the output pin back to its inactive level at the next match. A value representing the width of the pulse is added to the original value, and then written to the output compare register. Because the pin state changes occur at specific values of the free-running counter, the pulse width can be controlled accurately at the resolution of the free-running counter, independent of software latencies. To generate an output signal of a specific frequency and duty cycle, repeat this pulse-generating procedure.

The five 16-bit read/write output compare registers are: TOC1, TOC2, TOC3, and TOC4, and the TI4/O5. TI4/O5 functions under software control as either IC4 or OC5. Each of the OC registers is set to $FFFF on reset. A value written to an OC register is compared to the free-running counter value during each E-clock cycle. If a match is found, the particular output compare flag is set in timer interrupt flag register 1 (TFLG1). If that particular interrupt is enabled in the timer interrupt mask register 1 (TMSK1), an interrupt is generated. In addition to an interrupt, a specified action can be initiated at one or more timer output pins. For OC[5:2], the pin action is controlled by pairs of bits (OMx and OLx) in the TCTL1 register. The output action is taken on each successful compare, regardless of whether or not the OCxF flag in the TFLG1 register was previously cleared.

10.6 Supply Currents and Power Dissipation

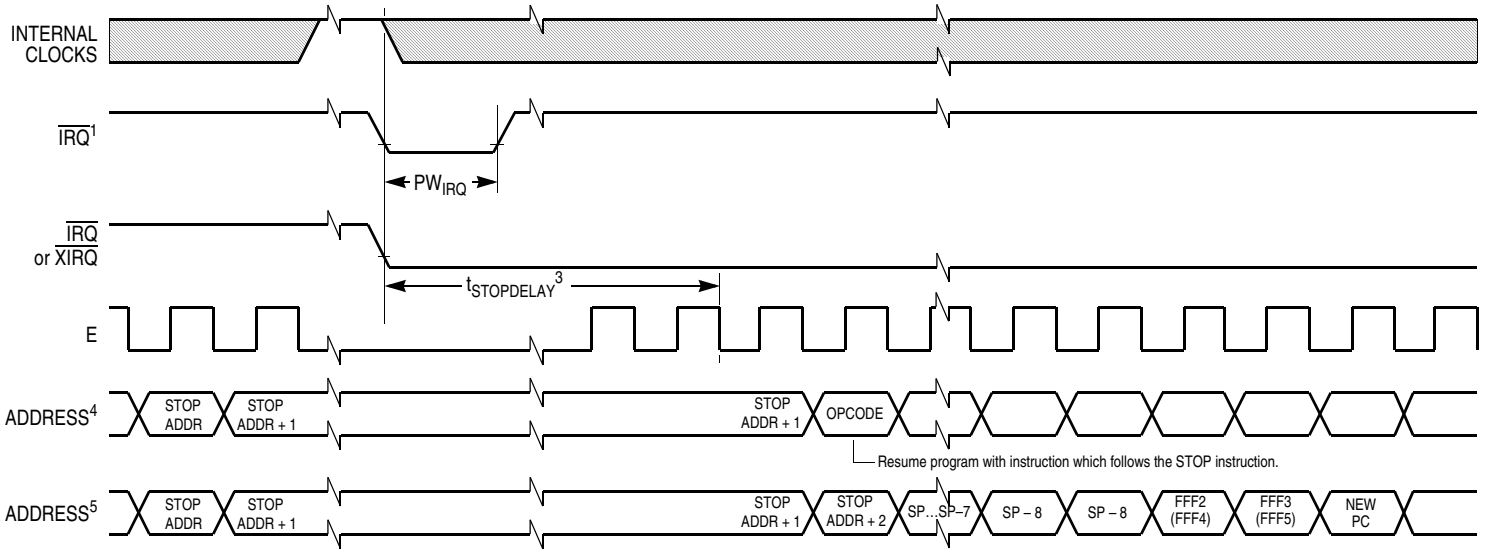
Characteristics[1]	Symbol	Min	Max	Unit
Run maximum total supply current[2] Single-chip mode2 MHz 3 MHz Expanded multiplexed mode2 MHz 3 MHz	I_{DD}	— — — —	15 27 27 35	mA
Wait maximum total supply current[2] (all peripheral functions shut down) Single-chip mode2 MHz 3 MHz Expanded multiplexed mode2 MHz 3 MHz	W_{IDD}	— — — —	6 15 10 20	mA
Stop maximum total supply current[2] Single-chip mode, no clocks–40°C to +85°C > +85°C to +105°C > +105°C to +125°C	S_{IDD}	— — —	25 50 100	µA
Maximum power dissipation Single-chip mode2 MHz 3 MHz Expanded multiplexed mode2 MHz 3 MHz	P_D	— — — —	85 150 150 195	mW

1. V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H, unless otherwise noted
2. EXTAL is driven with a square wave, and
   t_{CYC}= 500 ns for 2 MHz rating
   t_{CYC}= 333 ns for 3 MHz rating
   $V_{IL} \leq 0.2$ V
   $V_{IH} \geq V_{DD} - 0.2$ V
   no dc loads



Notes:
1. Full test loads are applied during all dc electrical tests and ac timing measurements.
2. During ac timing measurements, inputs are driven to 0.4 volts and V_{DD} – 0.8 volts while timing measurements are taken at 20% and 70% of V_{DD} points.

Figure 10-1. Test Methods



Notes:
1. Edge Sensitive $\overline{\text{IRQ}}$ pin (IRQE bit = 1)
2. Level sensitive $\overline{\text{IRQ}}$ pin (IRQE bit = 0)
3. $t_{STOPDELAY}$ = 4064 t_{CYC} if DLY bit = 1 or 4 t_{CYC} if DLY = 0.
4. $\overline{\text{XIRQ}}$ with X bit in CCR = 1.
5. $\overline{\text{IRQ}}$ or ($\overline{\text{XIRQ}}$ with X bit in CCR = 0).

Figure 10-4. STOP Recovery Timing Diagram

Description	CONFIG	Temperature	Frequency	MC Order Number
52-pin plastic leaded chip carrier (PLCC) (Continued)				
OTPROM	$0F	–40°C to +85°C	2 MHz	MC68HC711E9CFN2
			3 MHz	MC68HC711E9CFN3
		–40°C to +105°C	2 MHz	MC68HC711E9VFN2
		–40°C to +125°C	2 MHz	MC68HC711E9MFN2
OTPROM, enhanced security feature	$0F	–40°C to +85°C	2 MHz	MC68S711E9CFN2
20 Kbytes OTPROM	$0F	0°C to +70°C	3 MHz	MC68HC711E20FN3
		–40°C to +85°C	2 MHz	MC68HC711E20CFN2
			3 MHz	MC68HC711E20CFN3
		–40°C to +105°C	2 MHz	MC68HC711E20VFN2
		–40°C to +125°C	2 MHz	MC68HC711E20MFN2
No ROM, 2 Kbytes EEPROM	$FF	0°C to +70°C	2 MHz	MC68HC811E2FN2
		–40°C to +85°C	2 MHz	MC68HC811E2CFN2
		–40°C to +105°C	2 MHz	MC68HC811E2VFN2
		–40°C to +125°C	2 MHz	MC68HC811E2MFN2
64-pin quad flat pack (QFP)				
BUFFALO ROM	$0F	–40°C to +85°C	2 MHz	MC68HC11E9BCFU2
			3 MHz	MC68HC11E9BCFU3
No ROM	$0D	–40°C to +85°C	2 MHz	MC68HC11E1CFU2
			3 MHz	MC68HC11E1CFU3
		–40°C to +105°C	2 MHz	MC68HC11E1VFU2
No ROM, no EEPROM	$0C	–40°C to +85°C	2 MHz	MC68HC11E0CFU2
		–40°C to +105°C	2 MHz	MC68HC11E0VFU2
20 Kbytes OTPROM	$0F	0°C to +70°°C	3 MHz	MC68HC711E20FU3
		–40°C to +85°C	2 MHz	MC68HC711E20CFU2
			3 MHz	MC68HC711E20CFU3
		–40°C to +105°C	2 MHz	MC68HC711E20VFU2
		–40°C to +125°C	2 MHz	MC68HC711E20MFU2
52-pin thin quad flat pack (TQFP)				
BUFFALO ROM	$0F	–40°C to +85°C	2 MHz	MC68HC11E9BCPB2
			3 MHz	MC68HC11E9BCPB3

Appendix A
Development Support

A.1 Introduction

This section provides information on the development support offered for the E-series devices.

A.2 M68HC11 E-Series Development Tools

Device	Package	Emulation Module[1] [2]	Flex Cable[1] [2]	MMDS11 Target Head[1] [2]	SPGMR Programming Adapter[3]
MC68HC11E9 MC68HC711E9	52 FN	M68EM11E20	M68CBL11C	M68TC11E20FN52	M68PA11E20FN52
	52 PB	M68EM11E20	M68CBL11C	M68TC11E20PB52	M68PA11E20PB52
	56 B	M68EM11E20	M68CBL11B	M68TC11E20B56	M68PA11E20B56
	64 FU	M68EM11E20	M68CBL11C	M68TC11E20FU64	M68PA11E20FU64
MC68HC11E20 MC68HC711E20	52 FN	M68EM11E20	M68CBL11C	M68TC11E20FN52	M68PA11E20FN52
	64 FU	M68EM11E20	M68CBL11C	M68TC11E20FU64	M68PA11E20FU64
MC68HC811E2	48 P	M68EM11E20	M68CBL11B	M68TB11E20P48	M68PA11A8P48
	52 FN	M68EM11E20	M68CBL11C	M68TC11E20FN52	M68PA11E20FN52

1. Each MMDS11 system consists of a system console (M68MMDS11), an emulation module, a flex cable, and a target head.
2. A complete EVS consists of a platform board (M68HC11PFB), an emulation module, a flex cable, and a target head.
3. Each SPGMR system consists of a universal serial programmer (M68SPGMR11) and a programming adapter. It can be used alone or in conjunction with the MMDS11.

A.3 EVS — Evaluation System

The EVS is an economical tool for designing, debugging, and evaluating target systems based on the M68HC11. EVS features include:

- Monitor/debugger firmware
- One-line assembler/disassembler
- Host computer download capability
- Dual memory maps:
  - 64-Kbyte monitor map that includes 16 Kbytes of monitor EPROM
  - M68HC11 E-series user map that includes 64 Kbytes of emulation RAM
- MCU extension input/output (I/O) port for single-chip, expanded, and special-test operation modes
- RS-232C terminal and host I/O ports
- Logic analyzer connector

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 process, in this manner:

- First, the on-chip SCI (serial communications interface) 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, Freescale uses 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 many MCUS. These boards are then placed in burn-in ovens. Driver boards outside the ovens download an EEPROM exercise and diagnostic program to all 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.

Freescale also uses bootstrap mode for programming target devices on the M68HC11 evaluation modules (EVM). 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.

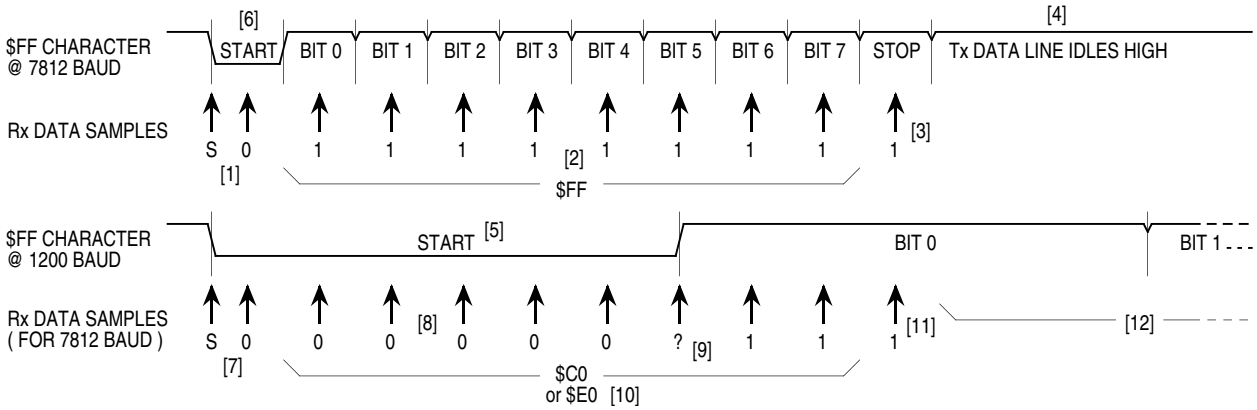


Figure 2. Automatic Detection of Baud Rate

Samples taken at [7] detect the failing edge of the start bit and verify it is a logic 0. Samples taken at the middle of what the receiver interprets as the first five bit times [8] detect logic 0s. The sample taken at the middle of what the receiver interprets as bit 5 [9] may detect either a 0 or a 1 because the receive data has a rising transition at about this time. The samples for bits 6 and 7 detect 1s, 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 1 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 Listing 3. MC68HC711E9 Bootloader ROM at the end of this application note, and to **Appendix B** of the *M68HC11 Reference Manual*, Freescale document order number M68HC11RM/AD.

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

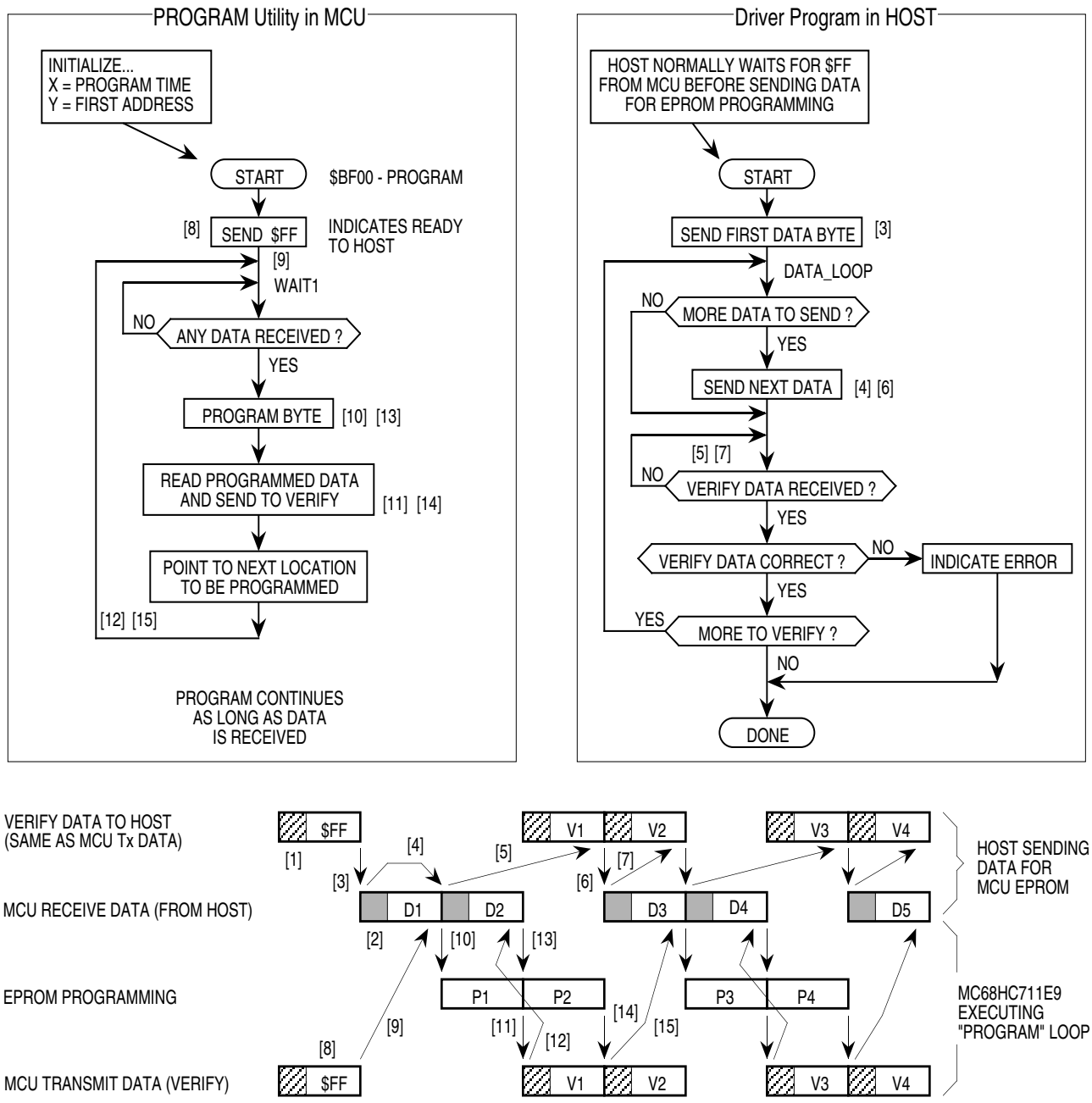


Figure 4. Host and MCU Activity during EPROM PROGRAM Utility

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 non-volatile memories in target M68HC11s in Freescale 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 Universal Evaluation Board User's Manual*, Freescale document order number M68HC11EVBU/D.

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 be restored easily 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 12-Kbyte 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 0 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 0, it is configured for the PROG EPROM emulation mode, and PB7 is the output enable signal for the EPROM data I/O (input/output) 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. MCU-to-MCU Duplicator Program.

Listing 2. BASIC Program for Personal Computer

```
1 ' ************************************************************************
2 ' *
3 ' *    E9BUF.BAS - A PROGRAM TO DEMONSTRATE THE USE OF THE BOOT MODE
4 ' *                   ON THE HC11 BY PROGRAMMING AN HC711E9 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.
14 '*
15 '************************************************************************
25 H$ = "0123456789ABCDEF"      'STRING TO USE FOR HEX CONVERSIONS
30 DEFINT B, I: CODESIZE% = 8192: ADRSTART= 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 Q$
60 A$ = MID$(Q$, 1, 1)
65 GOSUB 7000                   'CONVERTS HEX DIGIT TO DECIMAL
70 TEMP = 16 * X                'HANG ON TO UPPER DIGIT
75 A$ = MID$(Q$, 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$;") ";
107 INPUT Q$
110 IF Q$<>"" THEN FILNAM$=Q$
120 OPEN FILNAM$ FOR INPUT AS #1
130 PRINT : PRINT "Converting "; FILNAM$; " to binary..."
999 REM ***** SCANS FOR 'S1' RECORDS *****
1000 GOSUB 6000                  'GET 1 CHARACTER FROM INPUT FILE
1010 IF FLAG THEN 1250           'FLAG IS EOF FLAG FROM SUBROUTINE
1020 IF A$ <> "S" THEN 1000
1022 GOSUB 6000
1024 IF A$ <> "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
```