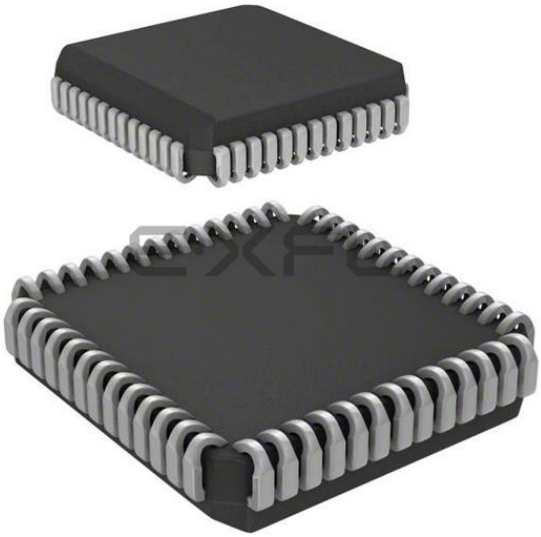


Welcome to E-XFL.COM

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	Obsolete
Core Processor	HC11
Core Size	8-Bit
Speed	3MHz
Connectivity	SCI, SPI
Peripherals	POR, WDT
Number of I/O	38
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	512 x 8
RAM Size	512 x 8
Voltage - Supply (Vcc/Vdd)	4.5V ~ 5.5V
Data Converters	A/D 8x8b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	52-LCC (J-Lead)
Supplier Device Package	52-PLCC (19.1x19.1)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68hc11e1cfne2r

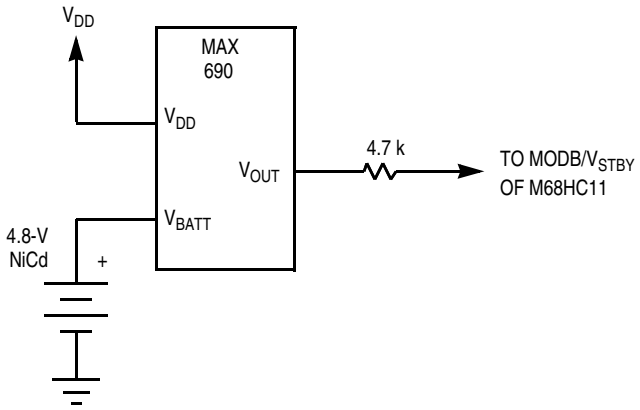


Figure 2-8. RAM Standby MODB/V_{STBY} Connections

The bootloader program is contained in the internal bootstrap ROM. This ROM, which appears as internal memory space at locations $BF00–$BFFF, is enabled only if the MCU is reset in special bootstrap mode.

In expanded modes, the ROM/EPROM/OTPROM (if present) is enabled out of reset and located at the top of the memory map if the ROMON bit in the CONFIG register is set. ROM or EPROM is enabled out of reset in single-chip and bootstrap modes, regardless of the state of ROMON.

For devices with 512 bytes of EEPROM, the EEPROM is located at $B600–$B7FF and has the same read cycle time as the internal ROM. The 512 bytes of EEPROM cannot be remapped to other locations.

For the MC68HC811E2, EEPROM is located at $F800–$FFFF and can be remapped to any 4-Kbyte boundary. EEPROM mapping control bits (EE[3:0] in CONFIG) determine the location of the 2048 bytes of EEPROM and are present only on the MC68HC811E2. Refer to 2.3.3.1 System Configuration Register for a description of the MC68HC811E2 CONFIG register.

EEPROM can be programmed or erased by software and an on-chip charge pump, allowing EEPROM changes using the single V_{DD} supply.

2.3.2 Mode Selection

The four mode variations are selected by the logic states of the MODA and MODB pins during reset. The MODA and MODB logic levels determine the logic state of SMOD and the MDA control bits in the highest priority I-bit interrupt and miscellaneous (HPRIO) register.

After reset is released, the mode select pins no longer influence the MCU operating mode. In single-chip operating mode, the MODA pin is connected to a logic level 0. In expanded mode, MODA is normally connected to V_{DD} through a pullup resistor of 4.7 kΩ. The MODA pin also functions as the load instruction register $\overline{LIR}$ pin when the MCU is not in reset. The open-drain active low $\overline{LIR}$ output pin drives low during the first E cycle of each instruction. The MODB pin also functions as standby power input (V_{STBY}), which allows RAM contents to be maintained in absence of V_{DD}.

Refer to Table 2-1, which is a summary of mode pin operation, the mode control bits, and the four operating modes.

NOSEC — Security Disable Bit

NOSEC is invalid unless the security mask option is specified before the MCU is manufactured. If the security mask option is omitted NOSEC always reads 1. The enhanced security feature is available in the MC68S711E9 MCU. The enhancement to the standard security feature protects the EPROM as well as RAM and EEPROM.

0 = Security enabled
1 = Security disabled

NOCOP — COP System Disable Bit

Refer to Chapter 5 Resets and Interrupts.

1 = COP disabled
0 = COP enabled

ROMON — ROM/EPROM/OTPROM Enable Bit

When this bit is 0, the ROM or EPROM is disabled and that memory space becomes externally addressed. In single-chip mode, ROMON is forced to 1 to enable ROM/EPROM regardless of the state of the ROMON bit.

0 = ROM disabled from the memory map
1 = ROM present in the memory map

EEON — EEPROM Enable Bit

When this bit is 0, the EEPROM is disabled and that memory space becomes externally addressed.

0 = EEPROM removed from the memory map
1 = EEPROM present in the memory map

2.3.3.2 RAM and I/O Mapping Register

The internal registers used to control the operation of the MCU can be relocated on 4-Kbyte boundaries within the memory space with the use of the RAM and I/O mapping register (INIT). This 8-bit special-purpose register can change the default locations of the RAM and control registers within the MCU memory map. It can be written only once within the first 64 E-clock cycles after a reset in normal modes, and then it becomes a read-only register.

Address: $103D

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0
Reset:	0	0	0	0	0	0	0	1

Figure 2-12. RAM and I/O Mapping Register (INIT)

RAM[3:0] — RAM Map Position Bits

These four bits, which specify the upper hexadecimal digit of the RAM address, control position of RAM in the memory map. RAM can be positioned at the beginning of any 4-Kbyte page in the memory map. It is initialized to address $0000 out of reset. Refer to Table 2-4.

REG[3:0] — 64-Byte Register Block Position

These four bits specify the upper hexadecimal digit of the address for the 64-byte block of internal registers. The register block, positioned at the beginning of any 4-Kbyte page in the memory map, is initialized to address $1000 out of reset. Refer to Table 2-5.

Address: $1036

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	MBE		ELAT	EXCOL	EXROW	T1	T0	PGM
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 2-15. MC68HC711E20 EPROM Programming Control Register (EPROG)

MBE — Multiple-Byte Programming Enable Bit

When multiple-byte programming is enabled, address bit 5 is considered a don't care so that bytes with address bit 5 = 0 and address bit 5 = 1 both get programmed. MBE can be read in any mode and always reads 0 in normal modes. MBE can be written only in special modes.

0 = EPROM array configured for normal programming
1 = Program two bytes with the same data

Bit 6 — Unimplemented

Always reads 0

ELAT — EPROM/OTPROM Latch Control Bit

When ELAT = 1, writes to EPROM cause address and data to be latched and the EPROM/OTPROM cannot be read. ELAT can be read any time. ELAT can be written any time except when PGM = 1; then the write to ELAT is disabled.

0 = EPROM/OTPROM address and data bus configured for normal reads
1 = EPROM/OTPROM address and data bus configured for programming

EXCOL — Select Extra Columns Bit

0 = User array selected
1 = User array is disabled and extra columns are accessed at bits [7:0]. Addresses use bits [13:5] and bits [4:0] are don't care. EXCOL can be read and written only in special modes and always returns 0 in normal modes.

EXROW — Select Extra Rows Bit

0 = User array selected
1 = User array is disabled and two extra rows are available. Addresses use bits [7:0] and bits [13:8] are don't care. EXROW can be read and written only in special modes and always returns 0 in normal modes.

T[1:0] — EPROM Test Mode Select Bits

These bits allow selection of either gate stress or drain stress test modes. They can be read and written only in special modes and always read 0 in normal modes.

T1	T0	Function Selected
0	0	Normal mode
0	1	Reserved
1	0	Gate stress
1	1	Drain stress

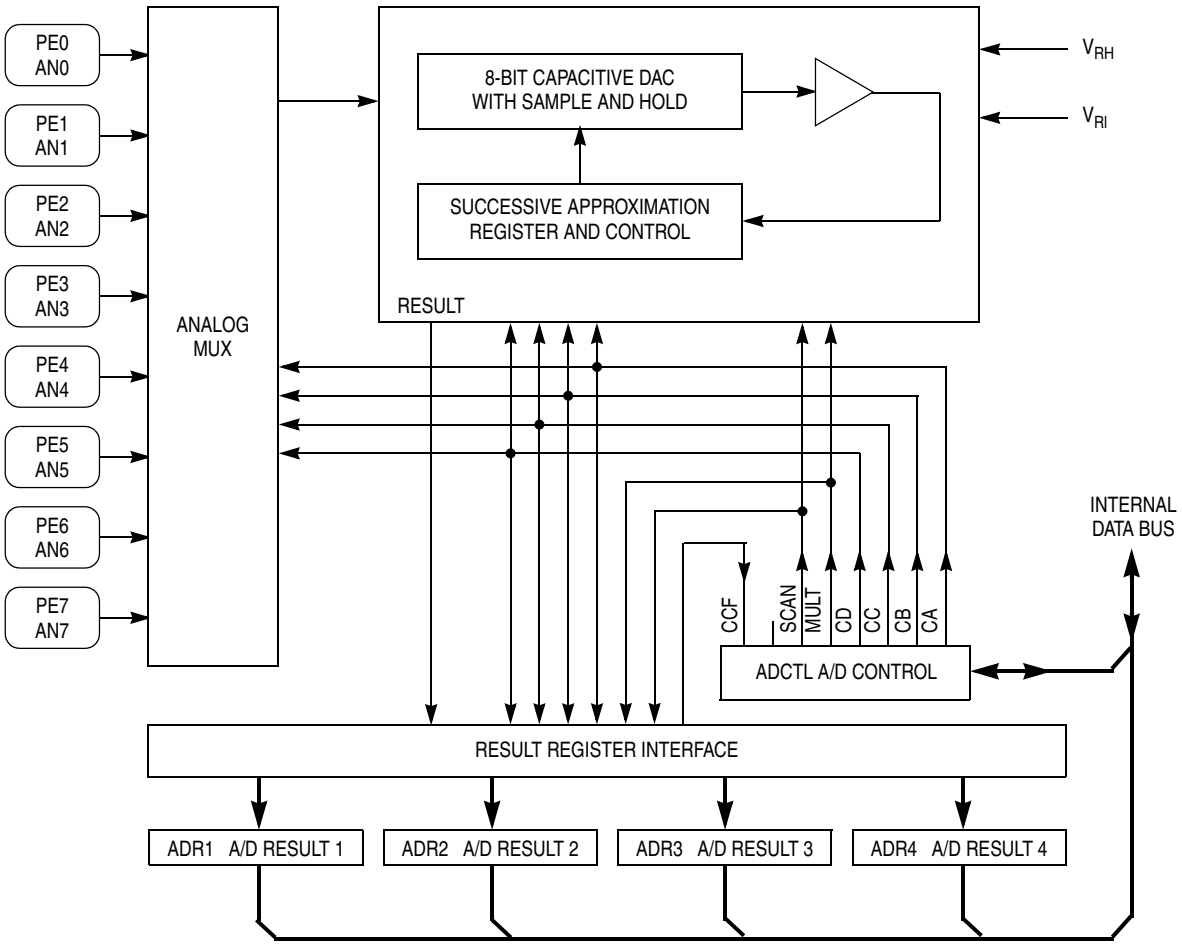
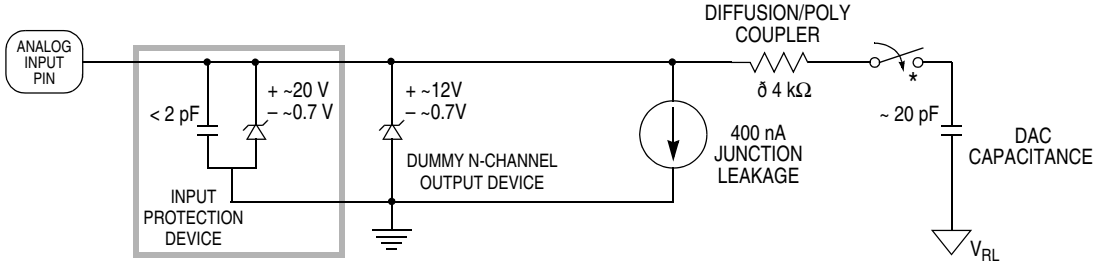


Figure 3-1. A/D Converter Block Diagram



* THIS ANALOG SWITCH IS CLOSED ONLY DURING THE 12-CYCLE SAMPLE TIME.

Figure 3-2. Electrical Model of an A/D Input Pin (Sample Mode)

end of the interrupt service routine, the return-from-interrupt instruction is executed and the saved registers are pulled from the stack in reverse order so that normal program execution can resume. Refer to Chapter 4 Central Processor Unit (CPU).

Table 5-5. Stacking Order on Entry to Interrupts

Memory Location	CPU Registers
SP	PCL
SP–1	PCH
SP–2	IYL
SP–3	IYH
SP–4	IXL
SP–5	IXH
SP–6	ACCA
SP–7	ACCB
SP–8	CCR

5.5.2 Non-Maskable Interrupt Request ($\overline{\text{XIRQ}}$)

Non-maskable interrupts are useful because they can always interrupt CPU operations. The most common use for such an interrupt is for serious system problems, such as program runaway or power failure. The $\overline{\text{XIRQ}}$ input is an updated version of the $\overline{\text{NMI}}$ (non-maskable interrupt) input of earlier MCUs.

Upon reset, both the X bit and I bit of the CCR are set to inhibit all maskable interrupts and $\overline{\text{XIRQ}}$. After minimum system initialization, software can clear the X bit by a TAP instruction, enabling $\overline{\text{XIRQ}}$ interrupts. Thereafter, software cannot set the X bit. Thus, an $\overline{\text{XIRQ}}$ interrupt is a non-maskable interrupt. Because the operation of the I-bit-related interrupt structure has no effect on the X bit, the internal $\overline{\text{XIRQ}}$ pin remains unmasked. In the interrupt priority logic, the $\overline{\text{XIRQ}}$ interrupt has a higher priority than any source that is maskable by the I bit. All I-bit-related interrupts operate normally with their own priority relationship.

When an I-bit-related interrupt occurs, the I bit is automatically set by hardware after stacking the CCR byte. The X bit is not affected. When an X-bit-related interrupt occurs, both the X and I bits are automatically set by hardware after stacking the CCR. A return-from-interrupt instruction restores the X and I bits to their pre-interrupt request state.

5.5.3 Illegal Opcode Trap

Because not all possible opcodes or opcode sequences are defined, the MCU includes an illegal opcode detection circuit, which generates an interrupt request. When an illegal opcode is detected and the interrupt is recognized, the current value of the program counter is stacked. After interrupt service is complete, reinitialize the stack pointer so repeated execution of illegal opcodes does not cause stack underflow. Left uninitialized, the illegal opcode vector can point to a memory location that contains an illegal opcode. This condition causes an infinite loop that causes stack underflow. The stack grows until the system crashes.

The illegal opcode trap mechanism works for all unimplemented opcodes on all four opcode map pages. The address stacked as the return address for the illegal opcode interrupt is the address of the first byte of the illegal opcode. Otherwise, it would be almost impossible to determine whether the illegal opcode had been one or two bytes. The stacked return address can be used as a pointer to the illegal opcode so the illegal opcode service routine can evaluate the offending opcode.

masked), the MCU starts up, beginning with the stacking sequence leading to normal service of the $\overline{\text{XIRQ}}$ request. If X is set to 1 ($\overline{\text{XIRQ}}$ masked or inhibited), then processing continues with the instruction that immediately follows the STOP instruction, and no $\overline{\text{XIRQ}}$ interrupt service is requested or pending.

Because the oscillator is stopped in stop mode, a restart delay may be imposed to allow oscillator stabilization upon leaving stop. If the internal oscillator is being used, this delay is required; however, if a stable external oscillator is being used, the DLY control bit can be used to bypass this startup delay. The DLY control bit is set by reset and can be optionally cleared during initialization. If the DLY equal to 0 option is used to avoid startup delay on recovery from stop, then reset should not be used as the means of recovering from stop, as this causes DLY to be set again by reset, imposing the restart delay. This same delay also applies to power-on reset, regardless of the state of the DLY control bit, but does not apply to a reset while the clocks are running.

Table 7-1. Baud Rate Values

Prescaler Selects						Prescale Divide	Baud Set Divide	Crystal Frequency (MHz)					
								4.00	4.9152	8.00	10.00	12.00	16.00
								Bus Frequency (MHz)					
SCP2	SCP1	SCP0	SCR2	SCR1	SCR0			1.00	1.23	2.00	2.50	3.00	4.00
0	0	0	0	0	0	1	1	62500	76800	125000	156250	187500	250000
0	0	0	0	0	1	1	2	31250	38400	62500	78125	93750	125000
0	0	0	0	1	0	1	4	15625	19200	31250	39063	46875	62500
0	0	0	0	1	1	1	8	7813	9600	15625	19531	23438	31250
0	0	0	1	0	0	1	16	3906	4800	7813	9766	11719	15625
0	0	0	1	0	1	1	32	1953	2400	3906	4883	5859	7813
0	0	0	1	1	0	1	64	977	1200	1953	2441	2930	3906
0	0	0	1	1	1	1	128	488	600	977	1221	1465	1953
0	0	1	0	0	0	3	1	20833	25600	41667	52083	62500	83333
0	0	1	0	0	1	3	2	10417	12800	20833	26042	31250	41667
0	0	1	0	1	0	3	4	5208	6400	10417	13021	15625	20833
0	0	1	0	1	1	3	8	2604	3200	5208	6510	7813	10417
0	0	1	1	0	0	3	16	1302	1600	2604	3255	3906	5208
0	0	1	1	0	1	3	32	651	800	1302	1628	1953	2604
0	0	1	1	1	0	3	64	326	400	651	814	977	1302
0	0	1	1	1	1	3	128	163	200	326	407	488	651
0	1	0	0	0	0	4	1	15625	19200	31250	39063	46875	62500
0	1	0	0	0	1	4	2	7813	9600	15625	19531	23438	31250
0	1	0	0	1	0	4	4	3906	4800	7813	9766	11719	15625
0	1	0	0	1	1	4	8	1953	2400	3906	4883	5859	7813
0	1	0	1	0	0	4	16	977	1200	1953	2441	2930	3906
0	1	0	1	0	1	4	32	488	600	977	1221	1465	1953
0	1	0	1	1	0	4	64	244	300	488	610	732	977
0	1	0	1	1	1	4	128	122	150	244	305	366	488
0	1	1	0	0	0	13	1	4808	5908	9615	12019	14423	19231
0	1	1	0	0	1	13	2	2404	2954	4808	6010	7212	9615
0	1	1	0	1	0	13	4	1202	1477	2404	3005	3606	4808
0	1	1	0	1	1	13	8	601	738	1202	1502	1803	2404
0	1	1	1	0	0	13	16	300	369	601	751	901	1202
0	1	1	1	0	1	13	32	150	185	300	376	451	601
0	1	1	1	1	0	13	64	75	92	150	188	225	300
0	1	1	1	1	1	13	128	38	46	75	94	113	150
1	0	0	0	0	0	39	1	1603	1969	3205	4006	4808	6410
1	0	0	0	0	1	39	2	801	985	1603	2003	2404	3205
1	0	0	0	1	0	39	4	401	492	801	1002	1202	1603
1	0	0	0	1	1	39	8	200	246	401	501	601	801
1	0	0	1	0	0	39	16	100	123	200	250	300	401
1	0	0	1	0	1	39	32	50	62	100	125	150	200
1	0	0	1	1	0	39	64	25	31	50	63	75	100
1	0	0	1	1	1	39	128	13	15	25	31	38	50

Shaded areas reflect standard baud rates.
On MC68HC(7)11E20 do not set SCP1 or SCP0 when SCP2 is 1.

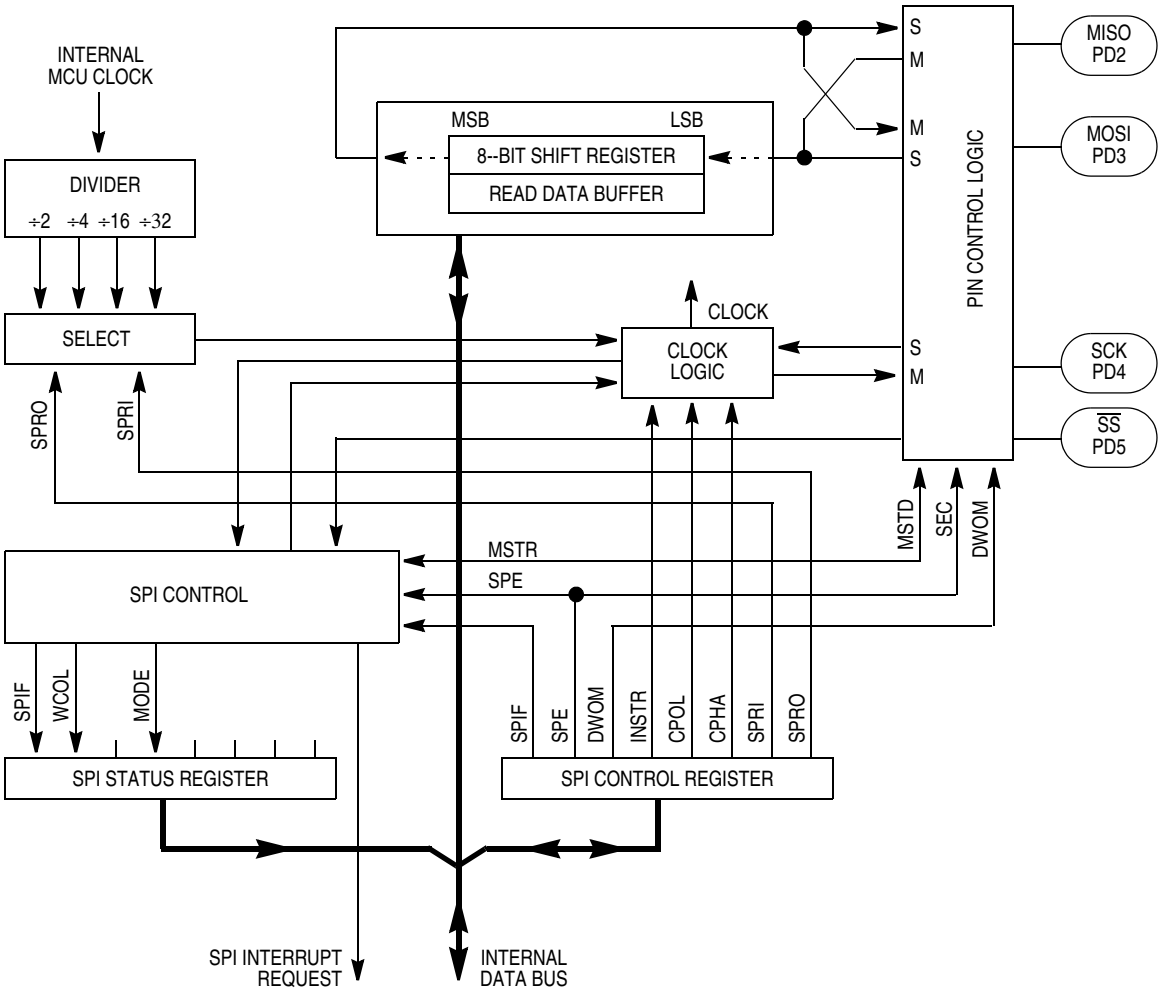


Figure 8-1. SPI Block Diagram

8.4 Clock Phase and Polarity Controls

Software can select one of four combinations of serial clock phase and polarity using two bits in the SPI control register (SPCR). The clock polarity is specified by the CPOL control bit, which selects an active high or active low clock, and has no significant effect on the transfer format. The clock phase (CPHA) control bit selects one of two different transfer formats. The clock phase and polarity should be identical for the master SPI device and the communicating slave device. In some cases, the phase and polarity are changed between transfers to allow a master device to communicate with peripheral slaves having different requirements.

When CPHA equals 0, the $\overline{SS}$ line must be negated and reasserted between each successive serial byte. Also, if the slave writes data to the SPI data register (SPDR) while $\overline{SS}$ is low, a write collision error results.

When CPHA equals 1, the $\overline{SS}$ line can remain low between successive transfers.

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.

Register name: Timer Output Compare 3 Register (High) Address: $101A

	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 Output Compare 3 Register (Low) Address: $101B

	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 9-10. Timer Output Compare 3 Register Pair (TOC3)

Register name: Timer Output Compare 4 Register (High) Address: $101C

	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 Output Compare 4 Register (Low) Address: $101D

	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 9-11. Timer Output Compare 4 Register Pair (TOC4)

9.4.2 Timer Compare Force Register

The CFORC register allows forced early compares. FOC[1:5] correspond to the five output compares. These bits are set for each output compare that is to be forced. The action taken as a result of a forced compare is the same as if there were a match between the OCx register and the free-running counter, except that the corresponding interrupt status flag bits are not set. The forced channels trigger their programmed pin actions to occur at the next timer count transition after the write to CFORC.

The CFORC bits should not be used on an output compare function that is programmed to toggle its output on a successful compare because a normal compare that occurs immediately before or after the force can result in an undesirable operation.

Address: $100B

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	FOC1	FOC2	FOC3	FOC4	FOC5			
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 9-12. Timer Compare Force Register (CFORC)

9.7.1 Pulse Accumulator Control Register

Four of this register's bits control an 8-bit pulse accumulator system. Another bit enables either the OC5 function or the IC4 function, while two other bits select the rate for the real-time interrupt system.

Address: $1026

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	DDRA7	PAEN	PAMOD	PEDGE	DDRA3	I4/O5	RTR1	RTR0
Reset:	0	0	0	0	0	0	0	0

Figure 9-25. Pulse Accumulator Control Register (PACTL)

DDRA7 — Data Direction for Port A Bit 7

Refer to Chapter 6 Parallel Input/Output (I/O) Ports.

PAEN — Pulse Accumulator System Enable Bit

0 = Pulse accumulator disabled
1 = Pulse accumulator enabled

PAMOD — Pulse Accumulator Mode Bit

0 = Event counter
1 = Gated time accumulation

PEDGE — Pulse Accumulator Edge Control Bit

This bit has different meanings depending on the state of the PAMOD bit, as shown in Table 9-7.

Table 9-7. Pulse Accumulator Edge Control

PAMOD	PEDGE	Action on Clock
0	0	PAI falling edge increments the counter.
0	1	PAI rising edge increments the counter.
1	0	A 0 on PAI inhibits counting.
1	1	A 1 on PAI inhibits counting.

DDRA3 — Data Direction for Port A Bit 3

Refer to Chapter 6 Parallel Input/Output (I/O) Ports.

I4/O5 — Input Capture 4/Output Compare 5 Bit

0 = Output compare 5 function enable (no IC4)
1 = Input capture 4 function enable (no OC5)

RTR[1:0] — RTI Interrupt Rate Select Bits

Refer to 9.5 Real-Time Interrupt (RTI).

10.9 Control Timing

Characteristic[1] [2]	Symbol	1.0 MHz		2.0 MHz		3.0 MHz		Unit
		Min	Max	Min	Max	Min	Max	
Frequency of operation	f_o	dc	1.0	dc	2.0	dc	3.0	MHz
E-clock period	t_{CYC}	1000	—	500	—	333	—	ns
Crystal frequency	f_{XTAL}	—	4.0	—	8.0	—	12.0	MHz
External oscillator frequency	4 f_o	dc	4.0	dc	8.0	dc	12.0	MHz
Processor control setup time t_{PCSU} = 1/4 t_{CYC}+ 50 ns	t_{PCSU}	300	—	175	—	133	—	ns
Reset input pulse width To guarantee external reset vector Minimum input time (can be pre-empted by internal reset)	PW_{RSTL}	8 1	— —	8 1	— —	8 1	— —	t_{CYC}
Mode programming setup time	t_{MPS}	2	—	2	—	2	—	t_{CYC}
Mode programming hold time	t_{MPH}	10	—	10	—	10	—	ns
Interrupt pulse width, $\overline{IRQ}$ edge-sensitive mode PW_{IRQ} = t_{CYC} + 20 ns	PW_{IRQ}	1020	—	520	—	353	—	ns
Wait recovery startup time	t_{WRS}	—	4	—	4	—	4	t_{CYC}
Timer pulse width input capture pulse accumulator input PW_{TIM} = t_{CYC} + 20 ns	PW_{TIM}	1020	—	520	—	353	—	ns

1. V_{DD} = 5.0 Vdc ±10%, V_{SS} = 0 Vdc, T_A = T_L to T_H, all timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted
2. $\overline{RESET}$ is recognized during the first clock cycle it is held low. Internal circuitry then drives the pin low for four clock cycles, releases the pin, and samples the pin level two cycles later to determine the source of the interrupt. Refer to Chapter 5 Resets and Interrupts for further detail.

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

A.4 Modular Development System (MMDS11)

The M68MMDS11 modular development system (MMDS11) is an emulator system for developing embedded systems based on an M68HC11 microcontroller unit (MCU). The MMDS11 provides a bus state analyzer (BSA) and real-time memory windows. The unit's integrated development environment includes an editor, an assembler, user interface, and source-level debug. These features significantly reduce the time necessary to develop and debug an embedded MCU system. The unit's compact size requires a minimum of desk space.

The MMDS11 is one component of Freescale's modular approach to MCU-based product development. This modular approach allows easy configuration of the MMDS11 to fit a wide range of requirements. It also reduces development system cost by allowing the user to purchase only the modular components necessary to support the particular MCU derivative.

MMDS11 features include:

- Real-time, non-intrusive, in-circuit emulation at the MCU’s operating frequency
- Real-time bus state analyzer
  - 8 K x 64 real-time trace buffer
  - Display of real-time trace data as raw data, disassembled instructions, raw data and disassembled instructions, or assembly-language source code
  - Four hardware triggers for commencing trace and to provide breakpoints
  - Nine triggering modes
  - As many as 8190 pre- or post-trigger points for trace data
  - 16 general-purpose logic clips, four of which can be used to trigger the bus state analyzer sequencer
  - 16-bit time tag or an optional 24-bit time tag that reduces the logic clips traced from 16 to eight
- Four data breakpoints (hardware breakpoints)
- Hardware instruction breakpoints over either the 64-Kbyte M68HC11 memory map or over a 1-Mbyte bank switched memory map
- 32 real-time variables, nine of which can be displayed in the variables window. These variables may be read or written while the MCU is running
- 32 bytes of real-time memory can be displayed in the memory window. This memory may be read or written while the MCU is running
- 64 Kbytes of fast emulation memory (SRAM)
- Current-limited target input/output connections
- Six software-selectable oscillator clock sources: five internally generated frequencies and an external frequency via a bus analyzer logic clip
- Command and response logging to MS-DOS® disk files to save session history
- SCRIPT command for automatic execution of a sequence of MMDS11 commands
- Assembly or C-language source-level debugging with global variable viewing
- Host/emulator communications speeds as high as 57,600 baud for quick program loading

® MS-DOS is a registered trademark of Microsoft Corporation.

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 or use it as an example.

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 also must 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 0 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.

```
27                ************************************************
28                *
29 B600 7F103D   BEGIN      CLR    INIT             Moves Registers to $0000-3F
30 B603 8604               LDAA   #$04             Pattern for DWOM off, no SPI
31 B605 9728               STAA   SPCR             Turns off DWOM in EVBU MCU
32 B607 8680               LDAA   #RESET
33 B609 9704               STAA   PORTB            Release reset to target MCU
34 B60B 132E20FC WT4BRK    BRCLR  SCSR RDRF WT4BRK Loop till char received
35 B60F 86FF               LDAA   #$FF             Leading char for bootload ...
36 B611 972F               STAA   SCDR             to target MCU
37 B613 CEB675             LDX    #BLPROG          Point at program for target
38 B616 8D53     BLLOOP    BSR    SEND1            Bootload to target
39 B618 8CB67D             CPX    #ENDBPR          Past end ?
40 B61B 26F9               BNE    BLLOOP           Continue till all sent
41                *****
42                * Delay for about 4 char times to allow boot related
43                * SCI communications to finish before clearing
44                * Rx related flags
45 B61D CE06A7             LDX    #1703            # of 6 cyc loops
46 B620 09       DLYLP     DEX                     [3]
47 B621 26FD               BNE    DLYLP            [3] Total loop time = 6 cyc
48 B623 962E               LDAA   SCSR             Read status (RDRF will be set)
49 B625 962F               LDAA   SCDR             Read SCI data reg to clear RDRF
50                *****
51                * Now wait for character from target to indicate it's ready for
52                * data to be programmed into EPROM
53 B627 132E20FC WT4FF     BRCLR  SCSR RDRF WT4FF  Wait for RDRF
54 B62B 962F               LDAA   SCDR             Clear RDRF, don't need data
55 B62D CED000             LDX    #EPSTRT          Point at start of EPROM
56                * Handle turn-on of Vpp
57 B630 18CE523D WT4VPP    LDY    #21053           Delay counter (about 200ms)
58 B634 150402             BCLR   PORTB RED        Turn off RED LED
59 B637 960A     DLYLP2    LDAA   PORTE            [3] Wait for Vpp to be ON
60 B639 2AF5               BPL    WT4VPP           [3] Vpp sense is on port E MSB
61 B63B 140402             BSET   PORTB RED        [6] Turn on RED LED
62 B63E 1809               DEY                     [4]
63 B640 26F5               BNE    DLYLP2           [3] Total loop time = 19 cyc
64                * Vpp has been stable for 200ms
65
66 B642 18CED000           LDY    #EPSTRT          X=Tx pointer, Y=verify pointer
67 B646 8D23               BSR    SEND1            Send first data to target
68 B648 8C0000   DATALP    CPX    #0               X points at $0000 after last
69 B64B 2702               BEQ    VERF             Skip send if no more
70 B64D 8D1C               BSR    SEND1            Send another data char
71 B64F 132E20FC VERF      BRCLR  SCSR RDRF VERF   Wait for Rx ready
72 B653 962F               LDAA   SCDR             Get char and clr RDRF
73 B655 18A100             CMPA   0,Y              Does char verify ?
74 B658 2705               BEQ    VERFOK           Skip error if OK
75 B65A 150403             BCLR   PORTB (RED+GREEN) Turn off LEDs
76 B65D 2007               BRA    DUNPRG           Done (programming failed)
77 B65F
78 B65F 1808     VERFOK    INY                     Advance verify pointer
79 B661 26E5               BNE    DATALP           Continue till all done
80 B663
81 B663 140401             BSET   PORTB GREEN      Grn LED ON
```

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 2-Mbyte Macintosh and a 640-Kbyte 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:

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.

Listing 3. MC68HC711E9 Bootloader ROM

```
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
```

Listing 3. MC68HC711E9 Bootloader ROM

Freescale Semiconductor
Engineering Bulletin

EB188
Rev. 0.1, 07/2005

Enabling the Security Feature on M68HC811E2 Devices with PCbug11 on the M68HC711E9PGMR

By **Edgar Saenz**
Austin, Texas

Introduction

The PCbug11 software, needed along with the M68HC711E9PGMR to program MC68HC811E2 devices, is available from the download section of the Microcontroller Worldwide Web site

http://www.freescale.com

Retrieve the file pcbug342.exe (a self-extracting archive) from the MCU11 directory.

Some Freescale evaluation board products also are shipped with PCbug11.

NOTE

For specific information about any of the PCbug11 commands, see the appropriate sections in the PCbug11 User's Manual (part number M68PCBUG11/D2), which is available from the Freescale Literature http://www.freescale.com. The file is also on the software download system and is called pcbug11.pdf.