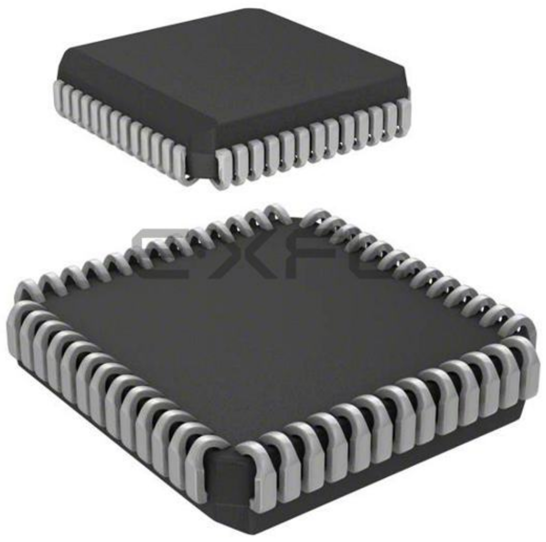


What is "Embedded - Microcontrollers"?

"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	2MHz
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)	3V ~ 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/mc68l11e1cfne2

1.4.15 Port D

Pins PD5–PD0 can be used for general-purpose I/O signals. These pins alternately serve as the serial communication interface (SCI) and serial peripheral interface (SPI) signals when those subsystems are enabled.

- PD0 is the receive data input (RxD) signal for the SCI.
- PD1 is the transmit data output (TxD) signal for the SCI.
- PD5–PD2 are dedicated to the SPI:
  - PD2 is the master in/slave out (MISO) signal.
  - PD3 is the master out/slave in (MOSI) signal.
  - PD4 is the serial clock (SCK) signal.
  - PD5 is the slave select ($\overline{\text{SS}}$) input.

1.4.16 Port E

Use port E for general-purpose or analog-to-digital (A/D) inputs.

CAUTION

If high accuracy is required for A/D conversions, avoid reading port E during sampling, as small disturbances can reduce the accuracy of that result.

2.3.3.1 System Configuration Register

The system configuration register (CONFIG) consists of an EEPROM byte and static latches that control the startup configuration of the MCU. The contents of the EEPROM byte are transferred into static working latches during reset sequences. The operation of the MCU is controlled directly by these latches and not by CONFIG itself. In normal modes, changes to CONFIG do not affect operation of the MCU until after the next reset sequence. When programming, the CONFIG register itself is accessed. When the CONFIG register is read, the static latches are accessed. See 2.5.1 EEPROM and CONFIG Programming and Erasure for information on modifying CONFIG.

To take full advantage of the MCU's functionality, customers can program the CONFIG register in bootstrap mode. This can be accomplished by setting the mode pins to logic 0 and downloading a small program to internal RAM. For more information, Freescale application note AN1060 entitled M68HC11 Bootstrap Mode has been included at the back of this document. The downloadable talker will consist of:

- Bulk erase
- Byte programming
- Communication server

All of this functionality is provided by PCbug11 which can be found on the Freescale Web site at http://www.freescale.com. For more information on using PCbug11 to program an E-series device, Freescale engineering bulletin EB296 entitled Programming MC68HC711E9 Devices with PCbug11 and the M68HC11EVBU has been included at the back of this document.

NOTE

The CONFIG register on the 68HC11 is an EEPROM cell and must be programmed accordingly.

Operation of the CONFIG register in the MC68HC811E2 differs from other devices in the M68HC11 E series. See Figure 2-10 and Figure 2-11.

Address: $103F

	Bit 7	6	5	4	3	2	1	Bit 0
Read: / Write:	▨	▨	▨	▨	NOSEC	NOCOP	ROMON	EEON
Resets:								
Single chip:	0	0	0	0	U	U	1	U
Bootstrap:	0	0	0	0	U	U(L)	U	U
Expanded:	0	0	0	0	1	U	U	U
Test:	0	0	0	0	1	U(L)	U	U

▨ = Unimplemented

U indicates a previously programmed bit. U(L) indicates that the bit resets to the logic level held in the latch prior to reset, but the function of COP is controlled by the DISR bit in TEST1 register.

Figure 2-10. System Configuration Register (CONFIG)

2.4.3 EPROM and EEPROM Programming Control Register

The EPROM and EEPROM programming control register (PPROG) enables the EPROM programming voltage and controls the latching of data to be programmed.

- For MC68HC711E9, PPROG is also the EEPROM programming control register.
- For the MC68HC711E20, EPROM programming is controlled by the EPROG register and EEPROM programming is controlled by the PPROG register.

Address: $103B

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	ODD	EVEN	ELAT(1)	BYTE	ROW	ERASE	EELAT	EPGM
Reset:	0	0	0	0	0	0	0	0

1. MC68HC711E9 only

Figure 2-14. EPROM and EEPROM Programming Control Register (PPROG)

ODD — Program Odd Rows in Half of EEPROM (Test) Bit
Refer to 2.5 EEPROM.

EVEN — Program Even Rows in Half of EEPROM (Test) Bit
Refer to 2.5 EEPROM.

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 EPGM = 1; then the write to ELAT is disabled.
0 = EPROM address and data bus configured for normal reads
1 = EPROM address and data bus configured for programming

For the MC68HC711E9:
a. EPGM enables the high voltage necessary for both EEPROM and EPROM/OTPROM programming.
b. ELAT and EELAT are mutually exclusive and cannot both equal 1.

BYTE — Byte/Other EEPROM Erase Mode Bit
Refer to 2.5 EEPROM.

ROW — Row/All EEPROM Erase Mode Bit
Refer to 2.5 EEPROM.

ERASE — Erase Mode Select Bit
Refer to 2.5 EEPROM.

EELAT — EEPROM Latch Control Bit
Refer to 2.5 EEPROM.

EPGM —EPROM/OTPROM/EEPROM Programming Voltage Enable Bit
EPGM can be read any time and can be written only when ELAT = 1 (for EPROM/OTPROM programming) or when EELAT = 1 (for EEPROM programming).
0 = Programming voltage to EPROM/OTPROM/EEPROM array disconnected
1 = Programming voltage to EPROM/OTPROM/EEPROM array connected

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

PGM — EPROM Programming Voltage Enable Bit

PGM can be read any time and can be written only when ELAT = 1.

0 = Programming voltage to EPROM array disconnected

1 = Programming voltage to EPROM array connected

2.5 EEPROM

Some E-series devices contain 512 bytes of on-chip EEPROM. The MC68HC811E2 contains 2048 bytes of EEPROM with selectable base address. All E-series devices contain the EEPROM-based CONFIG register.

2.5.1 EEPROM and CONFIG Programming and Erasure

The erased state of an EEPROM bit is 1. During a read operation, bit lines are precharged to 1. The floating gate devices of programmed bits conduct and pull the bit lines to 0. Unprogrammed bits remain at the precharged level and are read as ones. Programming a bit to 1 causes no change. Programming a bit to 0 changes the bit so that subsequent reads return 0.

When appropriate bits in the BPROT register are cleared, the PPROG register controls programming and erasing the EEPROM. The PPROG register can be read or written at any time, but logic enforces defined programming and erasing sequences to prevent unintentional changes to EEPROM data. When the EELAT bit in the PPROG register is cleared, the EEPROM can be read as if it were a ROM.

The on-chip charge pump that generates the EEPROM programming voltage from V_{DD} uses MOS capacitors, which are relatively small in value. The efficiency of this charge pump and its drive capability are affected by the level of V_{DD} and the frequency of the driving clock. The load depends on the number of bits being programmed or erased and capacitances in the EEPROM array.

The clock source driving the charge pump is software selectable. When the clock select (CSEL) bit in the OPTION register is 0, the E clock is used; when CSEL is 1, an on-chip resistor-capacitor (RC) oscillator is used.

The EEPROM programming voltage power supply voltage to the EEPROM array is not enabled until there has been a write to PPROG with EELAT set and PGM cleared. This must be followed by a write to a valid EEPROM location or to the CONFIG address, and then a write to PPROG with both the EELAT and EPGM bits set. Any attempt to set both EELAT and EPGM during the same write operation results in neither bit being set.

2.5.1.1 Block Protect Register

This register prevents inadvertent writes to both the CONFIG register and EEPROM. The active bits in this register are initialized to 1 out of reset and can be cleared only during the first 64 E-clock cycles after reset in the normal modes. When these bits are cleared, the associated EEPROM section and the CONFIG register can be programmed or erased. EEPROM is only visible if the EEON bit in the CONFIG register is set. The bits in the BPROT register can be written to 1 at any time to protect EEPROM and the CONFIG register. In test or bootstrap modes, write protection is inhibited and BPROT can be written repeatedly. Address ranges for protected areas of EEPROM differ significantly for the MC68HC811E2. Refer to Figure 2-16.

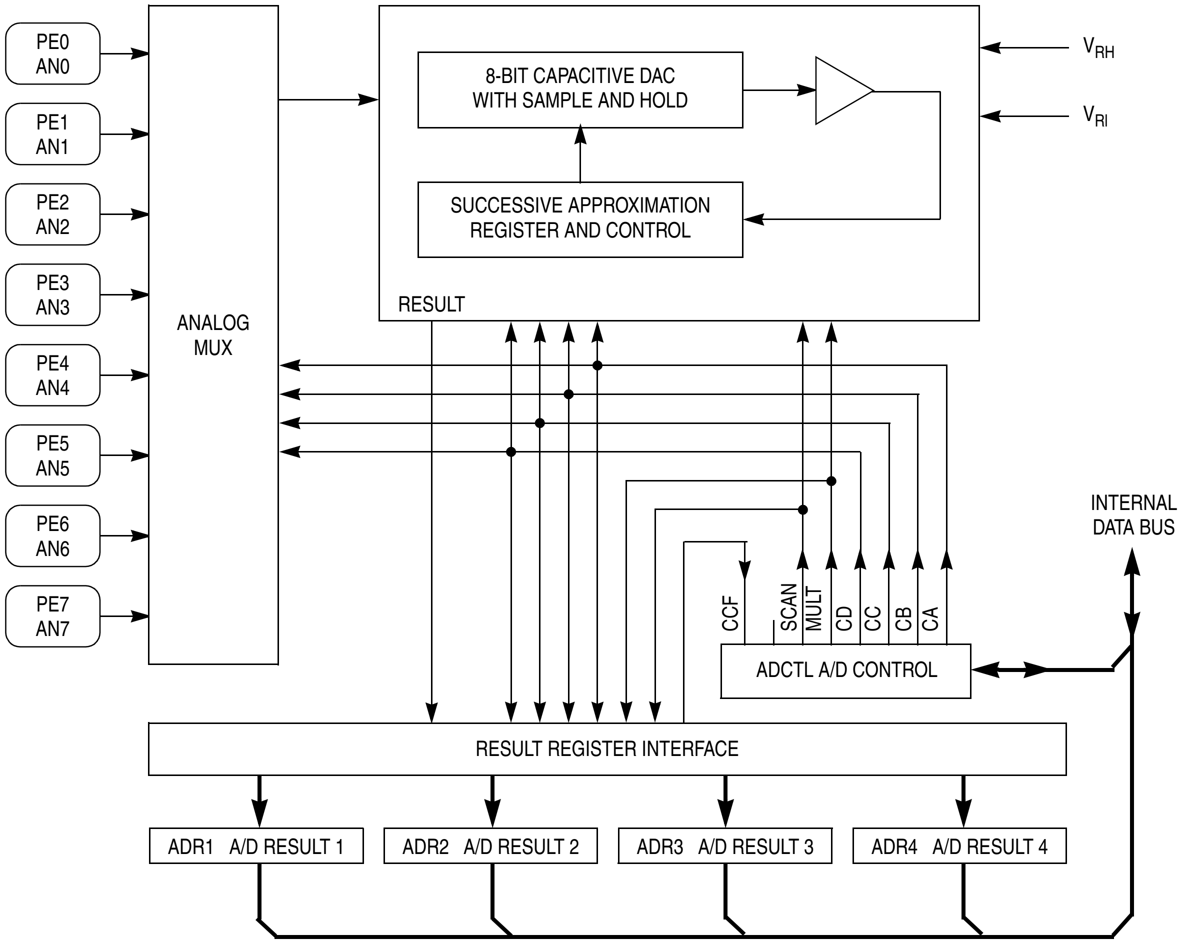
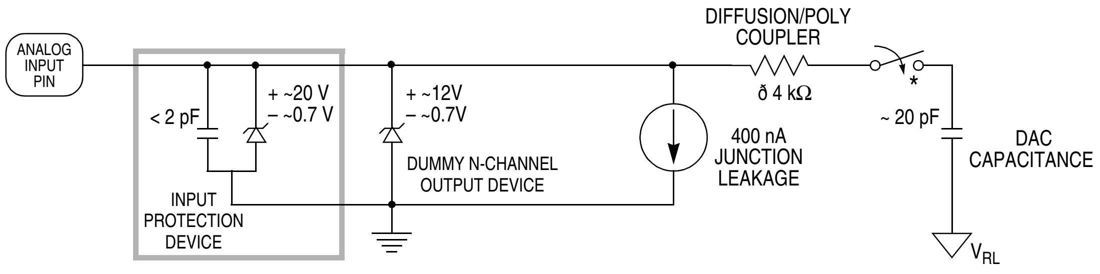


Figure 3-1. A/D Converter Block Diagram



* 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)

6.6 Port E

Port E is used for general-purpose static inputs or pins that share functions with the analog-to-digital (A/D) converter system. When some port E pins are being used for general-purpose input and others are being used as A/D inputs, PORTE should not be read during the sample portion of an A/D conversion.

Address:	$100A							
	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
Reset:	Indeterminate after reset							
Alternate Function:	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0

Figure 6-9. Port E Data Register (PORTE)

6.7 Handshake Protocol

Simple and full handshake input and output functions are available on ports B and C pins in single-chip mode. In simple strobed mode, port B is a strobed output port and port C is a latching input port. The two activities are available simultaneously.

The STRB output is pulsed for two E-clock periods each time there is a write to the PORTB register. The INVB bit in the PIOC register controls the polarity of STRB pulses. Port C levels are latched into the alternate port C latch (PORTCL) register on each assertion of the STRA input. STRA edge select, flag, and interrupt enable bits are located in the PIOC register. Any or all of the port C lines can still be used as general-purpose I/O while in strobed input mode.

Full handshake modes use port C pins and the STRA and STRB lines. Input and output handshake modes are supported, and output handshake mode has a 3-stated variation. STRA is an edge-detecting input and STRB is a handshake output. Control and enable bits are located in the PIOC register.

In full input handshake mode, the MCU asserts STRB to signal an external system that it is ready to latch data. Port C logic levels are latched into PORTCL when the STRA line is asserted by the external system. The MCU then negates STRB. The MCU reasserts STRB after the PORTCL register is read. In this mode, a mix of latched inputs, static inputs, and static outputs is allowed on port C, differentiated by the data direction bits and use of the PORTC and PORTCL registers.

In full output handshake mode, the MCU writes data to PORTCL which, in turn, asserts the STRB output to indicate that data is ready. The external system reads port C data and asserts the STRA input to acknowledge that data has been received.

In the 3-state variation of output handshake mode, lines intended as 3-state handshake outputs are configured as inputs by clearing the corresponding DDRC bits. The MCU writes data to PORTCL and asserts STRB. The external system responds by activating the STRA input, which forces the MCU to drive the data in PORTC out on all of the port C lines. After the trailing edge of the active signal on STRA, the MCU negates the STRB signal. The 3-state mode variation does not allow part of port C to be used for static inputs while other port C pins are being used for handshake outputs. Refer to the 6.8 Parallel I/O Control Register for further information.

Chapter 8
Serial Peripheral Interface (SPI)

8.1 Introduction

The serial peripheral interface (SPI), an independent serial communications subsystem, allows the MCU to communicate synchronously with peripheral devices, such as:

- Frequency synthesizers
- Liquid crystal display (LCD) drivers
- Analog-to-digital (A/D) converter subsystems
- Other microprocessors

The SPI is also capable of inter-processor communication in a multiple master system. The SPI system can be configured as either a master or a slave device. When configured as a master, data transfer rates can be as high as one-half the E-clock rate (1.5 Mbits per second for a 3-MHz bus frequency). When configured as a slave, data transfers can be as fast as the E-clock rate (3 Mbits per second for a 3-MHz bus frequency).

8.2 Functional Description

The central element in the SPI system is the block containing the shift register and the read data buffer. The system is single buffered in the transmit direction and double buffered in the receive direction. This means that new data for transmission cannot be written to the shifter until the previous transfer is complete; however, received data is transferred into a parallel read data buffer so the shifter is free to accept a second serial character. As long as the first character is read out of the read data buffer before the next serial character is ready to be transferred, no overrun condition occurs. A single MCU register address is used for reading data from the read data buffer and for writing data to the shifter.

The SPI status block represents the SPI status functions (transfer complete, write collision, and mode fault) performed by the serial peripheral status register (SPSR). The SPI control block represents those functions that control the SPI system through the serial peripheral control register (SPCR).

Refer to Figure 8-1, which shows the SPI block diagram.

8.3 SPI Transfer Formats

During an SPI transfer, data is simultaneously transmitted and received. A serial clock line synchronizes shifting and sampling of the information on the two serial data lines. A slave select line allows individual selection of a slave SPI device; slave devices that are not selected do not interfere with SPI bus activities. On a master SPI device, the select line can optionally be used to indicate a multiple master bus contention. Refer to Figure 8-2.

A write collision is normally a slave error because a slave has no control over when a master initiates a transfer. A master knows when a transfer is in progress, so there is no reason for a master to generate a write-collision error, although the SPI logic can detect write collisions in both master and slave devices.

The SPI configuration determines the characteristics of a transfer in progress. For a master, a transfer begins when data is written to SPDR and ends when SPIF is set. For a slave with CPHA equal to 0, a transfer starts when $\overline{SS}$ goes low and ends when $\overline{SS}$ returns high. In this case, SPIF is set at the middle of the eighth SCK cycle when data is transferred from the shifter to the parallel data register, but the transfer is still in progress until $\overline{SS}$ goes high. For a slave with CPHA equal to 1, transfer begins when the SCK line goes to its active level, which is the edge at the beginning of the first SCK cycle. The transfer ends in a slave in which CPHA equals 1 when SPIF is set.

8.7 SPI Registers

The three SPI registers are:

- Serial peripheral control register (SPCR)
- Serial peripheral status register (SPSR)
- Serial peripheral data register (SPDR)

These registers provide control, status, and data storage functions.

8.7.1 Serial Peripheral Control Register

Address: $1028

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0
Reset:	0	0	0	0	0	1	U	U

U = Unaffected

Figure 8-3. Serial Peripheral Control Register (SPCR)

SPIE — Serial Peripheral Interrupt Enable Bit

Set the SPE bit to 1 to request a hardware interrupt sequence each time the SPIF or MODF status flag is set. SPI interrupts are inhibited if this bit is clear or if the I bit in the condition code register is 1.

0 = SPI system interrupts disabled
1 = SPI system interrupts enabled

SPE — Serial Peripheral System Enable Bit

When the SPE bit is set, the port D bit 2, 3, 4, and 5 pins are dedicated to the SPI function. If the SPI is in the master mode and DDRD bit 5 is set, then the port D bit 5 pin becomes a general-purpose output instead of the $\overline{SS}$ input.

0 = SPI system disabled
1 = SPI system enabled

DWOM — Port D Wired-OR Mode Bit

DWOM affects all port D pins.

0 = Normal CMOS outputs
1 = Open-drain outputs

MSTR — Master Mode Select Bit

It is customary to have an external pullup resistor on lines that are driven by open-drain devices.
- 0 = Slave mode
- 1 = Master mode

CPOL — Clock Polarity Bit

When the clock polarity bit is cleared and data is not being transferred, the SCK pin of the master device has a steady state low value. When CPOL is set, SCK idles high. Refer to Figure 8-2 and 8.4 Clock Phase and Polarity Controls.

CPHA — Clock Phase Bit

The clock phase bit, in conjunction with the CPOL bit, controls the clock-data relationship between master and slave. The CPHA bit selects one of two different clocking protocols. Refer to Figure 8-2 and 8.4 Clock Phase and Polarity Controls.

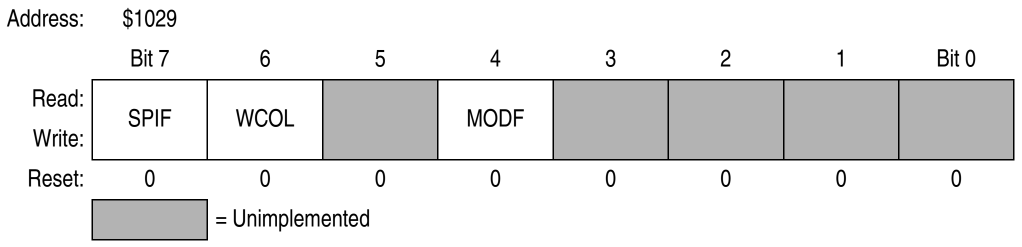
SPR[1:0] — SPI Clock Rate Select Bits

These two bits select the SPI clock (SCK) rate when the device is configured as master. When the device is configured as slave, these bits have no effect. Refer to Table 8-1.

Table 8-1. SPI Clock Rates

SPR[1:0]	Divide E Clock By	Frequency at E = 1 MHz (Baud)	Frequency at E = 2 MHz (Baud)	Frequency at E = 3 MHz (Baud)	Frequency at E = 4 MHz (Baud)
0 0	2	500 kHz	1.0 MHz	1.5 MHz	2 MHz
0 1	4	250 kHz	500 kHz	750 kHz	1 MHz
1 0	16	62.5 kHz	125 kHz	187.5 kHz	250 kHz
1 1	32	31.3 kHz	62.5 kHz	93.8 kHz	125 kHz

8.7.2 Serial Peripheral Status Register



Address: $1029

	Bit 7	6	5	4	3	2	1	Bit 0
Read: / Write:	SPIF	WCOL	▨	MODF	▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 8-4. Serial Peripheral Status Register (SPSR)

SPIF — SPI Interrupt Complete Flag

SPIF is set upon completion of data transfer between the processor and the external device. If SPIF goes high, and if SPIE is set, a serial peripheral interrupt is generated. To clear the SPIF bit, read the SPSR with SPIF set, then access the SPDR. Unless SPSR is read (with SPIF set) first, attempts to write SPDR are inhibited.

WCOL — Write Collision Bit

Clearing the WCOL bit is accomplished by reading the SPSR (with WCOL set) followed by an access of SPDR. Refer to 8.5.4 Slave Select and 8.6 SPI System Errors.
- 0 = No write collision
- 1 = Write collision

independent of the software latencies associated with flag clearing and service. For this reason, an RTI period starts from the previous timeout, not from when RTIF is cleared.

Every timeout causes the RTIF bit in TFLG2 to be set, and if RTII is set, an interrupt request is generated. After reset, one entire RTI period elapses before the RTIF is set for the first time. Refer to the 9.4.9 Timer Interrupt Mask 2 Register, 9.5.2 Timer Interrupt Flag Register 2, and 9.5.3 Pulse Accumulator Control Register.

9.5.1 Timer Interrupt Mask Register 2

This register contains the real-time interrupt enable bits.

Address: $1024

	Bit 7	6	5	4	3	2	1	Bit 0
Read: / Write:	TOI	RTI	PAOVI	PAII	▨	▨	PR1	PR0
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 9-21. Timer Interrupt Mask 2 Register (TMSK2)

TOI — Timer Overflow Interrupt Enable Bit
0 = TOF interrupts disabled
1 = Interrupt requested when TOF is set to 1

RTII — Real-Time Interrupt Enable Bit
0 = RTIF interrupts disabled
1 = Interrupt requested when RTIF set to 1

PAOVI — Pulse Accumulator Overflow Interrupt Enable Bit
Refer to 9.7 Pulse Accumulator.

PAII — Pulse Accumulator Input Edge Bit
Refer to 9.7 Pulse Accumulator.

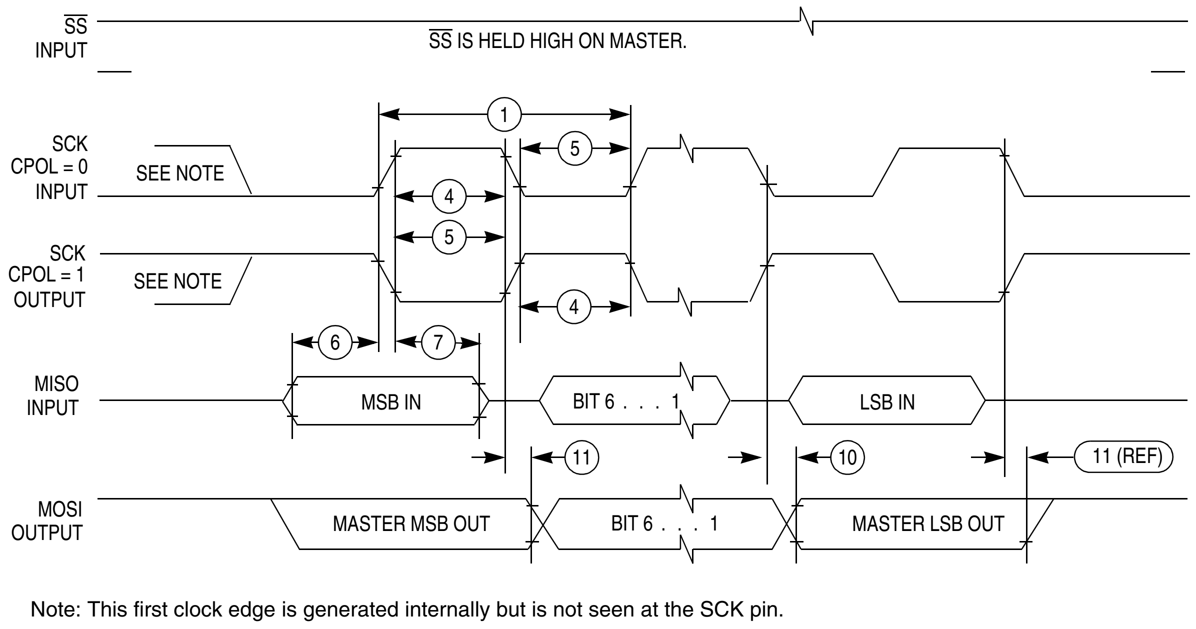
Bits [3:2] — Unimplemented
Always read 0

PR[1:0] — Timer Prescaler Select Bits
Refer to Table 9-4.

NOTE
Bits in TMSK2 correspond bit for bit with flag bits in TFLG2. Bits in TMSK2 enable the corresponding interrupt sources.

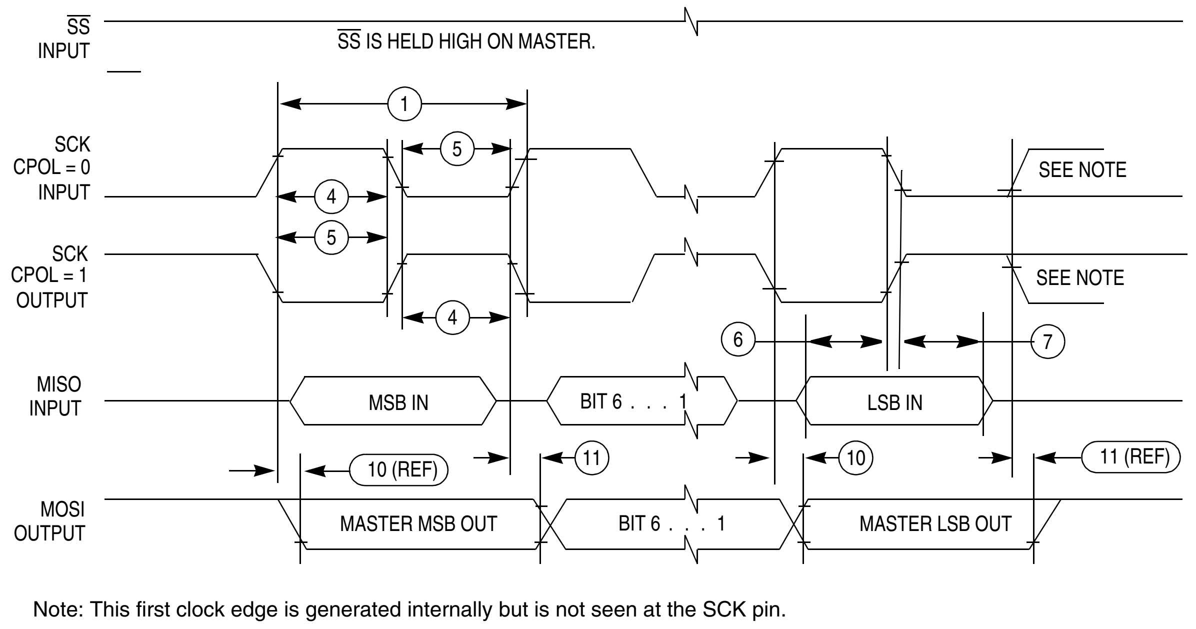
PAII and PAIF — Pulse Accumulator Input Edge Interrupt Enable Bit and Flag

The PAIF status bit is automatically set each time a selected edge is detected at the PA7/PAI/OC1 pin. To clear this status bit, write to the TFLG2 register with a 1 in the corresponding data bit position (bit 4). The PAII control bit allows configuring the pulse accumulator input edge detect for polled or interrupt-driven operation but does not affect setting or clearing the PAIF bit. When PAII is 0, pulse accumulator input interrupts are inhibited, and the system operates in a polled mode. In this mode, the PAIF bit must be polled by user software to determine when an edge has occurred. When the PAII control bit is set, a hardware interrupt request is generated each time PAIF is set. Before leaving the interrupt service routine, software must clear PAIF by writing to the TFLG2 register.



Note: This first clock edge is generated internally but is not seen at the SCK pin.

A) SPI Master Timing (CPHA = 0)



Note: This first clock edge is generated internally but is not seen at the SCK pin.

B) SPI Master Timing (CPHA = 1)

Figure 10-15. SPI Timing Diagram (Sheet 1 of 2)

Description	Temperature	Frequency	MC Order Number
20 Kbytes custom ROM	0°C to +70°°C	3 MHz	MC68HC11E20FN3
	–40°C to +85°C	2 MHz	MC68HC11E20CFN2
		3 MHz	MC68HC11E20CFN3
	–40°C to +105°C	2 MHz	MC68HC11E20VFN2
	–40°C to +125°C	2 MHz	MC68HC11E20MFN2

64-pin quad flat pack (QFP)

Custom ROM	0°C to +70°°C	3 MHz	MC68HC11E9FU3
	–40°C to +85°C	2 MHz	MC68HC11E9CFU2
		3 MHz	MC68HC11E9CFU3
	–40°C to +105°C	2 MHz	MC68HC11E9VFU2
	–40°C to +125°C	2 MHz	MC68HC11E9MFU2

64-pin quad flat pack (continued)

20 Kbytes Custom ROM	0°C to +70°°C	3 MHz	MC68HC11E20FU3
	–40°C to +85°C	2 MHz	MC68HC11E20CFU2
		3 MHz	MC68HC11E20CFU3
	–40°C to +105°C	2 MHz	MC68HC11E20VFU2
	–40°C to +125°C	2 MHz	MC68HC11E20MFU2

52-pin thin quad flat pack (10 mm x 10 mm)

Custom ROM	0°C to +70°°C	3 MHz	MC68HC11E9PB3
	–40°C to +85°C	2 MHz	MC68HC11E9CPB2
		3 MHz	MC68HC11E9CPB3
	–40°C to +105°C	2 MHz	MC68HC11E9VPB2
	–40°C to +125°C	2 MHz	MC68HC11E9MPB2

56-pin dual in-line package with 0.70-inch lead spacing (SDIP)

Custom ROM	0°C to +70°°C	3 MHz	MC68HC11E9B3
	–40°C to +85°C	2 MHz	MC68HC11E9CB2
		3 MHz	MC68HC11E9CB3
	–40°C to +105°C	2 MHz	MC68HC11E9VB2
	–40°C to +125°C	2 MHz	MC68HC11E9MB2

11.4 Extended Voltage Device Ordering Information (3.0 Vdc to 5.5 Vdc)

Description	Temperature	Frequency	MC Order Number
52-pin plastic leaded chip carrier (PLCC)			
Custom ROM	–20°C to +70°C	2 MHz	MC68L11E9FN2 MC68L11E20FN2
No ROM		2 MHz	MC68L11E1FN2
No ROM, no EEPROM		2 MHz	MC68L11E0FN2
64-pin quad flat pack (QFP)			
Custom ROM	–20°C to +70°C	2 MHz	MC68L11E9FU2 MC68L11E20FU2
No ROM		2 MHz	MC68L11E1FU2
No ROM, no EEPROM		2 MHz	MC68L11E0FU2
52-pin thin quad flat pack (10 mm x 10 mm)			
Custom ROM	–20°C to +70°C	2 MHz	MC68L11E9PB2
No ROM		2 MHz	MC68L11E1PB2
No ROM, no EEPROM		2 MHz	MC68L11E0PB2
56-pin dual in-line package with 0.70-inch lead spacing (SDIP)			
Custom ROM	–20°C to +70°C	2 MHz	MC68L11E9B2
No ROM		2 MHz	MC68L11E1B2
No ROM, no EEPROM		2 MHz	MC68L11E0B2

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.

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 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
```

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 character to be $00 (or break). The data reception logic in the SCI looks for a 1-to-0 transition on the RxD pin to synchronize to the beginning of a receive character. If the RxD pin is tied to ground, no 1-to-0 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.

$FF Character Is Required before Loading 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 pullup resistor.