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	eZ8
Core Size	8-Bit
Speed	20MHz
Connectivity	I²C, IrDA, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LED, LVD, POR, PWM, Temp Sensor, WDT
Number of I/O	23
Program Memory Size	16KB (16K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	3K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z8f1680sj020eg

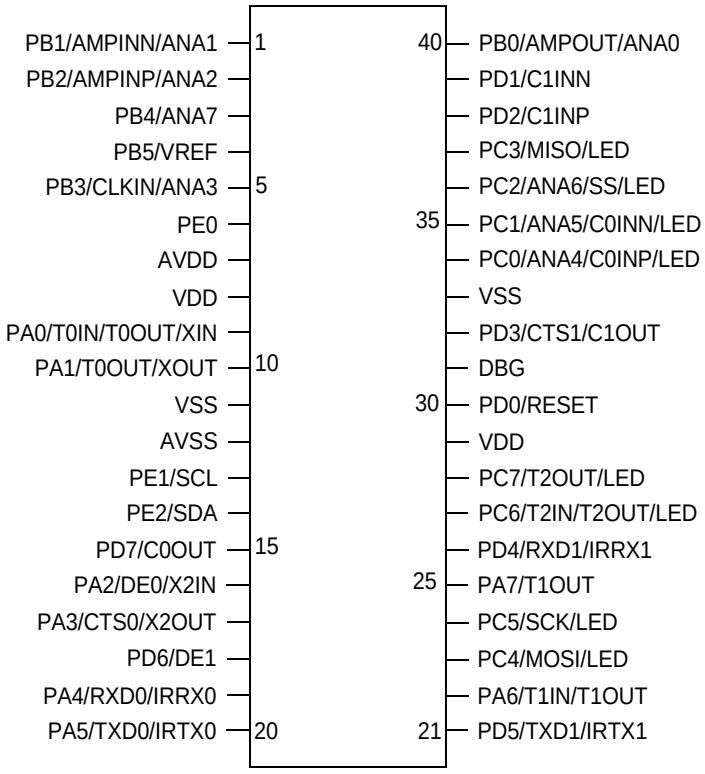

Figure 4. Z8F2480, Z8F1680 and Z8F0880 in 40-Pin Dual Inline Package (PDIP)

Table 5. Pin Characteristics (20-, 28-, 40- and 44-pin Devices) (Continued)

Symbol Mnemonic	Direction	Reset Direction	Active Low or Active High	Tristate Output	Internal Pull-up or Pull-down	Schmitt Trigger Input	Open Drain Output	5V Tolerance
PE[6:0]	I/O	I	N/A	Yes	Programmable pull-up	Yes	Yes, programmable	Yes, 5V tolerant inputs unless pull-ups are enabled
RESET/ PD0	I/O	I/O (defaults to $\overline{\text{RESET}}$)	Low (in RESET mode)	Yes (PD0 only)	Programmable for PD0; always On for $\overline{\text{RESET}}$	Yes	Programmable for PD0; always On for $\overline{\text{RESET}}$	Yes, 5V tolerant inputs unless pull-ups are enabled
V_{DD}	N/A	N/A	N/A	N/A			N/A	N/A
V_{SS}	N/A	N/A	N/A	N/A			N/A	N/A

need to use this on-chip Program RAM to shadow Interrupt Service Routines (ISR). For details, see the PRAM_M section on page 278.

3.2. Program Memory

The eZ8 CPU supports 64KB of Program Memory address space. The F1680 Series MCU contains 8KB to 24KB of on-chip Flash memory in the Program Memory address space, depending on the device.

In addition, the F1680 Series MCU contains up to 1KB of on-chip Program RAM. The Program RAM is mapped in the Program Memory address space beyond the on-chip Flash memory. The Program RAM is entirely under user control and is meant to store interrupt service routines of high-frequency interrupts. Since interrupts bring the CPU out of low-power mode, it is important to ensure that interrupts that occur very often use as low a current as possible. For battery operated systems, Program RAM based handling of high-frequency interrupts provides power savings by keeping the Flash block disabled. Program RAM (PRAM) is optimized for low-current operation and can be easily boot-strapped with interrupt code at power up.

Reading from Program Memory addresses present outside the available Flash memory and PRAM addresses returns FFH. Writing to these unimplemented Program Memory addresses produces no effect. Table 6 describes the Program Memory maps for the F1680 Series MCU.

Table 6. F1680 Series MCU Program Memory Maps

Program Memory Address (Hex)	Function
Z8F2480 Device	
0000–0001	Flash option bits
0002–0003	Reset vector
0004–0005	WDT interrupt vector
0006–0007	Illegal instruction trap
0008–0037	Interrupt vectors*
0038–003D	Oscillator fail traps*
003E–5FFF	Program Flash
E000–E3FF	1KB PRAM

Note: *See Table 36 on page 69 for a list of interrupt vectors and traps.

Example 1. A poor coding style that can result in lost interrupt requests:

```
LDX r0, IRQ0
AND r0, MASK
LDX IRQ0, r0
```

To avoid missing interrupts, use the coding style in Example 2 to clear bits in the Interrupt Request 0 Register:

Example 2. A good coding style that avoids lost interrupt requests:

```
ANDX IRQ0, MASK
```

8.3.4. Software Interrupt Assertion

Program code can generate interrupts directly. Writing a 1 to the correct bit in the Interrupt Request Register triggers an interrupt (assuming that the interrupt is enabled). When the interrupt request is acknowledged by the eZ8 CPU, the bit in the Interrupt Request Register is automatically cleared to 0.

! **Caution:** Zilog recommends not using a coding style to generate software interrupts by setting bits in the Interrupt Request registers. All incoming interrupts received between execution of the first LDX command and the final LDX command are lost. See Example 3, which follows.

Example 3. A poor coding style that can result in lost interrupt requests:

```
LDX r0, IRQ0
OR r0, MASK
LDX IRQ0, r0
```

To avoid missing interrupts, use the coding style in Example 4 to set bits in the Interrupt Request registers:

Example 4. A good coding style that avoids lost interrupt requests:

```
ORX IRQ0, MASK
```

8.4. Interrupt Control Register Definitions

For all interrupts other than the Watchdog Timer interrupt, the Primary Oscillator Fail Trap and the Watchdog Oscillator Fail Trap, the Interrupt Control registers enable individual interrupts, set interrupt priorities and indicate interrupt requests.

Bit	Description
[5] PA5CS	**PA5/Comparator 1 Selection** 0 = PA5 is used for the interrupt for PA5CS interrupt request. 1 = The Comparator 1 is used for the interrupt for PA5CS interrupt request.
[4:1] PAD*x*S	**PAx/PDx Selection** 0 = PAx is used for the interrupt for PAx/PDx interrupt request 1 = PDx is used for the interrupt for PAx/PDx interrupt request; an *x* indicates the specific GPIO port pin number (1–4).
[0]	Reserved; must be 0.

8.4.9. Interrupt Control Register

The Interrupt Control (IRQCTL) Register, shown in Table 51, contains the master enable bit for all interrupts.

Table 51. Interrupt Control Register (IRQCTL)

Bits	7	6	5	4	3	2	1	0
Field	IRQE	Reserved						
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R	R	R	R	R	R	R
Address	FCFH							

Bit	Description
[7] IRQE	**Interrupt Request Enable** This bit is set to 1 by executing an Enable Interrupts (EI) or IRET (Interrupt Return) instruction, or by a direct register write of a 1 to this bit. It is reset to 0 by executing a DI instruction, eZ8 CPU acknowledgement of an interrupt request, a Reset, or by a direct register write of a 0 to this bit. 0 = Interrupts are disabled. 1 = Interrupts are enabled.
[6:0]	Reserved; must be 0.

$$\text{COMPARE Mode Time (s)} = \frac{(\text{Compare Value - Start Value}) \times \text{Prescale}}{\text{Timer Clock Frequency (Hz)}}$$

generated the interrupt, read the associated GPIO input value and compare to the value stored in the TPOL bit.

The timer counts up to the 16-bit reload value stored in the Timer Reload High and Low Byte registers. The timer input is the timer clock. When reaching the reload value, the timer generates an interrupt, the count value in the Timer High and Low Byte registers is reset to 0001H and counting resumes (assuming the Timer Input signal is still asserted). Also, if the Timer Output alternate function is enabled, the Timer Output pin changes state (from Low to High or from High to Low) at timer reset.

Observe the following steps to configure a timer for GATED Mode and initiate the count:

1. Write to the Timer Control 1 Register to:
   - Disable the timer
   - Configure the timer for GATED Mode
   - Set the prescale value
2. Write to the Timer Control 2 Register to choose the timer clock source.
3. Write to the Timer Control 0 Register to set the timer interrupt configuration field TICONFIG.
4. Write to the Timer High and Low Byte registers to set the starting count value. This value only affects the first pass in GATED Mode. After the first timer reset in GATED Mode, counting always begins at the reset value of 0001H.
5. Write to the Timer Reload High and Low Byte registers to set the reload value.
6. If required, enable the timer interrupt and set the timer interrupt priority by writing to the relevant interrupt registers. By default, the timer interrupt will be generated for both input deassertion and reload events. If required, configure the timer interrupt to be generated only at the Input Deassertion event or the Reload event by setting TICONFIG field of the Timer Control 0 Register.
7. Configure the associated GPIO port pin for the Timer Input alternate function.
8. Write to the Timer Control 1 Register to enable the timer.
9. Assert the Timer Input signal to initiate the counting.

12.3.3. LIN-UART Status 0 Register

The LIN-UART Status 0 Register identifies the current LIN-UART operating configuration and status. Table 85 describes the Status 0 Register for standard UART mode. Table 86 describes the Status 0 Register for LIN mode.

Table 85. LIN-UART Status 0 Register—Standard UART Mode (U0STAT0 = F41H)

Bit	7	6	5	4	3	2	1	0
Field	RDA	PE	OE	FE	BRKD	TDRE	TXE	CTS
Reset	0	0	0	0	0	1	1	X
R/W	R	R	R	R	R	R	R	R
Address	F41H, F49H							

Note: R = Read; X = undefined.

Bit	Description
[7] RDA	**Receive Data Available** This bit indicates that the LIN-UART Receive Data Register has received data. Reading the LIN-UART Receive Data Register clears this bit. 0 = The LIN-UART Receive Data Register is empty. 1 = There is a byte in the LIN-UART Receive Data Register.
[6] PE	**Parity Error** This bit indicates that a parity error has occurred. Reading the Receive Data Register clears this bit. 0 = No parity error occurred. 1 = A parity error occurred.
[5] OE	**Overrun Error** This bit indicates that an overrun error has occurred. An overrun occurs when new data is received and the Receive Data Register is not read. Reading the Receive Data Register clears this bit. 0 = No overrun error occurred. 1 = An overrun error occurred.
[4] FE	**Framing Error** This bit indicates that a framing error (no stop bit following data reception) was detected. Reading the Receive Data Register clears this bit. 0 = No framing error occurred. 1 = A framing error occurred.
[3] BRKD	**Break Detect** This bit indicates that a break occurred. If the data bits, parity/multiprocessor bit and stop bit(s) are all zeros then this bit is set to 1. Reading the Receive Data Register clears this bit. 0 = No break occurred. 1 = A break occurred.

Bit	Description (Continued)
[2] TDRE	**Transmitter Data Register Empty** This bit indicates that the Transmit Data Register is empty and ready for additional data. Writing to the Transmit Data Register resets this bit. 0 = Do not write to the Transmit Data Register. 1 = The Transmit Data Register is ready to receive an additional byte for transmission.
[1] TXE	**Transmitter Empty** This bit indicates that the Transmit Shift Register is empty and character transmission is finished. 0 = Data is currently transmitting. 1 = Transmission is complete.
[0] CTS	**Clear to Send Signal** When this bit is read it returns the level of the $\overline{\text{CTS}}$ signal. If LBEN = 1, the $\overline{\text{CTS}}$ input signal is replaced by the internal Receive Data Available signal to provide flow control in loopback mode. CTS only affects transmission if the CTSE bit = 1.

Table 86. LIN-UART Status 0 Register—LIN Mode (U0STAT0 = F41H)

Bit	7	6	5	4	3	2	1	0
Field	RDA	PLE	OE	FE	BRKD	TDRE	TXE	ATB
Reset	0	0	0	0	0	1	1	0
R/W	R	R	R	R	R	R	R	R
Address	F41H, F49H							

Note: R = Read.

Bit	Description
[7] RDA	**Receive Data Available** This bit indicates that the Receive Data Register has received data. Reading the Receive Data Register clears this bit. 0 = The Receive Data Register is empty. 1 = There is a byte in the Receive Data Register.
[6] PLE	**Physical Layer Error** This bit indicates that transmit and receive data do not match when a LIN slave or master is transmitting. This could be by a fault in the physical layer or multiple devices driving the bus simultaneously. Reading the Status 0 Register or the Receive Data Register clears this bit. 0 = Transmit and Receive data match. 1 = Transmit and Receive data do not match.

A receive interrupt is generated by the RDRNE status bit when the ESPI block is enabled, the DIRQE bit is set and a character transfer completes. At the end of the character transfer, the contents of the Shift Register are transferred into the Data Register, causing the RDRNE bit to assert. The RDRNE bit is cleared when the Data Buffer is read as empty. If information is being transmitted but not received by the software application, the receive interrupt can be eliminated by selecting Transmit Only mode (ESPIEN1,0 = 10) in either MASTER or SLAVE modes. When information is being sent and received under interrupt control, RDRNE and TDRE will both assert simultaneously at the end of a character transfer. Since the new receive data is in the Data Register, the receive interrupt must be serviced before the transmit interrupt.

ESPI error interrupts occur if any of the TUND, COL, ABT and ROVR bits in the ESPI Status Register are set. These bits are cleared by writing a 1. If the ESPI is disabled (ESPIEN1, 0 = 00), an ESPI interrupt can be generated by a Baud Rate Generator time-out. This timer function must be enabled by setting the BRGCTL bit in the ESPICTL register. This timer interrupt does not set any of the bits of the ESPI Status Register.

16.3.7. ESPI Baud Rate Generator

In ESPI MASTER Mode, the Baud Rate Generator creates a lower frequency serial clock (SCK) for data transmission synchronization between the Master and the external Slave. The input to the Baud Rate Generator is the system clock. The ESPI Baud Rate High and Low Byte registers combine to form a 16-bit reload value, BRG[15:0], for the ESPI Baud Rate Generator. The ESPI baud rate is calculated using the following equation:

$$\text{SPI Baud Rate (bits § s)} = \frac{\text{System Clock Frequency (Hz)}}{2 \times \text{BRG[15:0]}}$$

Minimum baud rate is obtained by setting BRG[15:0] to 0000H for a clock divisor value of (2 x 65536 = 131072).

When the ESPI is disabled, the Baud Rate Generator can function as a basic 16-bit timer with interrupt on time-out. Observe the following steps to configure the Baud Rate Generator as a timer with interrupt on time-out:

1. Disable the ESPI by clearing the ESPIEN1,0 bits in the ESPI Control Register.
2. Load the appropriate 16-bit count value into the ESPI Baud Rate High and Low Byte registers.
3. Enable the Baud Rate Generator timer function and associated interrupt by setting the BRGCTL bit in the ESPI Control Register to 1.

When configured as a general purpose timer, the SPI BRG interrupt interval is calculated using the following equation:

Bit	Description
[1] TEOF	**Transmit End of Frame** This bit is used in MASTER Mode to indicate that the data in the Transmit Data Register is the last byte of the transfer or frame. When the last byte has been sent $\overline{SS}$ (and SSV) will change state and TEOF will automatically clear. 0 = The data in the Transmit Data Register is not the last character in the message. 1 = The data in the Transmit Data Register is the last character in the message.
[0] SSV	**Slave Select Value** When SSIO = 1, writes to this register will control the value output on the $\overline{SS}$ pin. For more details, see the SSMD field of the ESPI Mode Register on page 217.

16.4.3. ESPI Control Register

The ESPI Control Register, shown in Table 111, configures the ESPI for transmit and receive operations.

Table 111. ESPI Control Register

Bits	7	6	5	4	3	2	1	0
Field	DIRQE	ESPIEN1	BRGCTL	PHASE	CLKPOL	WOR	MMEN	ESPIEN0
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Address	F62H							

Bit	Description
[7] DIRQE	**Data Interrupt Request Enable** This bit is used to disable or enable data (TDRE and RDRNE) interrupts. Disabling the data interrupts is needed to control data transfer by polling. Error interrupts are not disabled. To block all ESPI interrupt sources, clear the ESPI interrupt enable bit in the Interrupt Controller. 0 = TDRE and RDRNE assertions do not cause an interrupt. Use this setting if controlling data transfer by software polling of TDRE and RDRNE. The TUND, COL, ABT and ROVR bits will cause an interrupt. 1 = TDRE and RDRNE assertions will cause an interrupt. TUND, COL, ABT and ROVR will also cause interrupts. Use this setting when controlling data transfer via interrupt handlers.

Bit	Description (Continued)
[6,0] ESPIEN1, ESPIEN0	**ESPI Enable and Direction Control** 00 = The ESPI block is disabled. BRG can be used as a general-purpose timer by setting BRGCTL = 1. 01 = Receive Only Mode. Use this setting in SLAVE Mode if software application is receiving data but not sending. TDRE will not assert. Transmitted data will be all 1s. Not valid in MASTER Mode since Master must source data to drive the transfer. 10 = Transmit Only Mode Use this setting in MASTER or SLAVE Mode when the software application is sending data but not receiving. RDRNE will not assert. 11 = Transmit/Receive Mode Use this setting if the software application is both sending and receiving information. Both TDRE and RDRNE will be active.
[5] BRGCTL	**Baud Rate Generator Control** The function of this bit depends upon ESPIEN1,0. When ESPIEN1,0 = 00, this bit allows enabling the BRG to provide periodic interrupts. **If the ESPI is disabled** 0 = The Baud Rate Generator timer function is disabled. Reading the Baud Rate High and Low registers returns the BRG reload value. 1 = The Baud Rate Generator timer function and time-out interrupt is enabled. Reading the Baud Rate High and Low registers returns the BRG Counter value. **If the ESPI is enabled** 0 = Reading the Baud Rate High and Low registers returns the BRG reload value. If MMEN = 1, the BRG is enabled to generate SCK. If MMEN = 0, the BRG is disabled. 1 = Reading the Baud Rate High and Low registers returns the BRG Counter value. If MMEN = 1, the BRG is enabled to generate SCK. If MMEN = 0 the BRG is enabled to provide a Slave SCK time-out. See the SLAVE Mode Abort error description on page 211. **Caution:** If reading the counter one byte at a time while the BRG is counting keep in mind that the values will not be in sync. Zilog recommends reading the counter using (2-byte) word reads.
[4] PHASE	**Phase Select** Sets the phase relationship of the data to the clock. For more information about operation of the PHASE bit, see the ESPI Clock Phase and Polarity Control section on page 201.
[3] CLKPOL	**Clock Polarity** 0 = SCK idles Low (0). 1 = SCK idles High (1).
[2] WOR	**Wire-OR (Open-Drain) Mode Enabled** 0 = ESPI signal pins not configured for open-drain. 1 = All four ESPI signal pins (SCK, $\overline{SS}$, MISO and MOSI) configured for open-drain function. This setting is typically used for multi-Master and/or Multi-Slave configurations.
[1] MMEN	**ESPI MASTER Mode Enable** This bit controls the data I/O pin selection and SCK direction. 0 = Data out on MISO, data in on MOSI (used in SPI SLAVE Mode), SCK is an input. 1 = Data out on MOSI, data in on MISO (used in SPI MASTER Mode), SCK is an output.

17.2.3. Start and Stop Conditions

The Master generates the start and stop conditions to start or end a transaction. To start a transaction, the I^2C controller generates a start condition by pulling the SDA signal Low while SCL is High. To complete a transaction, the I^2C controller generates a stop condition by creating a Low-to-High transition of the SDA signal while the SCL signal is High. These start and stop events occur when the start and stop bits in the I^2C Control Register are written by software to begin or end a transaction. Any byte transfer currently under way including the Acknowledge phase finishes before the start or stop condition occurs.

17.2.4. Software Control of I^2C Transactions

The I^2C controller is configured via the I^2C Control and I^2C Mode registers. The MODE[1:0] field of the I^2C Mode Register allows the configuration of the I^2C controller for MASTER/SLAVE or SLAVE ONLY mode and configures the slave for 7-bit or 10-bit addressing recognition.

MASTER/SLAVE Mode can be used for:

- MASTER ONLY operation in a Single Master/One or More Slave I^2C system
- MASTER/SLAVE in a Multimaster/multislave I^2C system
- SLAVE ONLY operation in an I^2C system

In SLAVE ONLY mode, the start bit of the I^2C Control Register is ignored (software cannot initiate a master transaction by accident) and operation to SLAVE ONLY Mode is restricted thereby preventing accidental operation in MASTER Mode. The software controls I^2C transactions by enabling the I^2C controller interrupt in the interrupt controller or by polling the I^2C Status Register.

To use interrupts, the I^2C interrupt must be enabled in the interrupt controller and followed by executing an EI instruction. The TXI bit in the I^2C Control Register must be set to enable transmit interrupts. An I^2C interrupt service routine then checks the I^2C Status Register to determine the cause of the interrupt.

To control transactions by polling, the TDRE, RDRF, SAM, ARBLST, SPRS and NCKI interrupt bits in the I^2C Status Register should be polled. The TDRE bit asserts regardless of the state of the TXI bit.

17.2.5. Master Transactions

The following sections describe Master Read and Write transactions to both 7-bit and 10-bit slaves.

17.2.5.1. Master Arbitration

If a Master loses arbitration during the address byte it releases the SDA line, switches to SLAVE Mode and monitors the address to determine if it is selected as a Slave. If a Master loses arbitration during the transmission of a data byte, it releases the SDA line and waits for the next stop or start condition.

The Master detects a loss of arbitration when a 1 is transmitted but a 0 is received from the bus in the same bit-time. This loss occurs if more than one Master is simultaneously accessing the bus. Loss of arbitration occurs during the address phase (two or more Masters accessing different slaves) or during the data phase, when the masters are attempting to Write different data to the same Slave.

When a Master loses arbitration, the software is informed by means of the Arbitration Lost interrupt. The software can repeat the same transaction at a later time.

A special case can occur when a Slave transaction starts just before the software attempts to start a new master transaction by setting the start bit. In this case, the state machine enters its Slave states before the start bit is set and as a result the I^2C controller will not arbitrate. If a Slave address match occurs and the I^2C controller receives/transmits data, the start bit is cleared and an Arbitration Lost interrupt is asserted. The software can minimize the chance of this instance occurring by checking the busy bit in the I2CSTATE Register before initiating a Master transaction. If a slave address match does not occur, the Arbitration Lost interrupt will not occur and the start bit will not be cleared. The I^2C controller will initiate the master transaction after the I^2C bus is no longer busy.

17.2.5.2. Master Address-Only Transactions

It is sometimes preferable to perform an address-only transaction to determine if a particular slave device is able to respond. This transaction can be performed by monitoring the ACKV bit in the I2CSTATE Register after the address has been written to the I2CDATA Register and the start bit has been set. After the ACKV bit is set, the ACK bit in the I2CSTATE Register determines if the slave is able to communicate. The stop bit must be set in the I2CCTL Register to terminate the transaction without transferring data. For a 10-bit slave address, if the first address byte is acknowledged, the second address byte should also be sent to determine if the preferred Slave is responding.

Another approach is to set both the stop and start bits (for sending a 7-bit address). After both bits have been cleared (7-bit address has been sent and transaction is complete), the ACK bit can be read to determine if the Slave has acknowledged. For a 10-bit Slave, set the stop bit after the second TDRE interrupt (which indicates that the second address byte is being sent).

17.2.5.3. Master Transaction Diagrams

In the following transaction diagrams, the shaded regions indicate the data that is transferred from the Master to the Slave and the unshaded regions indicate the data that is

11. The I^2C slave sends an Acknowledge (by pulling the SDA signal Low) during the next High period of SCL. The I^2C controller sets the ACK bit in the I^2C Status Register.

    If the slave does not acknowledge the address byte, the I^2C controller sets the NCKI bit in the I^2C Status Register, sets the ACKV bit and clears the ACK bit in the I^2C State Register. The software responds to the Not Acknowledge interrupt by setting the stop bit and clearing the TXI bit. The I^2C controller flushes the Transmit Data Register, sends a stop condition on the bus and clears the stop and NCKI bits. The transaction is complete and the following steps can be ignored.
12. The I^2C controller loads the contents of the I^2C Shift Register with the contents of the I^2C Data Register.
13. The I^2C controller shifts the data out via the SDA signal. After the first bit is sent, the transmit interrupt asserts.
14. If more bytes remain to be sent, return to Step 9.
15. When there is no more data to be sent, the software responds by setting the stop bit of the I^2C Control Register (or the start bit to initiate a new transaction).
16. If no additional transaction is queued by the master, the software can clear the TXI bit of the I^2C Control Register.
17. The I^2C controller completes transmission of the data on the SDA signal.
18. The I^2C controller sends a stop condition to the I^2C bus.

Note: If the slave terminates the transaction early by responding with a Not Acknowledge during the transfer, the I^2C controller asserts the NCKI interrupt and halts. The software must terminate the transaction by setting either the stop bit (end transaction) or the start bit (end this transaction, start a new one). In this case, it is not necessary for software to set the FLUSH bit of the I2CCTL Register to flush the data that was previously written but not transmitted. The I^2C controller hardware automatically flushes transmit data in the not acknowledge case.

17.2.5.5. Master Write Transaction with a 10-Bit Address

Figure 44 displays the data transfer format from a Master to a 10-bit addressed slave.

S	Slave Address 1st Byte	W = 0	A	Slave Address 2nd Byte	A	Data	A	Data	A/$\overline{A}$	F/S

Figure 44. Data Transfer Format—Master Write Transaction with a 10-Bit Address

4. If this operation is a single-byte transfer, the software asserts the NAK bit of the I2C Control Register so that after the first byte of data has been read by the I2C controller, a Not Acknowledge instruction is sent to the I2C slave.
5. The I2C controller sends a start condition.
6. The I2C controller sends the address and Read bit out via the SDA signal.
7. The I2C slave acknowledges the address by pulling the SDA signal Low during the next High period of SCL.

   If the slave does not acknowledge the address byte, the I2C controller sets the NCKI bit in the I2C Status Register, sets the ACKV bit and clears the ACK bit in the I2C State Register. The software responds to the Not Acknowledge interrupt by setting the stop bit and clearing the TXI bit. The I2C controller flushes the Transmit Data Register, sends a stop condition on the bus and clears the stop and NCKI bits. The transaction is complete and the following steps can be ignored.
8. The I2C controller shifts in the first byte of data from the I2C slave on the SDA signal.
9. The I2C controller asserts the receive interrupt.
10. The software responds by reading the I2C Data Register. If the next data byte is to be the final byte, the software must set the NAK bit of the I2C Control Register.
11. The I2C controller sends a Not Acknowledge to the I2C slave if the next byte is the final byte; otherwise, it sends an Acknowledge.
12. If there are more bytes to transfer, the I2C controller returns to Step 7.
13. A NAK interrupt (NCKI bit in I2CISTAT) is generated by the I2C controller.
14. The software responds by setting the stop bit of the I2C Control Register.
15. A stop condition is sent to the I2C slave.

17.2.5.7. Master Read Transaction with a 10-Bit Address

Figure 46 displays the read transaction format for a 10-bit addressed Slave.

S	Slave Address 1st Byte	W=0	A	Slave Address 2nd Byte	A	S	Slave Address 1st Byte	R=1	A	Data	A	Data	$\overline{A}$	P

Figure 46. Data Transfer Format—Master Read Transaction with a 10-Bit Address

The first 7 bits transmitted in the first byte are 11110XX. The two XX bits are the two most-significant bits of the 10-bit address. The lowest bit of the first byte transferred is the write control bit.

Observe the following data transfer procedure for a Read operation to a 10-bit addressed slave:

State Register. The software responds to the Not Acknowledge interrupt by setting the stop bit and clearing the TXI bit. The I2C controller flushes the Transmit Data Register, sends the stop condition on the bus and clears the stop and NCKI bits. The transaction is complete and the following steps can be ignored.

16. The I2C controller sends a repeated start condition.
17. The I2C controller loads the I2C Shift Register with the contents of the I2C Data Register (the third address transfer).
18. The I2C controller sends `11110b`, followed by the two most-significant bits of the slave read address and a 1 (Read).
19. The I2C slave sends an Acknowledge by pulling the SDA signal Low during the next High period of SCL.
20. The I2C controller shifts in a byte of data from the slave.
21. The I2C controller asserts the Receive interrupt.
22. The software responds by reading the I2C Data Register. If the next data byte is to be the final byte, the software must set the NAK bit of the I2C Control Register.
23. The I2C controller sends an Acknowledge or Not Acknowledge to the I2C Slave, based on the value of the NAK bit.
24. If there are more bytes to transfer, the I2C controller returns to Step 18.
25. The I2C controller generates a NAK interrupt (the NCKI bit in the I2CISTAT Register).
26. The software responds by setting the stop bit of the I2C Control Register.
27. A stop condition is sent to the I2C Slave.

17.2.6. Slave Transactions

The following sections describe Read and Write transactions to the I2C controller configured for 7-bit and 10-bit Slave modes.

17.2.6.1. Slave Address Recognition

The following two slave address recognition options are supported; a description of each follows.

- Slave 7-Bit Address Recognition Mode
- Slave 10-Bit Address Recognition Mode

Slave 7-Bit Address Recognition Mode. If IRM = 0 during the address phase and the controller is configured for MASTER/SLAVE or SLAVE 7-bit address mode, the

17.3.3. I²C Control Register

The I²C Control Register, shown in Table 121, enables and configures I²C operation.

Note: The R/W1 bit can be set (written to 1) when IEN = 1, but cannot be cleared (written to 0).

Table 121. I²C Control Register (I2CCTL)

Bits	7	6	5	4	3	2	1	0
Field	IEN	START	STOP	BIRQ	TXI	NAK	FLUSH	FILTEN
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W1	R/W1	R/W	R/W	R/W1	W	R/W
Address	F52H							

Bit	Description
[7] IEN	**I²C Enable** This bit enables the I²C controller.
[6] START	**Send Start Condition** When set, this bit causes the I²C controller (when configured as the master) to send a start condition. After it is asserted, this bit is cleared by the I²C controller after it sends the start condition or by deasserting the IEN bit. If this bit is 1, it cannot be cleared by writing to the bit. After this bit is set, a start condition is sent if there is data in the I2CDATA or I²C Shift Register. If there is no data in one of these registers, the I²C controller waits until data is loaded. If this bit is set while the I²C controller is shifting out data, it generates a restart condition after the byte shifts and the Acknowledge phase completes. If the stop bit is also set, it waits until the stop condition is sent before the start condition. If start is set while a SLAVE Mode transaction is underway to this device, the start bit will be cleared and ARBLST bit in the Interrupt Status Register will be set.
[5] STOP	**Send Stop Condition** When set, this bit causes the I²C controller (when configured as the master) to send the stop condition after the byte in the I²C Shift Register has completed transmission or after a byte is received in a receive operation. When set, this bit is reset by the I²C controller after a stop condition has been sent or by deasserting the IEN bit. If this bit is 1, it cannot be cleared to 0 by writing to the register. If stop is set while a SLAVE Mode transaction is underway, the stop bit is cleared by hardware.
[4] BIRQ	**Baud Rate Generator Interrupt Request** This bit is ignored when the I²C controller is enabled. If this bit is set = 1 when the I²C controller is disabled (IEN = 0), the baud rate generator is used as an additional timer causing an interrupt to occur every time the baud rate generator counts down to one. The baud rate generator runs continuously in this mode, generating periodic interrupts.

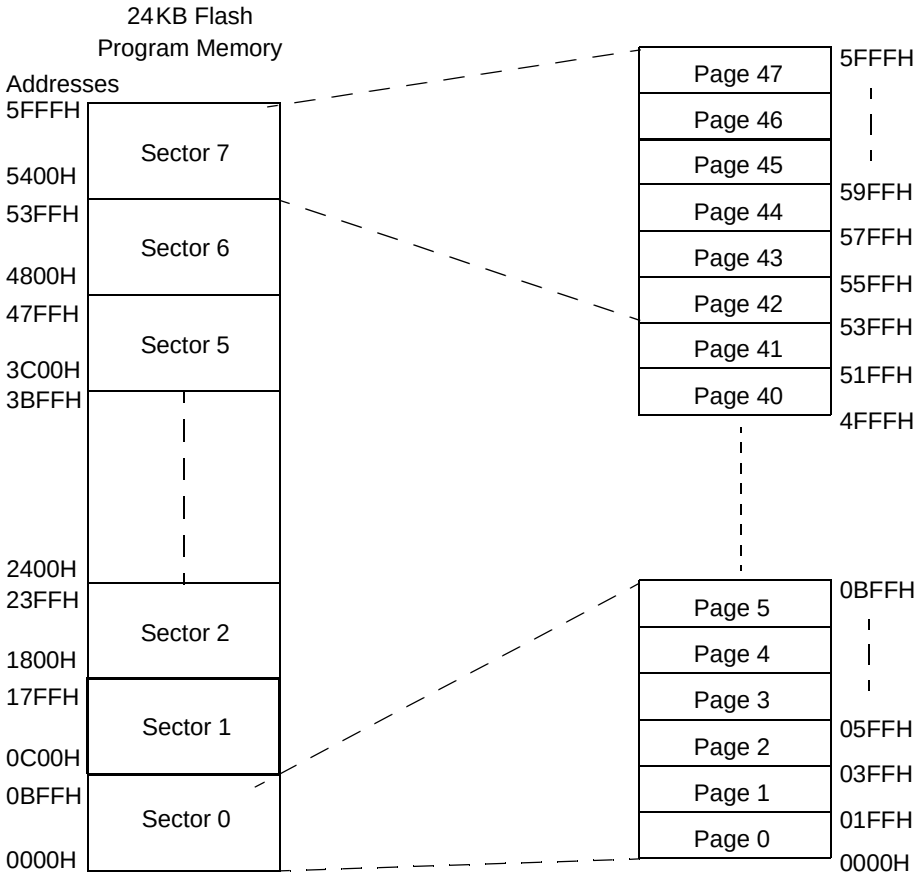

Figure 53. 24KB Flash Memory Arrangement

20.2. Operation

The Flash Controller programs and erases Flash memory. The Flash Controller provides the proper Flash controls and timing for byte programming, Page Erase and Mass Erase of Flash memory.

The Flash Controller contains several protection mechanisms to prevent accidental programming or erasure. These mechanisms operate on the page, sector and full-memory levels.

The Flow Chart in Figure 54 displays basic Flash Controller operation. The sections that follow provide details about the various operations (Lock, Unlock, Byte Programming, Page Protect, Page Unprotect, Page Select Page Erase and Mass Erase) shown in Figure 54.

H

I