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	eZ8
Core Size	8-Bit
Speed	20MHz
Connectivity	I²C, IrDA, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	46
Program Memory Size	24KB (24K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	3V ~ 3.6V
Data Converters	A/D 12x10b
Oscillator Type	Internal
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	64-LQFP
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z8f2422ar020sc

Table 6. Z8 Encore! XP 64K Series Flash Microcontrollers Information Area Map

Program Memory Address (Hex)	Function
FE00H-FE3FH	Reserved
FE40H-FE53H	Part Number 20-character ASCII alphanumeric code Left justified and filled with zeros (ASCII Null character)
FE54H-FFFFH	Reserved

UART0 Transmit Data
U0TXD (F40H - Write Only)

D7	D6	D5	D4	D3	D2	D1	D0

UART0 transmitter data byte [7:0]

UART0 Receive Data
U0RXD (F40H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

UART0 receiver data byte [7:0]

UART0 Status 0
U0STAT0 (F41H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

$\overline{\text{CTS}}$ signal
Returns the level of the $\overline{\text{CTS}}$ signal

Transmitter Empty
0 = Data is currently transmitting
1 = Transmission is complete

Transmitter Data Register Empty
0 = Transmit Data Register is full
1 = Transmit Data register is empty

Break Detect
0 = No break occurred
1 = A break occurred

Framing Error
0 = No framing error occurred
1 = A framing occurred

Overrun Error
0 = No overrun error occurred
1 = An overrun error occurred

Parity Error
0 = No parity error occurred
1 = A parity error occurred

Receive Data Available
0 = Receive Data Register is empty
1 = A byte is available in the Receive Data Register

UART0 Control 0
U0CTL0 (F42H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

Loop Back Enable
0 = Normal operation
1 = Transmit data is looped back to the receiver

Stop Bit Select
0 = Transmitter sends 1 Stop bit
1 = Transmitter sends 2 Stop bits

Send Break
0 = No break is sent
1 = Output of the transmitter is zero

Parity Select
0 = Even parity
1 = Odd parity

Parity Enable
0 = Parity is disabled
1 = Parity is enabled

$\overline{\text{CTS}}$ Enable
0 = CTS signal has no effect on the transmitter
1 = UART recognizes $\overline{\text{CTS}}$ signal as a transmit enable control signal

Receive Enable
0 = Receiver disabled
1 = Receiver enabled

Transmit Enable
0 = Transmitter disabled
1 = Transmitter enabled

UART0 Control 1
U0CTL1 (F43H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- Infrared Encoder/Decoder Enable
  0 = Infrared endec is disabled
  1 = Infrared endec is enabled
- Received Data Interrupt Enable
  0 = Received data and errors generate interrupt requests
  1 = Only errors generate interrupt requests. Received data does not.
- Baud Rate Registers Control
  Refer to UART chapter for operation
- Driver Enable Polarity
  0 = DE signal is active High
  1 = DE signal is active Low
- Multiprocessor Bit Transmit
  0 = Send a 0 as the multiprocessor bit
  1 = Send a 1 as the multiprocessor bit
- Multiprocessor Mode [0]
  See Multiprocessor Mode [1] below
- Multiprocessor (9-bit) Enable
  0 = Multiprocessor mode is disabled
  1 = Multiprocessor mode is enabled
- Multiprocessor Mode [1]
  with Multiprocess Mode bit 0:
  00 = Interrupt on all received bytes
  01 = Interrupt only on address bytes
  10 = Interrupt on address match and following data
  11 = Interrupt on data following an address match

UART0 Status 1
U0STAT1 (F44H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

- Mulitprocessor Receive
  Returns value of last multiprocessor bit
- New Frame
  0 = Current byte is not start of frame
  1 = Current byte is start of new frame
- Reserved

UART0 Address Compare
U0ADDR (F45H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- UART0 Address Compare [7:0]

UART0 Baud Rate Generator High Byte
U0BRH (F46H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- UART0 Baud Rate divisor [15:8]

UART0 Baud Rate Generator Low Byte
U0BRL (F47H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- UART0 Baud Rate divisor [7:0]

UART1 Transmit Data
U1TXD (F48H - Write Only)

D7	D6	D5	D4	D3	D2	D1	D0

- UART1 transmitter data byte[7:0]

UART1 Receive Data
U1RXD (F48H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

- UART receiver data byte [7:0]

I²C Status

I2CSTAT (F51H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

- D0 — NACK Interrupt
  0 = No action required to service NAK
  1 = START/STOP not set after NAK
- D1 — Data Shift State
  0 = Data is not being transferred
  1 = Data is being transferred
- D2 — Transmit Address State
  0 = Address is not being transferred
  1 = Address is being transferred
- D3 — Read
  0 = Write operation
  1 = Read operation
- D4 — 10-Bit Address
  0 = 7-bit address being transmitted
  1 = 10-bit address being transmitted
- D5 — Acknowledge
  0 = Acknowledge not transmitted/received
  1 = For last byte, Acknowledge was transmitted/received
- D6 — Receive Data Register Full
  0 = I2C has not received data
  1 = Data register contains received data
- D7 — Transmit Data Register Empty
  0 = Data register is full
  1 = Data register is empty

I²C Control

I2CCTL (F52H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- D0 — I2C Signal Filter Enable
  0 = Digital filtering disabled
  1 = Low-pass digital filters enabled on SDA and SCL input signals
- D1 — Flush Data
  0 = No effect
  1 = Clears I2C Data register
- D2 — Send NAK
  0 = Do not send NAK
  1 = Send NAK after next byte received from slave
- D3 — Enable TDRE Interrupts
  0 = Do not generate an interrupt when the I2C Data register is empty
  1 = Generate an interrupt when the I2C Transmit Data register is empty
- D4 — Baud Rate Generator Interrupt
  0 = Interrupts behave as set by I2C control
  1 = BRG generates an interrupt when it counts down to zero
- D5 — Send Stop Condition
  0 = Do not issue Stop condition after data transmission is complete
  1 = Issue Stop condition after data transmission is complete
- D6 — Send Start Condition
  0 = Do not send Start Condition
  1 = Send Start Condition
- D7 — I2C Enable
  0 = I2C is disabled
  1 = I2C is enabled

I2C Baud Rate Generator High Byte

I2CBRH (F53H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- D7–D0 — I2C Baud Rate divisor [15:8]

I2C Baud Rate Generator Low Byte

I2CBRL (F54H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- D7–D0 — I2C Baud Rate divisor [7:0]

SPI Data
SPIDATA (F60H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

SPI Data [7:0]

SPI Control
SPICTL (F61H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

SPI Enable
0 = SPI disabled
1 = SPI enabled

Master Mode Enabled
0 = SPI configured in Slave mode
1 = SPI configured in Master mode

Wire-OR (open-drain) Mode
0 = SPI signals not configured for open-drain
1 = SPI signals (SCK, $\overline{SS}$, MISO, and MOSI) configured for open-drain

Clock Polarity
0 = SCK idles Low
1 = SPI idles High

Phase Select
Sets the phase relationship of the data to the clock.

BRG Timer Interrupt Request
0 = BRG timer function is disabled
1 = BRG time-out interrupt is enabled

Start an SPI Interrupt Request
0 = No effect
1 = Generate an SPI interrupt request

Interrupt Request Enable
0 = SPI interrupt requests are disabled
1 = SPI interrupt requests are enabled

SPI Status
SPISTAT (F62H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

Slave Select
0 = If Slave, $\overline{SS}$ pin is asserted
1 = If Slave, SS pin is not asserted

Transmit Status
0 = No data transmission in progress
1 = Data transmission now in progress

Reserved

Slave Mode Transaction Abort
0 = No slave mode transaction abort detected
1 = Slave mode transaction abort was detected

Collision
0 = No multi-master collision detected
1 = Multi-master collision was detected

Overrun
0 = No overrun error detected
1 = Overrun error was detected

Interrupt Request
0 = No SPI interrupt request pending
1 = SPI interrupt request is pending

SPI Mode
SPIMODE (F63H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

Slave Select Value
If Master and SPIMODE[1] = 1:
0 = $\overline{SS}$ pin driven Low
1 = SS pin driven High

Slave Select I/O
0 = $\overline{SS}$ pin configured as an input
1 = SS pin configured as an output (Master mode only)

Number of Data Bits Per Character
000 = 8 bits
001 = 1 bit
010 = 2 bits
011 = 3 bits
100 = 4 bits
101 = 5 bit
110 = 6 bits

Port A–H Input Data Registers

Reading from the Port A–H Input Data registers (Table 21) returns the sampled values from the corresponding port pins. The Port A–H Input Data registers are Read-only.

Table 21. Port A–H Input Data Registers (PxIN)

BITS	7	6	5	4	3	2	1	0
FIELD	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
RESET	X							
R/W	R							
ADDR	FD2H, FD6H, FDAH, FDEH, FE2H, FE6H, FEAH, FEEH							

PIN[7:0]—Port Input Data
Sampled data from the corresponding port pin input.
0 = Input data is logical 0 (Low).
1 = Input data is logical 1 (High).

Port A–H Output Data Register

The Port A–H Output Data register (Table 22) writes output data to the pins.

Table 22. Port A–H Output Data Register (P*x*OUT)

BITS	7	6	5	4	3	2	1	0
FIELD	POUT7	POUT6	POUT5	POUT4	POUT3	POUT2	POUT1	POUT0
RESET	0							
R/W	R/W							
ADDR	FD3H, FD7H, FDBH, FDFH, FE3H, FE7H, FEBH, FEFH							

POUT[7:0]—Port Output Data
These bits contain the data to be driven out from the port pins. The values are only driven if the corresponding pin is configured as an output and the pin is not configured for alternate function operation.
0 = Drive a logical 0 (Low).
1= Drive a logical 1 (High). High value is not driven if the drain has been disabled by setting the corresponding Port Output Control register bit to 1.

Interrupt Controller

Overview

The interrupt controller on the 64K Series products prioritizes the interrupt requests from the on-chip peripherals and the GPIO port pins. The features of the interrupt controller include the following:

- 24 unique interrupt vectors:
  - 12 GPIO port pin interrupt sources
  - 12 on-chip peripheral interrupt sources
- Flexible GPIO interrupts
  - Eight selectable rising and falling edge GPIO interrupts
  - Four dual-edge interrupts
- Three levels of individually programmable interrupt priority
- Watchdog Timer can be configured to generate an interrupt

Interrupt requests (IRQs) allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start an interrupt service routine (ISR). Usually this interrupt service routine is involved with the exchange of data, status information, or control information between the CPU and the interrupting peripheral. When the service routine is completed, the CPU returns to the operation from which it was interrupted.

The eZ8 CPU supports both vectored and polled interrupt handling. For polled interrupts, the interrupt control has no effect on operation. For more information on interrupt servicing by the eZ8 CPU, refer to *eZ8™ CPU Core User Manual (UM0128)* available for download at www.zilog.com.

Interrupt Vector Listing

Table 23 lists all of the interrupts available in order of priority. The interrupt vector is stored with the most-significant byte (MSB) at the even Program Memory address and the least-significant byte (LSB) at the following odd Program Memory address.

Caution: *The following style of coding to generate software interrupts by setting bits in the Interrupt Request registers is NOT recommended. All incoming interrupts that are received between execution of the first LDX command and the last LDX command are lost.*

Poor coding style that can result in lost interrupt requests:

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

To avoid missing interrupts, the following style of coding to set bits in the Interrupt Request registers is recommended:

Good coding style that avoids lost interrupt requests:

```
ORX IRQ0, MASK
```

Interrupt Control Register Definitions

For all interrupts other than the Watchdog Timer interrupt, the interrupt control registers enable individual interrupts, set interrupt priorities, and indicate interrupt requests.

Interrupt Request 0 Register

The Interrupt Request 0 (IRQ0) register (Table 24) stores the interrupt requests for both vectored and polled interrupts. When a request is presented to the interrupt controller, the corresponding bit in the IRQ0 register becomes 1. If interrupts are globally enabled (vectored interrupts), the interrupt controller passes an interrupt request to the eZ8™ CPU. If interrupts are globally disabled (polled interrupts), the eZ8 CPU can read the Interrupt Request 0 register to determine if any interrupt requests are pending

Table 24. Interrupt Request 0 Register (IRQ0)

BITS	7	6	5	4	3	2	1	0
FIELD	T2I	T1I	T0I	U0RXI	U0TXI	I2CI	SPII	ADCI
RESET	0							
R/W	R/W							
ADDR	FC0H							

T2I—Timer 2 Interrupt Request
0 = No interrupt request is pending for Timer 2.
1 = An interrupt request from Timer 2 is awaiting service.

2. Write to the Timer High and Low Byte registers to set the starting count value (usually 0001H), affecting only the first pass in CONTINUOUS mode. After the first timer Reload in CONTINUOUS mode, counting always begins at the reset value of 0001H.
3. Write to the Timer Reload High and Low Byte registers to set the Reload value.
4. If desired, enable the timer interrupt and set the timer interrupt priority by writing to the relevant interrupt registers.
5. If using the Timer Output function, configure the associated GPIO port pin for the Timer Output alternate function.
6. Write to the Timer Control 1 register to enable the timer and initiate counting.

In CONTINUOUS mode, the system clock always provides the timer input. The timer period is given by the following equation:

$$\text{CONTINUOUS Mode Time-Out Period (s)} = \frac{\text{Reload Value} \times \text{Prescale}}{\text{System Clock Frequency (Hz)}}$$

If an initial starting value other than 0001H is loaded into the Timer High and Low Byte registers, the ONE-SHOT mode equation must be used to determine the first time-out period.

COUNTER Mode

In COUNTER mode, the timer counts input transitions from a GPIO port pin. The timer input is taken from the GPIO Port pin Timer Input alternate function. The TPOL bit in the Timer Control 1 Register selects whether the count occurs on the rising edge or the falling edge of the Timer Input signal. In COUNTER mode, the prescaler is disabled.

Caution: *The input frequency of the Timer Input signal must not exceed one-fourth the system clock frequency.*

Upon reaching the Reload value stored in the Timer Reload High and Low Byte registers, the timer generates an interrupt, the count value in the Timer High and Low Byte registers is reset to 0001H and counting resumes. 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 Reload.

Follow the steps below for configuring a timer for COUNTER mode and initiating the count:

1. Write to the Timer Control 1 register to:
   - Disable the timer
   - Configure the timer for COUNTER mode

– Set the prescale value

2. Write to the Timer High and Low Byte registers to set the starting count value. This 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.
3. Write to the Timer Reload High and Low Byte registers to set the Reload value.
4. If desired, enable the timer interrupt and set the timer interrupt priority by writing to the relevant interrupt registers.
5. Configure the associated GPIO port pin for the Timer Input alternate function.
6. Write to the Timer Control 1 register to enable the timer.
7. Assert the Timer Input signal to initiate the counting.

CAPTURE/COMPARE Mode

In CAPTURE/COMPARE mode, the timer begins counting on the *first* external Timer Input transition. The desired transition (rising edge or falling edge) is set by the TPOL bit in the Timer Control 1 Register. The timer input is the system clock.

Every subsequent desired transition (after the first) of the Timer Input signal captures the current count value. The Capture value is written to the Timer PWM High and Low Byte Registers. When the Capture event occurs, an interrupt is generated, the count value in the Timer High and Low Byte registers is reset to 0001H, and counting resumes.

If no Capture event occurs, the timer counts up to the 16-bit Compare value stored in the Timer Reload High and Low Byte registers. Upon reaching the Compare value, the timer generates an interrupt, the count value in the Timer High and Low Byte registers is reset to 0001H and counting resumes.

Follow the steps below for configuring a timer for CAPTURE/COMPARE mode and initiating the count:

1. Write to the Timer Control 1 register to:
   – Disable the timer
   – Configure the timer for CAPTURE/COMPARE mode
   – Set the prescale value
   – Set the Capture edge (rising or falling) for the Timer Input
2. Write to the Timer High and Low Byte registers to set the starting count value (typically 0001H).
3. Write to the Timer Reload High and Low Byte registers to set the Compare value.
4. If desired, enable the timer interrupt and set the timer interrupt priority by writing to the relevant interrupt registers.
5. Configure the associated GPIO port pin for the Timer Input alternate function.

Table 47. Watchdog Timer Approximate Time-Out Delays

WDT Reload Value (Hex)	WDT Reload Value (Decimal)	Approximate Time-Out Delay (with 10 kHz typical WDT oscillator frequency)	
		Typical	Description
000004	4	400 µs	Minimum time-out delay
FFFFFF	16,777,215	1677.5 s	Maximum time-out delay

Watchdog Timer Refresh

When first enabled, the Watchdog Timer is loaded with the value in the Watchdog Timer Reload registers. The Watchdog Timer then counts down to 000000H unless a WDT instruction is executed by the eZ8™ CPU. Execution of the WDT instruction causes the downcounter to be reloaded with the WDT Reload value stored in the Watchdog Timer Reload registers. Counting resumes following the reload operation.

When the 64K Series devices are operating in DEBUG Mode (through the On-Chip Debugger), the Watchdog Timer is continuously refreshed to prevent spurious Watchdog Timer time-outs.

Watchdog Timer Time-Out Response

The Watchdog Timer times out when the counter reaches 000000H. A time-out of the Watchdog Timer generates either an interrupt or a Reset. The WDT_RES Option Bit determines the time-out response of the Watchdog Timer. For information on programming of the WDT_RES Option Bit, see Option Bits on page 195.

WDT Interrupt in Normal Operation

If configured to generate an interrupt when a time-out occurs, the Watchdog Timer issues an interrupt request to the interrupt controller and sets the WDT status bit in the Watchdog Timer Control register. If interrupts are enabled, the eZ8 CPU responds to the interrupt request by fetching the Watchdog Timer interrupt vector and executing code from the vector address. After time-out and interrupt generation, the Watchdog Timer counter rolls over to its maximum value of FFFFFH and continues counting. The Watchdog Timer counter is not automatically returned to its Reload Value.

WDT Interrupt in STOP Mode

If configured to generate an interrupt when a time-out occurs and the 64K Series devices are in STOP mode, the Watchdog Timer automatically initiates a Stop Mode Recovery and generates an interrupt request. Both the WDT status bit and the STOP bit in the Watchdog Timer Control register are set to 1 following WDT time-out in STOP mode. For more information on Stop Mode Recovery, see Reset and Stop Mode Recovery on page 47.

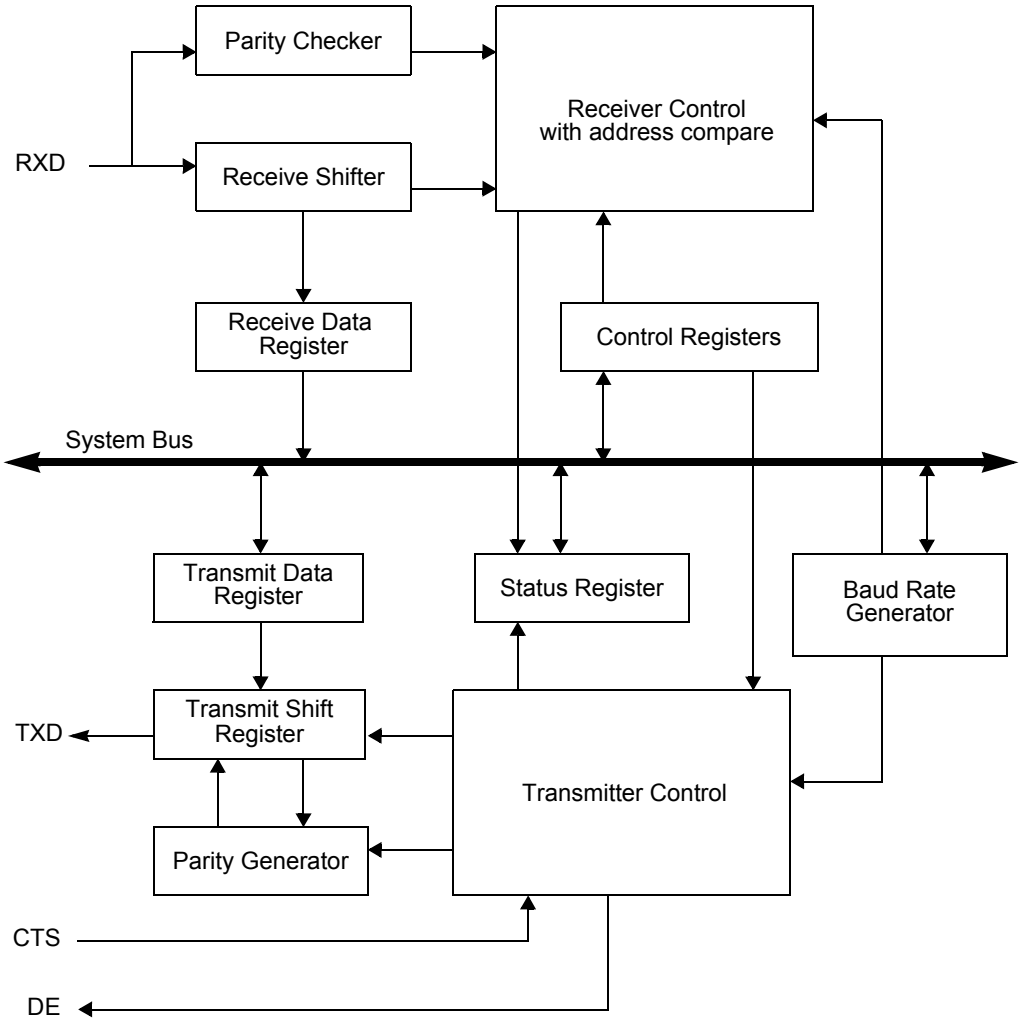

Figure 13. UART Block Diagram

Operation

Data Format

The UART always transmits and receives data in an 8-bit data format, least-significant bit first. An even or odd parity bit can be optionally added to the data stream. Each character begins with an active Low Start bit and ends with either 1 or 2 active High Stop bits. Figure 14 and Figure 15 on page 105 displays the asynchronous data format employed by the UART without parity and with parity, respectively.

Receiver Interrupts

The receiver generates an interrupt when any of the following occurs:

- A data byte has been received and is available in the UART Receive Data register. This interrupt can be disabled independent of the other receiver interrupt sources. The received data interrupt occurs once the receive character has been received and placed in the Receive Data register. Software must respond to this received data available condition before the next character is completely received to avoid an overrun error.

Note: *In MULTIPROCESSOR mode (MPEN = 1), the receive data interrupts are dependent on the multiprocessor configuration and the most recent address byte.*

- A break is received
- An overrun is detected
- A data framing error is detected

UART Overrun Errors

When an overrun error condition occurs the UART prevents overwriting of the valid data currently in the Receive Data register. The Break Detect and Overrun status bits are not displayed until after the valid data has been read.

After the valid data has been read, the UART Status 0 register is updated to indicate the overrun condition (and Break Detect, if applicable). The RDA bit is set to 1 to indicate that the Receive Data register contains a data byte. However, because the overrun error occurred, this byte may not contain valid data and should be ignored. The BRKD bit indicates if the overrun was caused by a break condition on the line. After reading the status byte indicating an overrun error, the Receive Data register must be read again to clear the error bits is the UART Status 0 register. Updates to the Receive Data register occur only when the next data word is received.

UART Data and Error Handling Procedure

Figure 18 on page 113 displays the recommended procedure for use in UART receiver interrupt service routines.

The Master and Slave are each capable of exchanging a character of data during a sequence of NUMBITS clock cycles (see NUMBITS field in the SPI Mode Register on page 140). In both Master and Slave SPI devices, data is shifted on one edge of the SCK and is sampled on the opposite edge where data is stable. Edge polarity is determined by the SPI phase and polarity control.

Slave Select

The active Low Slave Select ($\overline{SS}$) input signal selects a Slave SPI device. $\overline{SS}$ must be Low prior to all data communication to and from the Slave device. $\overline{SS}$ must stay Low for the full duration of each character transferred. The $\overline{SS}$ signal may stay Low during the transfer of multiple characters or may deassert between each character.

When the SPI is configured as the only Master in an SPI system, the $\overline{SS}$ pin can be set as either an input or an output. For communication between the Z8F642x family Z8R642x family device's SPI Master and external Slave devices, the $\overline{SS}$ signal, as an output, can assert the $\overline{SS}$ input pin on one of the Slave devices. Other GPIO output pins can also be employed to select external SPI Slave devices.

When the SPI is configured as one Master in a multi-master SPI system, the $\overline{SS}$ pin must be set as an input. The $\overline{SS}$ input signal on the Master must be High. If the $\overline{SS}$ signal goes Low (indicating another Master is driving the SPI bus), a Collision error Flag is set in the SPI Status register.

SPI Clock Phase and Polarity Control

The SPI supports four combinations of serial clock phase and polarity using two bits in the SPI Control register. The clock polarity bit, CLKPOL, selects an active high or active Low clock and has no effect on the transfer format. Table 62 lists the SPI Clock Phase and Polarity Operation parameters. The clock phase bit, PHASE, selects one of two fundamentally different transfer formats. For proper data transmission, the clock phase and polarity must be identical for the SPI Master and the SPI Slave. The Master always places data on the MOSI line a half-cycle before the receive clock edge (SCK signal), in order for the Slave to latch the data.

Table 62. SPI Clock Phase (PHASE) and Clock Polarity (CLKPOL) Operation

PHASE	CLKPOL	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	Falling	Rising	Low
0	1	Rising	Falling	High
1	0	Rising	Falling	Low
1	1	Falling	Rising	High

In order for a receive (read) DMA transaction to send a Not Acknowledge on the last byte, the receive DMA must be set up to receive n-1 bytes, then software must set the NAK bit and receive the last (nth) byte directly.

Start and Stop Conditions

The master (I^2C) drives all Start and Stop signals and initiates all transactions. 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. The START and STOP bits in the I^2C Control register control the sending of the Start and Stop conditions. A master is also allowed to end one transaction and begin a new one by issuing a Restart. This is accomplished by setting the START bit at the end of a transaction, rather than the STOP bit. Note that the Start condition not sent until the START bit is set and data has been written to the I^2C Data register.

Master Write and Read Transactions

The following sections provide a recommended procedure for performing I^2C write and read transactions from the I^2C Controller (master) to slave I^2C devices. In general software should rely on the TDRE, RDRF and NCKI bits of the status register (these bits generate interrupts) to initiate software actions. When using interrupts or DMA, the TXI bit is set to start each transaction and cleared at the end of each transaction to eliminate a ‘trailing’ Transmit interrupt.

Caution should be used in using the ACK status bit within a transaction because it is difficult for software to tell when it is updated by hardware.

When writing data to a slave, the I^2C pauses at the beginning of the Acknowledge cycle if the data register has not been written with the next value to be sent (TDRE bit in the I^2C Status register = 1). In this scenario where software is not keeping up with the I^2C bus (TDRE asserted longer than one byte time), the Acknowledge clock cycle for byte n is delayed until the Data register is written with byte n + 1, and appears to be grouped with the data clock cycles for byte n+1. If either the START or STOP bit is set, the I^2C does not pause prior to the Acknowledge cycle because no additional data is sent.

When a Not Acknowledge condition is received during a write (either during the address or data phases), the I^2C Controller generates the Not Acknowledge interrupt (NCKI = 1) and pause until either the STOP or START bit is set. Unless the Not Acknowledge was received on the last byte, the Data register will already have been written with the next address or data byte to send. In this case the FLUSH bit of the Control register should be set at the same time the STOP or START bit is set to remove the stale transmit data and enable subsequent Transmit interrupts.

When reading data from the slave, the I^2C pauses after the data Acknowledge cycle until the receive interrupt is serviced and the RDRF bit of the status register is cleared by

Write Transaction with a 7-Bit Address

Figure 29 displays the data transfer format for a 7-bit addressed slave. Shaded regions indicate data transferred from the I²C Controller to slaves and unshaded regions indicate data transferred from the slaves to the I²C Controller.

S	Slave Address	W = 0	A	Data	A	Data	A	Data	A/$\overline{A}$	P/S

Figure 29. 7-Bit Addressed Slave Data Transfer Format

Follow the steps below for a transmit operation to a 7-bit addressed slave:

1. Software asserts the IEN bit in the I²C Control register.
2. Software asserts the TXI bit of the I²C Control register to enable Transmit interrupts.
3. The I²C interrupt asserts, because the I²C Data register is empty
4. Software responds to the TDRE bit by writing a 7-bit slave address plus write bit (=0) to the I²C Data register.
5. Software asserts the START bit of the I²C Control register.
6. The I²C Controller sends the START condition to the I²C slave.
7. The I²C Controller loads the I²C Shift register with the contents of the I²C Data register.
8. After one bit of address has been shifted out by the SDA signal, the Transmit interrupt is asserted (TDRE = 1).
9. Software responds by writing the transmit data into the I²C Data register.
10. The I²C Controller shifts the rest of the address and write bit out by the SDA signal.
11. If the I²C slave sends an acknowledge (by pulling the SDA signal low) during the next high period of SCL the I²C Controller sets the ACK bit in the I²C Status register. Continue with step 12.

    If the slave does not acknowledge, the Not Acknowledge interrupt occurs (NCKI bit is set in the Status register, ACK bit is cleared). Software responds to the Not Acknowledge interrupt by setting the STOP and FLUSH bits and clearing the TXI bit. The I²C Controller sends the STOP condition on the bus and clears the STOP and NCKI bits. The transaction is complete (ignore the following steps).
12. The I²C Controller loads the contents of the I²C Shift register with the contents of the I²C Data register.

Electrical Characteristics

Absolute Maximum Ratings

Stresses greater than those listed in Table 105 may cause permanent damage to the device. These ratings are stress ratings only. Operation of the device at any condition outside those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability. For improved reliability, unused inputs must be tied to one of the supply voltages (V_{DD} or V_{SS}).

Table 105. Absolute Maximum Ratings

Parameter	Minimum	Maximum	Units	Notes
Ambient temperature under bias	-40	+125	C	
Storage temperature	-65	+150	C	
Voltage on any pin with respect to V_{SS}	-0.3	+5.5	V	1
Voltage on V_{DD} pin with respect to V_{SS}	-0.3	+3.6	V	
Maximum current on input and/or inactive output pin	-5	+5	µA	
Maximum output current from active output pin	-25	+25	mA	
80-Pin QFP Maximum Ratings at –40 °C to 70 °C				
Total power dissipation		550	mW	
Maximum current into V_{DD} or out of V_{SS}		150	mA	
80-Pin QFP Maximum Ratings at 70 °C to 125 °C				
Total power dissipation		200	mW	
Maximum current into V_{DD} or out of V_{SS}		56	mA	
68-Pin PLCC Maximum Ratings at –40 °C to 70 °C				
Total power dissipation		1000	mW	
Maximum current into V_{DD} or out of V_{SS}		275	mA	
68-Pin PLCC Maximum Ratings at 70 °C to 125 °C				
Total power dissipation		500	mW	

eZ8™ CPU Instruction Set

Assembly Language Programming Introduction

The eZ8 CPU assembly language provides a means for writing an application program without having to be concerned with actual memory addresses or machine instruction formats. A program written in assembly language is called a source program. Assembly language allows the use of symbolic addresses to identify memory locations. It also allows mnemonic codes (opcodes and operands) to represent the instructions themselves. The opcodes identify the instruction while the operands represent memory locations, registers, or immediate data values.

Each assembly language program consists of a series of symbolic commands called statements. Each statement can contain labels, operations, operands and comments.

Labels can be assigned to a particular instruction step in a source program. The label identifies that step in the program as an entry point for use by other instructions.

The assembly language also includes assembler directives that supplement the machine instruction. The assembler directives, or pseudo-ops, are not translated into a machine instruction. Rather, the pseudo-ops are interpreted as directives that control or assist the assembly process.

The source program is processed (assembled) by the assembler to obtain a machine language program called the object code. The object code is executed by the eZ8 CPU. An example segment of an assembly language program is detailed in the following example.

Assembly Language Source Program Example

```
JP START          ; Everything after the semicolon is a comment.
START:            ; A label called “START”. The first instruction (JP START) in this
                  ; example causes program execution to jump to the point within the
                  ; program where the START label occurs.
LD R4, R7         ; A Load (LD) instruction with two operands. The first operand,
                  ; Working Register R4, is the destination. The second operand,
                  ; Working Register R7, is the source. The contents of R7 is
                  ; written into R4.
LD 234H, #%01     ; Another Load (LD) instruction with two operands.
                  ; The first operand, Extended Mode Register Address 234H,
                  ; identifies the destination. The second operand, Immediate Data
```

Table 124. Condition Codes (Continued)

Binary	Hex	Assembly Mnemonic	Definition	Flag Test Operation
0011	3	ULE	Unsigned Less Than or Equal	(C OR Z) = 1
0100	4	OV	Overflow	V = 1
0101	5	MI	Minus	S = 1
0110	6	Z	Zero	Z = 1
0110	6	EQ	Equal	Z = 1
0111	7	C	Carry	C = 1
0111	7	ULT	Unsigned Less Than	C = 1
1000	8	T (or blank)	Always True	–
1001	9	GE	Greater Than or Equal	(S XOR V) = 0
1010	A	GT	Greater Than	(Z OR (S XOR V)) = 0
1011	B	UGT	Unsigned Greater Than	(C = 0 AND Z = 0) = 1
1100	C	NOV	No Overflow	V = 0
1101	D	PL	Plus	S = 0
1110	E	NZ	Non-Zero	Z = 0
1110	E	NE	Not Equal	Z = 0
1111	F	NC	No Carry	C = 0
1111	F	UGE	Unsigned Greater Than or Equal	C = 0

eZ8 CPU Instruction Classes

eZ8 CPU instructions can be divided functionally into the following groups:

- Arithmetic
- Bit Manipulation
- Block Transfer
- CPU Control
- Load
- Logical
- Program Control
- Rotate and Shift