



Welcome to [E-XFL.COM](https://www.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	I <sup>2</sup> C, IrDA, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	46
Program Memory Size	16KB (16K 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	<a href="https://www.e-xfl.com/product-detail/zilog/z8f1622ar020sc2104">https://www.e-xfl.com/product-detail/zilog/z8f1622ar020sc2104</a>

**Table 3. Signal Descriptions (Continued)**

Signal Mnemonic	I/O	Description
SCK	I/O	SPI Serial Clock. The SPI master supplies this pin. If the Z8 Encore! XP 64K Series Flash Microcontrollers is the SPI master, this pin is an output. If the Z8 Encore! XP 64K Series Flash Microcontrollers is the SPI slave, this pin is an input. It is multiplexed with a general-purpose I/O pin.
MOSI	I/O	Master-Out/Slave-In. This signal is the data output from the SPI master device and the data input to the SPI slave device. It is multiplexed with a general-purpose I/O pin.
MISO	I/O	Master-In/Slave-Out. This pin is the data input to the SPI master device and the data output from the SPI slave device. It is multiplexed with a general-purpose I/O pin.
<b>UART Controllers</b>		
TXD0 / TXD1	O	Transmit Data. These signals are the transmit outputs from the UARTs. The TXD signals are multiplexed with general-purpose I/O pins.
RXD0 / RXD1	I	Receive Data. These signals are the receiver inputs for the UARTs and IrDAs. The RXD signals are multiplexed with general-purpose I/O pins.
$\overline{\text{CTS0}}$ / $\overline{\text{CTS1}}$	I	Clear To Send. These signals are control inputs for the UARTs. The $\overline{\text{CTS}}$ signals are multiplexed with general-purpose I/O pins.
DE0 / DE1	O	Driver Enable. This signal allows automatic control of external RS-485 drivers. This signal is approximately the inverse of the Transmit Empty (TXE) bit in the UART Status 0 register. The DE signal may be used to ensure an external RS-485 driver is enabled when data is transmitted by the UART.
<b>Timers</b>		
T0OUT/T1OUT/ T2OUT/T3OUT	O	Timer Output 0-3. These signals are output pins from the timers. The Timer Output signals are multiplexed with general-purpose I/O pins. T3OUT is not available in 44-pin package devices.
T0IN/T1IN/ T2IN/T3IN	I	Timer Input 0-3. These signals are used as the capture, gating and counter inputs. The Timer Input signals are multiplexed with general-purpose I/O pins. T3IN is not available in 44-pin package devices.
<b>Analog</b>		
ANA[11:0]	I	Analog Input. These signals are inputs to the ADC. The ADC analog inputs are multiplexed with general-purpose I/O pins.
VREF	I	Analog-to-Digital converter reference voltage input. The VREF pin must be left unconnected (or capacitively coupled to analog ground) if the internal voltage reference is selected as the ADC reference voltage.
<b>Oscillators</b>		

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

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

### 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,  $\overline{SS}$  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 =  $\overline{SS}$  pin driven High
- Slave Select I/O  
0 =  $\overline{SS}$  pin configured as an input  
1 =  $\overline{SS}$  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

**Table 8. Reset and Stop Mode Recovery Characteristics and Latency**

Reset Type	Reset Characteristics and Latency		
	Control Registers	eZ8 <sup>™</sup> CPU	Reset Latency (Delay)
System reset	Reset (as applicable)	Reset	66 WDT Oscillator cycles + 16 System Clock cycles
Stop Mode Recovery	Unaffected, except WDT_CTL register	Reset	66 WDT Oscillator cycles + 16 System Clock cycles

### System Reset

During a system reset, the 64K Series devices are held in Reset for 66 cycles of the Watchdog Timer oscillator followed by 16 cycles of the system clock. At the beginning of Reset, all GPIO pins are configured as inputs.

During Reset, the eZ8 CPU and on-chip peripherals are idle; however, the on-chip crystal oscillator and Watchdog Timer oscillator continue to run. The system clock begins operating following the Watchdog Timer oscillator cycle count. The eZ8 CPU and on-chip peripherals remain idle through the 16 cycles of the system clock.

Upon Reset, control registers within the Register File that have a defined Reset value are loaded with their reset values. Other control registers (including the Stack Pointer, Register Pointer, and Flags) and general-purpose RAM are undefined following Reset. The eZ8 CPU fetches the Reset vector at Program Memory addresses 0002H and 0003H and loads that value into the Program Counter. Program execution begins at the Reset vector address.

### Reset Sources

[Table 9](#) lists the reset sources as a function of the operating mode. The text following provides more detailed information on the individual Reset sources. A Power-On Reset/Voltage Brownout event always takes priority over all other possible reset sources to ensure a full system reset occurs.

# General-Purpose I/O

## Overview

The 64K Series products support a maximum of seven 8-bit ports (Ports A–G) and one 4-bit port (Port H) for general-purpose input/output (GPIO) operations. Each port consists of control and data registers. The GPIO control registers are used to determine data direction, open-drain, output drive current and alternate pin functions. Each port pin is individually programmable. All ports (except B and H) support 5 V-tolerant inputs.

## GPIO Port Availability By Device

[Table 11](#) lists the port pins available with each device and package type.

**Table 11. Port Availability by Device and Package Type**

Device	Packages	Port A	Port B	Port C	Port D	Port E	Port F	Port G	Port H
Z8X1621	40-pin	[7:0]	[7:0]	[6:0]	[6:3, 1:0]	–	–	–	–
Z8X1621	44-pin	[7:0]	[7:0]	[7:0]	[6:0]	–	–	–	–
Z8X1622	64- and 68-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7]	[3]	[3:0]
Z8X2421	40-pin	[7:0]	[7:0]	[6:0]	[6:3, 1:0]	–	–	–	–
Z8X2421	44-pin	[7:0]	[7:0]	[7:0]	[6:0]	–	–	–	–
Z8X2422	64- and 68-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7]	[3]	[3:0]
Z8X3221	40-pin	[7:0]	[7:0]	[6:0]	[6:3, 1:0]	–	–	–	–
Z8X3221	44-pin	[7:0]	[7:0]	[7:0]	[6:0]	–	–	–	–
Z8X3222	64- and 68-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7]	[3]	[3:0]
Z8X4821	40-pin	[7:0]	[7:0]	[6:0]	[6:3, 1:0]	–	–	–	–
Z8X4821	44-pin	[7:0]	[7:0]	[7:0]	[6:0]	–	–	–	–
Z8X4822	64- and 68-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7]	[3]	[3:0]

# 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<sup>™</sup> CPU Core User Manual (UM0128)* available for download at [www.zilog.com](http://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.

- Executing a Trap instruction.
- Illegal Instruction trap.

## Interrupt Vectors and Priority

The interrupt controller supports three levels of interrupt priority. Level 3 is the highest priority, Level 2 is the second highest priority, and Level 1 is the lowest priority. If all of the interrupts were enabled with identical interrupt priority (all as Level 2 interrupts, for example), then interrupt priority would be assigned from highest to lowest as specified in [Table 23](#) on page 68. Level 3 interrupts always have higher priority than Level 2 interrupts which, in turn, always have higher priority than Level 1 interrupts. Within each interrupt priority level (Level 1, Level 2, or Level 3), priority is assigned as specified in [Table 23](#) on page 68. Reset, Watchdog Timer interrupt (if enabled), and Illegal Instruction Trap always have highest priority.

## Interrupt Assertion

Interrupt sources assert their interrupt requests for only a single system clock period (single pulse). When the interrupt request is acknowledged by the eZ8 CPU, the corresponding bit in the Interrupt Request register is cleared until the next interrupt occurs. Writing a 0 to the corresponding bit in the Interrupt Request register likewise clears the interrupt request.



**Caution:** *The following style of coding to clear 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
AND r0, MASK
LDX IRQ0, r0
```

*To avoid missing interrupts, the following style of coding to clear bits in the Interrupt Request 0 register is recommended:*

### Good coding style that avoids lost interrupt requests:

```
ANDX IRQ0, MASK
```

## Software Interrupt Assertion

Program code can generate interrupts directly. Writing a 1 to the desired bit in the Interrupt Request register triggers an interrupt (assuming that 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:** *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<sup>™</sup> 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.

T1I—Timer 1 Interrupt Request

0 = No interrupt request is pending for Timer 1.

1 = An interrupt request from Timer 1 is awaiting service.

T0I—Timer 0 Interrupt Request

0 = No interrupt request is pending for Timer 0.

1 = An interrupt request from Timer 0 is awaiting service.

U0RXI—UART 0 Receiver Interrupt Request

0 = No interrupt request is pending for the UART 0 receiver.

1 = An interrupt request from the UART 0 receiver is awaiting service.

U0TXI—UART 0 Transmitter Interrupt Request

0 = No interrupt request is pending for the UART 0 transmitter.

1 = An interrupt request from the UART 0 transmitter is awaiting service.

I<sup>2</sup>CI— I<sup>2</sup>C Interrupt Request

0 = No interrupt request is pending for the I<sup>2</sup>C.

1 = An interrupt request from the I<sup>2</sup>C is awaiting service.

SPII—SPI Interrupt Request

0 = No interrupt request is pending for the SPI.

1 = An interrupt request from the SPI is awaiting service.

ADCI—ADC Interrupt Request

0 = No interrupt request is pending for the Analog-to-Digital Converter.

1 = An interrupt request from the Analog-to-Digital Converter is awaiting service.

## Interrupt Request 1 Register

The Interrupt Request 1 (IRQ1) register ([Table 25](#)) stores interrupt requests for both vectored and polled interrupts. When a request is presented to the interrupt controller, the corresponding bit in the IRQ1 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 1 register to determine if any interrupt requests are pending.

**Table 25. Interrupt Request 1 Register (IRQ1)**

BITS	7	6	5	4	3	2	1	0
FIELD	PAD7I	PAD6I	PAD5I	PAD4I	PAD3I	PAD2I	PAD1I	PAD0I
RESET	0							
R/W	R/W							
ADDR	FC3H							

## Timer 0-3 Control 1 Registers

The Timer 0-3 Control 1 (TxCTL1) registers enable/disable the timers, set the prescaler value, and determine the timer operating mode.

**Table 46. Timer 0-3 Control 1 Register (TxCTL1)**

BITS	7	6	5	4	3	2	1	0
FIELD	TEN	TPOL	PRES			TMODE		
RESET	0							
R/W	R/W							
ADDR	F07H, F0FH, F17H, F1FH							

TEN—Timer Enable

0 = Timer is disabled.

1 = Timer enabled to count.

TPOL—Timer Input/Output Polarity

Operation of this bit is a function of the current operating mode of the timer.

### ONE-SHOT mode

When the timer is disabled, the Timer Output signal is set to the value of this bit.

When the timer is enabled, the Timer Output signal is complemented upon timer Reload.

### CONTINUOUS mode

When the timer is disabled, the Timer Output signal is set to the value of this bit.

When the timer is enabled, the Timer Output signal is complemented upon timer Reload.

### COUNTER mode

When the timer is disabled, the Timer Output signal is set to the value of this bit.

When the timer is enabled, the Timer Output signal is complemented upon timer Reload.

0 = Count occurs on the rising edge of the Timer Input signal.

1 = Count occurs on the falling edge of the Timer Input signal.

### PWM mode

0 = Timer Output is forced Low (0) when the timer is disabled. When enabled, the Timer Output is forced High (1) upon PWM count match and forced Low (0) upon Reload.

defined to be 1 through 8 bits by the NUMBITS field in the SPI Mode register. In slave mode it is not necessary for  $\overline{SS}$  to deassert between characters to generate the interrupt. The SPI in Slave mode can also generate an interrupt if the  $\overline{SS}$  signal deasserts prior to transfer of all the bits in a character (see description of slave abort error above). Writing a 1 to the IRQ bit in the SPI Status Register clears the pending SPI interrupt request. The IRQ bit must be cleared to 0 by the Interrupt Service Routine to generate future interrupts. To start the transfer process, an SPI interrupt may be forced by software writing a 1 to the STR bit in the SPICTL register.

If the SPI is disabled, an SPI interrupt can be generated by a Baud Rate Generator time-out. This timer function must be enabled by setting the BIRQ bit in the SPICTL register. This Baud Rate Generator time-out does not set the IRQ bit in the SPISTAT register, just the SPI interrupt bit in the interrupt controller.

## SPI Baud Rate Generator

In SPI 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 SPI Baud Rate High and Low Byte registers combine to form a 16-bit reload value, BRG[15:0], for the SPI Baud Rate Generator. The SPI 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 SPI is disabled, the Baud Rate Generator can function as a basic 16-bit timer with interrupt on time-out. Follow the steps below to configure the Baud Rate Generator as a timer with interrupt on time-out:

1. Disable the SPI by clearing the SPIEN bit in the SPI Control register to 0.
2. Load the desired 16-bit count value into the SPI Baud Rate High and Low Byte registers.
3. Enable the Baud Rate Generator timer function and associated interrupt by setting the BIRQ bit in the SPI Control register to 1.

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

$$\text{Interrupt Interval (s)} = \text{System Clock Period (s)} \times \text{BRG}[15:0]$$

## SPI Control Register Definitions

### SPI Data Register

The SPI Data register ([Table 63](#)) stores both the outgoing (transmit) data and the incoming (receive) data. Reads from the SPI Data register always return the current contents of the 8-bit shift register. Data is shifted out starting with bit 7. The last bit received resides in bit position 0.

With the SPI configured as a Master, writing a data byte to this register initiates the data transmission. With the SPI configured as a Slave, writing a data byte to this register loads the shift register in preparation for the next data transfer with the external Master. In either the Master or Slave modes, if a transmission is already in progress, writes to this register are ignored and the Overrun error Flag, OVR, is set in the SPI Status register.

When the character length is less than 8 bits (as set by the NUMBITS field in the SPI Mode register), the transmit character must be left justified in the SPI Data register. A received character of less than 8 bits is right justified (last bit received is in bit position 0). For example, if the SPI is configured for 4-bit characters, the transmit characters must be written to SPIDATA[7:4] and the received characters are read from SPIDATA[3:0].

**Table 63. SPI Data Register (SPIDATA)**

BITS	7	6	5	4	3	2	1	0
FIELD	DATA							
RESET	X							
R/W	R/W							
ADDR	F60H							

DATA—Data  
Transmit and/or receive data.

### SPI Control Register

The SPI Control register (see [Table 64](#) on page 138) configures the SPI for transmit and receive operations.

16. If the I<sup>2</sup>C slave sends an acknowledge by pulling the SDA signal low during the next high period of SCL, the I<sup>2</sup>C Controller sets the ACK bit in the I<sup>2</sup>C Status register. Continue with [step 17](#).

If the slave does not acknowledge the second address byte or one of the data bytes, the I<sup>2</sup>C Controller sets the NCKI bit and clears the ACK bit in the I<sup>2</sup>C Status register. Software responds to the Not Acknowledge interrupt by setting the STOP and FLUSH bits and clearing the TXI bit. The I<sup>2</sup>C Controller sends the STOP condition on the bus and clears the STOP and NCKI bits. The transaction is complete (ignore the following steps).

17. The I<sup>2</sup>C Controller shifts the data out by the SDA signal. After the first bit is sent, the Transmit interrupt is asserted.
18. If more bytes remain to be sent, return to [step 14](#).
19. If the last byte is currently being sent, software sets the STOP bit of the I<sup>2</sup>C Control register (or START bit to initiate a new transaction). In the STOP case, software also clears the TXI bit of the I<sup>2</sup>C Control register at the same time.
20. The I<sup>2</sup>C Controller completes transmission of the last data byte on the SDA signal.
21. The slave may either Acknowledge or Not Acknowledge the last byte. Because either the STOP or START bit is already set, the NCKI interrupt does not occur.
22. The I<sup>2</sup>C Controller sends the STOP (or RESTART) condition to the I<sup>2</sup>C bus and clears the STOP (or START) bit.

### Read Transaction with a 7-Bit Address

[Figure 32](#) displays the data transfer format for a read operation to a 7-bit addressed slave. The shaded regions indicate data transferred from the I<sup>2</sup>C Controller to slaves and unshaded regions indicate data transferred from the slaves to the I<sup>2</sup>C Controller.

S	Slave Address	R = 1	A	Data	A	Data	$\bar{A}$	P/S
---	---------------	-------	---	------	---	------	-----------	-----

**Figure 32. Receive Data Transfer Format for a 7-Bit Addressed Slave**

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

1. Software writes the I<sup>2</sup>C Data register with a 7-bit slave address plus the read bit (=1).
2. Software asserts the START bit of the I<sup>2</sup>C Control register.
3. If this is a single byte transfer, Software asserts the NAK bit of the I<sup>2</sup>C Control register so that after the first byte of data has been read by the I<sup>2</sup>C Controller, a Not Acknowledge is sent to the I<sup>2</sup>C slave.

#### DMAA\_ADDR—DMA\_ADC Address

These bits specify the seven most-significant bits of the 12-bit Register File addresses used for storing the ADC output data. The ADC Analog Input Number defines the five least-significant bits of the Register File address. Full 12-bit address is {DMAA\_ADDR[7:1], 4-bit ADC Analog Input Number, 0}.

Reserved

This bit is reserved and must be 0.

### DMA\_ADC Control Register

The DMA\_ADC Control register (Table 84 on page 172) enables and sets options (DMA enable and interrupt enable) for ADC operation.

**Table 84. DMA\_ADC Control Register (DMAACTL)**

BITS	7	6	5	4	3	2	1	0
FIELD	DAEN	IRQEN	Reserved		ADC_IN			
RESET	0							
R/W	R/W							
ADDR	FBEH							

#### DAEN—DMA\_ADC Enable

0 = DMA\_ADC is disabled and the ADC Analog Input Number (ADC\_IN) is reset to 0.

1 = DMA\_ADC is enabled.

#### IRQEN—Interrupt Enable

0 = DMA\_ADC does not generate any interrupts.

1 = DMA\_ADC generates an interrupt after transferring data from the last ADC Analog Input specified by the ADC\_IN field.

Reserved

These bits are reserved and must be 0.

#### ADC\_IN—ADC Analog Input Number

These bits set the number of ADC Analog Inputs to be used in the continuous update (data conversion followed by DMA data transfer). The conversion always begins with ADC Analog Input 0 and then progresses sequentially through the other selected ADC Analog Inputs.

0000 = ADC Analog Input 0 updated.

0001 = ADC Analog Inputs 0-1 updated.

0010 = ADC Analog Inputs 0-2 updated.

0011 = ADC Analog Inputs 0-3 updated.

0100 = ADC Analog Inputs 0-4 updated.







; value 01H, is the source. The value 01H is written into the  
; Register at address 234H.

Assembly Language Syntax

For proper instruction execution, eZ8 CPU assembly language syntax requires that the operands be written as ‘destination, source’. After assembly, the object code usually has the operands in the order ‘source, destination’, but ordering is opcode-dependent. The following instruction examples illustrate the format of some basic assembly instructions and the resulting object code produced by the assembler. This binary format must be followed if you prefer manual program coding or intend to implement your own assembler.

**Example 1:** If the contents of Registers 43H and 08H are added and the result is stored in 43H, the assembly syntax and resulting object code is:

Assembly Language Syntax Example 1

Assembly Language Code	ADD	43H,	08H	(ADD dst, src)
Object Code	04	08	43	(OPC src, dst)

**Example 2:** In general, when an instruction format requires an 8-bit register address, that address can specify any register location in the range 0–255 or, using Escaped Mode Addressing, a Working Register R0 - R15. If the contents of Register 43H and Working Register R8 are added and the result is stored in 43H, the assembly syntax and resulting object code is:

Assembly Language Syntax Example 2

Assembly Language Code	ADD	43H,	R8	(ADD dst, src)
Object Code	04	E8	43	(OPC src, dst)

Refer to the device-specific Product Specification to determine the exact register file range available. The register file size varies, depending on the device type.

eZ8 CPU Instruction Notation

In the eZ8 CPU Instruction Summary and Description sections, the operands, condition codes, status Flags, and address modes are represented by a notational shorthand that is described in [Table 122](#).

**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

Table 133. eZ8 CPU Instruction Summary (Continued)

Assembly Mnemonic	Symbolic Operation	Address Mode		Opcode(s) (Hex)	Flags						Fetch Cycles	Instr. Cycles
		dst	src		C	Z	S	V	D	H		
COM dst	dst ← ~dst	R		60	-	*	*	0	-	-	2	2
		IR		61							2	3
CP dst, src	dst - src	r	r	A2	*	*	*	*	-	-	2	3
		r	lr	A3							2	4
		R	R	A4							3	3
		R	IR	A5							3	4
		R	IM	A6							3	3
		IR	IM	A7							3	4
CPC dst, src	dst - src - C	r	r	1F A2	*	*	*	*	-	-	3	3
		r	lr	1F A3							3	4
		R	R	1F A4							4	3
		R	IR	1F A5							4	4
		R	IM	1F A6							4	3
		IR	IM	1F A7							4	4
CPCX dst, src	dst - src - C	ER	ER	1F A8	*	*	*	*	-	-	5	3
		ER	IM	1F A9							5	3
CPX dst, src	dst - src	ER	ER	A8	*	*	*	*	-	-	4	3
		ER	IM	A9							4	3
DA dst	dst ← DA(dst)	R		40	*	*	*	X	-	-	2	2
		IR		41							2	3
DEC dst	dst ← dst - 1	R		30	-	*	*	*	-	-	2	2
		IR		31							2	3
DECW dst	dst ← dst - 1	RR		80	-	*	*	*	-	-	2	5
		IRR		81							2	6
DI	IRQCTL[7] ← 0			8F	-	-	-	-	-	-	1	2
DJNZ dst, RA	dst ← dst - 1 if dst ≠ 0 PC ← PC + X	r		0A-FA	-	-	-	-	-	-	2	3

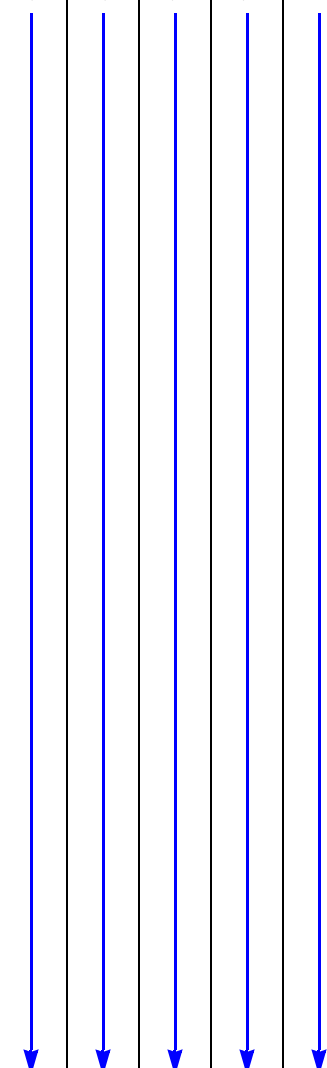
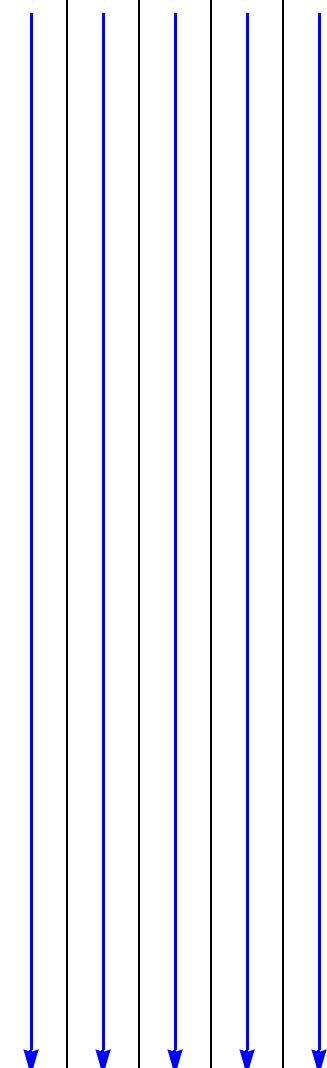
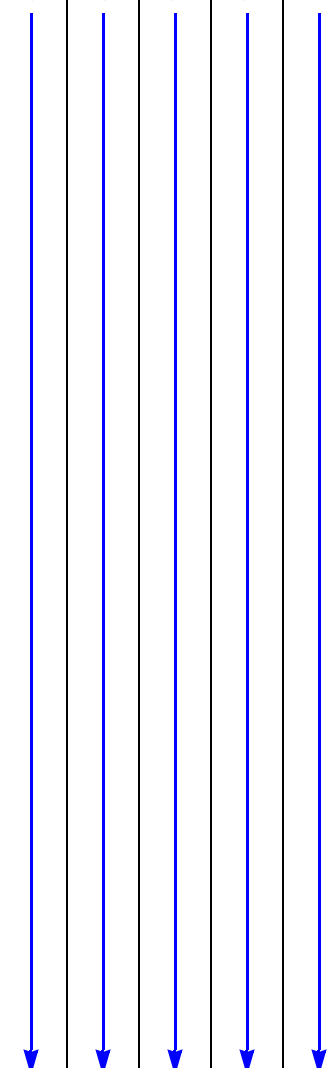
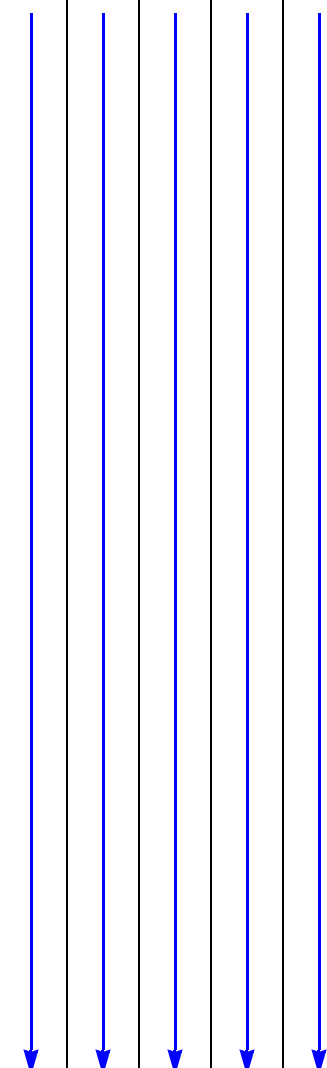
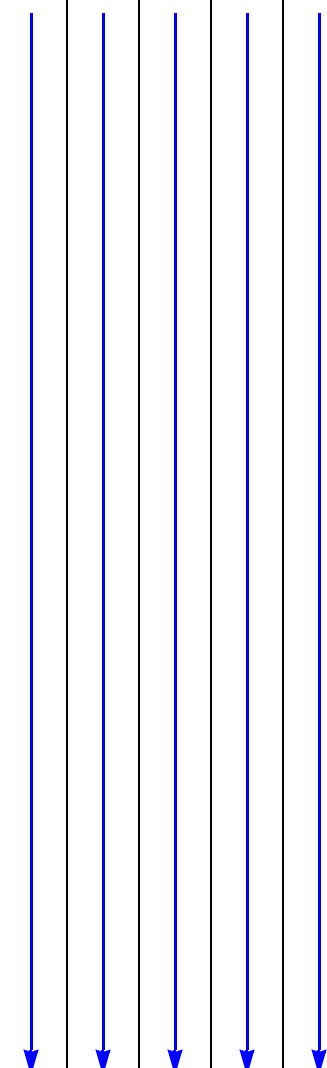
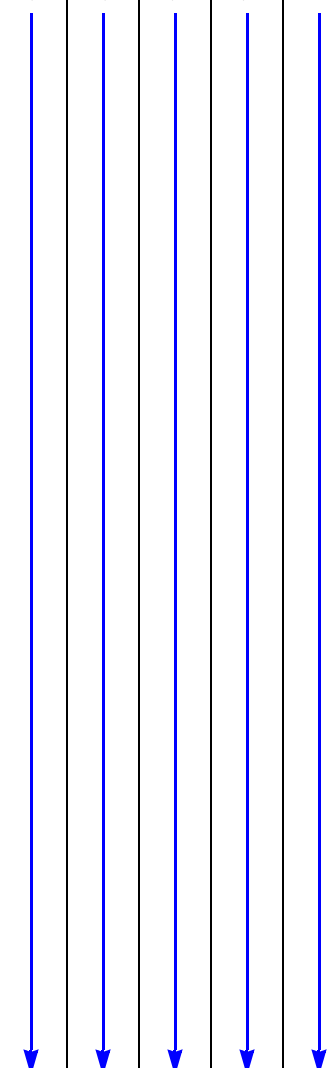
		Lower Nibble (Hex)																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
Upper Nibble (Hex)	0	1.2 BRK	2.2 SRP	2.3 ADD	2.4 ADD	3.3 ADD	3.4 ADD	3.3 ADD	3.4 ADD	4.3 ADDX	4.3 ADDX	2.3 DJNZ	2.2 JR	2.2 LD	3.2 JP	1.2 INC	1.2 NOP	
	1	2.2 RLC	2.3 RLC	2.3 ADC	2.4 ADC	3.3 ADC	3.4 ADC	3.3 ADC	3.4 ADC	4.3 ADCX	4.3 ADCX							See 2nd Opcode Map
	2	2.2 INC	2.3 INC	2.3 SUB	2.4 SUB	3.3 SUB	3.4 SUB	3.3 SUB	3.4 SUB	4.3 SUBX	4.3 SUBX							1.2 ATM
	3	2.2 DEC	2.3 DEC	2.3 SBC	2.4 SBC	3.3 SBC	3.4 SBC	3.3 SBC	3.4 SBC	4.3 SBCX	4.3 SBCX							
	4	2.2 DA	2.3 DA	2.3 OR	2.4 OR	3.3 OR	3.4 OR	3.3 OR	3.4 OR	4.3 ORX	4.3 ORX							
	5	2.2 POP	2.3 POP	2.3 AND	2.4 AND	3.3 AND	3.4 AND	3.3 AND	3.4 AND	4.3 ANDX	4.3 ANDX							1.2 WDT
	6	2.2 COM	2.3 COM	2.3 TCM	2.4 TCM	3.3 TCM	3.4 TCM	3.3 TCM	3.4 TCM	4.3 TCMX	4.3 TCMX							1.2 STOP
	7	2.2 PUSH	2.3 PUSH	2.3 TM	2.4 TM	3.3 TM	3.4 TM	3.3 TM	3.4 TM	4.3 TMX	4.3 TMX							1.2 HALT
	8	2.5 DECW	2.6 DECW	2.5 LDE	2.9 LDEI	3.2 LDX	3.3 LDX	3.4 LDX	3.5 LDX	3.4 LDX	3.4 LDX							1.2 DI
	9	2.2 RL	2.3 RL	2.5 LDE	2.9 LDEI	3.2 LDX	3.3 LDX	3.4 LDX	3.5 LDX	3.3 LEA	3.5 LEA							1.2 EI
	A	2.5 INCW	2.6 INCW	2.3 CP	2.4 CP	3.3 CP	3.4 CP	3.3 CP	3.4 CP	4.3 CPX	4.3 CPX							1.4 RET
	B	2.2 CLR	2.3 CLR	2.3 XOR	2.4 XOR	3.3 XOR	3.4 XOR	3.3 XOR	3.4 XOR	4.3 XORX	4.3 XORX							1.5 IRET
	C	2.2 RRC	2.3 RRC	2.5 LDC	2.9 LDCI	2.3 JP	2.9 LDC		3.4 LD	3.2 PUSHX								1.2 RCF
	D	2.2 SRA	2.3 SRA	2.5 LDC	2.9 LDCI	2.6 CALL	2.2 BSWAP	3.3 CALL	3.4 LD	3.2 POPX								1.2 SCF
	E	2.2 RR	2.3 RR	2.2 BIT	2.3 LD	3.2 LD	3.3 LD	3.2 LD	3.3 LD	4.2 LDX	4.2 LDX							1.2 CCF
	F	2.2 SWAP	2.3 SWAP	2.6 TRAP	2.3 LD	2.8 MULT	3.3 LD	3.3 BTJ	3.4 BTJ									

Figure 60. First Opcode Map

- flash page select (FPS) 191
- flash status (FSTAT) 190
- GPIO port A-H address (PxADDR) 61
- GPIO port A-H alternate function sub-registers 63
- GPIO port A-H control address (PxCTL) 62
- GPIO port A-H data direction sub-registers 63
- I2C baud rate high (I2CBRH) 160, 161, 163
- I2C control (I2CCTL) 158
- I2C data (I2CDATA) 157
- I2C status 157
- I2C status (I2CSTAT) 157
- I2Cbaud rate low (I2CBRL) 161
- mode, SPI 140
- OCD control 209
- OCD status 210
- SPI baud rate high byte (SPIBRH) 142
- SPI baud rate low byte (SPIBRL) 142
- SPI control (SPICTL) 138
- SPI data (SPIDATA) 137
- SPI status (SPISTAT) 139
- status, I2C 157
- status, SPI 139
- UARTx baud rate high byte (UxBRH) 121
- UARTx baud rate low byte (UxBRL) 121
- UARTx Control 0 (UxCTL0) 117, 120
- UARTx control 1 (UxCTL1) 118
- UARTx receive data (UxRXD) 115
- UARTx status 0 (UxSTAT0) 115
- UARTx status 1 (UxSTAT1) 117
- UARTx transmit data (UxTXD) 114
- watch-dog timer control (WDTCTL) 100
- watch-dog timer reload high byte (WDTH) 102
- watch-dog timer reload low byte (WDTL) 102
- watch-dog timer reload upper byte (WDTU) 102
- register file 19
- register file address map 23
- register pair 243
- register pointer 244
- reset
  - and STOP mode characteristics 48
  - carry flag 247
  - controller 5
  - sources 48
- RET 249
- return 249
- RL 249
- RLC 249
- rotate and shift instructions 249
- rotate left 249
- rotate left through carry 249
- rotate right 249
- rotate right through carry 249
- RP 244
- RR 243, 249
- rr 243
- RRC 249
- S**
- SBC 246
- SCF 247
- SDA and SCL (IrDA) signals 145
- second opcode map after 1FH 264
- serial clock 131
- serial peripheral interface (SPI) 129
- set carry flag 247
- set register pointer 247
- shift right arithmetic 249
- shift right logical 250
- signal descriptions 14
- single-shot conversion (ADC) 177
- SIO 5
- slave data transfer formats (I2C) 151
- slave select 132
- software trap 249
- source operand 244
- SP 244
- SPI
  - architecture 129
  - baud rate generator 136
  - baud rate high and low byte register 142
  - clock phase 132
  - configured as slave 130
  - control register 137
  - control register definitions 137
  - data register 137