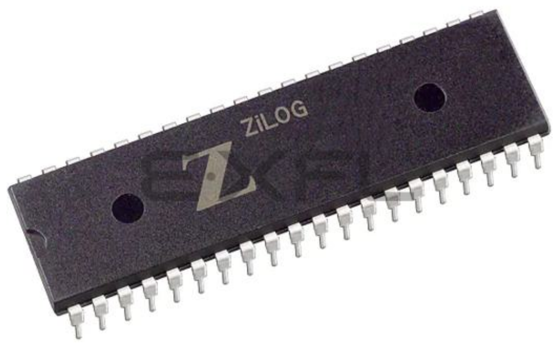


What is "Embedded - Microcontrollers"?

"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	31
Program Memory Size	48KB (48K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	4K x 8
Voltage - Supply (Vcc/Vdd)	3V ~ 3.6V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Through Hole
Package / Case	40-DIP (0.620", 15.75mm)
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z8f4801pm020sc

Table 10. Port Availability by Device and Package Type (Continued)

Device	Packages	Port A	Port B	Port C	Port D	Port E	Port F	Port G	Port H
Z8F6401	44-pin	[7:0]	[7:0]	[7:0]	[6:0]	-	-	-	-
Z8F6402	64- and 68-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7]	[3]	[3:0]
Z8F6403	80-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[3:0]

Architecture

Figure 64 illustrates a simplified block diagram of a GPIO port pin. In this figure, the ability to accommodate alternate functions and variable port current drive strength are not illustrated.

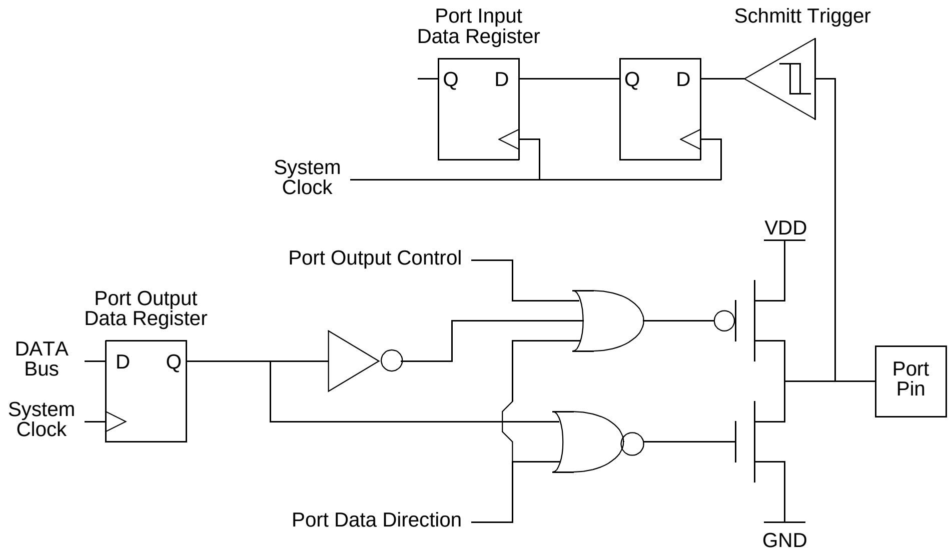


Figure 64. GPIO Port Pin Block Diagram

GPIO Alternate Functions

Many of the GPIO port pins can be used as both general-purpose I/O and to provide access to on-chip peripheral functions such as the timers and serial communication devices. The Port A-H Alternate Function sub-registers configure these pins for either general-purpose I/O or alternate function operation. When a pin is configured for alternate function, control

Architecture

Figure 65 illustrates a block diagram of the interrupt controller.

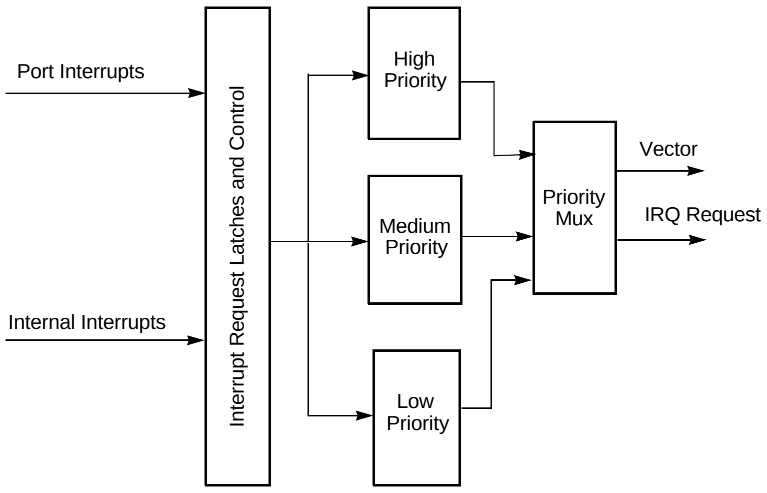


Figure 65. Interrupt Controller Block Diagram

Operation

Master Interrupt Enable

The master interrupt enable bit (`IRQE`) in the Interrupt Control register globally enables and disables interrupts.

Interrupts are globally enabled by any of the following actions:

- Execution of an EI (Enable Interrupt) instruction
- Execution of an IRET (Return from Interrupt) instruction
- Writing a 1 to the `IRQE` bit in the Interrupt Control register

Interrupts are globally disabled by any of the following actions:

- Execution of a DI (Disable Interrupt) instruction
- eZ8 CPU acknowledgement of an interrupt service request from the interrupt controller
- Writing a 0 to the `IRQE` bit in the Interrupt Control register
- Reset

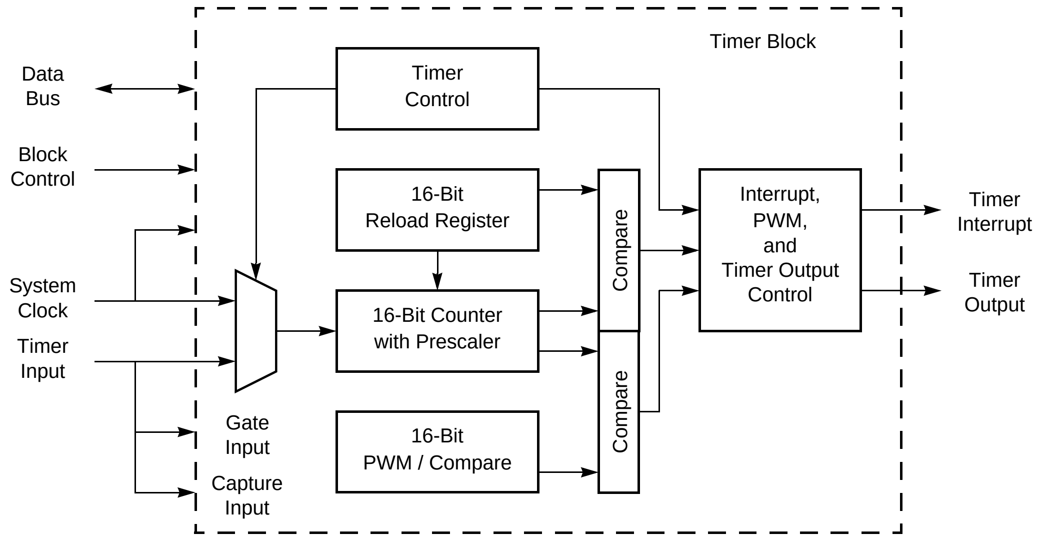


Figure 66. Timer Block Diagram

Operation

The timers are 16-bit up-counters. Minimum time-out delay is set by loading the value `0001H` into the Timer Reload High and Low Byte registers and setting the prescale value to 1. Maximum time-out delay is set by loading the value `0000H` into the Timer Reload High and Low Byte registers and setting the prescale value to 128. If the Timer reaches `FFFFH`, the timer rolls over to `0000H` and continues counting.

Timer Operating Modes

The timers can be configured to operate in the following modes:

One-Shot Mode

In One-Shot mode, 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 system clock. Upon reaching the Reload value, the timer generates an interrupt and the count value in the Timer High and Low Byte registers is reset to `0001H`. Then, the timer is automatically disabled and stops counting.

Also, if the Timer Output alternate function is enabled, the Timer Output pin changes state for one system clock cycle (from Low to High or from High to Low) upon timer Reload. If it is desired to have the Timer Output make a permanent state change upon One-Shot time-

Reserved
These bits are reserved and must be 0.

Watch-Dog Timer Reload Upper, High and Low Byte Registers

The Watch-Dog Timer Reload Upper, High and Low Byte (WDTU, WDTH, WDTL) registers (Tables 47 through 49) form the 24-bit reload value that is loaded into the Watch-Dog Timer when a WDT instruction executes. The 24-bit reload value is {WDTU[7:0], WDTH[7:0], WDTL[7:0]. Writing to these registers sets the desired Reload Value. Reading from these registers returns the current Watch-Dog Timer count value.

Caution: The 24-bit WDT Reload Value must not be set to a value less than `000004H` or unpredictable behavior may result.

Table 47. Watch-Dog Timer Reload Upper Byte Register (WDTU)

BITS	7	6	5	4	3	2	1	0
FIELD	WDTU							
RESET	1	1	1	1	1	1	1	1
R/W	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*
ADDR	FF1H							
R/W* - Read returns the current WDT count value. Write sets the desired Reload Value.								

WDTU—WDT Reload Upper Byte

Most significant byte (MSB), Bits[23:16], of the 24-bit WDT reload value.

Table 48. Watch-Dog Timer Reload High Byte Register (WDTH)

BITS	7	6	5	4	3	2	1	0
FIELD	WDTH							
RESET	1	1	1	1	1	1	1	1
R/W	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*
ADDR	FF2H							
R/W* - Read returns the current WDT count value. Write sets the desired Reload Value.								

WDTH—WDT Reload High Byte

Middle byte, Bits[15:8], of the 24-bit WDT reload value.

Table 49. Watch-Dog Timer Reload Low Byte Register (WDTL)

BITS	7	6	5	4	3	2	1	0
FIELD	WDTL							
RESET	1	1	1	1	1	1	1	1
R/W	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*	R/W*
ADDR	FF3H							
R/W* - Read returns the current WDT count value. Write sets the desired Reload Value.								

WDTL—WDT Reload Low

Least significant byte (LSB), Bits[7:0], of the 24-bit WDT reload value.

5. Check the `TDRE` bit in the UART Status 0 register to determine if the Transmit Data register is empty (indicated by a 1). If empty, continue to Step 6. If the Transmit Data register is full (indicated by a 0), continue to monitor the `TDRE` bit until the Transmit Data register becomes available to receive new data.
6. Write the data byte to the UART Transmit Data register. The transmitter automatically transfers the data to the Transmit Shift register and transmit the data.
7. To transmit additional bits, return to Step 5.

Transmitting Data using the Interrupt-Driven Method

The UART Transmitter interrupt indicates the availability of the Transmit Data register to accept new data for transmission. Follow these steps to configure the UART for interrupt-driven data transmission:

1. Write to the UART Baud Rate High and Low Byte registers to set the desired baud rate.
2. Enable the UART pin functions by configuring the associated GPIO Port pins for alternate function operation.
3. Execute a DI instruction to disable interrupts.
4. Write to the Interrupt control registers to enable the UART Transmitter interrupt and set the desired priority.
5. Write to the UART Control 1 register to enable Multiprocessor (9-bit) mode functions, if desired.
6. Write to the UART Control 0 register to:
   – Set the transmit enable bit (`TEN`) to enable the UART for data transmission
   – Enable parity, if desired, and select either even or odd parity.
   – Set or clear the `CTSE` bit to enable or disable control from the receiver via the $\overline{\text{CTS}}$ pin.
7. Execute an EI instruction to enable interrupts.

The UART is now configured for interrupt-driven data transmission. When the UART Transmit interrupt is detected, the associated interrupt service routine (ISR) should perform the following:

8. Write the data byte to the UART Transmit Data register. The transmitter will automatically transfer the data to the Transmit Shift register and transmit the data.
9. Clear the UART Transmit interrupt bit in the applicable Interrupt Request register.
10. Execute the IRET instruction to return from the interrupt-service routine and wait for the Transmit Data register to again become empty.

UART*x* Receive Data Register

Data bytes received through the RXD*x* pin are stored in the UART*x* Receive Data register (Table 51). The Read-only UART*x* Receive Data register shares a Register File address with the Write-only UART*x* Transmit Data register.

Table 51. UART*x* Receive Data Register (U*x*RXD)

BITS	7	6	5	4	3	2	1	0
FIELD	RXD							
RESET	X	X	X	X	X	X	X	X
R/W	R	R	R	R	R	R	R	R
ADDR	F40H and F48H							

RXD—Receive Data
UART receiver data byte from the RXD*x* pin

UART*x* Status 0 and Status 1 Registers

The UART*x* Status 0 and Status 1 registers (Table 52 and 53) identify the current UART operating configuration and status.

Table 52. UART*x* Status 0 Register (U*x*STAT0)

BITS	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
ADDR	F41H and F49H							

RDA—Receive Data Available
This bit indicates that the UART Receive Data register has received data. Reading the UART Receive Data register clears this bit.
0 = The UART Receive Data register is empty.
1 = There is a byte in the UART Receive Data register.

PE—Parity Error
This bit indicates that a parity error has occurred. Reading the UART Receive Data register clears this bit.

Infrared Encoder/Decoder

Overview

The Z8F640x family products contain two fully-functional, high-performance UART to Infrared Encoder/Decoders (Endecs). Each Infrared Endec is integrated with an on-chip UART to allow easy communication between the Z8F640x family device and IrDA Physical Layer Specification Version 1.3-compliant infrared transceivers. Infrared communication provides secure, reliable, low-cost, point-to-point communication between PCs, PDAs, cell phones, printers and other infrared enabled devices.

Architecture

Figure 71 illustrates the architecture of the Infrared Endec.

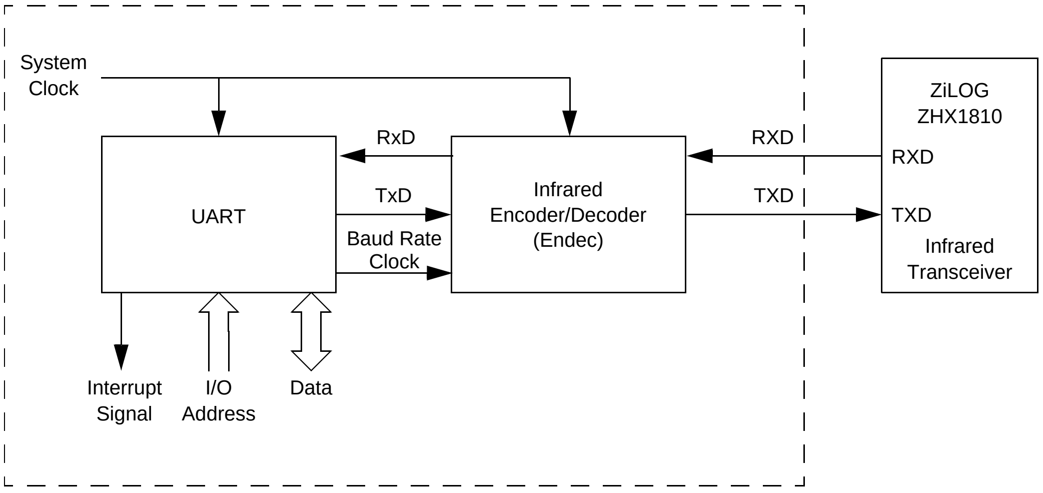


Figure 71. Infrared Data Communication System Block Diagram

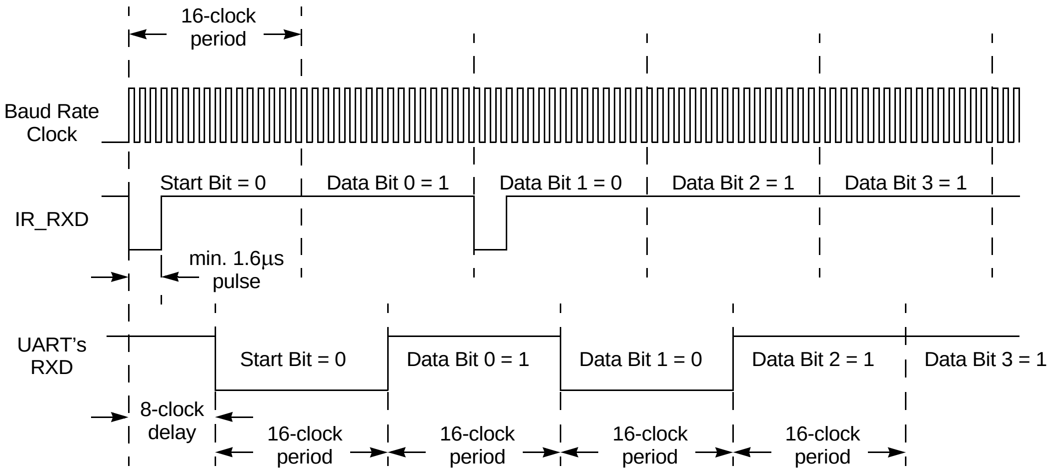


Figure 73. Infrared Data Reception

Jitter

Because of the inherent sampling of the received IR_RXD signal by the bit rate clock, some jitter can be expected on the first bit in any sequence of data. All subsequent bits in the received data stream are a fixed 16-clock periods wide.

Infrared Encoder/Decoder Control Register Definitions

All Infrared Endec configuration and status information is set by the UART control registers as defined beginning on page 86.

Caution: To prevent spurious signals during IrDA data transmission, set the `IREN` bit in the UART*x* Control 1 register to 1 to enable the Infrared Encoder/Decoder *before* enabling the GPIO Port alternate function for the corresponding pin.

The Master and Slave are each capable of exchanging a byte of data during a sequence of eight clock cycles. 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 ($\overline{\text{SS}}$) input signal is used to select a Slave SPI device. $\overline{\text{SS}}$ must be Low prior to all data communication to and from the Slave device. $\overline{\text{SS}}$ must stay Low for the full duration of each character transferred. The $\overline{\text{SS}}$ signal may stay Low during the transfer of multiple characters or may deassert between each character.

When the SPI on the Z8F640x family device is configured as the only Master in an SPI system, the $\overline{\text{SS}}$ pin can be set as either an input or an output. For communication between the Z8F640x family device SPI Master and external Slave devices, the $\overline{\text{SS}}$ signal, as an output, can assert the $\overline{\text{SS}}$ 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 on the Z8F640x family device is configured as one Master in a multi-master SPI system, the $\overline{\text{SS}}$ pin on the should be set as an input. The $\overline{\text{SS}}$ input signal on the Master must be High. If the $\overline{\text{SS}}$ signal goes Low (indicating another Master is driving the SPI bus), a Mode Fault 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 59 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 clock edge (SCK signal), in order for the Slave to latch the data.

Table 59. 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

limit to the amount of data transferred in one operation. When transmitting data or acknowledging read data from the slave, the SDA signal changes in the middle of the low period of SCL and is sampled in the middle of the high period of SCL.

I^{2}C Interrupts

the I^{2}C Controller contains three sources of interrupts—Transmit, Receive and Not Acknowledge (NAK) interrupts. NAK interrupts occur when a Not Acknowledge is received from the slave or sent by the I^{2}C Controller and the Start or Stop bit is set. This source sets bit 0 and can only be cleared by setting the Start or Stop bit. When this interrupt occurs, the I^{2}C Controller waits until it is cleared before performing any action. In an interrupt service routine, this interrupt must be the first thing polled. Receive interrupts occur when a byte of data has been received by the I^{2}C master. This interrupt is cleared by reading from the I^{2}C Data register. If no action is taken, the I^{2}C Controller waits until this interrupt is cleared before performing any other action.

For Transmit interrupts to occur, the `TXI` bit must be 1 in the I^{2}C Control register. Transmit interrupts occur under the following conditions when the transmit data register is empty:

- The I^{2}C Controller is idle (not performing an operation).
- The `START` bit is set and there is no valid data in the I^{2}C Shift or I^{2}C Data register to shift out.
- The first bit of the byte of an address is shifting out and the `RD` bit of the I^{2}C Status register is deasserted.
- The first bit of a 10-bit address shifts out.
- The first bit of write data shifted out.

Note: Writing to the I^{2}C Data register always clears a Transmit interrupt.

Start and Stop Conditions

The master (I^{2}C) drives all Start and Stop signals and initiates all transactions. To start a transaction, the I^{2}C Controller generates a START condition by pulling the SDA signal low while SCL is high. Then a high-to-low transition occurs on the SDA signal while the clock is High. To complete a transaction, the I^{2}C Controller generates a Stop condition by creating a low-to-high transition of the SDA signal in the middle of the high period of the SCL signal.When the SCL signal is High, the master generates a Start bit by pulling a High SDA signal Low and generates a Stop bit by releasing the SDA signal. The Start and Stop signals are found in the I^{2}C Control register and must be written by software when the Z8F640x family device must begin or end a transaction.

Writing a Transaction with a 7-Bit Address

1. The I^{2}C Controller shifts the I^{2}C Shift register out onto SDA signal.

Flash Page Select Register

The Flash Page Select register is used to select one of the 128 available Flash memory pages to be erased in a Page Erase operation. Each Flash Page contains 512 bytes of Flash memory. During a Page Erase operation, all Flash memory having addresses with the most significant 7-bits given by `FPS[6:0]` are erased (all bytes written to `FFH`).

Table 87. Flash Page Select Register (FPS)

BITS	7	6	5	4	3	2	1	0
FIELD	Reserved	PAGE						
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
ADDR	FF9H							

Reserved
This bit is reserved and must be 0.

PAGE—Page Select
This 7-bit field identifies the Flash memory page for Page Erase operation.
Program Memory Address[15:9] = PAGE[6:0]

ister. When the Watchpoint event occurs, the Z8F640x family device enters Debug mode and the `DBGMODE` bit in the OCDCTL register becomes 1.

Runtime Counter

The On-Chip Debugger contains a 16-bit Runtime Counter. It counts system clock cycles between Breakpoints. The counter starts counting when the On-Chip Debugger leaves Debug mode and stops counting when it enters Debug mode again or when it reaches the maximum count of `FFFFH`.

On-Chip Debugger Commands

The host communicates to the On-Chip Debugger by sending OCD commands using the DBG interface. During normal operation of the Z8F640x family device, only a subset of the OCD commands are available. In Debug mode, all OCD commands become available unless the user code and control registers are protected by programming the Read Protect Option Bit (`RP`). The Read Protect Option Bit prevents the code in memory from being read out of the Z8F640x family device. When this option is enabled, several of the OCD commands are disabled. Table 93 contains a summary of the On-Chip Debugger commands. Each OCD command is described in further detail in the bulleted list following Table 93. Table 93 indicates those commands that operate when the Z8F640x family device is not in Debug mode (normal operation) and those commands that are disabled by programming the Read Protect Option Bit.

Table 93. On-Chip Debugger Commands

Debug Command	Command Byte	Enabled when NOT in Debug mode?	Disabled by Read Protect Option Bit
Read OCD Revision	00H	Yes	-
Reserved	01H	-	-
Read OCD Status Register	02H	Yes	-
Read Runtime Counter	03H	-	-
Write OCD Control Register	04H	Yes	Cannot clear `DBGMODE` bit
Read OCD Control Register	05H	Yes	-
Write Program Counter	06H	-	Disabled
Read Program Counter	07H	-	Disabled
Write Register	08H	-	Only writes of the Flash Memory Control registers are allowed. Additionally, only the Mass Erase command is allowed to be written to the Flash Control register.
Read Register	09H	-	Disabled

RPEN—Read Protect Option Bit Enabled
0 = The Read Protect Option Bit is disabled (1).
0 = The Read Protect Option Bit is enabled (0), disabling many OCD commands.

Reserved
These bits are always 0.

OCD Watchpoint Control Register

The OCD Watchpoint Control register is used to configure the debug Watchpoint.

Table 96. OCD Watchpoint Control/Address (WPTCTL)

BITS	7	6	5	4	3	2	1	0
FIELD	WPW	WPR	WPDM	Reserved	WPTADDR[11:8]			
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

WPW—Watchpoint Break on Write
This bit cannot be set if the Read Protect Option Bit is enabled.
0 = Watchpoint Break on Register File write is disabled.
1 = Watchpoint Break on Register File write is enabled.

WPR—Watchpoint Break on Read
This bit cannot be set if the Read Protect Option Bit is enabled.
0 = Watchpoint Break on Register File read is disabled.
1 = Watchpoint Break on Register File write is enabled.

WPDM—Watchpoint Data Match
If this bit is set, then the Watchpoint only generates a Debug Break if the data being read or written matches the specified Watchpoint data. Either the `WPR` and/or `WPW` bits must also be set for this bit to affect operation. This bit cannot be set if the Read Protect Option Bit is enabled.
0 = Watchpoint Break on read and/or write does not require a data match.
1 = Watchpoint Break on read and/or write requires a data match.

Reserved
This bit is reserved and must be 0.

RADDR[11:8]—Register address
These bits specify the upper 4 bits of the Register File address to match when generating a Watchpoint Debug Break. The full 12-bit Register File address is given by {WPTCTL3:0], WPTADDR[7:0]}.

General Purpose I/O Port Input Data Sample Timing

Figure 93 illustrates timing of the GPIO Port input sampling. The input value on a GPIO Port pin is sampled on the rising edge of the system clock. The Port value is then available to the eZ8 CPU on the second rising clock edge following the change of the Port value.

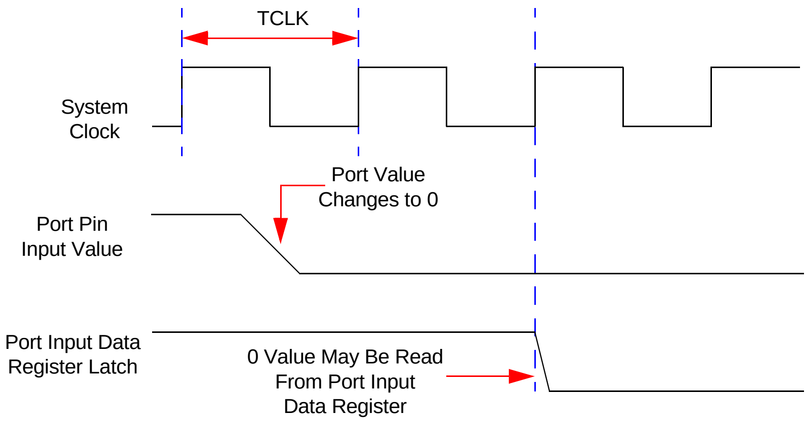

TCLK
System Clock
Port Value Changes to 0
Port Pin Input Value
Port Input Data Register Latch
0 Value May Be Read From Port Input Data Register

Figure 93. Port Input Sample Timing

Table 107. GPIO Port Input Timing

		Delay (ns)	
Parameter	**Abbreviation**	**Minimum**	**Maximum**
T_{S_PORT}	Port Input Transition to XIN Rise Setup Time (Not pictured)	5	–
T_{H_PORT}	XIN Rise to Port Input Transition Hold Time (Not pictured)	5	–
T_{SMR}	GPIO Port Pin Pulse Width to Insure Stop Mode Recovery (for GPIO Port Pins enabled as SMR sources)	1µs	

; 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 by users that prefer manual program coding or intend to implement their 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:

Table 113. 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:

Table 114. Assembly Language Syntax Example 2

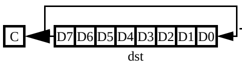
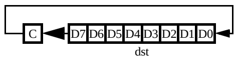
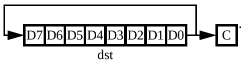
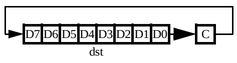
Assembly Language Code	ADD	43H,	R8	(ADD dst, src)
Object Code	04	E8	43	(OPC src, dst)

See 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 115

Table 126. eZ8 CPU Instruction Summary (Continued)

Assembly Mnemonic	Symbolic Operation	Address Mode dst	Address Mode src	Opcode(s) (Hex)	C	Z	S	V	D	H	Fetch Cycles	Instr. Cycles
POP dst	dst ← @SP SP ← SP + 1	R		50	-	-	-	-	-	-	2	2
		IR		51							2	3
POPX dst	dst ← @SP SP ← SP + 1	ER		D8	-	-	-	-	-	-	3	2
PUSH src	SP ← SP – 1 @SP ← src	R		70	-	-	-	-	-	-	2	2
		IR		71							2	3
PUSHX src	SP ← SP – 1 @SP ← src	ER		C8	-	-	-	-	-	-	3	2
RCF	C ← 0			CF	0	-	-	-	-	-	1	2
RET	PC ← @SP SP ← SP + 2			AF	-	-	-	-	-	-	1	4
RL dst		R		90	*	*	*	*	-	-	2	2
		IR		91							2	3
RLC dst		R		10	*	*	*	*	-	-	2	2
		IR		11							2	3
RR dst		R		E0	*	*	*	*	-	-	2	2
		IR		E1							2	3
RRC dst		R		C0	*	*	*	*	-	-	2	2
		IR		C1							2	3

Flags Notation: * = Value is a function of the result of the operation.
- = Unaffected
X = Undefined
0 = Reset to 0
1 = Set to 1

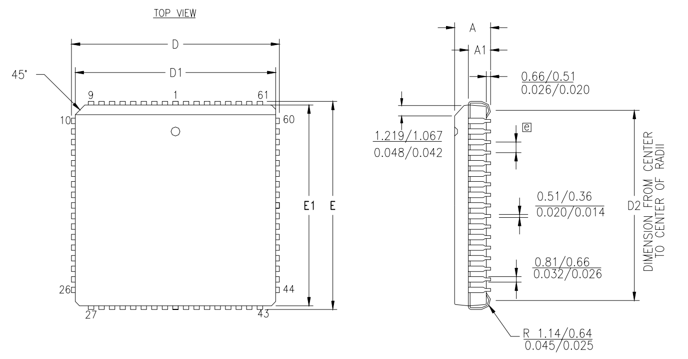
Lower Nibble (Hex)

Upper Nibble (Hex)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1.2 **BRK**	2.2 **SRP** IM	2.3 **ADD** r1,r2	2.4 **ADD** r1,Ir2	3.3 **ADD** R2,R1	3.4 **ADD** IR2,R1	3.3 **ADD** R1,IM	3.4 **ADD** IR1,IM	4.3 **ADDX** ER2,ER1	4.3 **ADDX** IM,ER1	2.3 **DJNZ** r1,X	2.2 **JR** cc,X	2.2 **LD** r1,IM	3.2 **JP** cc,DA	1.2 **INC** r1	1.2 **NOP**
1	2.2 **RLC** R1	2.3 **RLC** IR1	2.3 **ADC** r1,r2	2.4 **ADC** r1,Ir2	3.3 **ADC** R2,R1	3.4 **ADC** IR2,R1	3.3 **ADC** R1,IM	3.4 **ADC** IR1,IM	4.3 **ADCX** ER2,ER1	4.3 **ADCX** IM,ER1	↓	↓	↓	↓	↓	See 2nd Opcode Map
2	2.2 **INC** R1	2.3 **INC** IR1	2.3 **SUB** r1,r2	2.4 **SUB** r1,Ir2	3.3 **SUB** R2,R1	3.4 **SUB** IR2,R1	3.3 **SUB** R1,IM	3.4 **SUB** IR1,IM	4.3 **SUBX** ER2,ER1	4.3 **SUBX** IM,ER1	↓	↓	↓	↓	↓	
3	2.2 **DEC** R1	2.3 **DEC** IR1	2.3 **SBC** r1,r2	2.4 **SBC** r1,Ir2	3.3 **SBC** R2,R1	3.4 **SBC** IR2,R1	3.3 **SBC** R1,IM	3.4 **SBC** IR1,IM	4.3 **SBCX** ER2,ER1	4.3 **SBCX** IM,ER1	↓	↓	↓	↓	↓	
4	2.2 **DA** R1	2.3 **DA** IR1	2.3 **OR** r1,r2	2.4 **OR** r1,Ir2	3.3 **OR** R2,R1	3.4 **OR** IR2,R1	3.3 **OR** R1,IM	3.4 **OR** IR1,IM	4.3 **ORX** ER2,ER1	4.3 **ORX** IM,ER1	↓	↓	↓	↓	↓	
5	2.2 **POP** R1	2.3 **POP** IR1	2.3 **AND** r1,r2	2.4 **AND** r1,Ir2	3.3 **AND** R2,R1	3.4 **AND** IR2,R1	3.3 **AND** R1,IM	3.4 **AND** IR1,IM	4.3 **ANDX** ER2,ER1	4.3 **ANDX** IM,ER1	↓	↓	↓	↓	↓	1.2 **WDT**
6	2.2 **COM** R1	2.3 **COM** IR1	2.3 **TCM** r1,r2	2.4 **TCM** r1,Ir2	3.3 **TCM** R2,R1	3.4 **TCM** IR2,R1	3.3 **TCM** R1,IM	3.4 **TCM** IR1,IM	4.3 **TCMX** ER2,ER1	4.3 **TCMX** IM,ER1	↓	↓	↓	↓	↓	1.2 **STOP**
7	2.2 **PUSH** R2	2.3 **PUSH** IR2	2.3 **TM** r1,r2	2.4 **TM** r1,Ir2	3.3 **TM** R2,R1	3.4 **TM** IR2,R1	3.3 **TM** R1,IM	3.4 **TM** IR1,IM	4.3 **TMX** ER2,ER1	4.3 **TMX** IM,ER1	↓	↓	↓	↓	↓	1.2 **HALT**
8	2.5 **DECW** RR1	2.6 **DECW** IRR1	2.5 **LDE** r1,Irr2	2.9 **LDEI** Ir1,Irr2	3.2 **LDX** r1,ER2	3.3 **LDX** Ir1,ER2	3.4 **LDX** IRR2,R1	3.5 **LDX** IRR2,IR1	3.4 **LDX** r1,rr2,X	3.4 **LDX** rr1,r2,X	↓	↓	↓	↓	↓	1.2 **DI**
9	2.2 **RL** R1	2.3 **RL** IR1	2.5 **LDE** r2,Irr1	2.9 **LDEI** Ir2,Irr1	3.2 **LDX** r2,ER1	3.3 **LDX** Ir2,ER1	3.4 **LDX** R2,IRR1	3.5 **LDX** IR2,IRR1	3.3 **LEA** r1,r2,X	3.5 **LEA** rr1,rr2,X	↓	↓	↓	↓	↓	1.2 **EI**
A	2.5 **INCW** RR1	2.6 **INCW** IRR1	2.3 **CP** r1,r2	2.4 **CP** r1,Ir2	3.3 **CP** R2,R1	3.4 **CP** IR2,R1	3.3 **CP** R1,IM	3.4 **CP** IR1,IM	4.3 **CPX** ER2,ER1	4.3 **CPX** IM,ER1	↓	↓	↓	↓	↓	1.4 **RET**
B	2.2 **CLR** R1	2.3 **CLR** IR1	2.3 **XOR** r1,r2	2.4 **XOR** r1,Ir2	3.3 **XOR** R2,R1	3.4 **XOR** IR2,R1	3.3 **XOR** R1,IM	3.4 **XOR** IR1,IM	4.3 **XORX** ER2,ER1	4.3 **XORX** IM,ER1	↓	↓	↓	↓	↓	1.5 **IRET**
C	2.2 **RRC** R1	2.3 **RRC** IR1	2.5 **LDC** r1,Irr2	2.9 **LDCI** Ir1,Irr2	2.3 **JP** IRR1	2.9 **LDC** Ir1,Irr2		3.3 **LD** r1,r2,X	3.2 **PUSHX** ER2		↓	↓	↓	↓	↓	1.2 **RCF**
D	2.2 **SRA** R1	2.3 **SRA** IR1	2.5 **LDC** r2,Irr1	2.9 **LDCI** Ir2,Irr1	2.6 **CALL** IRR1	2.2 **BSWAP** R1	3.3 **CALL** DA	3.4 **LD** r2,r1,X	3.2 **POPX** ER1		↓	↓	↓	↓	↓	1.2 **SCF**
E	2.2 **RR** R1	2.3 **RR** IR1	2.2 **BIT** p,b,r1	2.3 **LD** r1,Ir2	3.2 **LD** R2,R1	3.3 **LD** IR2,R1	3.2 **LD** R1,IM	3.3 **LD** IR1,IM	4.2 **LDX** ER2,ER1	4.2 **LDX** IM,ER1	↓	↓	↓	↓	↓	1.2 **CCF**
F	2.2 **SWAP** R1	2.3 **SWAP** IR1	2.6 **TRAP** Vector	2.3 **LD** Ir1,r2	2.8 **MULT** RR1	3.3 **LD** R2,IR1	3.3 **BTJ** p,b,r1,X	3.4 **BTJ** p,b,Ir1,X			↓	↓	↓	↓	↓	

Figure 101. First Opcode Map

Figure 107 illustrates the 68-pin PLCC (plastic lead chip carrier) package available for the Z8F1602, Z8F2402, Z8F3202, Z8F4802, and Z8F6402 devices.



SYMBOL	MILLIMETER		INCH	
	MIN	MAX	MIN	MAX
A	4.32	4.57	.170	.180
A1	2.43	2.92	.095	.115
D/E	25.02	25.40	.985	1.000
D1/E1	24.13	24.33	.950	.958
D2	22.86	23.62	.900	.930
e	1.27 BSC		.050 BSC	

NOTE:
1. CONTROLLING DIMENSIONS : INCH.
2. LEADS ARE COPLANAR WITHIN 0.004 IN. RANGE.
3. DIMENSION : $\frac{\text{MM}}{\text{INCH.}}$

Figure 107. 68-Lead Plastic Lead Chip Carrier Package (PLCC)