



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 ² C, 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	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	44-LQFP
Supplier Device Package	44-LQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z8f4801an020ec



Table 101. Absolute Maximum Ratings	167
Table 102. DC Characteristics	169
Table 103. AC Characteristics	172
Table 104. Power-On Reset and Voltage Brown-Out Electrical Characteristics and Timing	173
Table 105. Flash Memory Electrical Characteristics and Timing . . .	173
Table 106. Watch-Dog Timer Electrical Characteristics and Timing	174
Table 107. Analog-to-Digital Converter Electrical Characteristics and Timing	174
Table 108. GPIO Port Input Timing	176
Table 109. GPIO Port Output Timing	177
Table 110. On-Chip Debugger Timing	178
Table 111. SPI Master Mode Timing	179
Table 112. SPI Slave Mode Timing	180
Table 113. I2C Timing	181
Table 114. Assembly Language Syntax Example 1	183
Table 115. Assembly Language Syntax Example 2	183
Table 116. Notational Shorthand	184
Table 117. Additional Symbols	185
Table 118. Condition Codes	186
Table 119. Arithmetic Instructions	187
Table 120. Bit Manipulation Instructions	188
Table 121. Block Transfer Instructions	188
Table 122. CPU Control Instructions	189
Table 123. Load Instructions	189
Table 124. Logical Instructions	190
Table 125. Program Control Instructions	190
Table 126. Rotate and Shift Instructions	191
Table 127. eZ8 CPU Instruction Summary	191
Table 128. Opcode Map Abbreviations	203
Table 129. Ordering Information	211



- Software stack allows much greater depth in subroutine calls and interrupts than hardware stacks
- Compatible with existing Z8 code
- Expanded internal Register File allows access of up to 4KB
- New instructions improve execution efficiency for code developed using higher-level programming languages, including C
- Pipelined instruction fetch and execution
- New instructions for improved performance including BIT, BSWAP, BTJ, CPC, LDC, LDCI, LEA, MULT, and SRL
- New instructions support 12-bit linear addressing of the Register File
- Up to 10 MIPS operation
- C-Compiler friendly
- 2-9 clock cycles per instruction

For more information regarding the eZ8 CPU, refer to the *eZ8 CPU User Manual* available for download at www.zilog.com.

General Purpose I/O

The Z8 Encore!® features seven 8-bit ports (Ports A-G) and one 4-bit port (Port H) for general purpose I/O (GPIO). Each pin is individually programmable.

Flash Controller

The Flash Controller programs and erases the Flash memory.

10-Bit Analog-to-Digital Converter

The Analog-to-Digital Converter (ADC) converts an analog input signal to a 10-bit binary number. The ADC accepts inputs from up to 12 different analog input sources.

UARTs

Each UART is full-duplex and capable of handling asynchronous data transfers. The UARTs support 8- and 9-bit data modes and selectable parity.

I²C

The inter-integrated circuit (I²C®) controller makes the Z8 Encore!® compatible with the I²C protocol. The I²C controller consists of two bidirectional bus lines, a serial data (SDA) line and a serial clock (SCL) line.



Address Space

Overview

The eZ8 CPU can access three distinct address spaces:

- The Register File contains addresses for the general-purpose registers and the eZ8 CPU, peripheral, and general-purpose I/O port control registers.
- The Program Memory contains addresses for all memory locations having executable code and/or data.
- The Data Memory contains addresses for all memory locations that hold data only.

These three address spaces are covered briefly in the following subsections. For more detailed information regarding the eZ8 CPU and its address space, refer to the *eZ8 CPU User Manual* available for download at www.zilog.com.

Register File

The Register File address space in the Z8 Encore!® is 4KB (4096 bytes). The Register File is composed of two sections—control registers and general-purpose registers. When instructions are executed, registers are read from when defined as sources and written to when defined as destinations. The architecture of the eZ8 CPU allows all general-purpose registers to function as accumulators, address pointers, index registers, stack areas, or scratch pad memory.

The upper 256 bytes of the 4KB Register File address space are reserved for control of the eZ8 CPU, the on-chip peripherals, and the I/O ports. These registers are located at addresses from F00H to FFFH. Some of the addresses within the 256-byte control register section are reserved (unavailable). Reading from an reserved Register File addresses returns an undefined value. Writing to reserved Register File addresses is not recommended and can produce unpredictable results.

The on-chip RAM always begins at address 000H in the Register File address space. The Z8F640x family products contain 2KB to 4KB of on-chip RAM depending upon the device. Reading from Register File addresses outside the available RAM addresses (and not within in the control register address space) returns an undefined value. Writing to these Register File addresses produces no effect. Refer to the **Part Selection Guide** section of the **Introduction** chapter to determine the amount of RAM available for the specific Z8F640x family device.



Register File Address Map

Table 6 provides the address map for the Register File of the Z8F640x family of products. Not all devices and package styles in the Z8F640x family support Timer 3 and all of the GPIO Ports. Consider registers for unimplemented peripherals as Reserved.

Table 6. Register File Address Map

Address (Hex)	Register Description	Mnemonic	Reset (Hex)	Page #
General Purpose RAM				
000-EFF	General-Purpose Register File RAM	—	XX	
Timer 0				
F00	Timer 0 High Byte	T0H	00	66
F01	Timer 0 Low Byte	T0L	01	66
F02	Timer 0 Reload High Byte	T0RH	FF	67
F03	Timer 0 Reload Low Byte	T0RL	FF	67
F04	Timer 0 PWM High Byte	T0PWMH	00	69
F05	Timer 0 PWM Low Byte	T0PWML	00	69
F06	Reserved	—	XX	
F07	Timer 0 Control	T0CTL	00	70
Timer 1				
F08	Timer 1 High Byte	T1H	00	66
F09	Timer 1 Low Byte	T1L	01	66
F0A	Timer 1 Reload High Byte	T1RH	FF	67
F0B	Timer 1 Reload Low Byte	T1RL	FF	67
F0C	Timer 1 PWM High Byte	T1PWMH	00	69
F0D	Timer 1 PWM Low Byte	T1PWML	00	69
F0E	Reserved	—	XX	
F0F	Timer 1 Control	T1CTL	00	70
Timer 2				
F10	Timer 2 High Byte	T2H	00	66
F11	Timer 2 Low Byte	T2L	01	66
F12	Timer 2 Reload High Byte	T2RH	FF	67
F13	Timer 2 Reload Low Byte	T2RL	FF	67
F14	Timer 2 PWM High Byte	T2PWMH	00	69
F15	Timer 2 PWM Low Byte	T2PWML	00	69
F16	Reserved	—	XX	
F17	Timer 2 Control	T2CTL	00	70

XX=Undefined

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.

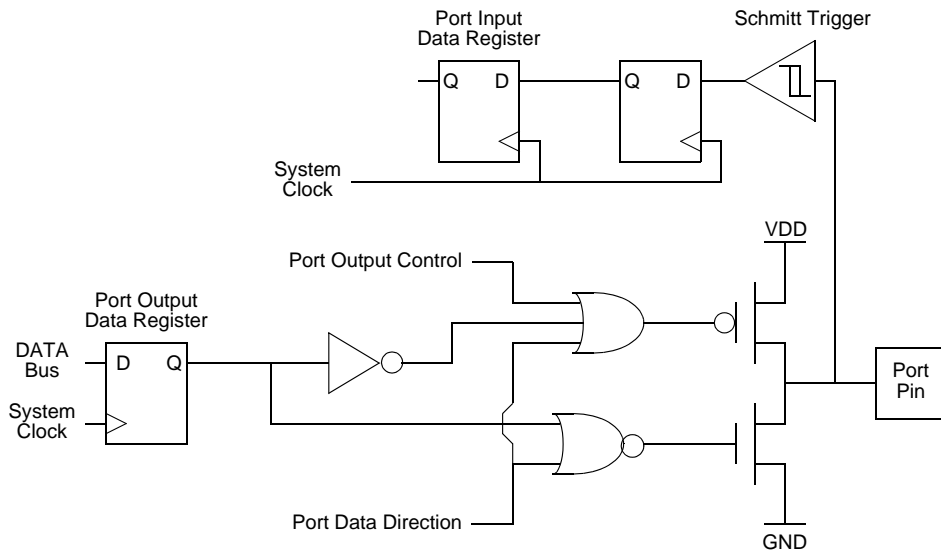


Figure 64. GPIO Port Pin Block Diagram

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

Interrupt Control Register Definitions

For all interrupts other than the Watch-Dog 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 23) 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 23. 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	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	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.



mode. Refer to the **Reset and Stop Mode Recovery** chapter for more information on STOP Mode Recovery.

If interrupts are enabled, following completion of the Stop Mode Recovery the eZ8 CPU responds to the interrupt request by fetching the Watch-Dog Timer interrupt vector and executing code from the vector address.

WDT Reset in Normal Operation

If configured to generate a Reset when a time-out occurs, the Watch-Dog Timer forces the Z8F640x family device into the Short Reset state. The WDT status bit in the Watch-Dog Timer Control register is set to 1. Refer to the **Reset and Stop Mode Recovery** chapter for more information on Short Reset.

WDT Reset in Stop Mode

If configured to generate a Reset when a time-out occurs and the Z8F640x family device is in STOP mode, the Watch-Dog Timer initiates a Stop Mode Recovery. Both the WDT status bit and the STOP bit in the Watch-Dog Timer Control register are set to 1 following WDT time-out in STOP mode. Refer to the **Reset and Stop Mode Recovery** chapter for more information.

Watch-Dog Timer Reload Unlock Sequence

Writing the unlock sequence to the Watch-Dog Timer Control register (WDTCTL) unlocks the three Watch-Dog Timer Reload Byte registers (WDTU, WDTH, and WDTL) to allow changes to the time-out period. These write operations to the WDTCTL register address produce no effect on the bits in the WDTCTL register. The locking mechanism prevents spurious writes to the Reload registers. The follow sequence is required to unlock the Watch-Dog Timer Reload Byte registers (WDTU, WDTH, and WDTL) for write access.

1. Write 55H to the Watch-Dog Timer Control register (WDTCTL)
2. Write AAH to the Watch-Dog Timer Control register (WDTCTL)
3. Write the Watch-Dog Timer Reload Upper Byte register (WDTU)
4. Write the Watch-Dog Timer Reload High Byte register (WDTH)
5. Write the Watch-Dog Timer Reload Low Byte register (WDTL)

All three Watch-Dog Timer Reload registers must be written in the order just listed. There must be no other register writes between each of these operations. If a register write occurs, the lock state machine resets and no further writes can occur, unless the sequence is restarted. The value in the Watch-Dog Timer Reload registers is loaded into the counter when the Watch-Dog Timer is first enabled and every time a WDT instruction is executed.

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 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.

UARTx 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. UARTx Receive Data Register (UxRXD)

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

UARTx 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. UARTx Status 0 Register (UxSTAT0)

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.

Error Detection

The SPI contains error detection logic to support SPI communication protocols and recognize when communication errors have occurred. The SPI Status register indicates when a data transmission error has been detected.

Overrun (Write Collision)

An overrun error (write collision) indicates a write to the SPI Data register was attempted while a data transfer is in progress. An overrun sets the OVR bit in the SPI Status register to 1. Writing a 1 to OVR clears this error flag.

Mode Fault (Multi-Master Collision)

A mode fault indicates when more than one Master is trying to communicate at the same time (a multi-master collision). The mode fault is detected when the enabled Master's \overline{SS} pin is asserted. A mode fault sets the COL bit in the SPI Status register to 1. Writing a 1 to COL clears this error flag.

SPI Interrupts

When SPI interrupts are enabled, the SPI generates an interrupt after data transmission. The SPI in Master mode generates an interrupt after a character has been sent. A character can be defined to be 1 through 8 bits by the NUMBITS field in the SPI Mode register. The SPI in Slave mode generates an interrupt when the \overline{SS} signal deasserts to indicate completion of the data transfer. Writing a 1 to the IRQ bit in the SPI Status Register clears the pending interrupt request. If the SPI is disabled, an SPI interrupt can be generated by a Baud Rate Generator time-out.

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 reload value must be greater than or equal to 0002H for SPI operation (maximum baud rate is system clock frequency divided by 4). The SPI baud rate is calculated using the following equation:

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

When the SPI is disabled, the Baud Rate Generator can function as a basic 16-bit timer with interrupt on time-out. To configure the Baud Rate Generator as a timer with interrupt on time-out, complete the following procedure:

SPI Mode Register

The SPI Mode register configures the character bit width and the direction and value of the \overline{SS} pin.

Table 63. SPI Mode Register (SPIMODE)

BITS	7	6	5	4	3	2	1	0
FIELD	Reserved			NUMBITS[2:0]			SSIO	SSV
RESET	0			0	0	0	0	0
R/W	R			R/W	R/W	R/W	R/W	R/W
ADDR	F63H							

Reserved

These bits are reserved and must be 0.

NUMBITS[2:0]—Number of Data Bits Per Character to Transfer

This field contains the number of bits to shift for each character transfer. Refer to the SPI Data Register description for information on valid bit positions when the character length is less than 8-bits.

000 = 8 bits

001 = 1 bit

010 = 2 bits

011 = 3 bits

100 = 4 bits

101 = 5 bits

110 = 6 bits

111 = 7 bits.

SSIO—Slave Select I/O

0 = \overline{SS} pin configured as an input.

1 = \overline{SS} pin configured as an output (Master mode only).

SSV—Slave Select Value

If SSIO = 1 and SPI configured as a Master:

0 = \overline{SS} pin driven Low (0).

1 = \overline{SS} pin driven High (1).

This bit has no effect if SSIO = 0 or SPI configured as a Slave.

7. The I²C Controller loads the I²C Shift register with the contents of the I²C Data register.
8. After one bit of address is shifted out by the SDA signal, the Transmit interrupt is asserted.
9. Software responds by writing the second byte of address into the contents of the I²C Data register.
10. The I²C Controller shifts the rest of the first byte of address and write bit out by the SDA signal.
11. 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.
12. The I²C Controller loads the contents of the I²C Shift register with the contents of the I²C Data register.
13. The I²C Controller shifts the data out by the SDA signal. After the first bit has been sent, the Transmit interrupt is asserted.
14. Software responds by writing the data to be written out to the I²C Control register.
15. The I²C Controller shifts out the rest of the second byte of slave address by the SDA signal.
16. 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.
17. The I²C Controller shifts the data out by the SDA signal. After the first bit is sent, the Transmit interrupt is asserted.
18. Software responds by asserting the STOP bit of the I²C Control register.
19. The I²C Controller completes transmission of the data on the SDA signal.
20. The I²C Controller sends the STOP condition to the I²C bus.

Reading a Transaction with a 7-Bit Address

Figure 81 illustrates the data transfer format for a receive operation on a 7-bit addressed slave. The 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	R=1	A	Data	A	Data	\bar{A}	P
---	---------------	-----	---	------	---	------	-----------	---

Figure 81. Receive Data Transfer Format for a 7-Bit Addressed Slave

The data transfer format for a receive operation on a 7-bit addressed slave is as follows:



4. The I²C Controller loads the I²C Shift register with the contents of the I²C Data register.
5. After the first bit has been shifted out, a Transmit interrupt is asserted.
6. Software responds by writing eight bits of address to the I²C Data register.
7. The I²C Controller completes shifting of the two address bits and a 0 (write).
8. The I²C slave sends an acknowledge by pulling the SDA signal Low during the next high period of SCL.
9. The I²C Controller loads the I²C Shift register with the contents of the I²C Data register.
10. The I²C Controller shifts out the next eight bits of address. After the first bits are shifted, the I²C Controller generates a Transmit interrupt.
11. Software responds by setting the START bit of the I²C Control register to generate a repeated START.
12. Software responds by writing 11110B followed by the 2-bit slave address and a 1 (read).
13. Software responds by setting the NAK bit of the I²C Control register, so that a Not Acknowledge is sent after the first byte of data has been read. If you want to read only one byte, software responds by setting the NAK bit of the I²C Control register.
14. After the I²C Controller shifts out the address bits mentioned in step 9, the I²C slave sends an acknowledge by pulling the SDA signal Low during the next high period of SCL.
15. The I²C Controller sends the repeated START condition.
16. The I²C Controller loads the I²C Shift register with the contents of the I²C Data register.
17. The I²C Controller sends 11110B followed by the 2-bit slave read and a 1 (read).
18. The I²C slave sends an acknowledge by pulling the SDA signal Low during the next high period of SCL.
19. The I²C slave sends a byte of data.
20. A Receive interrupt is generated.
21. Software responds by reading the I²C Data register.
22. Software responds by setting the STOP bit of the I²C Control register.
23. A NAK condition is sent to the I²C slave.
24. A STOP condition is sent to the I²C slave.

Configuring DMA0 and DMA1 for Data Transfer

Follow these steps to configure and enable DMA0 or DMA1:

1. Write to the DMAx I/O Address register to set the Register File address identifying the on-chip peripheral control register. The upper nibble of the 12-bit address for on-chip peripheral control registers is always FH. The full address is {FH, DMAx_IO[7:0]}
2. Determine the 12-bit Start and End Register File addresses. The 12-bit Start Address is given by {DMAx_H[3:0], DMA_START[7:0]}. The 12-bit End Address is given by {DMAx_H[7:4], DMA_END[7:0]}.
3. Write the Start and End Register File address high nibbles to the DMAx End/Start Address High Nibble register.
4. Write the lower byte of the Start Address to the DMAx Start/Current Address register.
5. Write the lower byte of the End Address to the DMAx End Address register.
6. Write to the DMAx Control register to complete the following:
 - Select loop or single-pass mode operation
 - Select the data transfer direction (either from the Register File RAM to the on-chip peripheral control register; or from the on-chip peripheral control register to the Register File RAM)
 - Enable the DMAx interrupt request, if desired
 - Select Word or Byte mode
 - Select the DMAx request trigger
 - Enable the DMAx channel

DMA_ADC Operation

DMA_ADC transfers data from the ADC to the Register File. The sequence of operations in a DMA_ADC data transfer is:

1. ADC completes conversion on the current ADC input channel and signals the DMA controller that two-bytes of ADC data are ready for transfer.
2. DMA_ADC requests control of the system bus (address and data) from the eZ8 CPU.
3. After the eZ8 CPU acknowledges the bus request, DMA_ADC transfers the two-byte ADC output value to the Register File and then returns system bus control back to the eZ8 CPU.
4. If the current ADC Analog Input is the highest numbered input to be converted:
 - DMA_ADC resets the ADC Analog Input number to 0 and initiates data conversion on ADC Analog Input 0.
 - If configured to generate an interrupt, DMA_ADC sends an interrupt request to the Interrupt Controller



zero). If the Z8F640x family device is not in Debug mode or if the Read Protect Option Bit is enabled, this command returns FFH for all the data values.

```
DBG <-- 09H
DBG <-- {4'h0, Register Address[11:8]}
DBG <-- Register Address[7:0]
DBG <-- Size[7:0]
DBG --> 1-256 data bytes
```

- **Write Program Memory (0AH)**—The Write Program Memory command writes data to Program Memory. This command is equivalent to the LDC and LDCI instructions. Data can be written 1-65536 bytes at a time (65536 bytes can be written by setting size to zero). The on-chip Flash Controller must be written to and unlocked for the programming operation to occur. If the Flash Controller is not unlocked, the data is discarded. If the Z8F640x family device is not in Debug mode or if the Read Protect Option Bit is enabled, the data is discarded.

```
DBG <-- 0AH
DBG <-- Program Memory Address[15:8]
DBG <-- Program Memory Address[7:0]
DBG <-- Size[15:8]
DBG <-- Size[7:0]
DBG <-- 1-65536 data bytes
```

- **Read Program Memory (0BH)**—The Read Program Memory command reads data from Program Memory. This command is equivalent to the LDC and LDCI instructions. Data can be read 1-65536 bytes at a time (65536 bytes can be read by setting size to zero). If the Z8F640x family device is not in Debug mode or if the Read Protect Option Bit is enabled, this command returns FFH for the data.

```
DBG <-- 0BH
DBG <-- Program Memory Address[15:8]
DBG <-- Program Memory Address[7:0]
DBG <-- Size[15:8]
DBG <-- Size[7:0]
DBG --> 1-65536 data bytes
```

- **Write Data Memory (0CH)**—The Write Data Memory command writes data to Data Memory. This command is equivalent to the LDE and LDEI instructions. Data can be written 1-65536 bytes at a time (65536 bytes can be written by setting size to zero). If the Z8F640x family device is not in Debug mode or if the Read Protect Option Bit is enabled, the data is discarded.

```
DBG <-- 0CH
DBG <-- Data Memory Address[15:8]
DBG <-- Data Memory Address[7:0]
DBG <-- Size[15:8]
DBG <-- Size[7:0]
DBG <-- 1-65536 data bytes
```



BRKEN—Breakpoint Enable

This bit controls the behavior of the BRK instruction (opcode 00H). By default, Breakpoints are disabled and the BRK instruction behaves like a NOP. If this bit is set to 1, when a BRK instruction is decoded, the DBGMODE bit of the OCDCTL register is automatically set to one.

0 = Breakpoints are disabled.

1 = Breakpoints are enabled.

DBGACK—Debug Acknowledge

This bit enables the debug acknowledge feature. If this bit is set to 1, then the OCD sends an Debug Acknowledge character (FFH) to the host when a Breakpoint or Watchpoint occurs.

0 = Debug Acknowledge is disabled.

1 = Debug Acknowledge is enabled.

Reserved

These bits are reserved and must be 0.

RST—Reset

Setting this bit to 1 resets the Z8F640x family device. The device goes through a normal Power-On Reset sequence with the exception that the On-Chip Debugger is not reset. This bit is automatically cleared to 0 when the reset finishes.

0 = No effect.

1 = Reset Z8F640x family device.

OCD Status Register

The OCD Status register reports status information about the current state of the debugger and the Z8F640x family device.

Table 95. OCD Status Register (OCDSTAT)

BITS	7	6	5	4	3	2	1	0
FIELD	DBG	HALT	RPEN	Reserved				
RESET	0	0	0	0	0	0	0	0
R/W	R	R	R	R	R	R	R	R

DBG—Debug Status

0 = The Z8F640x family device is operating in normal mode.

1 = The Z8F640x family device is in Debug mode.

HALT—Halt Mode

0 = The Z8F640x family device is not in Halt mode.

1 = The Z8F640x family device is in Halt mode.



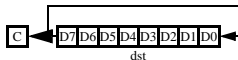

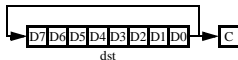
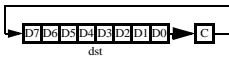
DC Characteristics

Table 101 lists the DC characteristics of the Z8F640x family devices. All voltages are referenced to V_{SS} , the primary system ground.

Table 101. DC Characteristics

Symbol	Parameter	$T_A = -40^{\circ}\text{C to } 105^{\circ}\text{C}$			Units	Conditions
		Minimum	Typical	Maximum		
V_{DD}	Supply Voltage	3.0	–	3.6	V	
V_{IL1}	Low Level Input Voltage	-0.3	–	$0.3 \cdot V_{DD}$	V	For all input pins except $\overline{\text{RESET}}$, DBG, and XIN.
V_{IL2}	Low Level Input Voltage	-0.3	–	$0.2 \cdot V_{DD}$	V	For $\overline{\text{RESET}}$, DBG, and XIN.
V_{IH1}	High Level Input Voltage	$0.7 \cdot V_{DD}$	–	5.5	V	Port A, C, D, E, F, and G pins.
V_{IH2}	High Level Input Voltage	$0.7 \cdot V_{DD}$	–	$V_{DD} + 0.3$	V	Port B and H pins.
V_{IH3}	High Level Input Voltage	$0.8 \cdot V_{DD}$	–	$V_{DD} + 0.3$	V	$\overline{\text{RESET}}$, DBG, and XIN pins.
V_{OL1}	Low Level Output Voltage	–	–	0.4	V	$V_{DD} = 3.0\text{V}$; $I_{OL} = 2\text{mA}$ High Output Drive disabled.
V_{OH1}	High Level Output Voltage	2.4	–	–	V	$V_{DD} = 3.0\text{V}$; $I_{OH} = -2\text{mA}$ High Output Drive disabled.
V_{OL2}	Low Level Output Voltage	–	–	0.6	V	$V_{DD} = 3.3\text{V}$; $I_{OL} = 20\text{mA}$ High Output Drive enabled. $T_A = -40^{\circ}\text{C to } +70^{\circ}\text{C}$
V_{OL3}	Low Level Output Voltage	–	–	0.6	V	$V_{DD} = 3.3\text{V}$; $I_{OL} = 15\text{mA}$ High Output Drive enabled. $T_A = 70^{\circ}\text{C to } +105^{\circ}\text{C}$
V_{OH2}	High Level Output Voltage	2.4	–	–	V	$V_{DD} = 3.3\text{V}$; $I_{OH} = -20\text{mA}$ High Output Drive enabled. $T_A = -40^{\circ}\text{C to } +70^{\circ}\text{C}$
V_{OH3}	High Level Output Voltage	2.4	–	–	V	$V_{DD} = 3.3\text{V}$; $I_{OH} = -15\text{mA}$ High Output Drive enabled. $T_A = 70^{\circ}\text{C to } +105^{\circ}\text{C}$
I_{IL}	Input Leakage Current	-5	–	+5	μA	$V_{DD} = 3.6\text{V}$; $V_{IN} = V_{DD}$ or V_{SS} ¹
I_{TL}	Tri-State Leakage Current	-5	–	+5	μA	$V_{DD} = 3.6\text{V}$
C_{PAD}	GPIO Port Pad Capacitance	–	8.0 ²	–	pF	
C_{XIN}	XIN Pad Capacitance	–	8.0 ²	–	pF	
C_{XOUT}	XOUT Pad Capacitance	–	9.5 ²	–	pF	

Table 126. 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		
POP dst	dst ← @SP	R		50	-	-	-	-	-	-	2	2
	SP ← SP + 1	IR		51							2	3
POPX dst	dst ← @SP	ER		D8	-	-	-	-	-	-	3	2
	SP ← SP + 1											
PUSH src	SP ← SP − 1	R		70	-	-	-	-	-	-	2	2
	@SP ← src	IR		71							2	3
PUSHX src	SP ← SP − 1	ER		C8	-	-	-	-	-	-	3	2
	@SP ← src											
RCF	C ← 0			CF	0	-	-	-	-	-	1	2
RET	PC ← @SP			AF	-	-	-	-	-	-	1	4
	SP ← SP + 2											
RL dst		R		90	*	*	*	*	-	-	2	2
		IR		91								2
RLC dst		R		10	*	*	*	*	-	-	2	2
		IR		11								2
RR dst		R		E0	*	*	*	*	-	-	2	2
		IR		E1								2
RRC dst		R		C0	*	*	*	*	-	-	2	2
		IR		C1								2
Flags Notation:												
* = Value is a function of the result of the operation.						0 = Reset to 0						
- = Unaffected						1 = Set to 1						
X = Undefined												



- read watchpoint (21H) 161
- step instruction (10H) 160
- stuff instruction (11H) 160
- write data memory (0CH) 159
- write OCD control register (04H) 158
- write program counter (06H) 158
- write program memory (0AH) 159
- write register (08H) 158
- write watchpoint (20H) 161
- on-chip debugger 5
- on-chip debugger (OCD) 151
- on-chip debugger signals 14
- on-chip oscillator 165
- one-shot mode 70
- opcode map
 - abbreviations 203
 - cell description 202
 - first 204
 - second after 1FH 205
- OR 190
- ordering information 211
- ORX 190
- oscillator signals 14

P

- p 184
- packaging
 - LQFP
 - 44 lead 207
 - 64 lead 208
 - PDIP 206
 - PLCC
 - 44 lead 207
 - 68 lead 209
 - QFP 210
- part number description 214
- part selection guide 2
- PC 185
- PDIP 206
- peripheral AC and DC electrical characteristics 173
- PHASE=0 timing (SPI) 103
- PHASE=1 timing (SPI) 104
- pin characteristics 15

PLCC

- 44 lead 207
- 68-lead 209
- polarity 184
- POP 189
- pop using extended addressing 189
- POPX 189
- port availability, device 33
- port input timing (GPIO) 176
- port output timing, GPIO 177
- power supply signals 15
- power-down, automatic (ADC) 133
- power-on and voltage brown-out 173
- power-on reset (POR) 27
- problem description or suggestion 217
- product information 216
- program control instructions 190
- program counter 185
- program memory 18
- PUSH 189
- push using extended addressing 189
- PUSHX 189
- PWM mode 70
- PxADDR register 37
- PxCTL register 38

Q

- QFP 210

R

- R 184
- r 184
- RA, register address 184
- RCF 188, 189
- receive
 - 10-bit data format (I2C) 116
 - 7-bit data transfer format (I2C) 115
 - IrDA data 97
- receive interrupt 112
- receiving UART data-DMA controller 83
- receiving UART data-interrupt-driven method 82
- receiving UART data-pollled method 82