**Welcome to E-XFL.COM**

## What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

## Applications of "Embedded - Microcontrollers"

| Details | |
|---|---|
| Product Status | Obsolete |
| Core Processor | eZ8 |
| Core Size | 8-Bit |
| Speed | 20MHz |
| Connectivity | I²C, IrDA, UART/USART |
| Peripherals | Brown-out Detect/Reset, POR, PWM, WDT |
| Number of I/O | 11 |
| Program Memory Size | 8KB (8K x 8) |
| Program Memory Type | FLASH |
| EEPROM Size | - |
| RAM Size | 1K x 8 |
| Voltage - Supply (Vcc/Vdd) | 2.7V ~ 3.6V |
| Data Converters | A/D 2x10b |
| Oscillator Type | Internal |
| Operating Temperature | 0°C ~ 70°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 20-SSOP (0.209", 5.30mm Width) |
| Supplier Device Package | - |
| Purchase URL | https://www.e-xfl.com/product-detail/zilog/z8f0821hh020sc |

| Abbreviations/ Acronyms | Expansion |
|---|---|
| PDIP | Plastic Dual Inline Package |
| SOIC | Small Outline Integrated Circuit |
| SSOP | Small Shrink Outline Package |
| PC | Program Counter |
| IRQ | Interrupt Request |

# Address Space

The eZ8 CPU accesses three distinct address spaces:

- The Register File contains addresses for the general-purpose registers and the eZ8 CPU, Peripheral, and GPIO 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 sections. For more information on the eZ8 CPU and its address space, refer to *eZ8 CPU Core User Manual (UM0128)* available for download at www.zilog.com.

## Register File

The Register File address space in the Z8 Encore! XP® is 4 KB (4096 bytes). It 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 1 KB Register File address space is 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 is reserved (unavailable). Reading from the 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. Z8 Encore! XP F0822 Series contains 1 KB of on-chip RAM. Reading from Register File addresses outside the available RAM addresses (and not within the control register address space) returns an undefined value. Writing to these Register File addresses produces no effect.

## Program Memory

The eZ8 CPU supports 64 KB of Program Memory address space. Z8 Encore! XP® F0822 Series contain 4 KB to 8 KB on-chip Flash in the Program Memory address space, depending on the device. Reading from Program Memory addresses outside the available Flash addresses returns FFH. Writing to unimplemented Program Memory addresses produces no effect  Table 5 describes the Program Memory Maps for Z8 Encore! XP F0822 Series devices.

IRQ0 Enable Low Bit
IRQ0ENL (FC2H - Read/Write)

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

- D0: ADC IRQ Enable Hit Bit
- D1: SPI IRQ Enable Low Bit
- D2: I2C IRQ Enable Low Bit
- D3: UART 0 Transmitter IRQ
- D4: UART 0 Receiver IRQ Enable
- D5: Timer 0 IRQ Enable Low Bit
- D6: Timer 1 IRQ Enable Low Bit
- D7: Reserved

Interrupt Request 1
IRQ1 (FC3H - Read/Write)

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

- D0: Port A Pin Interrupt Request
  0 = IRQ from corresponding pin [7:0] is not pending
  1 = IRQ from corresponding pin [7:0] is awaiting service

IRQ1 Enable High Bit
IRQ1ENH (FC4H - Read/Write)

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

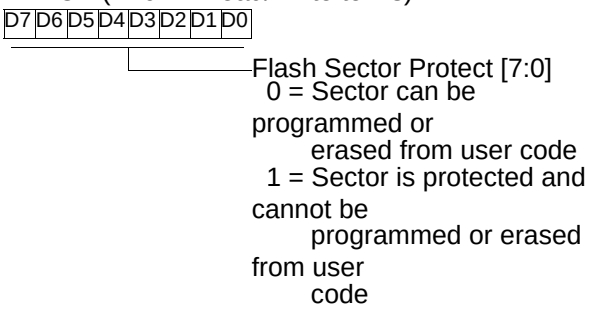
- D0: Port A Pin IRQ Enable High

IRQ1 Enable Low Bit
IRQ1ENL (FC5H - Read/Write)

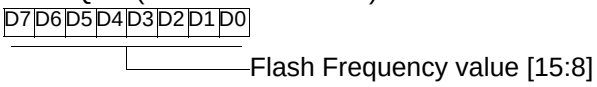
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

- D0: Port A Pin IRQ Enable Low

Flash Sector Protect
FPROT (FF9H - Read/Write to 1's)

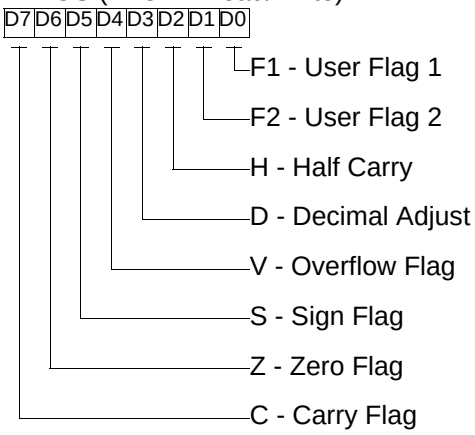



Flash Frequency High Byte
FFREQH (FFAH - Read/Write)

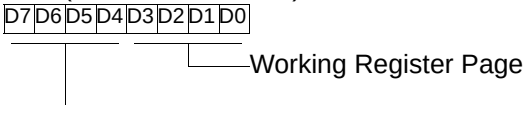



Flash Frequency Low Byte
FFREQL (FFBH - Read/Write)


Flags
FLAGS (FFCH - Read/Write)




Register Pointer
RP (FFDH- Read/Write)



pins.To determine the alternate function associated with each port pin, see GPIO Port Pin Block Diagram on page 48.

**Caution:** *Do not enable alternate function for GPIO port pins which do not have an associated alternate function. Failure to follow this guideline can result in unpredictable operation.*

**Table 17. Port A–CA–C Alternate Function Sub-Registers**

| BITS | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| FIELD | AF7 | AF6 | AF5 | AF4 | AF3 | AF2 | AF1 | AF0 |
| RESET | 0 | | | | | | | |
| R/W | R/W | | | | | | | |
| ADDR | If 02H in Port A–C Address Register, accessible through the Port A–C Control Register | | | | | | | |

**AF[7:0]—Port Alternate Function enabled**

0 = The port pin is in NORMAL mode and the DDx bit in the Port A–C Data Direction sub-register determines the direction of the pin.

1 = The alternate function is selected. Port pin operation is controlled by the alternate function.

### Port A–C Output Control Sub-Registers

The Port A–C Output Control sub-register (Table 18) is accessed through the Port A–C Control Register by writing 03H to the Port A–C Address Register. Setting the bits in the Port A–C Output Control sub-registers to 1 configures the specified port pins for open-drain operation. These sub-registers affect the pins directly and, as a result, alternate functions are also affected.

**Table 18. Port A–C Output Control Sub-Registers**

| BITS | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| FIELD | POC7 | POC6 | POC5 | POC4 | POC3 | POC2 | POC1 | POC0 |
| RESET | 0 | | | | | | | |
| R/W | R/W | | | | | | | |
| ADDR | If 03H in Port A–C Address Register, accessible through the Port A–C Control Register | | | | | | | |

**POC[7:0]—Port Output Control**

These bits function independently of the alternate function bit and always disable the drains if set to 1.

0 = The drains are enabled for any output mode (unless overridden by the

**Table 34. IRQ2 Enable and Priority Encoding**

| IRQ2ENH[*x*] | IRQ2ENL[*x*] | Priority | Description |
|---|---|---|---|
| 0 | 0 | Disabled | Disabled |
| 0 | 1 | Level 1 | Low |
| 1 | 0 | Level 2 | Nominal |
| 1 | 1 | Level 3 | High |

where *x* indicates the register bits from 0 through 7.

**Table 35. IRQ2 Enable High Bit Register (IRQ2ENH)**

| BITS | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| FIELD | Reserved | | | | C3ENH | C2ENH | C1ENH | C0ENH |
| RESET | 0 | | | | | | | |
| R/W | R/W | | | | | | | |
| ADDR | FC7H | | | | | | | |

**Reserved—Must be 0.**
**C3ENH**—Port C3 Interrupt Request Enable High Bit
**C2ENH**—Port C2 Interrupt Request Enable High Bit
**C1ENH**—Port C1 Interrupt Request Enable High Bit
**C0ENH**—Port C0 Interrupt Request Enable High Bit

**Table 36. IRQ2 Enable Low Bit Register (IRQ2ENL)**

| BITS | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| FIELD | Reserved | | | | C3ENL | C2ENL | C1ENL | C0ENL |
| RESET | 0 | | | | | | | |
| R/W | R/W | | | | | | | |
| ADDR | FC8H | | | | | | | |

**Reserved—Must be 0.**
**C3ENL**—Port C3 Interrupt Request Enable Low Bit
**C2ENL**—Port C2 Interrupt Request Enable Low Bit
**C1ENL**—Port C1 Interrupt Request Enable Low Bit
**C0ENL**—Port C0 Interrupt Request Enable Low Bit

## Timer 0–1 Control 1 Registers

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

**Table 46. Timer 0–1 Control Register (TxCTL)**

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

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

**CAPTURE Mode**
0 = Count is captured on the rising edge of the Timer Input signal.
1 = Count is captured on the falling edge of the Timer Input signal.

## Receiving Data Using Interrupt-Driven Method

The UART Receiver interrupt indicates the availability of new data (as well as error conditions). Follow the steps below to configure the UART receiver for interrupt-driven operation:

1. Write to the UART Baud Rate High and Low Byte Registers to set the required 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 Receiver interrupt and set the required priority.
5. Clear the UART Receiver interrupt in the applicable Interrupt Request Register.
6. Write to the UART Control 1 Register to enable MULTIPROCESSOR (9-bit) mode functions, if desired.
   – Set the Multiprocessor Mode Select (MPEN) to enable MULTIPROCESSOR mode.
   – Set the Multiprocessor Mode Bits, MPMD[1:0], to select the required address matching scheme.
   – Configure the UART to interrupt on received data and errors or errors only (interrupt on errors only is unlikely to be useful for Z8 Encore! XP devices without a DMA block)
7. Write the device address to the Address Compare Register (automatic multiprocessor modes only).
8. Write to the UART Control 0 Register to:
   – Set the receive enable bit (REN) to enable the UART for data reception
   – Enable parity, if required, and if MULTIPROCESSOR mode is not enabled, and select either even or odd parity.
9. Execute an EI instruction to enable interrupts.

The UART is now configured for interrupt-driven data reception. When the UART Receiver Interrupt is detected, the associated ISR performs the following:

1. Check the UART Status 0 Register to determine the source of the interrupt-error, break, or received data.
2. If the interrupt was due to data available, read the data from the UART Receive Data Register. If operating in MULTIPROCESSOR (9-bit) mode, further actions may be required depending on the Multiprocessor Mode bits MPMD[1:0].
3. Clear the UART Receiver Interrupt in the applicable Interrupt Request Register.
4. Execute the IRET instruction to return from the ISR and await more data.

Enable signal asserts at least one UART bit period and no greater than two UART bit periods before the Start bit is transmitted. This format allows a setup time to enable the transceiver. The Driver Enable signal deasserts one system clock period after the last STOP bit is transmitted. This one system clock delay allows both time for data to clear the transceiver before disabling it, as well as the ability to determine if another character follows the current character. In the event of back to back characters (new data must be written to the Transmit Data Register before the previous character is completely transmitted) the DE signal is not deasserted between characters. The DEPOL bit in the UART Control Register 1 sets the polarity of the Driver Enable signal.

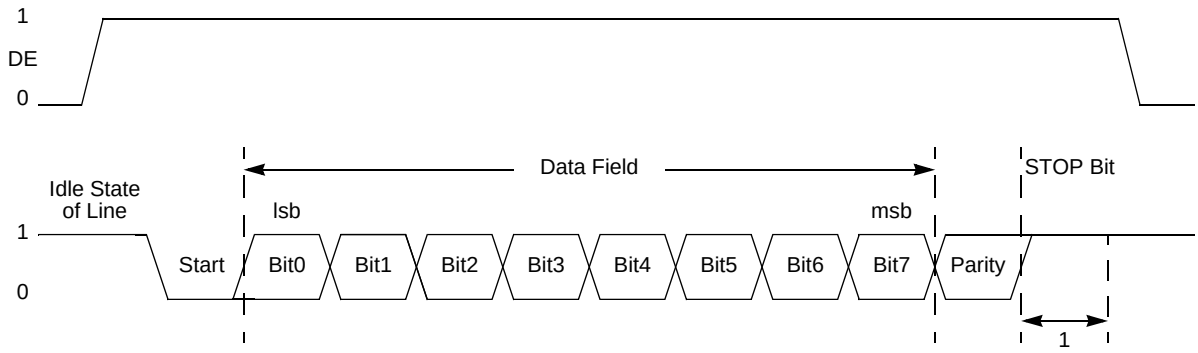



**Figure 15. UART Driver Enable Signal Timing (with 1 STOP Bit and Parity)**

The Driver Enable to Start bit setup time is calculated as follows:

$$\left(\frac{1}{\text{Baud Rate (Hz)}}\right) \leq \text{DE to Start Bit Setup Time (s)} \leq \left(\frac{2}{\text{Baud Rate (Hz)}}\right)$$

## UART Interrupts

The UART features separate interrupts for the transmitter and the receiver. In addition, when the UART primary functionality is disabled, the BRG also functions as a basic timer with interrupt capability.

### Transmitter Interrupts

The transmitter generates a single interrupt when the Transmit Data Register Empty bit (TDRE) is set to 1. This indicates that the transmitter is ready to accept new data for transmission. The TDRE interrupt occurs after the Transmit shift register has shifted the first bit of data out. At this point, the Transmit Data Register can be written with the next character to send. This provides 7 bit periods of latency to load the Transmit Data Register before the Transmit shift register completes shifting the current character. Writing to the UART Transmit Data Register clears the TDRE bit to 0.

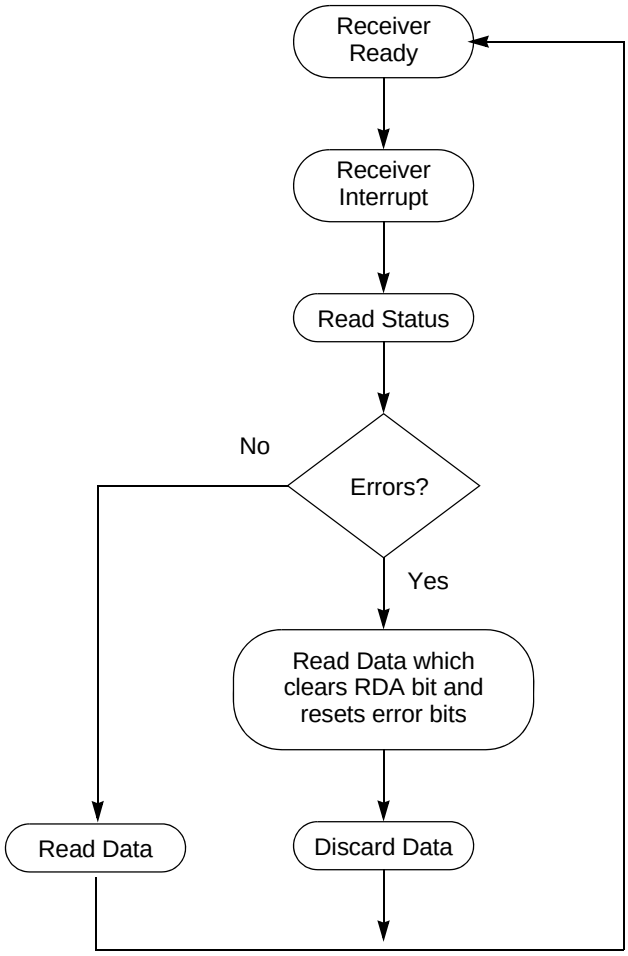



**Figure 16. UART Receiver Interrupt Service Routine Flow**

## UART Baud Rate Generator

The UART Baud Rate Generator creates a lower frequency baud rate clock for data transmission. The input to the BRG is the system clock. The UART Baud Rate High and Low Byte Registers combine to create a 16-bit baud rate divisor value (BRG[15:0]) that sets the data transmission rate (baud rate) of the UART. The UART data rate is calculated using the following equation:

$$\text{UART Data Rate (bits/s)} = \frac{\text{System Clock Frequency (Hz)}}{16 \times \text{UART Baud Rate Divisor Value}}$$

Receive interrupts occur when a byte of data has been received by the I2C Controller (Master reading data from Slave). This procedure sets the RDRF bit of the I2C Status Register. The RDRF bit is cleared by reading the I2C Data Register. The RDRF bit is set during the acknowledge phase. The I2C Controller pauses after the acknowledge phase until the receive interrupt is cleared before performing any other action.

Transmit interrupts occur when the TDRE bit of the I2C Status register sets and the TXI bit in the I2C Control Register is set. Transmit interrupts occur under the following conditions when the Transmit Data Register is empty:

- The I2C Controller is enabled
- The first bit of the byte of an address is shifting out and the RD bit of the I2C Status register is deasserted.
- The first bit of a 10-bit address shifts out.
- The first bit of write data shifts out.

**Note:** *Writing to the I2C Data Register always clears the TRDE bit to 0. When TDRE is asserted, the I2C Controller pauses at the beginning of the Acknowledge cycle of the byte currently shifting out until the data register is written with the next value to send or the STOP or START bits are set indicating the current byte is the last one to send.*

The fourth interrupt source is the BRG. If the I2C Controller is disabled (IEN bit in the I2CCTL Register = 0) and the BIRQ bit in the I2CCTL Register = 1, an interrupt is generated when the BRG counts down to 1. This allows the I2C Baud Rate Generator to be used by software as a general purpose timer when IEN = 0.

## Software Control of I2C Transactions

Software controls I2C transactions by using the I2C Controller interrupt, by polling the I2C Status register or by DMA. Note that not all products include a DMA Controller.

To use interrupts, the I2C interrupt must be enabled in the Interrupt Controller. The TXI bit in the I2C Control Register must be set to enable transmit interrupts.

To control transactions by polling, the interrupt bits (TDRE, RDRF and NCKI) in the I2C Status Register should be polled. The TDRE bit asserts regardless of the state of the TXI bit.

Either or both transmit and receive data movement can be controlled by the DMA Controller. The DMA Controller channel(s) must be initialized to select the I2C transmit and receive requests. Transmit DMA requests require that the TXI bit in the I2C Control Register be set.

**Caution:** *A transmit (write) DMA operation hangs if the slave responds with a Not Acknowledge before the last byte has been sent. After receiving the Not Acknowledge, the I2C Controller sets the NCKI bit in the Status Register and pauses until either the STOP or*

*START bits in the Control Register are set.*

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

## Start and Stop Conditions

The Master ($I^2C$) drives all Start and Stop signals and initiates all transactions. To start a transaction, the $I^2C$ Controller generates a START condition by pulling the SDA signal Low while SCL is High. To complete a transaction, the $I^2C$ Controller generates a Stop condition by creating a low-to-high transition of the SDA signal while the SCL signal is high. The START and STOP bits in the $I^2C$ Control Register control the sending of the Start and Stop conditions. A Master is also allowed to end one transaction and begin a new one by issuing a Restart. This is accomplished by setting the START bit at the end of a transaction, rather than the STOP bit.

**Note:** *The Start condition not sent until the START bit is set and data has been written to the $I^2C$ Data Register.*

## Master Write and Read Transactions

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

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

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

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

**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 | - |
| Write OCD Counter Register | 01H | - | - |
| Read OCD Status Register | 02H | Yes | - |
| Read OCD Counter Register | 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 peripheral control registers at address F00H-FFH are allowed. Additionally, only the Mass Erase command is allowed to be written to the Flash Control Register. |
| Read Register | 09H | - | Only reads of the peripheral control registers at address F00H-FFH are allowed. |
| Write Program Memory | 0AH | - | Disabled |
| Read Program Memory | 0BH | - | Disabled |
| Write Data Memory | 0CH | - | Disabled |
| Read Data Memory | 0DH | - | Disabled |
| Read Program Memory CRC | 0EH | - | - |
| Reserved | 0FH | - | - |
| Step Instruction | 10H | - | Disabled |
| Stuff Instruction | 11H | - | Disabled |

```
DBG ← 04H
DBG ← OCDCTL[7:0]
```

- **Read OCD Control Register (05H)**—The Read OCD Control Register command reads the value of the OCDCTL register.

```
DBG ← 05H
DBG → OCDCTL[7:0]
```

- **Write Program Counter (06H)**—The Write Program Counter command writes the data that follows to the eZ8 CPU's Program Counter. If the device is not in DEBUG mode or if the Read Protect Option Bit is enabled, the Program Counter values are discarded.

```
DBG ← 06H
DBG ← ProgramCounter[15:8]
DBG ← ProgramCounter[7:0]
```

- **Read Program Counter (07H)**—The Read Program Counter command reads the value in the eZ8 CPU's Program Counter. If the device is not in DEBUG mode or if the Read Protect Option Bit is enabled, this command returns FFFFH.

```
DBG ← 07H
DBG → ProgramCounter[15:8]
DBG → ProgramCounter[7:0]
```

- **Write Register (08H)**—The Write Register command writes data to the Register File. Data can be written 1-256 bytes at a time (256 bytes can be written by setting size to zero). If the device is not in DEBUG mode, the address and data values are discarded. If the Read Protect Option Bit is enabled, then only writes to the Flash Control Registers are allowed and all other register write data values are discarded.

```
DBG ← 08H
DBG ← {4'h0,Register Address[11:8]}
DBG ← Register Address[7:0]
DBG ← Size[7:0]
DBG ← 1-256 data bytes
```

- **Read Register (09H)**—The Read Register command reads data from the Register File. Data can be read 1-256 bytes at a time (256 bytes can be read by setting size to zero). Reading peripheral control registers through the OCD does not effect peripheral operation. For example, register bits that are normally cleared upon a read operation will not be effected (WDTSTAT register is affected by OCD read register operation). If the 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
```

A “reset and stop” function can be achieved by writing 81H to this register. A “reset and go” function can be achieved by writing 41H to this register. If the device is in DEBUG mode, a “run” function can be implemented by writing 40H to this register.

**Table 94. OCD Control Register (OCDCTL)**

| BITS | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| FIELD | DBGMODE | BRKEN | DBGACK | BRKLOOP | BRKPC | BRKZRO | Reserved | RST |
| RESET | 0 | | | | | | | |
| R/W | R/W | | | R | | | | R/W |

**DBGMODE—Debug Mode**
Setting this bit to 1 causes the device to enter DEBUG mode. When in DEBUG mode, the eZ8 CPU stops fetching new instructions. Clearing this bit causes the eZ8 CPU to start running again. This bit is automatically set when a BRK instruction is decoded and Breakpoints are enabled. If the Read Protect Option Bit is enabled, this bit can only be cleared by resetting the device, it cannot be written to 0.
0 = The Z8 Encore! XP F0822 Series device is operating in NORMAL mode.
1 = The Z8 Encore! XP F0822 Series device is in DEBUG mode.

**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 an NOP instruction. If this bit is set to 1 and a BRK instruction is decoded, the OCD takes action dependent upon the BRKLOOP bit.
0 = BRK instruction is disabled.
1 = BRK instruction is 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 occurs.
0 = Debug Acknowledge is disabled.
1 = Debug Acknowledge is enabled.

**BRKLOOP—Breakpoint Loop**
This bit determines what action the OCD takes when a BRK instruction is decoded if breakpoints are enabled (BRKEN is 1). If this bit is 0, then the DBGMODE bit is automatically set to 1 and the OCD enter DEBUG mode. If BRKLOOP is set to 1, then the eZ8 CPU loops on the BRK instruction.
0 = BRK instruction sets DBGMODE to 1.
1 = eZ8 CPU loops on BRK instruction.

**BRKPC—Break when PC == OCDCNTR**
If this bit is set to 1, then the OCDCNTR register is used as a hardware breakpoint. When the program counter matches the value in the OCDCNTR register, DBGMODE is

**Table 97. DC Characteristics (Continued)**

| Symbol | Parameter | $T_A$ = -40 °C to 105 °C | | | Units | Conditions |
|---|---|---|---|---|---|---|
| | | Minimum | Typical | Maximum | | |
| $V_{RAM}$ | RAM Data Retention | 0.7 | – | – | V | |
| $I_{IL}$ | Input Leakage Current | -5 | – | +5 | µA | $V_{DD}$ = 3.6 V; $V_{IN}$ = VDD or VSS[1] |
| $I_{TL}$ | Tri-State Leakage Current | -5 | – | +5 | µA | $V_{DD}$ = 3.6 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 | |
| $I_{PU1}$ | Weak Pull-up Current | 9 | 20 | 50 | µA | VDD = 2.7–3.6 V. $T_A$ = 0 °C to +70 °C |
| $I_{PU2}$ | Weak Pull-up Current | 7 | 20 | 75 | µA | VDD = 2.7–3.6 V. $T_A$ = -40 °C to +105 °C |

[1] This condition excludes all pins that have on-chip pull-ups, when driven Low.

[2] These values are provided for design guidance only and are not tested in production.

Figure 41 on page 189 displays the typical active mode current consumption while operating at 25 ºC, 3.3 V, versus the system clock frequency. All GPIO pins are configured as outputs and driven High.

## 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 on page 211.

Table 116 contains additional symbols that are used throughout the Instruction Summary and Instruction Set Description sections.

**Table 116. Additional Symbols**

| Symbol | Definition |
|---|---|
| dst | Destination Operand |
| src | Source Operand |
| @ | Indirect Address Prefix |
| SP | Stack Pointer |
| PC | Program Counter |
| FLAGS | Flags Register |
| RP | Register Pointer |
| # | Immediate Operand Prefix |
| B | Binary Number Suffix |
| % | Hexadecimal Number Prefix |
| H | Hexadecimal Number Suffix |

Assignment of a value is indicated by an arrow. For example,

dst ← dst + src

indicates the source data is added to the destination data and the result is stored in the destination location.

## D



## E



## F

## U



## V



## W