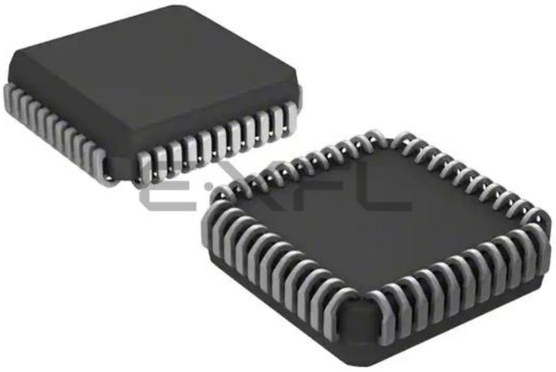

Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	eZ8
Core Size	8-Bit
Speed	20MHz
Connectivity	I²C, IrDA, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	31
Program Memory Size	32KB (32K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	2K 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	Surface Mount
Package / Case	44-LCC (J-Lead)
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z8f3201vn020sc

Block Diagram

Figure 55 illustrates the block diagram of the architecture of the Z8 Encore!™.

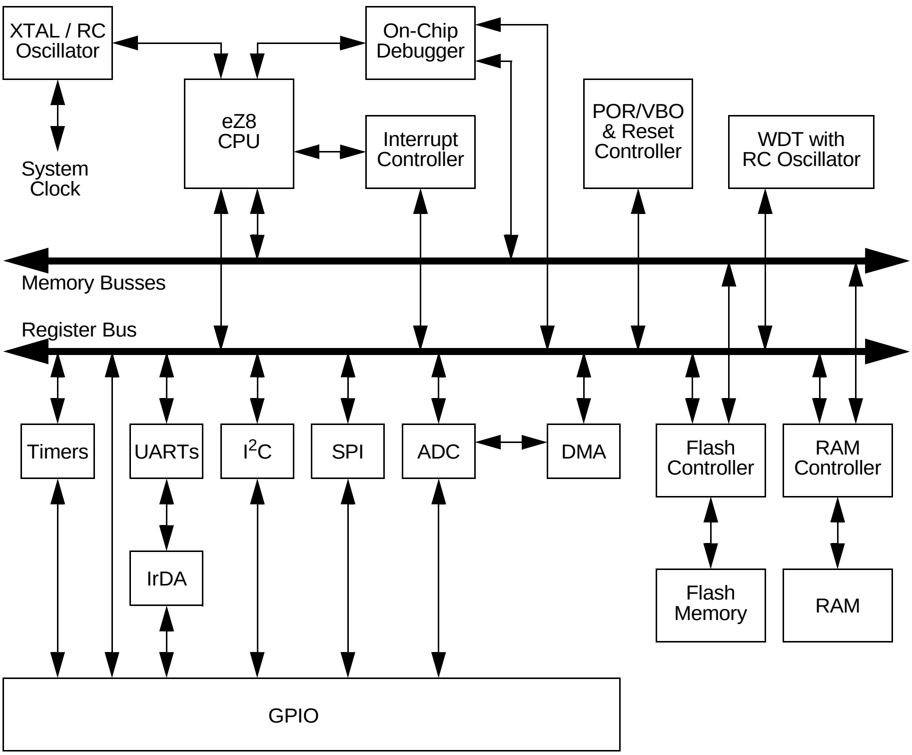

Figure 55. Z8 Encore!® Block Diagram

CPU and Peripheral Overview

eZ8 CPU Features

The eZ8, ZiLOG's latest 8-bit Central Processing Unit (CPU), meets the continuing demand for faster and more code-efficient microcontrollers. The eZ8 CPU executes a superset of the original Z8 instruction set. The eZ8 CPU features include:

- Direct register-to-register architecture allows each register to function as an accumulator, improving execution time and decreasing the required program memory

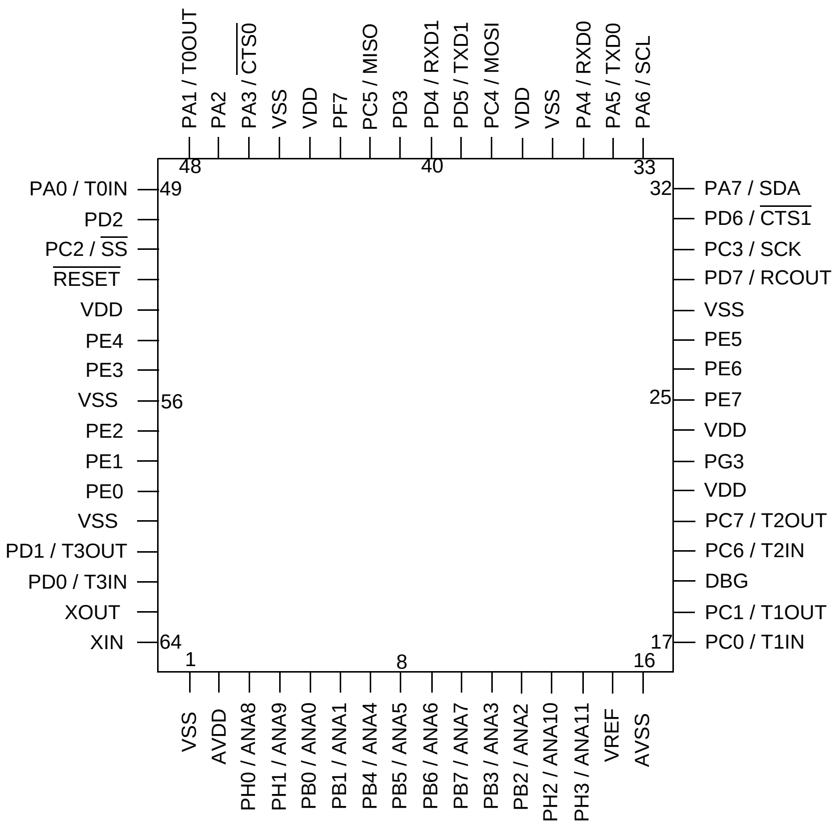

Figure 59. Z8Fxx02 in 64-Pin Low-Profile Quad Flat Package (LQFP)

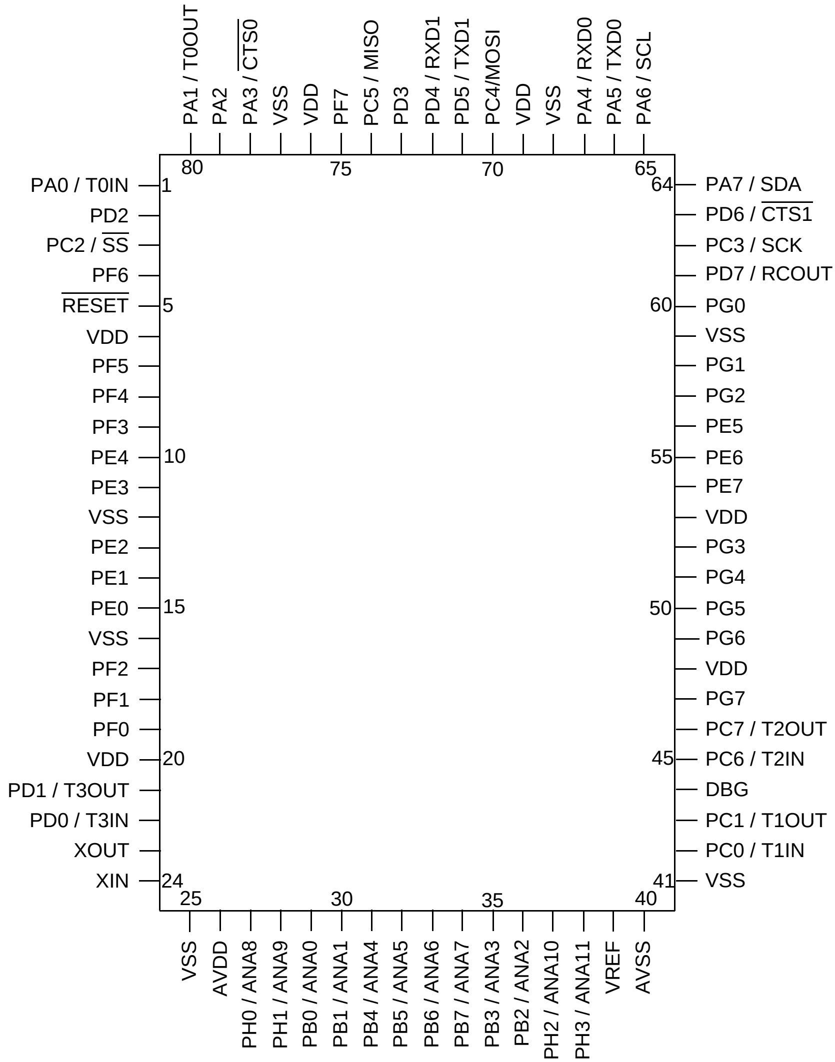

Figure 61. Z8Fxx03 in 80-Pin Quad Flat Package (QFP)

- Disable the timer
- Configure the timer for PWM mode.
- Set the prescale value.
- Set the initial logic level (High or Low) and PWM High/Low transition for the Timer Output alternate function.

2. Write to the Timer High and Low Byte registers to set the starting count value (typically `0001H`). This only affects the first pass in PWM mode. After the first timer reset in PWM mode, counting always begins at the reset value of `0001H`.
3. Write to the PWM High and Low Byte registers to set the PWM value.
4. Write to the Timer Reload High and Low Byte registers to set the Reload value (PWM period). The Reload value must be greater than the PWM value.
5. If desired, enable the timer interrupt and set the timer interrupt priority by writing to the relevant interrupt registers.
6. Configure the associated GPIO port pin for the Timer Output alternate function.
7. Write to the Timer Control register to enable the timer and initiate counting.

The PWM period is given by the following equation:

$$\textbf{PWM Period (s)} = \frac{\textbf{Reload Value} \times \textbf{Prescale}}{\textbf{System Clock Frequency (Hz)}}$$

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

If `TPOL` is set to 0, the ratio of the PWM output High time to the total period is given by:

$$\textbf{PWM Output High Time Ratio (\%)} = \frac{\textbf{Reload Value} - \textbf{PWM Value}}{\textbf{Reload Value}} \times \textbf{100}$$

If `TPOL` is set to 1, the ratio of the PWM output High time to the total period is given by:

$$\textbf{PWM Output High Time Ratio (\%)} = \frac{\textbf{PWM Value}}{\textbf{Reload Value}} \times \textbf{100}$$

Capture Mode

In Capture mode, the current timer count value is recorded when the desired external Timer Input transition occurs. The Capture count value is written to the Timer PWM High and Low Byte Registers. The timer input is the system clock. The `TPOL` bit in the Timer Control register determines if the Capture occurs on a rising edge or a falling edge of the

Watch-Dog Timer

Overview

The Watch-Dog Timer (WDT) helps protect against corrupt or unreliable software, power faults, and other system-level problems which may place the Z8 Encore!® into unsuitable operating states. The Watch-Dog Timer includes the following features:

- On-chip RC oscillator
- A selectable time-out response: Short Reset or interrupt
- 24-bit programmable time-out value

Operation

The Watch-Dog Timer (WDT) is a retriggerable one-shot timer that resets or interrupts the Z8F640x family device when the WDT reaches its terminal count. The Watch-Dog Timer uses its own dedicated on-chip RC oscillator as its clock source. The Watch-Dog Timer has only two modes of operation—on and off. Once enabled, it always counts and must be refreshed to prevent a time-out. An enable can be performed by executing the WDT instruction or by setting the `WDT_AO` Option Bit. The `WDT_AO` bit enables the Watch-Dog Timer to operate all the time, even if a WDT instruction has not been executed.

The Watch-Dog Timer is a 24-bit reloadable downcounter that uses three 8-bit registers in the eZ8 CPU register space to set the reload value. The nominal WDT time-out period is given by the following equation:

$$\textbf{WDT Time-out Period (ms)} = \frac{\textbf{WDT Reload Value}}{\textbf{50}}$$

where the WDT reload value is the decimal value of the 24-bit value given by {WDTU[7:0], WDTH[7:0], WDTL[7:0]} and the typical Watch-Dog Timer RC oscillator frequency is 50kHz. The Watch-Dog Timer cannot be refreshed once it reaches `000002H`. The WDT Reload Value must not be set to values below `000004H`. Table 45 provides

3. Enable the Baud Rate Generator timer function and associated interrupt by setting the `BIRQ` bit in the UART*x* Control 1 register to 1.

UART Control Register Definitions

The UART control registers support both the UARTs and the associated Infrared Encoder/ Decoders. For more information on the infrared operation, refer to the **Infrared Encoder/ Decoder** chapter on page 95.

UART*x* Transmit Data Register

Data bytes written to the UART*x* Transmit Data register (Table 50) are shifted out on the TXD*x* pin. The Write-only UART*x* Transmit Data register shares a Register File address with the Read-only UART*x* Receive Data register.

Table 50. UART*x* Transmit Data Register (U*x*TXD)

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

TXD—Transmit Data
UART transmitter data byte to be shifted out through the TXD*x* pin.

Table 57. UART*x* Baud Rate Low Byte Register (U*x*BRL)

BITS	7	6	5	4	3	2	1	0
FIELD	BRL							
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	F47H and F4FH							

The UART data rate is calculated using the following equation:

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

For a given UART data rate, the integer baud rate divisor value is calculated using the following equation:

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

The baud rate error relative to the desired baud rate is calculated using the following equation:

$$\text{UART Baud Rate Error (\%)} = 100 \times \left(\frac{\text{Actual Data Rate} - \text{Desired Data Rate}}{\text{Desired Data Rate}}\right)$$

For reliable communication, the UART baud rate error must never exceed 5 percent. Table 58 provides information on data rate errors for popular baud rates and commonly used crystal oscillator frequencies.

Transfer Format PHASE Equals Zero

Figure 77 illustrates the timing diagram for an SPI transfer in which PHASE is cleared to 0. The two SCK waveforms show polarity with CLKPOL reset to 0 and with CLKPOL set to one. The diagram may be interpreted as either a Master or Slave timing diagram since the SCK Master-In/Slave-Out (MISO) and Master-Out/Slave-In (MOSI) pins are directly connected between the Master and the Slave.

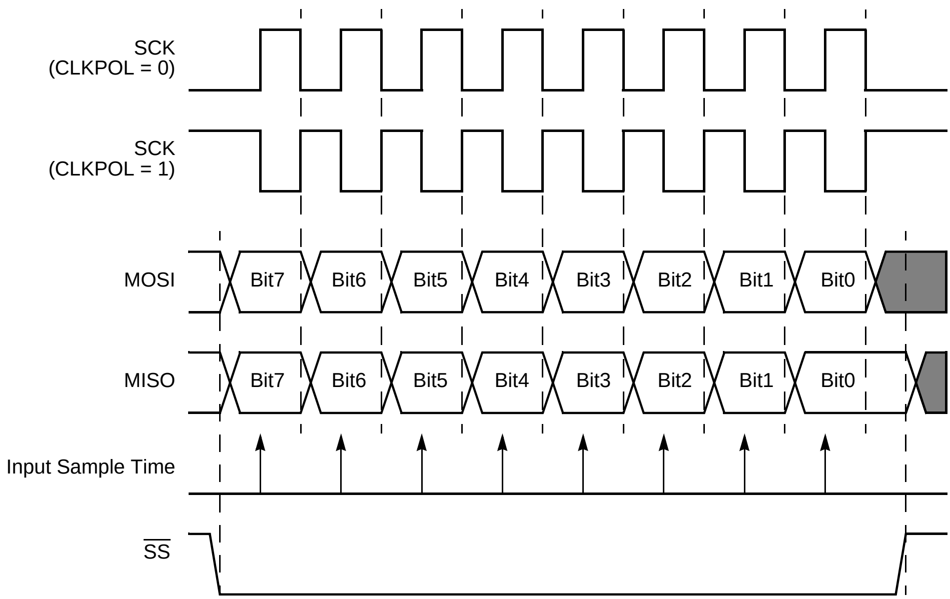

Figure 77. SPI Timing When PHASE is 0

Transfer Format PHASE Equals One

Figure 78 illustrates the timing diagram for an SPI transfer in which PHASE is one. Two waveforms are depicted for SCK, one for CLKPOL reset to 0 and another for CLKPOL set to 1.

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.

Direct Memory Access Controller

Overview

The Z8F640x family device's Direct Memory Access (DMA) Controller provides three independent Direct Memory Access channels. Two of the channels (DMA0 and DMA1) transfer data between the on-chip peripherals and the Register File. The third channel (DMA_ADC) controls the Analog-to-Digital Converter (ADC) operation and transfers the Single-Shot mode ADC output data to the Register File.

Operation

DMA0 and DMA1 Operation

DMA0 and DMA1, referred to collectively as DMA*x*, transfer data either from the on-chip peripheral control registers to the Register File, or from the Register File to the on-chip peripheral control registers. The sequence of operations in a DMA*x* data transfer is:

1. DMA*x* trigger source requests a DMA data transfer.
2. DMA*x* requests control of the system bus (address and data) from the eZ8 CPU.
3. After the eZ8 CPU acknowledges the bus request, DMA*x* transfers either a single byte or a two-byte word (depending upon configuration) and then returns system bus control back to the eZ8 CPU.
4. If Current Address equals End Address:
   - DMA*x* reloads the original Start Address
   - If configured to generate an interrupt, DMA*x* sends an interrupt request to the Interrupt Controller
   - If configured for single-pass operation, DMA*x* resets the `DEN` bit in the DMA*x* Control register to 0 and the DMA is disabled.

   If Current Address does not equal End Address, the Current Address increments by 1 (single-byte transfer) or 2 (two-byte word transfer).

Table 76 provides an example of the Register File addresses if the DMA_ADC Address register contains the value 72H.

Table 76. DMA_ADC Register File Address Example

ADC Analog Input	Register File Address (Hex)[1]
0	720H-721H
1	722H-723H
2	724H-725H
3	726H-727H
4	728H-729H
5	72AH-72BH
6	72CH-72DH
7	72EH-72FH
8	730H-731H
9	732H-733H
10	734H-735H
11	736H-737H

[1] DMAA_ADDR set to 72H.

Table 77. DMA_ADC Address Register (DMAA_ADDR)

BITS	7	6	5	4	3	2	1	0
FIELD	DMAA_ADDR							Reserved
RESET	X	X	X	X	X	X	X	X
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	FBDH							

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

Reserved
This bit is reserved and must be 0.

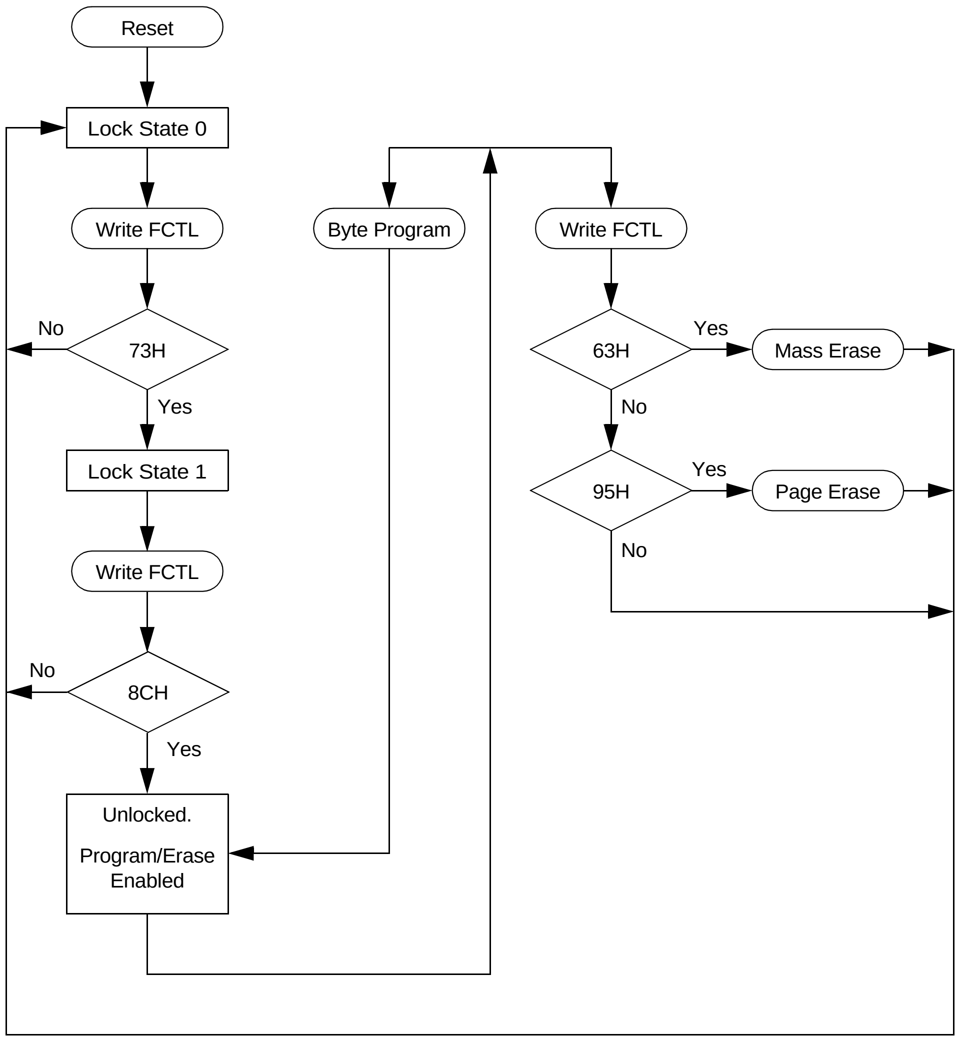

Figure 85. Flash Controller Operation Flow Chart

Flash Code Protection Using the Option Bits

The FHSWP and FWP Option Bits combine to provide three levels of Flash Program Memory protection as listed in Table 84. Refer to the **Option Bits** chapter for more information.

Table 84. Flash Code Protection Using the Option Bits

FHSWP	FWP	Flash Code Protection Description
0	0	Programming and erasure disabled for all of Flash Program Memory. In user code programming, Page Erase, and Mass Erase are all disabled. Mass Erase is available through the On-Chip Debugger.
1	0	Programming and Page Erase are enabled for the High Sector of the Flash Program Memory only. The High Sector on the Z8F640x family device contains 1KB to 4KB of Flash with addresses at the top of the available Flash memory. Programming and Page Erase are disabled for the other portions of the Flash Program Memory. Mass erase through user code is disabled. Mass Erase is available through the On-Chip Debugger.
0 or 1	1	Programming, Page Erase, and Mass Erase are enabled for all of Flash Program Memory.

Flash Code Protection Using the Flash Controller

At Reset, the Flash Controller locks to prevent accidental program or erasure of the Flash memory. To program or erase the Flash memory, unlock the Flash Controller by making two consecutive writes to the Flash Control register with the values 73H and 8CH, sequentially. After unlocking the Flash Controller, the Flash can be programmed or erased. When the Flash Controller is unlocked, any value written to the Flash Control register locks the Flash Controller. Writing the Mass Erase or Page Erase commands executes the function before locking the Flash Controller.

Byte Programming

When the Flash Controller is unlocked, all writes to Program Memory program a byte into the Flash. An erased Flash byte contains all 1's (FFH). The programming operation can only be used to change bits from 1 to 0. To change a Flash bit (or multiple bits) from 0 to 1 requires execution of either the Page Erase or Mass Erase commands.

Byte Programming can be accomplished using the On-Chip Debugger's Write Memory command or eZ8 CPU execution of the LDC or LDCI instructions. Refer to the **eZ8 CPU User Manual** for a description of the LDC and LDCI instructions. While the Flash Controller programs the Flash memory, the eZ8 CPU idles but the system clock and on-chip peripherals continue to operate. To exit programming mode and lock the Flash, write any value to the Flash Control register, except the Mass Erase or Page Erase commands.

If the OCD receives a Serial Break (nine or more continuous bits Low) the Auto-Baud Detector/Generator resets. The Auto-Baud Detector/Generator can then be reconfigured by sending 80H.

OCD Serial Errors

The On-Chip Debugger can detect any of the following error conditions on the DBG pin:

- Serial Break (a minimum of nine continuous bits Low)
- Framing Error (received Stop bit is Low)
- Transmit Collision (OCD and host simultaneous transmission detected by the OCD)

When the OCD detects one of these errors, it aborts any command currently in progress, transmits a four character long Serial Break back to the host, and resets the Auto-Baud Detector/Generator. A Framing Error or Transmit Collision may be caused by the host sending a Serial Break to the OCD. Because of the open-drain nature of the interface, returning a Serial Break break back to the host only extends the length of the Serial Break if the host releases the Serial Break early.

The host should transmit a Serial Break on the DBG pin when first connecting to the Z8F640x family device or when recovering from an error. A Serial Break from the host resets the Auto-Baud Generator/Detector but does not reset the OCD Control register. A Serial Break leaves the Z8F640x family device in Debug mode if that is the current mode. The OCD is held in Reset until the end of the Serial Break when the DBG pin returns High. Because of the open-drain nature of the DBG pin, the host can send a Serial Break to the OCD even if the OCD is transmitting a character.

Breakpoints

Execution Breakpoints are generated using the BRK instruction (opcode 00H). When the eZ8 CPU decodes a BRK instruction, it signals the On-Chip Debugger. If Breakpoints are enabled, the OCD enters Debug mode and idles the eZ8 CPU. If Breakpoints are not enabled, the OCD ignores the BRK signal and the BRK instruction operates as an NOP.

Breakpoints in Flash Memory

The BRK instruction is opcode 00H, which corresponds to the fully programmed state of a byte in Flash memory. To implement a Breakpoint, write 00H to the desired address, over-writing the current instruction. To remove a Breakpoint, the corresponding page of Flash memory must be erased and reprogrammed with the original data.

Watchpoints

The On-Chip Debugger can set one Watchpoint to cause a Debug Break. The Watchpoint identifies a single Register File address. The Watchpoint can be set to break on reads and/or writes of the selected Register File address. Additionally, the Watchpoint can be configured to break only when a specific data value is read and/or written from the specified reg-

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	$T_A = -40^0C$ to 105^0C			Units	Conditions
		Minimum	Typical	Maximum		
V_{DD}	Supply Voltage	3.0	–	3.6	V	
V_{IL1}	Low Level Input Voltage	-0.3	–	$0.3*V_{DD}$	V	For all input pins except $\overline{\text{RESET}}$, DBG, and XIN.
V_{IL2}	Low Level Input Voltage	-0.3	–	$0.2*V_{DD}$	V	For $\overline{\text{RESET}}$, DBG, and XIN.
V_{IH1}	High Level Input Voltage	$0.7*V_{DD}$	–	5.5	V	Port A, C, D, E, F, and G pins.
V_{IH2}	High Level Input Voltage	$0.7*V_{DD}$	–	$V_{DD}+0.3$	V	Port B and H pins.
V_{IH3}	High Level Input Voltage	$0.8*V_{DD}$	–	$V_{DD}+0.3$	V	$\overline{\text{RESET}}$, DBG, and XIN pins.
V_{OL1}	Low Level Output Voltage	–	–	0.4	V	$V_{DD} = 3.0V$; $I_{OL} = 2mA$ High Output Drive disabled.
V_{OH1}	High Level Output Voltage	2.4	–	–	V	$V_{DD} = 3.0V$; $I_{OH} = -2mA$ High Output Drive disabled.
V_{OL2}	Low Level Output Voltage	–	–	0.6	V	$V_{DD} = 3.3V$; $I_{OL} = 20mA$ High Output Drive enabled. $T_A = -40^0C$ to $+70^0C$
V_{OL3}	Low Level Output Voltage	–	–	0.6	V	$V_{DD} = 3.3V$; $I_{OL} = 15mA$ High Output Drive enabled. $T_A = 70^0C$ to $+105^0C$
V_{OH2}	High Level Output Voltage	2.4	–	–	V	$V_{DD} = 3.3V$; $I_{OH} = -20mA$ High Output Drive enabled. $T_A = -40^0C$ to $+70^0C$
V_{OH3}	High Level Output Voltage	2.4	–	–	V	$V_{DD} = 3.3V$; $I_{OH} = -15mA$ High Output Drive enabled. $T_A = 70^0C$ to $+105^0C$
I_{IL}	Input Leakage Current	-5	–	+5	µA	$V_{DD} = 3.6V$; V_{IN} = VDD or VSS[1]
I_{TL}	Tri-State Leakage Current	-5	–	+5	µA	$V_{DD} = 3.6V$
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	

; 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)	Flags C	Z	S	V	D	H	Fetch Cycles	Instr. Cycles
BTJZ bit, src, dst	if src[bit] = 0 PC ← PC + X		r	F6	-	-	-	-	-	-	3	3
			Ir	F7							3	4
CALL dst	SP ← SP -2 @SP ← PC PC ← dst	IRR		D4	-	-	-	-	-	-	2	6
		DA		D6							3	3
CCF	C ← ~C			EF	*	-	-	-	-	-	1	2
CLR dst	dst ← 00H	R		B0	-	-	-	-	-	-	2	2
		IR		B1							2	3
COM dst	dst ← ~dst	R		60	-	*	*	0	-	-	2	2
		IR		61							2	3
CP dst, src	dst - src	r	r	A2	*	*	*	*	-	-	2	3
		r	Ir	A3							2	4
		R	R	A4							3	3
		R	IR	A5							3	4
		R	IM	A6							3	3
		IR	IM	A7							3	4
CPC dst, src	dst - src - C	r	r	1F A2	*	*	*	*	-	-	3	3
		r	Ir	1F A3							3	4
		R	R	1F A4							4	3
		R	IR	1F A5							4	4
		R	IM	1F A6							4	3
		IR	IM	1F A7							4	4
CPCX dst, src	dst - src - C	ER	ER	1F A8	*	*	*	*	-	-	5	3
		ER	IM	1F A9							5	3
CPX dst, src	dst - src	ER	ER	A8	*	*	*	*	-	-	4	3
		ER	IM	A9							4	3
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

Problem Description or Suggestion

Provide a complete description of the problem or your suggestion. If you are reporting a specific problem, include all steps leading up to the occurrence of the problem. Attach additional pages as necessary.