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	46
Program Memory Size	24KB (24K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	3V ~ 3.6V
Data Converters	A/D 12x10b
Oscillator Type	Internal
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	68-LCC (J-Lead)
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z8f2402vs020sc

Block Diagram

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

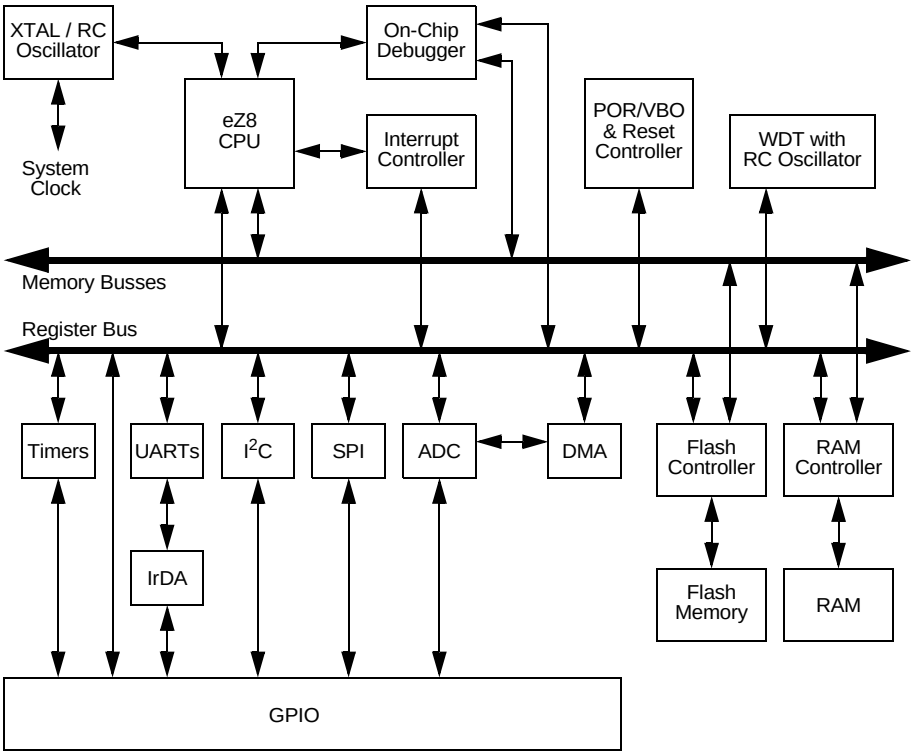

Figure 55. Z8 Encore!® Block Diagram

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

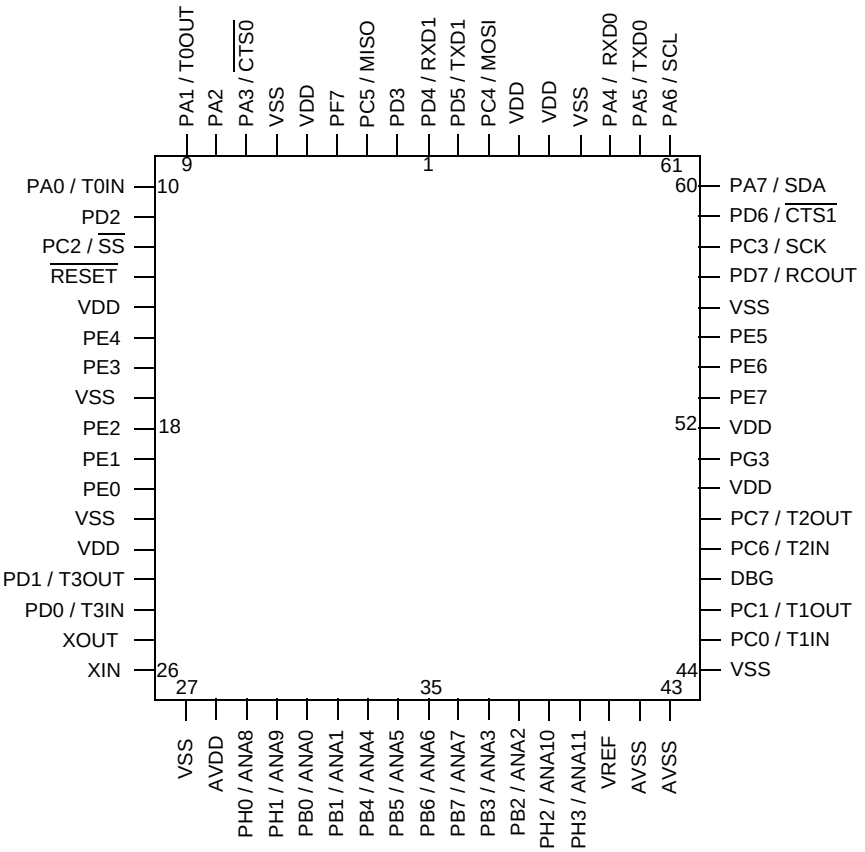

Figure 60. Z8Fxx02 in 68-Pin Plastic Leaded Chip Carrier (PLCC)

Program Memory

The eZ8 CPU supports 64KB of Program Memory address space. The Z8F640x family devices contain 16KB to 64KB of on-chip Flash memory in the Program Memory address space. Reading from Program Memory addresses outside the available Flash memory addresses returns FFH. Writing to these unemployments Program Memory addresses produces no effect. Table 4 describes the Program Memory Maps for the Z8F640x family products.

Table 4. Z8F640x Family Program Memory Maps

Program Memory Address (Hex)	Function
Z8F160x Products	
0000-0001	Flash Option Bits
0002-0003	Reset Vector
0004-0005	WDT Interrupt Vector
0006-0007	Illegal Instruction Trap
0008-0037	Interrupt Vectors*
0038-3FFFH	Program Memory
Z8F240x Products	
0000-0001	Flash Option Bits
0002-0003	Reset Vector
0004-0005	WDT Interrupt Vector
0006-0007	Illegal Instruction Trap
0008-0037	Interrupt Vectors*
0038-5FFFH	Program Memory
Z8F320x Products	
0000-0001	Flash Option Bits
0002-0003	Reset Vector
0004-0005	WDT Interrupt Vector
0006-0007	Illegal Instruction Trap
0008-0037	Interrupt Vectors*
0038-7FFFH	Program Memory

*** See Table 22 on page 45 for a list of the interrupt vectors.**

AF[7:0]—Port Alternate Function enabled
0 = The port pin is in normal mode and the DDx bit in the Port A-H 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-H Output Control Sub-Registers

The Port A-H Output Control sub-register (Table 17) is accessed through the Port A-H Control register by writing 03H to the Port A-H Address register. Setting the bits in the Port A-H 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 17. Port A-H Output Control Sub-Registers

BITS	7	6	5	4	3	2	1	0
FIELD	POC7	POC6	POC5	POC4	POC3	POC2	POC1	POC0
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	If 03H in Port A-H Address Register, accessible via Port A-H Control Register							

POC[7:0]—Port Output Control
These bits function independently of the alternate function bit and disables the drains if set to 1.
0 = The drains are enabled for any output mode.
1 = The drain of the associated pin is disabled (open-drain mode).

Port A-H Output Data Register

The Port A-H Output Data register (Table 21) writes output data to the pins.

Table 21. Port A-H Output Data Register (P*x*OUT)

BITS	7	6	5	4	3	2	1	0
FIELD	POUT7	POUT6	POUT5	POUT4	POUT3	POUT2	POUT1	POUT0
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	FD3H, FD7H, FDBH, FDFH, FE3H, FE7H, FEBH, FEFH							

POUT[7:0]—Port Output Data
These bits contain the data to be driven out from the port pins. The values are only driven if the corresponding pin is configured as an output and the pin is not configured for alternate function operation.
0 = Drive a logical 0 (Low).
1= Drive a logical 1 (High). High value is not driven if the drain has been disabled by setting the corresponding Port Output Control register bit to 1.

written during counting, the 8-bit written value is placed in the counter (High or Low Byte) at the next clock edge. The counter continues counting from the new value.

Table 38. Timer 0-3 High Byte Register (TxH)

BITS	7	6	5	4	3	2	1	0
FIELD	TH							
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	F00H, F08H, F10H, F18H							

Table 39>. Timer 0-3 Low Byte Register (TxL)

BITS	7	6	5	4	3	2	1	0
FIELD	TL							
RESET	0	0	0	0	0	0	0	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	F01H, F09H, F11H, F19H							

TH and TL—Timer High and Low Bytes
These 2 bytes, {TMRH[7:0], TMRL[7:0]}, contain the current 16-bit timer count value.

Timer Reload High and Low Byte Registers

The Timer 0-3 Reload High and Low Byte (TxRH and TxRL) registers (Tables 40 and 41) store a 16-bit reload value, {TRH[7:0], TRL[7:0]}. Values written to the Timer Reload High Byte register are stored in a temporary holding register. When a write to the Timer Reload Low Byte register occurs, the temporary holding register value is written to the Timer High Byte register. This operation allows simultaneous updates of the 16-bit Timer Reload value.

In Compare mode, the Timer Reload High and Low Byte registers store the 16-bit Compare value.

In single-byte DMA transactions to the Timer Reload High Byte register, the temporary holding register is bypassed and the value is written directly to the register. If the DMA is

Watch-Dog Timer Control Register Definitions

Watch-Dog Timer Control Register

The Watch-Dog Timer Control (WDTCTL) register, detailed in Table 46, is a Read-Only register that indicates the source of the most recent Reset event, indicates a Stop Mode Recovery event, and indicates a Watch-Dog Timer time-out. Reading this register resets the upper four bits to 0.

Writing the 55H, AAH unlock sequence to the Watch-Dog Timer Control (WDTCTL) register address 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.

Table 46. Watch-Dog Timer Control Register (WDTCTL)

BITS	7	6	5	4	3	2	1	0
FIELD	POR	STOP	WDT	EXT	Reserved			
RESET	X	X	X	0	0	0	0	0
R/W	R	R	R	R	R	R	R	R
ADDR	FF0							

POR—Power-On Reset Indicator
If this bit is set to 1, a Power-On Reset event occurred. This bit is reset to 0 if a WDT time-out or Stop Mode Recovery occurs. This bit is also reset to 0 when the register is read.

STOP—STOP Mode Recovery Indicator
If this bit is set to 1, a STOP Mode Recovery occurred. If the STOP and WDT bits are both set to 1, the STOP Mode Recovery occurred due to a WDT time-out. If the STOP bit is 1 and the WDT bit is 0, the STOP Mode Recovery was not caused by a WDT time-out. This bit is reset by a Power-On Reset or a WDT time-out that occurred while not in STOP mode. Reading this register also resets this bit.

WDT—Watch-Dog Timer Time-Out Indicator
If this bit is set to 1, a WDT time-out occurred. A Power-On Reset resets this pin. A Stop Mode Recovery from a change in an input pin also resets this bit. Reading this register resets this bit.

EXT—External Reset Indicator
If this bit is set to 1, a Reset initiated by the external $\overline{\text{RESET}}$ pin occurred. A Power-On Reset or a Stop Mode Recovery from a change in an input pin resets this bit. Reading this register resets this bit.

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

Transmitting Data using the Interrupt-Driven Method

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

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

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

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

SPI Baud Rate High and Low Byte Registers

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 proper SPI operation (maximum baud rate is system clock frequency divided by 4). The SPI baud rate is calculated using the following equation:

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

Table 64. SPI Baud Rate High Byte Register (SPIBRH)

BITS	7	6	5	4	3	2	1	0
FIELD	BRH							
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	F66H							

BRH = SPI Baud Rate High Byte
Most significant byte, BRG[15:8], of the SPI Baud Rate Generator's reload value.

Table 65. SPI Baud Rate Low Byte Register (SPIBRL)

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	F67H							

BRL = SPI Baud Rate Low Byte
Least significant byte, BRG[7:0], of the SPI Baud Rate Generator's reload value.

14. Software responds by setting the STOP bit of the I²C Control register.
15. If no new data is to be sent or address is to be sent, software responds by clearing the TXI bit of the I²C Control register.
16. The I²C Controller completes transmission of the data on the SDA signal.
17. The I²C Controller sends the STOP condition to the I²C bus.

Writing a Transaction with a 10-Bit Address

1. The I²C Controller shifts the I²C Shift register out onto SDA signal.
2. The I²C Controller waits for the slave to send an Acknowledge (by pulling the SDA signal Low). If the slave pulls the SDA signal High (Not-Acknowledge), the I²C Controller sends a Stop signal.
3. If the slave needs to service an interrupt, it pulls the SCL signal low, which halts I²C operation.
4. If there is no other data in the I²C Data register or the STOP bit in the I²C Control register is set by software, then the Stop signal is sent.

The data transfer format for a 10-bit addressed slave is illustrated in the figure below. Shaded regions indicate data transferred from the I²C Controller to slaves and unshaded regions indicate data transferred from the slaves to the I²C Controller.

Figure 80. 10-Bit Addressed Slave Data Transfer Format

The first seven bits transmitted in the first byte are 11110XX. The two bits XX are the two most-significant bits of the 10-bit address. The lowest bit of the first byte transferred is the write signal. The transmit operation is carried out in the same manner as 7-bit addressing.

The data transfer format for a transmit operation on a 10-bit addressed slave is as follows:

1. Software asserts the IEN bit in the I²C Control register.
2. Software asserts the TXI bit of the I²C Control register to enable Transmit interrupts.
3. The I²C interrupt asserts because the I²C Data register is empty.
4. Software responds to the TDRE bit by writing the first slave address byte. The least-significant bit must be 0 for the write operation.
5. Software asserts the START bit of the I²C Control register.
6. The I²C Controller sends the START condition to the I²C slave.

START—Send Start Condition
This bit sends the Start condition. Once asserted, it is cleared by the I2C Controller after it sends the START condition or by deasserting the IEN bit. After this bit is set, the Start condition is sent if there is data in the I2C Data or I2C Shift register. If there is no data in one of these registers, the I2C Controller waits until data is loaded. If this bit is set while the I2C Controller is shifting out data, it generates a START condition after the byte shifts and the acknowledge phase completed. If the STOP bit is also set, it also waits until the STOP condition is sent before the START condition. If this bit is 1, it cannot be cleared to 0 by writing to the register.This bit clears when the I2C is disabled.

STOP—Send Stop Condition
This bit causes the I2C Controller to issue a Stop condition after the byte in the I2C Shift register has completed transmission or after a byte has been received in a receive operation. Once set, this bit is reset by the I2C Controller after a Stop condition has been sent or by deasserting the IEN bit. If this bit is 1, it cannot be cleared to 0 by writing to the register.This bit clears when the I2C is disabled.

BIRQ—Baud Rate Generator Interrupt Request
This bit causes an interrupt to occur every time the baud rate generator counts down to zero. This bit allows the I2C Controller to be used as an additional counter when it is not being used elsewhere. This bit must only be set when the I2C Controller is disabled.

TXI—Enable TDRE interrupts
This bit enables interrupts when the I2C Data register is empty on the I2C Controller.

NAK—Send NAK
This bit sends a Not Acknowledge condition after the next byte of data has been read from the I2C slave. Once asserted, it is deasserted after a Not Acknowledge is sent or the IEN bit is deasserted.

FLUSH—Flush Data
Setting this bit to 1 clears the I2C Data register and sets the TDRE bit to 1. This bit allows flushing of the I2C Data register when an NAK is received after the data has been sent to the I2C Data register. Reading this bit always returns 0.

FILTEN—I2C Signal Filter Enable
Setting this bit to 1 enables low-pass digital filters on the SDA and SCL input signals. These filters reject any input pulse with periods less than a full system clock cycle. The filters introduce a 3-system clock cycle latency on the inputs.

On-Chip Debugger

Overview

The Z8F640x family devices have an integrated On-Chip Debugger (OCD) that provides advanced debugging features including:

- Reading and writing of the Register File
- Reading and writing of Program and Data Memory
- Setting of Breakpoints and Watchpoints
- Execution of eZ8 CPU instructions.

Architecture

The On-Chip Debugger consists of four primary functional blocks: transmitter, receiver, auto-baud generator, and debug controller. Figure 86 illustrates the architecture of the On-Chip Debugger

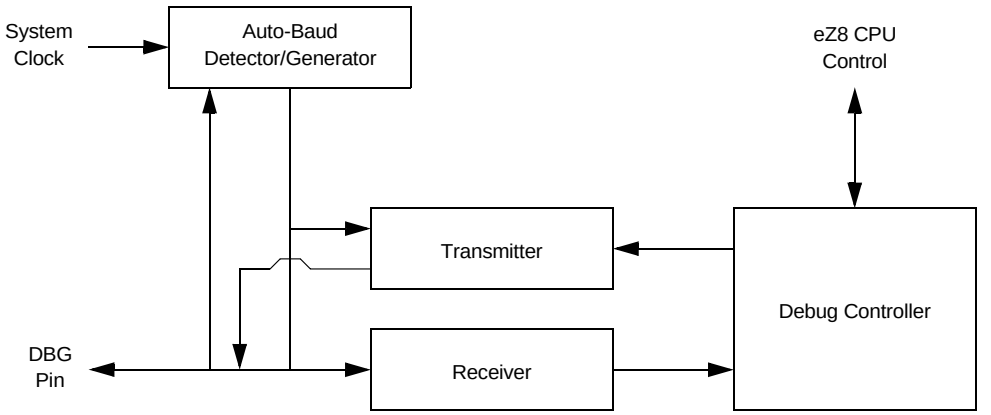

Figure 86. On-Chip Debugger Block Diagram

Table 101. DC Characteristics

Symbol	Parameter	T_A = -40°C to 105°C			Units	Conditions
		Minimum	Typical	Maximum		
I_{PU}	Weak Pull-up Current	30	100	350	µA	V_{DD} = 3.0 - 3.6V
I_{CCS}	Supply Current in Stop Mode		600		µA	V_{DD} = 3.3V

[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 91 illustrates the typical current consumption while operating at 25ºC, 3.3V, versus the system clock frequency.

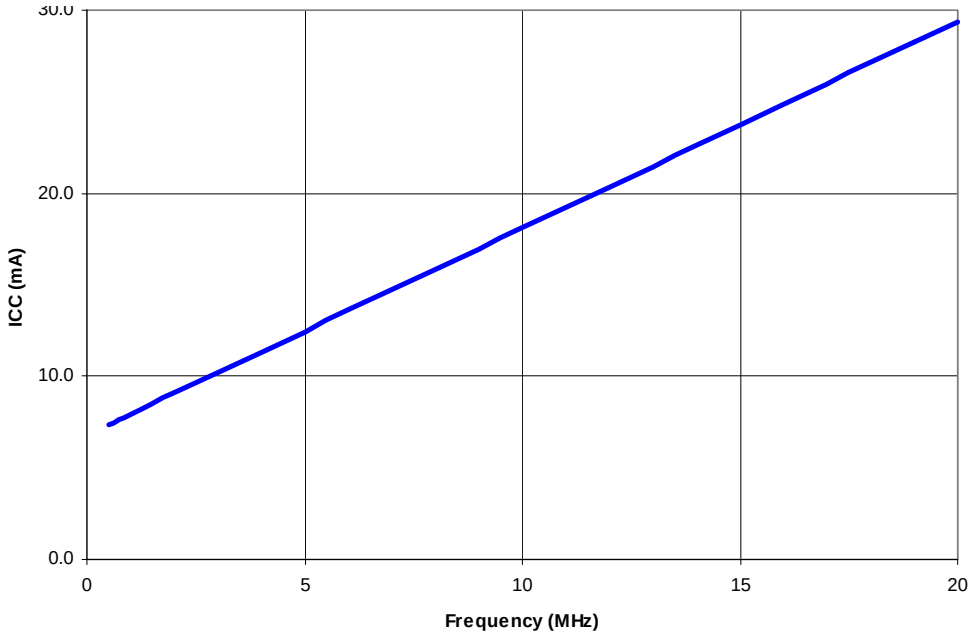

Figure 91. Nominal ICC Versus System Clock Frequency

eZ8 CPU Instruction Set

Assembly Language Programming Introduction

The eZ8 CPU assembly language provides a means for writing an application program without having to be concerned with actual memory addresses or machine instruction formats. A program written in assembly language is called a source program. Assembly language allows the use of symbolic addresses to identify memory locations. It also allows mnemonic codes (opcodes and operands) to represent the instructions themselves. The opcodes identify the instruction while the operands represent memory locations, registers, or immediate data values.

Each assembly language program consists of a series of symbolic commands called statements. Each statement can contain labels, operations, operands and comments.

Labels can be assigned to a particular instruction step in a source program. The label identifies that step in the program as an entry point for use by other instructions.

The assembly language also includes assembler directives that supplement the machine instruction. The assembler directives, or pseudo-ops, are not translated into a machine instruction. Rather, the pseudo-ops are interpreted as directives that control or assist the assembly process.

The source program is processed (assembled) by the assembler to obtain a machine language program called the object code. The object code is executed by the eZ8 CPU. An example segment of an assembly language program is detailed in the following example.

Assembly Language Source Program Example

```
JP START          ; Everything after the semicolon is a comment.

START:            ; A label called “START”. The first instruction (JP START) in this
                  ; example causes program execution to jump to the point within the
                  ; program where the START label occurs.

LD R4, R7         ; A Load (LD) instruction with two operands. The first operand,
                  ; Working Register R4, is the destination. The second operand,
                  ; Working Register R7, is the source. The contents of R7 is
                  ; written into R4.

LD 234H, #%01     ; Another Load (LD) instruction with two operands.
                  ; The first operand, Extended Mode Register Address 234H,
                  ; identifies the destination. The second operand, Immediate Data
```

Table 123. Logical Instructions

Mnemonic	Operands	Instruction
AND	dst, src	Logical AND
ANDX	dst, src	Logical AND using Extended Addressing
COM	dst	Complement
OR	dst, src	Logical OR
ORX	dst, src	Logical OR using Extended Addressing
XOR	dst, src	Logical Exclusive OR
XORX	dst, src	Logical Exclusive OR using Extended Addressing

Table 124. Program Control Instructions

Mnemonic	Operands	Instruction
BRK	—	On-Chip Debugger Break
BTJ	p, bit, src, DA	Bit Test and Jump
BTJNZ	bit, src, DA	Bit Test and Jump if Non-Zero
BTJZ	bit, src, DA	Bit Test and Jump if Zero
CALL	dst	Call Procedure
DJNZ	dst, src, RA	Decrement and Jump Non-Zero
IRET	—	Interrupt Return
JP	dst	Jump
JP cc	dst	Jump Conditional
JR	DA	Jump Relative
JR cc	DA	Jump Relative Conditional
RET	—	Return
TRAP	vector	Software Trap

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
ADD dst, src	dst ← dst + src	r	r	02	*	*	*	*	0	*	2	3
		r	Ir	03							2	4
		R	R	04							3	3
		R	IR	05							3	4
		R	IM	06							3	3
		IR	IM	07							3	4
ADDX dst, src	dst ← dst + src	ER	ER	08	*	*	*	*	0	*	4	3
		ER	IM	09							4	3
AND dst, src	dst ← dst AND src	r	r	52	-	*	*	0	-	-	2	3
		r	Ir	53							2	4
		R	R	54							3	3
		R	IR	55							3	4
		R	IM	56							3	3
		IR	IM	57							3	4
ANDX dst, src	dst ← dst AND src	ER	ER	58	-	*	*	0	-	-	4	3
		ER	IM	59							4	3
BCLR bit, dst	dst[bit] ← 0	r		E2	-	*	*	0	-	-	2	2
BIT p, bit, dst	dst[bit] ← p	r		E2	-	*	*	0	-	-	2	2
BRK	Debugger Break			00	-	-	-	-	-	-	1	1
BSET bit, dst	dst[bit] ← 1	r		E2	-	*	*	0	-	-	2	2
BSWAP dst	dst[7:0] ← dst[0:7]	R		D5	X	*	*	0	-	-	2	2
BTJ p, bit, src, dst	if src[bit] = p PC ← PC + X		r	F6	-	-	-	-	-	-	3	3
			Ir	F7							3	4
BTJNZ bit, src, dst	if src[bit] = 1 PC ← PC + X		r	F6	-	-	-	-	-	-	3	3
			Ir	F7							3	4

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

Table 127. Opcode Map Abbreviations

Abbreviation	Description	Abbreviation	Description
b	Bit position	IRR	Indirect Register Pair
cc	Condition code	p	Polarity (0 or 1)
X	8-bit signed index or displacement	r	4-bit Working Register
DA	Destination address	R	8-bit register
ER	Extended Addressing register	r1, R1, Ir1, Irr1, IR1, rr1, RR1, IRR1, ER1	Destination address
IM	Immediate data value	r2, R2, Ir2, Irr2, IR2, rr2, RR2, IRR2, ER2	Source address
Ir	Indirect Working Register	RA	Relative
IR	Indirect register	rr	Working Register Pair
Irr	Indirect Working Register Pair	RR	Register Pair

J

L

M

N

O