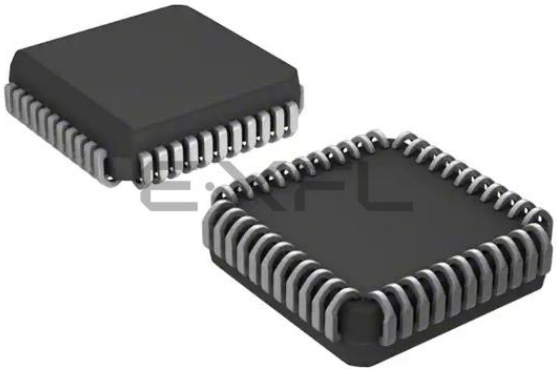


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

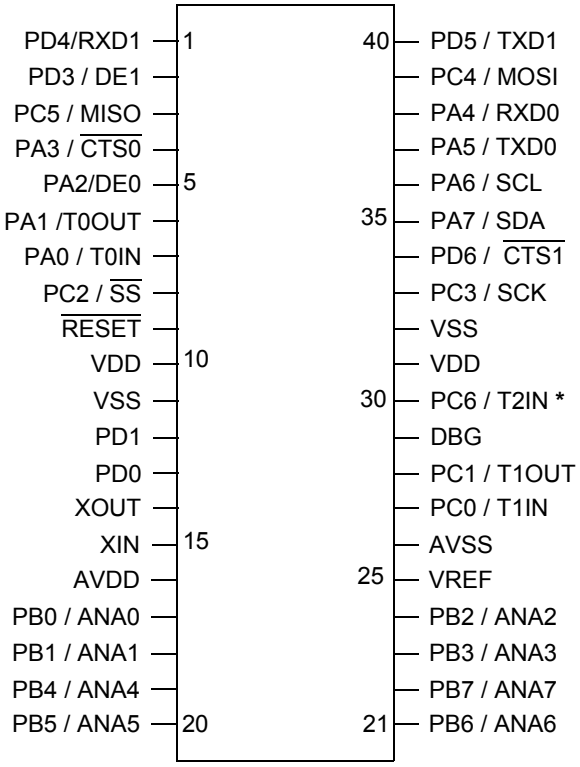
"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" 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	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 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/z8f2421vn020sc00tr

Pin Configurations

Figure 2 through Figure 7 on page 13 display the pin configurations for all of the packages available in the Z8 Encore! XP 64K Series Flash Microcontrollers. For description of the signals, see Table 3 on page 14. Timer 3 is not available in the 40-pin and 44-pin packages.



Note: Timer 3 is not supported.

* T2OUT is not supported.

Figure 2. Z8 Encore! XP 64K Series Flash Microcontrollers in 40-Pin Dual Inline Package (PDIP)

SPI Mode
SPIMODE (F63H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- D7: Reserved
- D6–D5: 111 = 7 bits
- D4: Diagnostic Mode Control
  0 = Reading from SPIBRH, SPIBRL returns reload values
  1 = Reading from SPIBRH, SPIBRL returns current BRG count value
- D3–D0: Reserved

SPI Diagnostic State
SPIDST (F64H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

- D7: Shift Clock Enable
  0 = Internal shift clock enable signal is deasserted
  1 = Internal shift clock enable signal is asserted
- D6: Transmit Clock Enable
  0 = Internal transmit clock enable signal is deasserted
  1 = Internal transmit clock enable signal is asserted
- D5–D0: SPI State

SPI Baud Rate Generator High Byte
SPIBRH (F66H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- D7–D0: SPI Baud Rate divisor [15:8]

SPI Baud Rate Generator Low Byte
SPIBRL (F67H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- D7–D0: SPI Baud Rate divisor [7:0]

ADC Control
ADCCTL (F70H - Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0

- D7: Conversion Enable
  0 = Conversion is complete
  1 = Begin conversion
- D6: Reserved
- D5: External VREF select
  0 = Internal voltage reference selected
  1 = External voltage reference selected
- D4: Continuous Mode Select
  0 = Single-shot conversion
  1 = Continuous conversion
- D3–D0: Analog Input Select
  0000 = ANA0 0001 = ANA1
  0010 = ANA2 0011 = ANA3
  0100 = ANA4 0101 = ANA5
  0110 = ANA6 0111 = ANA7
  1000 = ANA8 1001 = ANA9
  1010 = ANA10 1011 = ANA11
  11xx = Reserved

ADC Data High Byte
ADCD_H (F72H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

- D7–D0: ADC Data [9:2]

ADC Data Low Bits
ADCD_L (F73H - Read Only)

D7	D6	D5	D4	D3	D2	D1	D0

- D7–D6: ADC Data [1:0]
- D5–D0: Reserved

Table 11. 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
Z8X4823	80-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[3:0]
Z8X6421	40-pin	[7:0]	[7:0]	[6:0]	[6:3, 1:0]	-	-	-	-
Z8X6421	44-pin	[7:0]	[7:0]	[7:0]	[6:0]	-	-	-	-
Z8X6422	64- and 68-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7]	[3]	[3:0]
Z8X6423	80-pin	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[3:0]

Architecture

Figure 10 displays a simplified block diagram of a GPIO port pin. In Figure 10, the ability to accommodate alternate functions and variable port current drive strength are not illustrated.

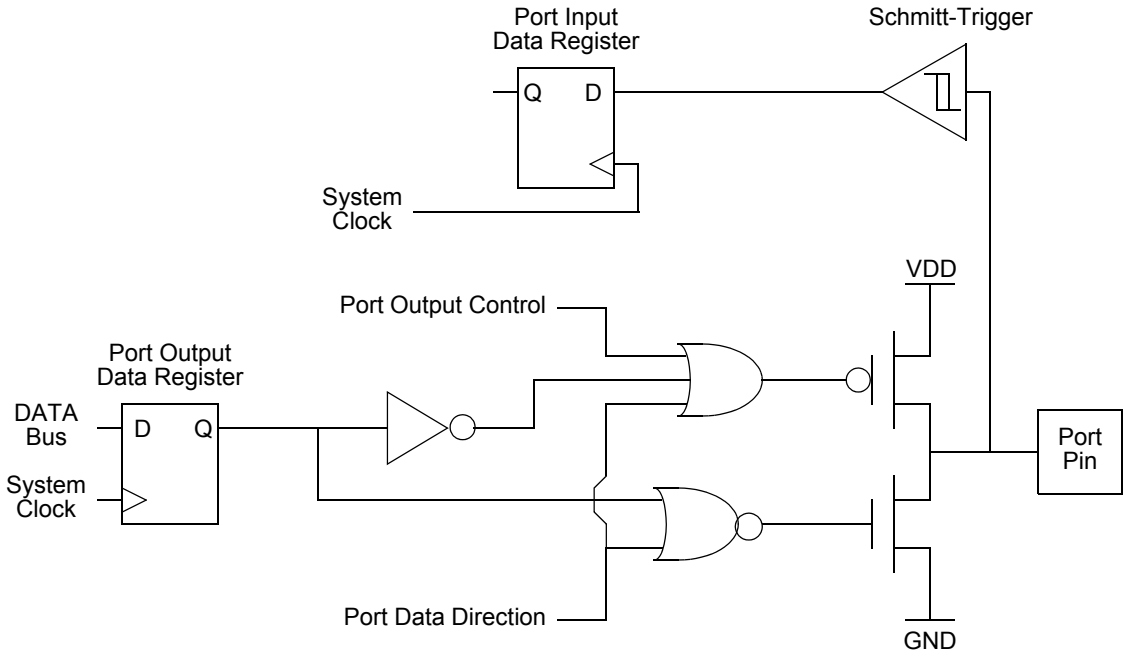


Figure 10. GPIO Port Pin Block Diagram

Interrupt Controller

Overview

The interrupt controller on the 64K Series products prioritizes the interrupt requests from the on-chip peripherals and the GPIO port pins. The features of the interrupt controller include the following:

- 24 unique interrupt vectors:
  - 12 GPIO port pin interrupt sources
  - 12 on-chip peripheral interrupt sources
- Flexible GPIO interrupts
  - Eight selectable rising and falling edge GPIO interrupts
  - Four dual-edge interrupts
- Three levels of individually programmable interrupt priority
- Watchdog Timer can be configured to generate an interrupt

Interrupt requests (IRQs) allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start an interrupt service routine (ISR). Usually this interrupt service routine is involved with the exchange of data, status information, or control information between the CPU and the interrupting peripheral. When the service routine is completed, the CPU returns to the operation from which it was interrupted.

The eZ8 CPU supports both vectored and polled interrupt handling. For polled interrupts, the interrupt control has no effect on operation. For more information on interrupt servicing by the eZ8 CPU, refer to *eZ8™ CPU Core User Manual (UM0128)* available for download at www.zilog.com.

Interrupt Vector Listing

Table 23 lists all of the interrupts available in order of priority. The interrupt vector is stored with the most-significant byte (MSB) at the even Program Memory address and the least-significant byte (LSB) at the following odd Program Memory address.

Timers

Overview

The 64K Series products contain up to four 16-bit reloadable timers that can be used for timing, event counting, or generation of pulse width modulated signals. The timers' features include:

- 16-bit reload counter
- Programmable prescaler with prescale values from 1 to 128
- PWM output generation
- Capture and compare capability
- External input pin for timer input, clock gating, or capture signal. External input pin signal frequency is limited to a maximum of one-fourth the system clock frequency.
- Timer output pin
- Timer interrupt

In addition to the timers described in this chapter, the Baud Rate Generators for any unused UART, SPI, or I^2C peripherals may also be used to provide basic timing functionality. For information on using the Baud Rate Generators as timers, see the respective serial communication peripheral. Timer 3 is unavailable in the 44-pin package devices.

Architecture

Figure 12 displays the architecture of the timers.

Watchdog Timer

Overview

The Watchdog Timer (WDT) helps protect against corrupt or unreliable software, power faults, and other system-level problems which may place the Z8 Encore! XP into unsuitable operating states. The features of Watchdog Timer include:

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

Operation

The Watchdog Timer (WDT) is a retriggerable one-shot timer that resets or interrupts the 64K Series devices when the WDT reaches its terminal count. The Watchdog Timer uses its own dedicated on-chip RC oscillator as its clock source. The Watchdog 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 Watchdog Timer to operate all the time, even if a WDT instruction has not been executed.

The Watchdog 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:

$$\text{WDT Time-out Period (ms)} = \frac{\text{WDT Reload Value}}{10}$$

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 Watchdog Timer RC oscillator frequency is 10 kHz. The Watchdog Timer cannot be refreshed once it reaches 000002H. The WDT Reload Value must not be set to values below 000004H. Table 47 provides information on approximate time-out delays for the minimum and maximum WDT reload values.

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

UART Receive Data Register

Data bytes received through the RXD*x* pin are stored in the UART Receive Data register (Table 53). The Read-only UART Receive Data register shares a Register File address with the Write-only UART Transmit Data register.

Table 53. UART Receive Data Register (U*x*RXD)

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

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

UART Status 0 Register

The UART Status 0 and Status 1 registers (Table 54 and Table 55 on page 117) identify the current UART operating configuration and status.

Table 54. UART Status 0 Register (U*x*STAT0)

BITS	7	6	5	4	3	2	1	0
FIELD	RDA	PE	OE	FE	BRKD	TDRE	TXE	CTS
RESET	0						1	X
R/W	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.

Serial Peripheral Interface

Overview

The Serial Peripheral Interface is a synchronous interface allowing several SPI-type devices to be interconnected. SPI-compatible devices include EEPROMs, Analog-to-Digital Converters, and ISDN devices. Features of the SPI include:

- Full-duplex, synchronous, character-oriented communication
- Four-wire interface
- Data transfers rates up to a maximum of one-half the system clock frequency
- Error detection
- Dedicated Baud Rate Generator

Architecture

The SPI may be configured as either a Master (in single or multi-master systems) or a Slave as displayed in Figure 22 through Figure 24.

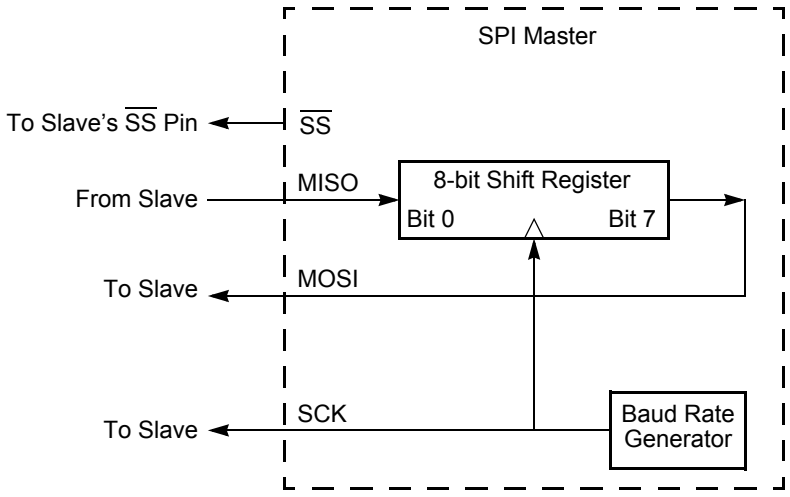


Figure 22. SPI Configured as a Master in a Single Master, Single Slave System

defined to be 1 through 8 bits by the NUMBITS field in the SPI Mode register. In slave mode it is not necessary for $\overline{SS}$ to deassert between characters to generate the interrupt. The SPI in Slave mode can also generate an interrupt if the $\overline{SS}$ signal deasserts prior to transfer of all the bits in a character (see description of slave abort error above). Writing a 1 to the IRQ bit in the SPI Status Register clears the pending SPI interrupt request. The IRQ bit must be cleared to 0 by the Interrupt Service Routine to generate future interrupts. To start the transfer process, an SPI interrupt may be forced by software writing a 1 to the STR bit in the SPICTL register.

If the SPI is disabled, an SPI interrupt can be generated by a Baud Rate Generator time-out. This timer function must be enabled by setting the BIRQ bit in the SPICTL register. This Baud Rate Generator time-out does not set the IRQ bit in the SPISTAT register, just the SPI interrupt bit in the interrupt controller.

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 SPI baud rate is calculated using the following equation:

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

Minimum baud rate is obtained by setting BRG[15:0] to 0000H for a clock divisor value of (2 X 65536 = 131072).

When the SPI is disabled, the Baud Rate Generator can function as a basic 16-bit timer with interrupt on time-out. Follow the steps below to configure the Baud Rate Generator as a timer with interrupt on time-out:

1. Disable the SPI by clearing the SPIEN bit in the SPI Control register to 0.
2. Load the desired 16-bit count value into the SPI Baud Rate High and Low Byte registers.
3. Enable the Baud Rate Generator timer function and associated interrupt by setting the BIRQ bit in the SPI Control register to 1.

When configured as a general purpose timer, the interrupt interval is calculated using the following equation:

$$\text{Interrupt Interval (s)} = \text{System Clock Period (s)} \times \text{BRG[15:0]}$$

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 baud rate generator. 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 baud rate generator 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 can control 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.*

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 read/write control bit (=0). The transmit operation is carried out in the same manner as 7-bit addressing.

Follow the steps below for a transmit operation on a 10-bit addressed slave:

1. Software asserts the IEN bit in the I^2C Control register.
2. Software asserts the TXI bit of the I^2C Control register to enable Transmit interrupts.
3. The I^2C interrupt asserts because the I^2C Data register is empty.
4. Software responds to the TDRE interrupt by writing the first slave address byte to the I^2C Data register. The least-significant bit must be 0 for the write operation.
5. Software asserts the START bit of the I^2C Control register.
6. The I^2C Controller sends the START condition to the I^2C slave.
7. The I^2C Controller loads the I^2C Shift register with the contents of the I^2C 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 I^2C Data register.
10. The I^2C Controller shifts the rest of the first byte of address and write bit out the SDA signal.
11. If the I^2C slave acknowledges the first address byte by pulling the SDA signal low during the next high period of SCL, the I^2C Controller sets the ACK bit in the I^2C Status register. Continue with step 12.

    If the slave does not acknowledge the first address byte, the I^2C Controller sets the NCKI bit and clears the ACK bit in the I^2C Status register. Software responds to the Not Acknowledge interrupt by setting the STOP and FLUSH bits and clearing the TXI bit. The I^2C Controller sends the STOP condition on the bus and clears the STOP and NCKI bits. The transaction is complete (ignore the following steps).
12. The I^2C Controller loads the I^2C Shift register with the contents of the I^2C Data register.
13. The I^2C Controller shifts the second address byte out the SDA signal. After the first bit has been sent, the Transmit interrupt is asserted.
14. Software responds by writing a data byte to the I^2C Data register.
15. The I^2C Controller completes shifting the contents of the shift register on the SDA signal.

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 (third address transfer).
17. The I2C Controller sends 11110B followed by the two most significant bits of the slave read address and a 1 (read).
18. The I2C slave sends an acknowledge by pulling the SDA signal Low during the next high period of SCL

    If the slave were to Not Acknowledge at this point (this should not happen because the slave did acknowledge the first two address bytes), software would respond by setting the STOP and FLUSH bits and clearing the TXI bit. The I2C Controller sends the STOP condition on the bus and clears the STOP and NCKI bits. The transaction is complete (ignore the following steps).
19. The I2C Controller shifts in a byte of data from the I2C slave on the SDA signal. The I2C Controller sends a Not Acknowledge to the I2C slave if the NAK bit is set (last byte), else it sends an Acknowledge.
20. The I2C Controller asserts the Receive interrupt (RDRF bit set in the Status register).
21. Software responds by reading the I2C Data register which clears the RDRF bit. If there is only one more byte to receive, set the NAK bit of the I2C Control register.
22. If there are one or more bytes to transfer, return to step 19.
23. After the last byte is shifted in, a Not Acknowledge interrupt is generated by the I2C Controller.
24. Software responds by setting the STOP bit of the I2C Control register.
25. A STOP condition is sent to the I2C slave and the STOP and NCKI bits are cleared.

I2C Control Register Definitions

I2C Data Register

The I2C Data register (see Table 70 on page 157) holds the data that is to be loaded into the I2C Shift register during a write to a slave. This register also holds data that is loaded from the I2C Shift register during a read from a slave. The I2C Shift Register is not accessible in the Register File address space, but is used only to buffer incoming and outgoing data.

Configuring DMA0 and DMA1 for Data Transfer

Follow the steps below to configure and enable DMA0 or DMA1:

1. Write to the DMA*x* 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, DMA*x*_IO[7:0]}.
2. Determine the 12-bit Start and End Register File addresses. The 12-bit Start Address is given by {DMA*x*_H[3:0], DMA_START[7:0]}. The 12-bit End Address is given by {DMA*x*_H[7:4], DMA_END[7:0]}.
3. Write the Start and End Register File address high nibbles to the DMA*x* End/Start Address High Nibble register.
4. Write the lower byte of the Start Address to the DMA*x* Start/Current Address register.
5. Write the lower byte of the End Address to the DMA*x* End Address register.
6. Write to the DMA*x* 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 DMA*x* interrupt request, if desired
   – Select Word or Byte mode
   – Select the DMA*x* request trigger
   – Enable the DMA*x* 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

On-Chip Debugger

Overview

The 64K Series products contain 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
- 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 36 displays the architecture of the On-Chip Debugger.

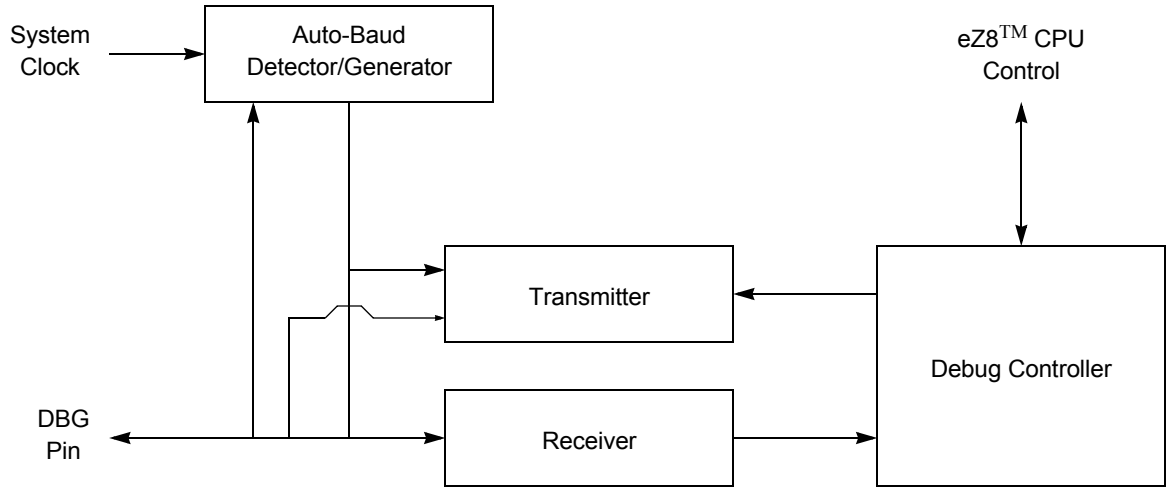


Figure 36. On-Chip Debugger Block Diagram

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 Serial Break 4096 system clock cycles long 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 transmits a Serial Break on the DBG pin when first connecting to the 64K Series devices 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 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 idles the eZ8 CPU and enters DEBUG mode. If Breakpoints are not enabled, the OCD ignores the BRK signal and the BRK instruction operates as an NOP.

If breakpoints are enabled, the OCD can be configured to automatically enter DEBUG mode, or to loop on the break instruction. If the OCD is configured to loop on the BRK instruction, then the CPU is still enabled to service DMA and interrupt requests.

The loop on BRK instruction can be used to service interrupts in the background. For interrupts to be serviced in the background, there cannot be any breakpoints in the interrupt service routine. Otherwise, the CPU stops on the breakpoint in the interrupt routine. For interrupts to be serviced in the background, interrupts must also be enabled. Debugging software should not automatically enable interrupts when using this feature, since interrupts are typically disabled during critical sections of code where interrupts should not occur (such as adjusting the stack pointer or modifying shared data).

Software can poll the IDLE bit of the OCDSTAT register to determine if the OCD is looping on a BRK instruction. When software wants to stop the CPU on the BRK instruction it is looping on, software should not set the DBGMODE bit of the OCDCTL register. The CPU may have vectored to and be in the middle of an interrupt service routine when this bit gets set. Instead, software must clear the BRKLP bit. This action allows the CPU to

Figure 57 and Table 121 provide timing information for UART pins for the case where the Clear To Send input signal ($\overline{CTS}$) is not used for flow control. In this example, it is assumed that the Driver Enable polarity has been configured to be Active Low and is represented here by $\overline{DE}$. $\overline{DE}$ asserts after the UART Transmit Data Register has been written. $\overline{DE}$ remains asserted for multiple characters as long as the Transmit Data register is written with the next character before the current character has completed.

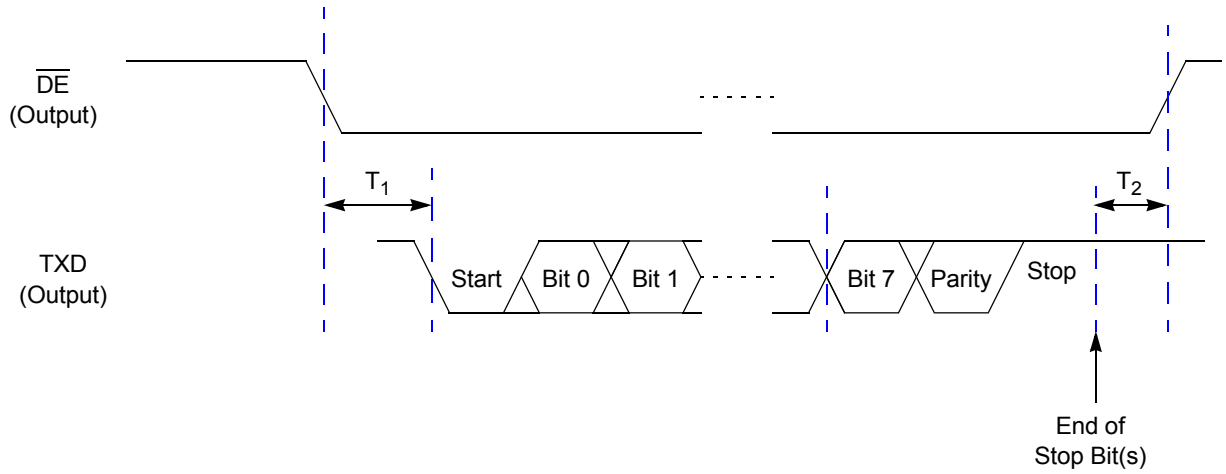


Figure 57. UART Timing without $\overline{CTS}$

Table 121. UART Timing without $\overline{CTS}$

		Delay (ns)	
Parameter	**Abbreviation**	**Minimum**	**Maximum**
T_1	$\overline{DE}$ Assertion to TXD Falling Edge (Start) Delay	1 Bit period	1 Bit period + 1 * XIN period
T_2	End of Stop Bit(s) to $\overline{DE}$ Deassertion Delay	1 * XIN period	2 * XIN period

Table 122. Notational Shorthand

Notation	Description	Operand	Range
b	Bit	b	b represents a value from 0 to 7 (000B to 111B).
cc	Condition Code	—	Refer to Condition Codes overview in the eZ8 CPU User Manual.
DA	Direct Address	Addrs	Addrs. represents a number in the range of 0000H to FFFFH
ER	Extended Addressing Register	Reg	Reg. represents a number in the range of 000H to FFFH
IM	Immediate Data	#Data	Data is a number between 00H to FFH
Ir	Indirect Working Register	@Rn	n = 0 –15
IR	Indirect Register	@Reg	Reg. represents a number in the range of 00H to FFH
Irr	Indirect Working Register Pair	@RRp	p = 0, 2, 4, 6, 8, 10, 12, or 14
IRR	Indirect Register Pair	@Reg	Reg. represents an even number in the range 00H to FEH
p	Polarity	p	Polarity is a single bit binary value of either 0B or 1B.
r	Working Register	Rn	n = 0 – 15
R	Register	Reg	Reg. represents a number in the range of 00H to FFH
RA	Relative Address	X	X represents an index in the range of +127 to -128 which is an offset relative to the address of the next instruction
rr	Working Register Pair	RRp	p = 0, 2, 4, 6, 8, 10, 12, or 14
RR	Register Pair	Reg	Reg. represents an even number in the range of 00H to FEH
Vector	Vector Address	Vector	Vector represents a number in the range of 00H to FFH
X	Indexed	#Index	The register or register pair to be indexed is offset by the signed Index value (#Index) in a +127 to -128 range.

Table 123 contains additional symbols that are used throughout the Instruction Summary and Instruction Set Description sections.

Table 132. Rotate and Shift Instructions (Continued)

Mnemonic	Operands	Instruction
SRL	dst	Shift Right Logical
SWAP	dst	Swap Nibbles

eZ8 CPU Instruction Summary

Table 133 summarizes the eZ8 CPU instructions. The table identifies the addressing modes employed by the instruction, the effect upon the Flags register, the number of CPU clock cycles required for the instruction fetch, and the number of CPU clock cycles required for the instruction execution.

Table 133. eZ8 CPU Instruction Summary

Assembly Mnemonic	Symbolic Operation	Address Mode dst	Address Mode src	Opcode(s) (Hex)	Flags C	Z	S	V	D	H	Fetch Cycles	Instr. Cycles
ADC dst, src	dst ← dst + src + C	r	r	12	*	*	*	*	0	*	2	3
		r	Ir	13							2	4
		R	R	14							3	3
		R	IR	15							3	4
		R	IM	16							3	3
		IR	IM	17							3	4
ADCX dst, src	dst ← dst + src + C	ER	ER	18	*	*	*	*	0	*	4	3
		ER	IM	19							4	3
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

Table 133. 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
LDC dst, src	dst ← src	r	Irr	C2	-	-	-	-	-	-	2	5
		Ir	Irr	C5							2	9
		Irr	r	D2							2	5
LDCI dst, src	dst ← src r ← r + 1 rr ← rr + 1	Ir	Irr	C3	-	-	-	-	-	-	2	9
		Irr	Ir	D3							2	9
LDE dst, src	dst ← src	r	Irr	82	-	-	-	-	-	-	2	5
		Irr	r	92							2	5
LDEI dst, src	dst ← src r ← r + 1 rr ← rr + 1	Ir	Irr	83	-	-	-	-	-	-	2	9
		Irr	Ir	93							2	9
LDWX dst, src	dst ← src	ER	ER	1F E8	-	-	-	-	-	-	5	4
LDX dst, src	dst ← src	r	ER	84	-	-	-	-	-	-	3	2
		Ir	ER	85							3	3
		R	IRR	86							3	4
		IR	IRR	87							3	5
		r	X(rr)	88							3	4
		X(rr)	r	89							3	4
		ER	r	94							3	2
		ER	Ir	95							3	3
		IRR	R	96							3	4
		IRR	IR	97							3	5
		ER	ER	E8							4	2
		ER	IM	E9							4	2
LEA dst, X(src)	dst ← src + X	r	X(r)	98	-	-	-	-	-	-	3	3
		rr	X(rr)	99							3	5
MULT dst	dst[15:0] ← dst[15:8] * dst[7:0]	RR		F4	-	-	-	-	-	-	2	8
NOP	No operation			0F	-	-	-	-	-	-	1	2

P

Q

R