eZ80190

eZ80 Webserver

Webserver Product Specification

PS006602-0101

ZiLOG Worldwide Headquarters • 910 E. Hamilton Avenue • Campbell, CA 95008
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

I^2C is a registered trademark of Phillips.

Document Disclaimer

Table of Contents

List of Figures

List of Tables

Webserver Architectural Overview

The eZ80 Webserver is a high speed, single cycle instruction fetch micro-controller operating at 48 MHz. It is the first part of a new set of products based around the eZ80 core and a Multiply Accumulator.

The eZ80 is one of the fastest 8-bit CPUs available today, executing code four times faster than a standard Z80 operating at the same clock speed. This increased processing efficiency can be used to improve available bandwidth or to decrease power consumption.

Considering both the increased clock speed and processor efficiency, the eZ80's processing power rivals the performance of 16-bit microprocessors.

Features

- Single Cycle Instruction Fetch, High Performance eZ80 CPU Core
- 16x16-bit Multiply and 40-bit Accumulate With 1K Dual Port SRAM
- Four Chip Selects with Individual Wait State Generators
- Six Counter Timers with Pre-scalars
- Watch Dog Timer
- 2 Channel DMA Controller
- 8K Bytes of High Speed SRAM
- 2 Universal ZiLOG Interface (UZI) Channels (I2C, SPI, UART)
- ZiLOG Debug Interface (ZDI)
- 32 Bits of General Purpose I/O
- On Chip Oscillator

Block Diagram

Figure 1 shows the block diagram of the eZ80 Webserver processor.

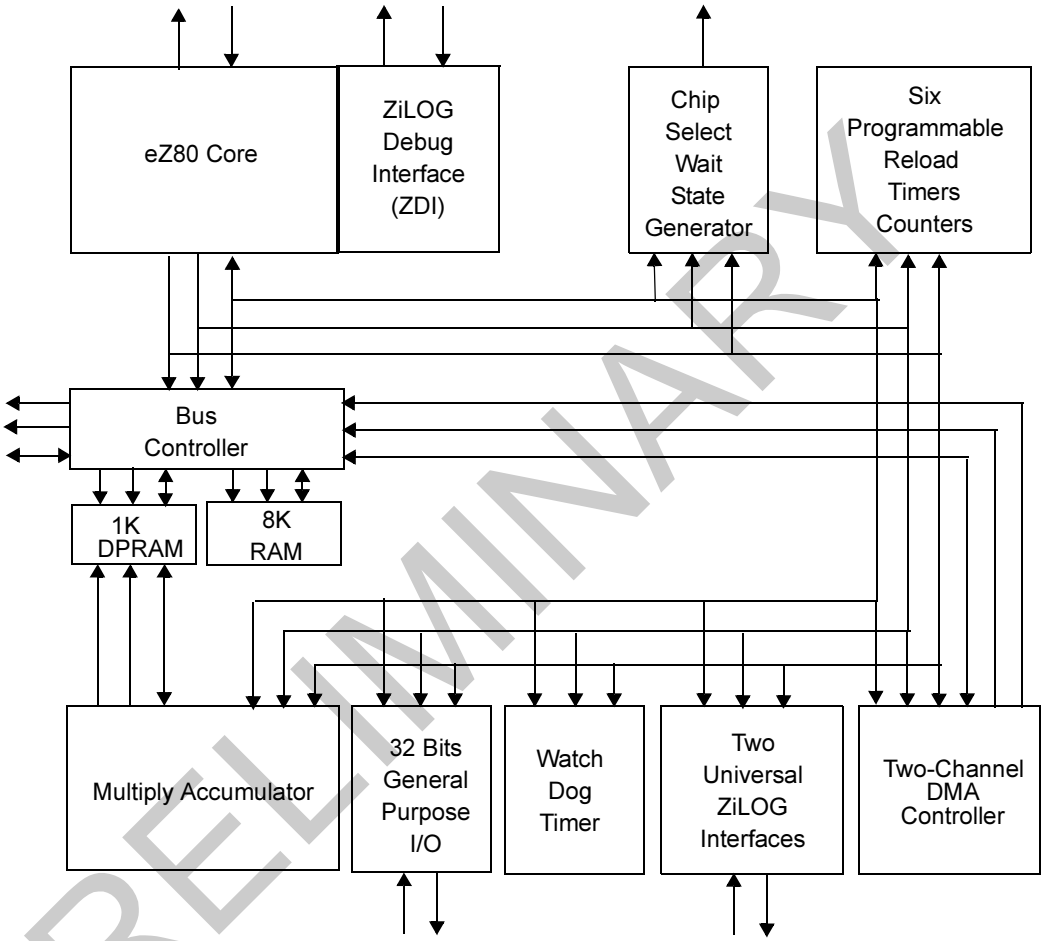


Figure 1. Block Level Diagram of eZ80 Webserver

Pin Description

Figure 2 shows the pin layout of the eZ80 Webserver in the 100 pin QFP package. Table 1 gives a description of the pins and their function.

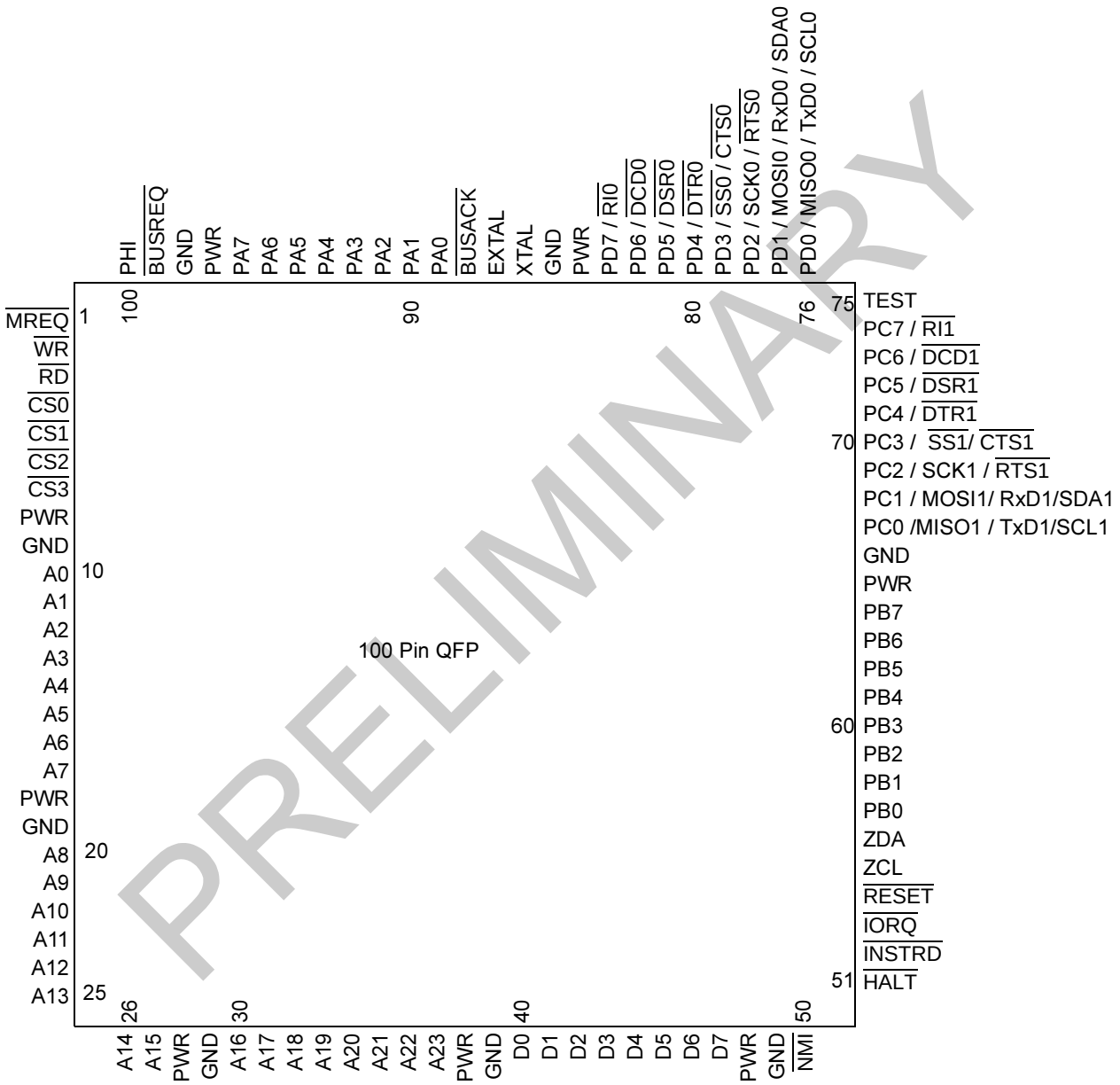


Figure 2. Pin Layout of eZ80 Webserver

Table 1. Processor Pin Signals

Symbol	Pin #	Function	Type	Description
ADR23:0	37-30, 27-20, 17-10.	Address Bus	Input/Output	These lines as output select a location in memory or I/O space to be read or written. These pins which are input during bus acknowledge cycles, drive CSWG block to generate Chip Selects.
$\overline{\text{BUSACK}}$	88	Bus Acknowledge	Output, Active Low	The eZ80 Webserver responds to a low on the $\overline{\text{BUSREQ}}$, by tristating the address, data and control signals and driving this line low. During Bus acknowledge cycles A23:0, $\overline{\text{IORQ}}$ and $\overline{\text{MREQ}}$ are inputs.
$\overline{\text{BUSREQ}}$	99	Bus Request	Input, Active Low	External devices can force the eZ80 Webserver to release the bus for their use, by driving this line low. To the internal CPU core, bus request signal can also originate from internal DMA controllers. In such cases, the DMA controllers have a higher priority compared to the request from an external bus master.
$\overline{\text{CTS1,0}}$	70,79	Clear to Send	Input, Active Low	Modem status signals to UART. These pins are multiplexed with PC3 and PD3.
D7:0	47-40	Data Bus	Bi-directional 3-state	These lines transfer information to and from I/O and memory devices. The eZ80 Webserver drives these lines only during write cycles in eZ80 Webserver bus master mode. During bus acknowledge cycle this bus is tristated.
$\overline{\text{DCD1,0}}$	73,82	Data Carrier Detect	Input, Active Low	Modem status signals to UART. These pins are multiplexed with PC6 and PD6.
$\overline{\text{DSR1,0}}$	72,81	Data Set Ready	Input, Active Low	Modem status signals to UART. These pins are multiplexed with PC5 and PD5.
$\overline{\text{DTR1,0}}$	71,80	Data Terminal Ready	Output, Active Low	Modem control signals to UART. These pins are multiplexed with PC4 and PD4.

Table 1. Processor Pin Signals (Continued)

Symbol	Pin #	Function	Type	Description
EXTAL	87	Oscillator Input	Input	This pin is the input to the onboard crystal oscillator. If an external oscillator is used, its clock output should be connected to this pin. When a crystal is used it should be connected between EXTAL and XTAL.
$\overline{\text{HALT}}$	51	Halt	Output, Active Low	A low on this pin indicates that the eZ80 Webserver has stopped because of a HALT instruction.
$\overline{\text{INSTRD}}$	52	Instruction Read	Output, Active Low, 3-state	$\overline{\text{INSTRD}}$ low (with $\overline{\text{MREQ}}$ and $\overline{\text{RD}}$ low) indicates that the eZ80 Webserver is fetching an instruction from code memory. The eZ80 Webserver does not drive this line during Reset or bus acknowledge cycles.
$\overline{\text{CS0}}$	4	Chip Select 0	Output, Active Low	$\overline{\text{CS0}}$ low indicates that an access is happening in the defined $\overline{\text{CS0}}$ memory or I/O address space. This signal is still driven during bus acknowledge cycles and is generated from the address and control provided on the external pins.
$\overline{\text{CS1}}$	5	Chip Select 1	Output, Active Low	$\overline{\text{CS1}}$ low indicates that an access is happening in the defined $\overline{\text{CS1}}$ memory or I/O address space. This signal is still driven during bus acknowledge cycles and is generated from the address and control provided on the external pins.
$\overline{\text{CS2}}$	6	Chip Select 2	Output, Active Low	$\overline{\text{CS2}}$ low indicates that an access is happening in the defined $\overline{\text{CS2}}$ memory or I/O address space. This signal is still driven during bus acknowledge cycles and is generated from the address and control provided on the external pins.
$\overline{\text{CS3}}$	7	Chip Select 3	Output, Active Low	$\overline{\text{CS3}}$ low indicates that an access is happening in the defined $\overline{\text{CS3}}$ memory or I/O address space. This signal is still driven during bus acknowledge cycles and is generated from the address and control provided on the external pins.

Table 1. Processor Pin Signals (Continued)

Symbol	Pin #	Function	Type	Description
$\overline{\text{IORQ}}$	53	Input/Output Request	Input/Output, Active Low	$\overline{\text{IORQ}}$ low indicates that the eZ80 Webserver is accessing a location in I/O space. $\overline{\text{RD}}$ and $\overline{\text{WR}}$ indicate the type of access. The eZ80 Webserver does not drive this line during Reset and is an input in bus acknowledge cycles.
MISO1,0	67,76	Master In Slave Out	Input/Output	The MISO line is configured as an input when eZ80 Webserver is a SPI master device and as an output when eZ80 Webserver is a SPI slave device. These pins are multiplexed with PC0 and PD0 respectively.
MOSI1,0	68,77	Master Out Slave In	Input/Output	The MOSI line is configured as an output when eZ80 Webserver is a SPI master device and as an input when eZ80 Webserver is a SPI slave device. These pins are multiplexed with PC1 and PD1.
$\overline{\text{MREQ}}$	1	Memory Request	Input/Output, Active Low	$\overline{\text{MREQ}}$ low indicates that the eZ80 Webserver is accessing a location in memory. The $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\overline{\text{INSTRD}}$ signals indicate the type of access. The eZ80 Webserver does not drive this line during Reset and is an input in bus acknowledge cycles.
$\overline{\text{NMI}}$	50	Non-Maskable Interrupt	Input, Active low	$\overline{\text{NMI}}$ has a higher priority than the maskable interrupt. It is always recognized at the end of an instruction, regardless of the state of the interrupt enable flip-flops. This signal forces processor execution to location 0066H. This input includes a Schmitt trigger to allow RC rise times. This signal is combined with internal $\overline{\text{NMI}}$ signal generated from WDT block, before being connected $\overline{\text{NMI}}$ of CPU core.

Table 1. Processor Pin Signals (Continued)

Symbol	Pin #	Function	Type	Description
PA7-0	96-89	GPIO	Input/Output	These pins can be used for general purpose I/O. They can be individually programmed as input or output. Each pin can also be used individually as an interrupt input. Each pin of the port when programmed as output can be selected to be an open drain or open source output.
PB7-0	64-57	GPIO	Input/Output	These pins can be used for general purpose I/O. They can be individually programmed as input or output. Each pin can also be used individually as an interrupt input. Each pin of the port when programmed as output can be selected to be an open drain or open source output.
PC7-0	74-67	GPIO	Input/Output	These pins can be used for general purpose I/O. They can be individually programmed as input or output. Each pin can also be used individually as an interrupt input. Each pin of the port when programmed as output can be selected to be an open drain or open source output. This port is multiplexed with one channel of the UZI interface.
PD7-0	83-76	GPIO	Input/Output	These pins can be used for general purpose I/O. They can be individually programmed as input or output. Each pin can also be used individually as an interrupt input. Each pin of the port when programmed as output can be selected to be an open drain or open source output. This port is multiplexed with one channel of the UZI interface.
PHI	100	System Clock	Output	This pin is an output driven by internal system clock. It can be used by the system for synchronization with the eZ80 Webserver.
$\overline{RD}$	3	Read	Output, Active Low	$\overline{RD}$ low indicates that the eZ80 Webserver is reading from the current Address location. This pin is tristated during bus acknowledge cycles.

Table 1. Processor Pin Signals (Continued)

Symbol	Pin #	Function	Type	Description
$\overline{\text{RESET}}$	54	Reset	Input, Active Low	This signal is used to initialize the eZ80 Webserver. This input must be low for a minimum of 3 Cycles, and must be held low until the clock is stable. This input includes a Schmitt trigger to allow RC rise times.
$\overline{\text{RI1,0}}$	74,83	Ring Indicator	Input, Active Low	Modem status signals to UART. These pins are multiplexed with PC7 and PD7.
$\overline{\text{RTS1,0}}$	69,78	Request to Send	Output, Active Low	Modem control signals from UART. These pins are multiplexed with PC2 and PD2.
RxD1,0	68,77	Receive Data	Input	These pins are used to receive data in asynchronous format to UART. These pins are multiplexed with PC1 and PD1.
SCK1,0	69,78	SPI Serial Clock	Input/Output	The SCK is the SPI serial clock. These pins are multiplexed with PC2 and PD2.
SCL1,0	67,76	I2C Serial Clock	Input/Output	These pins are used to receive and transmit the I2C clock. These pins are multiplexed with PC0 and PD0.
SDA1,0	68,77	I2C Serial Data	Input/Output	These pins carry the I2C data transceiver signals. These pins are multiplexed with PC1 and PD1.
$\overline{\text{SS1,0}}$	70,79	Slave Select	Input, Active Low	The slave select input line is used to select a slave device in SPI mode. These pins are multiplexed with PC3 and PD3.
$\overline{\text{TxD1,0}}$	67,76	Transmit Data	Output	These pins are used to transmit serial data in a asynchronous format from UART. These pins are multiplexed with PC0 and PD0.
$\overline{\text{WR}}$	2	Write	Output, Active Low	$\overline{\text{WR}}$ low indicates that the eZ80 Webserver is writing to the current Address location. The device accessed is determined by the $\overline{\text{IORQ}}$ and $\overline{\text{MREQ}}$ pins. This pin is tristated during bus acknowledge cycles.

Table 1. Processor Pin Signals (Continued)

Symbol	Pin #	Function	Type	Description
XTAL	86	Oscillator Output	Output	This pin is the output of the onboard crystal oscillator. When used, a crystal should be connected between EXTAL and XTAL.
ZDA	56	ZDI Data	Input/Output Open Drain with Pull-up	This pin is used to transfer data between the ZPAK and the internal eZ80 Webserver. This pin is open drain and has an internal pull-up.
ZCL	55	ZDI Clock	Input/Output Open Drain with Pull-up	This pin is used to clock the data between the ZPAK and the internal eZ80 Webserver. This pin is open drain and has an internal pull-up.
TEST	75	Test	Input, Active High	This pin places the chip in test mode. This is used only for factory testing. This signal should be tied low for normal operation in a system.

Register Map

All on-chip peripheral registers are accessed in the I/O address space. All I/O accesses in the address range 80 to FFh are routed to internal peripherals. External chip selects are not recognized if the address space programmed for the chip selects overlap the 80 to FFh address range.

Registers at unused addresses are not implemented. Read accesses to such addresses return unpredictable values and write accesses have no effect.

Table 2 shows the register map for the eZ80 Webserver.

Table 2. Register Map

Address (hex)	Mnemonic	Name	Reset (hex)	CPU Access
Programmable Reload Counter/Timer Block				
80	TMR_CTL0	Timer 0 Control Register	00	R/W
81	TMR_DRL0	Timer 0 Data Low Register	00	R
82	TMR_DRH0	Timer 0 Data High Register	00	R
81	TMR_RRL0	Timer 0 Low Reload Register	00	W
82	TMR_RRH0	Timer 0 High Reload Register	00	W
83	TMR_CTL1	Timer 1 Control Register	00	R/W
84	TMR_DRL1	Timer 1 Data Low Register	00	R
85	TMR_DRH1	Timer 1 Data High Register	00	R
84	TMR_RRL1	Timer 1 Low Reload Register	00	W
85	TMR_RRH1	Timer 1 High Reload Register	00	W
86	TMR_CTL2	Timer 2 Control Register	00	R/W
87	TMR_DRL2	Timer 2 Data Low Register	00	R
88	TMR_DRH2	Timer 2 Data High Register	00	R
87	TMR_RRL2	Timer 2 Low Reload Register	00	W
88	TMR_RRH2	Timer 2 High Reload Register	00	W
89	TMR_CTL3	Timer 3 Control Register	00	R/W
8A	TMR_DRL3	Timer 3 Data Low Register	00	R
8B	TMR_DRH3	Timer 3 Data High Register	00	R
8A	TMR_RRL3	Timer 3 Low Reload Register	00	W
8B	TMR_RRH3	Timer 3 High Reload Register	00	W

Table 2. Register Map (Continued)

Address (hex)	Mnemonic	Name	Reset (hex)	CPU Access
8C	TMR_CTL4	Timer 4 Control Register	00	R/W
8D	TMR_DRL4	Timer 4 Data Low Register	00	R
8E	TMR_DRH4	Timer 4 Data High Register	00	R
8D	TMR_RRL4	Timer 4 Low Reload Register	00	W
8E	TMR_RRH4	Timer 4 High Reload Register	00	W
8F	TMR_CTL5	Timer 5 Control Register	00	R/W
90	TMR_DRL5	Timer 5 Data Low Register	00	R
91	TMR_DRH5	Timer 5 Data High Register	00	R
90	TMR_RRL5	Timer 5 Low Reload Register	00	W
91	TMR_RRH5	Timer 5 High Reload Register	00	W
92	Not Accessible			
		Watch Dog Timer Block		
93	WDT_CTL	Watch Dog Timer Control Register	00/40#	R/W
94	WDT_RR	Watch Dog Timer Reset Register	XX	W
95	Not Accessible			
		General Purpose Input/Output Block		
96	PA_DR	Port A Data Register	XX	W*
97	PA_DDR	Port A Data Direction Register	FF	R/W
98	PA_ALT1	Port A Alternate Register 1	00	R/W
99	PA_ALT2	Port A Alternate Register 2	00	R/W
9A	PB_DR	Port B Data Register	XX	W*
9B	PB_DDR	Port B Data Direction Register	FF	R/W
9C	PB_ALT1	Port B Alternate Register 1	00	R/W
9D	PB_ALT2	Port B Alternate Register 2	00	R/W
9E	PC_DR	Port C Data Register	XX	W*
9F	PC_DDR	Port C Data Direction Register	FF	R/W
A0	PC_ALT1	Port C Alternate Register 1	00	R/W
A1	PC_ALT2	Port C Alternate Register 2	00	R/W
A2	PD_DR	Port D Data Register	XX	W*

Table 2. Register Map (Continued)

Address (hex)	Mnemonic	Name	Reset (hex)	CPU Access
A3	PD_DDR	Port D Data Direction Register	FF	R/W
A4	PD_ALT1	Port D Alternate Register 1	00	R/W
A5	PD_ALT2	Port D Alternate Register 2	00	R/W
A6	Not Accessible			
A7	Not Accessible			
		Chip Select/Wait State Generator Block		
A8	CS_LBR0	Chip Select 0 Lower Bound Register	00	R/W
A9	CS_UBR0	Chip Select 0 Upper Bound Register	FF	R/W
AA	CS_CTL0	Chip Select 0 Control Register	E8	R/W
AB	CS_LBR1	Chip Select 1 Lower Bound Register	00	R/W
AC	CS_UBR1	Chip Select 1 Upper Bound Register	00	R/W
AD	CS_CTL1	Chip Select 1 Control Register	00	R/W
AE	CS_LBR2	Chip Select 2 Lower Bound Register	00	R/W
AF	CS_UBR2	Chip Select 2 Upper Bound Register	00	R/W
B0	CS_CTL2	Chip Select 2 Control Register	00	R/W
B1	CS_LBR3	Chip Select 3 Lower Bound Register	00	R/W
B2	CS_UBR3	Chip Select 3 Upper Bound Register	00	R/W
B3	CS_CTL3	Chip Select 3 Control Register	00	R/W
		RAM Control Block		
B4	RAM_CTL0	RAM Control Register 0	00	R/W
B5	RAM_CTL1	RAM Control Register 1	00	R/W
		Universal ZiLOG Interface (UZI) Block		
B6	SPI_CTL0	SPI 0 Control Register	04	R/W
B7	SPI_SR0	SPI 0 Status Register	00	R
B8	SPI_RBR0	SPI 0 Receive Buffer Register	XX	R
B8	SPI_TSR0	SPI 0 Transmit Shift Register	XX	W
B9	Not Accessible			
BA	SPI_CTL1	SPI 1 Control Register	04	R/W
BB	SPI_SR1	SPI 1 Status Register	00	R

Table 2. Register Map (Continued)

Address (hex)	Mnemonic	Name	Reset (hex)	CPU Access
BC	SPI_RBR1	SPI 1 Receive Buffer Register	XX	R
BC	SPI_TSR1	SPI 1 Transmit Shift Register	XX	W
BD	Not Accessible			
BE	Not Accessible			
BF	Not Accessible			
C0	UART_RBR0	UART 0 Receive Buffer Register	XX	R
C0	UART_THR0	UART 0 Transmit Holding Register	XX	W
C0	BRG_DLRL0	BRG 0 Divisor Latch Low Register	02	R/W
C1	BRG_DLRH0	BRG 0 Divisor Latch High Register	00	R/W
C1	UART_IER0	UART 0 Interrupt Enable Register	00	R/W
C2	UART_IIR0	UART 0 Interrupt Identification Register	01	R
C2	UART_FCTL0	UART 0 FIFO Control Register	00	W
C3	UART_LCTL0	UART 0 Line Control Register	00	R/W
C4	UART_MCTL0	UART 0 Modem Control Register	00	R/W
C5	UART_LSR0	UART 0 Line Status Register	60	R
C6	UART_MSR0	UART 0 Modem Status Register	X0	R
C7	UART_SPR0	UART 0 Scratch Pin Register	00	R/W
C8	I2C_SAR0	I2C 0 Slave Address Register	00	R/W
C9	I2C_XSAR0	I2C 0 Extended Slave Address Register	00	R/W
CA	I2C_DR0	I2C 0 DATA Register	00	R/W
CB	I2C_CTL0	I2C 0 Control Register	00	R/W
CC	I2C_SR0	I2C 0 Status Register	F8	R
CD	I2C_SRR0	I2C 0 Software Reset Register	XX	W
CE	Not Accessible			
CF	UZI_CTL0	UZI 0 Control Register	00	R/W
D0	BRG_DLRL1	BRG 1 Divisor Latch Low Register	02	R/W
D1	BRG_DLRH1	BRG 1 Divisor Latch High Register	00	R/W
D0	UART_RBR1	UART 1 Receive Buffer Register	XX	R
D0	UART_THR1	UART 1 Transmit Holding Register	XX	W

Table 2. Register Map (Continued)

Address (hex)	Mnemonic	Name	Reset (hex)	CPU Access
D1	UART_IER1	UART 1 Interrupt Enable Register	00	R/W
D2	UART_IIR1	UART1 Interrupt Identification Register	01	R
D2	UART_FCTL1	UART1 FIFO Control Register	00	W
D3	UART_LCTL1	UART 1 Line Control Register	00	R/W
D4	UART_MCTL1	UART 1 Modem Control Register	00	R/W
D5	UART_LSR1	UART 1 Line Status Register	60	R/W
D6	UART_MSR1	UART 1 Modem Status Register	XX	R/W
D7	UART_SPR1	UART 1 Scratch Pin Register	00	R/W
D8	I2C_SAR1	I2C 1 Slave Address Register	00	R/W
D9	I2C_XSAR1	I2C 1 Extended Slave Address Register	00	R/W
DA	I2C_DR1	I2C 1 DATA Register	00	R/W
DB	I2C_CTL1	I2C 1 Control Register	00	R/W
DC	I2C_SR1	I2C 1 Status Register	F8	R
DD	I2C_SRR1	I2C 1 Software Reset Register	XX	W
DE	Not Accessible			
DF	UZI_CTL1	UZI 1 Control Register	00	R/W
		Multiplier Accumulator Block		
E0	MAC_XSTART	Multiply Accumulator X Start Address Register	00	R/W
E1	MAC_XEND	Multiply Accumulator X End Address Register	00	R/W
E2	MAC_XRELOAD	Multiply Accumulator X Reload Register	00	R/W
E3	MAC_LENGTH	Multiply Accumulator Length Register	00	R/W
E4	MAC_YSTART	Multiply Accumulator Y Start Address Register	00	R/W
E5	MAC_YEND	Multiply Accumulator Y End Address Register	00	R/W
E6	MAC_YRELOAD	Multiply Accumulator Y Reload Register	00	R/W
E7	MAC_CTL	Multiply Accumulator Control Register	00	R/W
E8	MAC_AC0	Multiply Accumulator [7-0] Register	XX	R/W
E9	MAC_AC1	Multiply Accumulator [15-8] Register	XX	R/W
EA	MAC_AC2	Multiply Accumulator [23-16] Register	XX	R/W
EB	MAC_AC3	Multiply Accumulator [31-24] Register	XX	R/W

Table 2. Register Map (Continued)

Address (hex)	Mnemonic	Name	Reset (hex)	CPU Access
EC	MAC_AC4	Multiply Accumulator [39-32] Register	XX	R/W
ED	MAC_SR	Multiply Accumulator Status Register	XX	R/W
		DMA Controller Block		
EE	DMA_SARL0	DMA0 Source Address Low Byte Register	XX	R/W
EF	DMA_SARM0	DMA0 Source Address Middle Byte Register	XX	R/W
F0	DMA_SARH0	DMA0 Source Address High Byte Register	XX	R/W
F1	DMA_DARL0	DMA0 Destination Address Low Byte Register	XX	R/W
F2	DMA_DARM0	DMA0 Destination Address Middle Byte Register	XX	R/W
F3	DMA_DARH0	DMA0 Destination Address High Byte Register	XX	R/W
F4	DMA_BCL0	DMA0 Byte Count Low Byte Register	XX	R/W
F5	DMA_BCH0	DMA0 Byte Count High Byte Register	XX	R/W
F6	DMA_CTL0	DMA0 Control Register	00	R/W
F7	DMA_SARL0	DMA1 Source Address Low Byte Register	XX	R/W
F8	DMA_SARM1	DMA1 Source Address Middle Byte Register	XX	R/W
F9	DMA_SARH1	DMA1 Source Address High Byte Register	XX	R/W
FA	DMA_DARL1	DMA1 Destination Address Low Byte Register	XX	R/W
FB	DMA_DARM1	DMA1 Destination Address Middle Byte Register	XX	R/W
FC	DMA_DARH1	DMA1 Destination Address High Byte Register	XX	R/W
FD	DMA_BCL1	DMA1 Byte Count Low Byte Register	XX	R/W
FE	DMA_BCH1	DMA1 Byte Count High Byte Register	XX	R/W
FF	DMA_CTL1	DMA1 Control Register	00	R/W

Note:

- (#) indicates that after external pin reset the value is 00 and after WDT reset it is 40h.
- (*) indicates that when the CPU reads, the PIN value of that particular port is read.

eZ80 CPU Core

Description

The eZ80 is a single cycle fetch 8-bit micro-controller. Its instruction set is a super-set of the Z180. It is binary compatible to the ZiLOG Z80 or the Z180 micro-controller. The processor has also been enhanced with a 24-bit linear address bus. Internal registers have been increased to 24 bits to handle the additional address bits.

Features

- Z80/Z180 Binary Compatible Instruction Set
- 48 MHz Operation
- 24-Bit Linear Address Space
- Single Cycle Instruction Fetch
- Dual Stack Pointers for Extended and Native Mode
- 24-Bit Internal Registers (e.g. IX,IY,HL,DE,BC,SP,PC)
- ZDI Hooks
- NMI + 128 Vectored Interrupts
- Additional Instructions to Work With the Multiply Accumulator

Programmable Reload Counter/Timers (PRT)

The eZ80 Webserver has six Programmable Reload Counter Timers (PRT). Each timer is a 16-bit down-counter and has a 4-bit clock pre-scaler with four selectable taps for CLK/2, CLK/4, CLK/8 and CLK/16. The timers' two modes of operation are single-pass and continuous count mode. The timer can be programmed to start, stop, restart to continue, or restart from the initial value.

An interrupt is generated whenever a timer reaches the end-of-count. This interrupt is disabled using the Timer Control Register. The timer can be configured to stop or continue during a CPU breakpoint. However, this particular feature is disabled in the current version of the eZ80 Webserver.

The CPU reads the current value of the counter while the timer is running. Minimum duration of the counter is achieved by loading `0001H` and maximum duration is achieved by loading `0000H`.

A programmable reload counter timer is controlled using five 8-bit registers. These registers are the Timer Control Register, Timer Low Reload Register, Timer High Reload Register, Timer Data Low Register and Timer Data High Register.

The Timer Control Register can be read or written to. The timer reload registers are write only and are at the same address as the timer data registers, which are read only.

Timer Block Diagram

Figure 3 shows the block diagram for the timer.

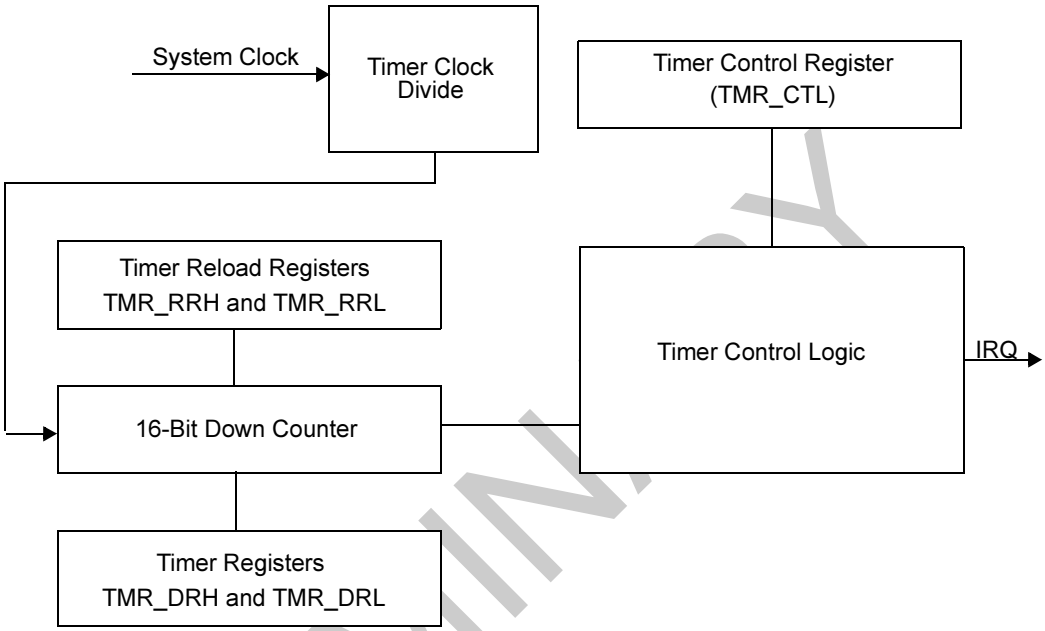


Figure 3. Timer Block Diagram

Timer Registers

Timer Control Register (TMR_CTL)

(CPU: Read / Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	Tmr IRQ	IRQ En	BRK CTL	Mult/Sing	CLKDiv1	CLKDiv0	LDRST	Timer En
Reset	0	0	0	0	0	0	0	0
CPU access	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The Timer Control Register is used to enable and control the operation of the timer, control the timeout value and define the software interface (interrupt or polling) to timer function.

Bit 7 Tmr IRQ - A read only bit and denotes the pending interrupt request. This bit is 1 when the timer reaches end-of-count. The bit is cleared when the CPU finishes reading this register.

Bit 6 IRQ En - This bit is used to enable or disable the timer interrupt. The interrupt request bit is 1 on the control register regardless of the status of this bit. This bit controls the only interrupt signal that goes to the CPU.

Bit 5 BRK CTL - This bit determines whether or not the timer counts during CPU breakpoints. If the timer is programmed to stop during breaks, an active high break signal stops the counter. The timer restarts when the break signal goes inactive. If this bit is 1, the timer stops the counter during breaks.

Bit 4 Mult/Sing - This bit selects the timer to either function in single-pass or continuous count mode. During a single-pass mode, the end-of-count results in clearing the timer enable bit (Bit 0). To restart the timer the CPU has to write a 1 to the timer enable bit (Bit 0).

Bit 2&3 CLKDiv1/CLKDiv0 - These bits are used to select the divide by ratio for the timer clock. The divide down ratios available are Clk/2, Clk/4, Clk/8 and Clk/16. See Table 3 on the proper bit settings for the clock divide ratio.

Table 3.Clock Divide Ratios

CLKDIV1,CLKDIV0	Divide Ratio
00	Clk/2
01	Clk/4
10	Clk/8
11	Clk/16

Bit 1 LDRST - Used to load the data and restart. The data in the timer reload registers is loaded in the next clock edge. This bit is cleared after data is loaded to the counter. When both Bit 0 and Bit 1 are 1, Bit 1 takes precedence and any action on Bit 0 is delayed until after loading is over.

Bit 0 Timer En - This bit enables the timer. When this bit is 0, the timer stops counting.

Timer Data High Register (TMR_DRH)

(CPU: Read Only)

This register mirrors the high byte of current value of the counter. This register can be read while the timer is in operation. To read the 16-bit data of the current counter value, first read the Timer Data Low Register and then read the Timer Data High Register. The Timer Data High Register value is latched when a read of the Timer Data Low Register occurs.

Timer Data Low Register (TMR_DRL)

(CPU: Read Only)

This register mirrors the low byte of the current counter's value. This register can be read while the timer is in operation. To read the16-bit data of the current counter value, first read the Timer Data Low Register and then read the Timer Data High Register. The Timer Data High Register value is latched when a read of the Timer Data Low Register occurs.

Timer High Reload Register (TMR_RRH)

(CPU: Write Only)

The Timer High Reload Register stores the most significant bit (MSB) of the value to be reloaded into the timer on end of count (if in continuous mode) or when Bit 1 of the Timer Control Register is 1.

Timer Low Reload Register (TMR_RRL)

(CPU: Write Only)

The Timer Low Reload Register stores the least significant bit (LSB) of the value which is to be reloaded into the timer on end of count (if in continuous mode) or when Bit 1 of the Timer Control Register is 1.

Note: The timer data registers and timer reload registers exist in the same address space.

Timer Operation

Mode of Operation

The timer operates in either single-pass or continuous count mode. At the end-of-count, 0000H, counting either stops or the initial value is reloaded and counting continues. For single-pass mode, once the timer reaches the end of the count, the timer enable bit is cleared. To restart the timer the CPU has to has to write a 1 to the timer enable bit (Bit 0) in the Timer Control Register.

Timer Interrupt:

An interrupt request is generated whenever the timer reaches an end-of-count. The interrupt signal to the CPU can be masked through the timer interrupt enable bit in the Timer Control Register. An interrupt request bit is set in the Timer Control Register (Bit 7) and is cleared when the CPU finishes reading the control register.

Watch Dog Timer (WDT)

Features

- Four Programmable Time-out Periods: 2^{18}, 2^{22}, 2^{25}, 2^{27} Clock Cycles
- Time-out Occurred Status
- Time-out Generates RESET or Non-Maskable Interrupt

Block Diagram

Figure 4 shows the block diagram for the Watch Dog Timer.

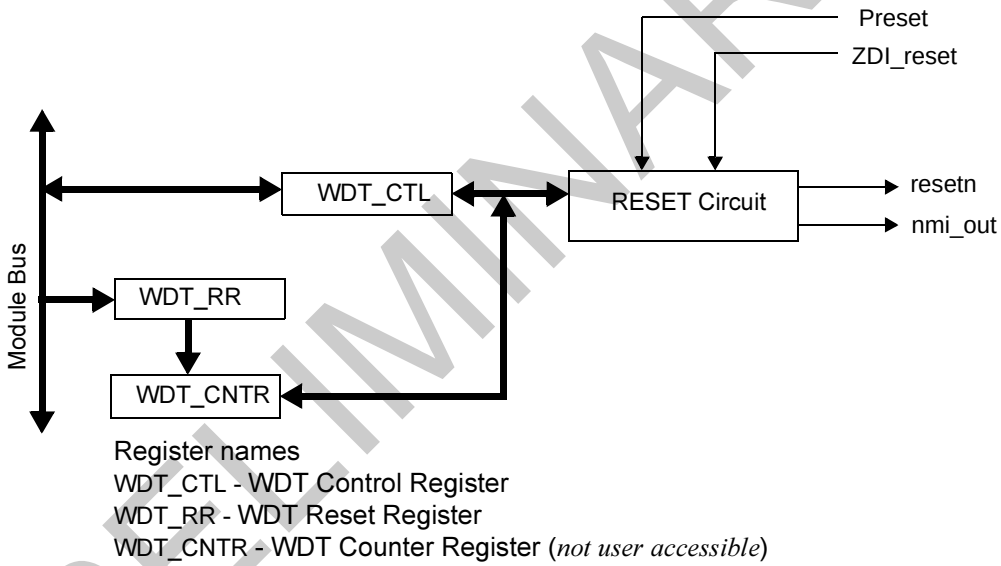


Figure 4. Watch Dog Timer Block Diagram

Watch Dog Timer Registers

Watch Dog Timer Control Register (WDT_CTL)

(CPU: Read / Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	WDTEN	NMIOUT	RESRC	Reserved	Reserved	Reserved	TOP1	TOP0
Reset	0	0	0/1	0	0	0	0	0
CPU access	R/W	R/W	R/W	R	R	R	R/W	R/W

The Watchdog Timer Control Register is an 8-bit read/write register used to enable the watchdog timer and select the time-out period.

Bit 7 WDTEN - Watchdog Timer Enable

0 - WDT is not enabled.

1- WDT is enabled. Once enabled the WDT cannot be disabled.

Bit 6 NMIOUT - Nonmaskable interrupt out

0 - Time-out generates CPU reset.

1 - Time-out generates NMI.

Bit 5 RESRC - Reset source

0 - Normal external chip reset and ZDI reset.

1 - Watch Dog timer reset.

Bits 4-2 Reserved.

Bits 1, 0 TOP1-0 - Time Out Period

Bits 1,0	Period
00	227 Clock Cycles
01	225 Clock Cycles
10	222 Clock Cycles
11	218 Clock Cycles

Watch Dog Timer Reset Register (WDT_RR)

(CPU: Write only)

The Watchdog Timer Reset Register is an 8-bit write only register. The watchdog timer is reset when an A5h value followed by 5Ah is written to this register.

Watch Dog Timer Operations

Initial Setting After System Reset

After a system reset, the Watch Dog Timer is disabled.

Normal Mode Operations

In normal mode, there are four choices of time-out periods as defined in the Time-out Periods field of the **WDT_CTL** register. The reset pulse generated by the Watch Dog Timer upon a timeout occurrence is 64 clock cycles wide. It asserts/de-asserts on the rising edge of the clock.

Based on the value programmed for the NMIOUT signal in the **WDT_CTL** register, the WDT timeout either resets the CPU or asserts an NMI for CPU processing. The RESRC bit (Bit 5) in the **WDT_CTL** register is used by software to determine if the reset was caused by the WDT's reset pin.

General Purpose Input/Output (GPIO)

The eZ80 Webserver has 32 bits of General Purpose Input or Output. All port signals can be used in either the Input or Output mode of operation. All of the port bits can be used as vectored interrupt sources.

GPIO Operation

The GPIO operation is the same for all GPIO ports (Port A to Port D). Every port has eight GPIO port pins. Each pin is controlled by four bits that are divided between four 8-bit registers. These registers are the Port Data Register, Port Data Direction Register, Port Alternate 1 Register and Port Alternate 2 Register. Set the pin mode by setting each register bit that pertains to the pin to be configured.

The combination of these register bits allow you to individually configure each port pin for nine modes. In all modes, reading of the Port Data Register returns the state/level of the signal on the corresponding pin. Table 4 shows the function of each port signal based upon these four register bits.

NOTE: Any read of the data register returns the value of the pins. The value returned is independent of the pins' mode.

Table 4. GPIO Pin Control

Mode	ALT2 Register	ALT1 Register	DDR Register	DATA Register	Port Mode	Output
1	0	0	0	0	Output a 0	Totem Pole
	0	0	0	1	Output a 1	Totem Pole
2	0	0	1	0	Input From Pin	High Z
	0	0	1	1	Input From Pin	High Z
3	0	1	0	0	Output a 0 (OD)	Open Drain
	0	1	0	1	I/O Open Drain	High Z
4	0	1	1	0	I/O Open Source	High Z
	0	1	1	1	Output a 1(OS)	Open Source
5	1	0	0	0	Reserved	High Z
6	1	0	0	1	Interrupt Dual Edge Triggered	High Z
7	1	0	1	0	Alternate Function	Alternate Function
	1	0	1	1	Alternate Function	Alternate Function

Table 4. GPIO Pin Control (Continued)

Mode	ALT2 Register	ALT1 Register	DDR Register	DATA Register	Port Mode	Output
8	1	1	0	0	Interrupt Active Low	High Z
	1	1	0	1	Interrupt Active High	High Z
9	1	1	1	0	Interrupt Low Edge Triggered	High Z
	1	1	1	1	Interrupt High Edge Triggered	High Z

Mode 1 The bit is in output mode. The value written to the Port Data Register is presented on the pin.

Mode 2 The bit is in input mode. The value stored in the Port Data Register has no effect. A read from the Port Data Register will return the pin's value.

Mode 3 The bit is in open drain mode. Writing a 0 to the data register outputs a low at the pin. Writing a 1 to the Port Data Register places the pin in High-Z.

Mode 4 The bit is in open source mode. Writing a 1 to the data register outputs a high at the pin. Writing a 0 to the data register places the pin in High-Z.

Mode 5 Reserved. Pin is High-Z.

Mode 6 The bit is in dual edge interrupt mode. Both a positive and a negative edge causes an interrupt request. Writing a 1 to the Port Data Register bit position resets the corresponding interrupt request. Writing a 0 will have no effect. Because writes to the data register are redefined in this mode, the data register must have a value of 1 before entering the edge interrupt mode.

Mode 7 Alternate function mode. The Source of pin output data and pin tri-state control come from alternate function's data output and tristate control respectively. The value in the Port Data Register has no effect.

Mode 8 Level sensitive interrupt mode. An interrupt request is generated when the level at the pin is the same as the level stored in the Port Data Register. The interrupt request remains active as long as this condition is maintained at the external source.

Mode 9 The bit is in edge interrupt mode. The value in the Port Data Register determines if a positive or negative edge causes an interrupt request. Write a 1 to the Port Data Register bit position to reset the corresponding interrupt request. Writing a 0 has no effect. Because writes to the data register are redefined in this mode, the Port Data Register must have a value of 1 before entering the edge interrupt mode.

GPIO Interrupts

Each port pin can be used as an interrupt pin. Interrupts can be either level or edge triggered.

Level Triggered Interrupts

When the port is configured for level triggered interrupts the corresponding port pin is tristated. For example, if a particular port pin is programmed for low level interrupt and the pin is forced low, an interrupt is generated from that port pin. The interrupt remains active until the external device supplying the interrupt forces the signal to a 1.

Edge Triggered Interrupts

When the port is configured for edge triggered interrupts the corresponding port pin is tristated. If the pin receives an appropriate edge, it generates an interrupt if no higher priority interrupt is pending. Anytime a port pin is configured for edge triggered interrupt, a write of a 1 to that pin's Port Data Register causes the edge detected interrupt to be reset. In addition, in edge triggered mode writes to the Port Data Register do not update the output data register. The programmer must set the Port Data Register before entering the edge triggered mode.

GPIO Rest Values

Table 5 lists the register contents after reset.

Table 5.GPIO Port Register Reset Values

Register	Reset Value
Port A Data Register	Undefined
Port A Data Direction Register	FFH
Port A Alternate Register 1	00H
Port A Alternate Register 2	00H
Port B Data Register	Undefined
Port B Data Direction Register	FFH
Port B Alternate Register 1	00H
Port B Alternate Register 2	00H
Port C Data Register	Undefined
Port C Data Direction Register	FFH
Port C Alternate Register 1	00H
Port C Alternate Register 2	00H
Port D Data Register	Undefined
Port D Data Direction Register	FFH
Port D Alternate Register 1	00H
Port D Alternate Register 2	00H

Chip Selects/Wait State Generator

The eZ80 Webserver generates four chip selects for external devices. Each chip select may be programed for either memory or I/O space. Each memory chip select can be individually programmed on a 64K boundary. The I/O chip selects can choose a 16-byte section of I/O space. Each chip select may be programmed for up to seven wait states.

Memory Chip Select Operation

Each memory chip select has three control registers. The eight high order address bits (A[23:16]) are compared to the values in the Chip Select Lower Bound Register (**CS_LBR**) and the Chip Select Upper Bound Register (CS_UBR). The chip select becomes active if the high order address is in the range specified, $\overline{\text{mreq}}$ is active and the chip select generation is enabled. The addresses indicated in the bound registers are inclusive.

The Chip Select Wait State Generator is active when (**CS_LBR**) <= A[23:16] <= (**CS_UBR**).

The 64K address space for the active chip select is chosen, when (**CS_LBR**) = (**CS_UBR**).

Memory Chip Select Priority

A lower numbered chip select has priority over a higher numbered chip select. If Chip Select 0's address space overlaps Chip Select 1's address space, Chip Select 0 is active.

If the address range programmed for any chip select signal overlaps with the address of the internal RAM, then RAM is given higher priority. When RAM is selected the particular chip select[s] which has an address range overlapping with RAM address, is not asserted.

Reset State

On reset Chip Select 0 is active for all addresses, as the corresponding registers for Chip Select 0 have equivalent default values.

Chip Select Registers

Chip Select Lower Bound Register (CS_LBR)

(CPU: Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	A23	A22	A21	A20	A19	A18	A17	A16
Reset	(See Table 7 on page 32 for reset values)							
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

This register defines the lower bound of the address range for which the corresponding chip select (if enabled) can be active.

Chip Select Upper Bound Register (CS_UBR)

(CPU: Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	A23	A22	A21	A20	A19	A18	A17	A16
Reset	(See Table 7 on page 32 for reset values)							
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

This register defines the upper bound of the address range for which the corresponding chip select (if enabled) can be active.

Chip Select Control Register (CS_CTL)

(CPU: Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	WAI2	WAI1	WAI0	MEMSEL	ENABLE	Reserved	Reserved	Reserved
Reset	(See Table 7 on page 32 for reset values)							
CPU access	R/W	R/W	R/W	R/W	R/W	N/A	N/A	N/A

This register controls the generation of the corresponding chip select signal.

Bits 7-5 **WAI2-0** - For bits 7-5 wait states, see Table 6 below.

Bit 4 **MEMSEL -** Memory select

0 = Memory CS

1 = I/O CS

Bit 3 **ENABLE**

0 = CS disabled

1 = CS Enabled

Bits 2-0 Reserved

Table 6.CS_CTL Wait States

Bits 7-5	Wait States
000	none
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Chip Select Registers' Reset Values

Table 7 shows the reset values for the chip select registers.

Table 7.Chip Select Register Reset Values

Register	Reset Value
CS_LBR0	00h
CS_UBR0	FFh
CS_CTL0	E8h
CS_LBR1	00h
CS_UBR1	00h
CS_CTL1	00h
CS_LBR2	00h
CS_UBR2	00h
CS_CTL2	00h
CS_LBR3	00h
CS_UBR3	00h
CS_CTL3	00h

I/O Chip Select Operation

The IOCS chip select logic decodes address lines 11:4, thus they are valid in all pages of I/O space. Bits 7-0 in the **CS_LBR** register are compared with addresses A[11:4]. If they match, $\overline{\text{IORQ}}$ signal is active, the chip select generation is enabled, and the $\overline{\text{CSN}}$ signal is asserted.

RAM

The eZ80 Webserver has 8K x 8 data RAM for general purpose and a 1K x 8 dual-port RAM for the Multiply Accumulator unit. Both RAMs can be individually enabled or disabled and can be relocated to the top of any 64K byte page.

The data RAM occupies address xxE000h - xxFFFFh, where xx is the value in the RAM_CTL1 register. The Multiply Accumulator dual-port RAM is at address range xxDC00h - xxDFFFh.

RAM Control

RAM Control Register 0 (RAM_CTL0)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	RAMEN	MRAM	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R	R	R	R	R	R

The internal RAMs, data RAM and Multiply Accumulator RAM, can be enabled by setting corresponding bits in this register.

Bit 7 **RAMEN** - RAM Enable, set to 0 on RESET

1 = RAM Enabled

0 = RAM Disabled

Bit 6 **MRAM** - Multiply Accumulator RAM Enable, set to 0 on RESET

1 = RAM Enabled

0 = RAM Disabled

Bits 5-0 Reserved, read as 0.

RAM Control Register 1 (RAM_CTL1)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The RA bits in this register define the 64K page where the internal RAM is located by matching to A[23:16] of the address bus. The internal RAM resides at the top of the selected page, provided the RAMEN bit is `1` in the **RAM_CTL0** register. For example if RA[7:0] = `0` and RAMEN = `1`; the RAM is accessed at `00E000h` through `00FFFFh`.

Internal RAM has priority over the chip selects. If the enabled RAM and chip select addresses overlap, the external chip select from Chip Select block is not asserted.

Universal ZiLOG Interface (UZI)

The Universal ZiLOG Interface (UZI) consists of four blocks. There are three serial communication controller blocks, SPI, UART, and I2C, along with the control registers and a Baud Rate Generator (BRG) which is common to all three serial devices. Only one of the serial devices is active at any time. The entire UZI port (including the Baud Rate Generator), is inactive when none of the serial devices are selected. Figure 5 shows the UZI block diagram.

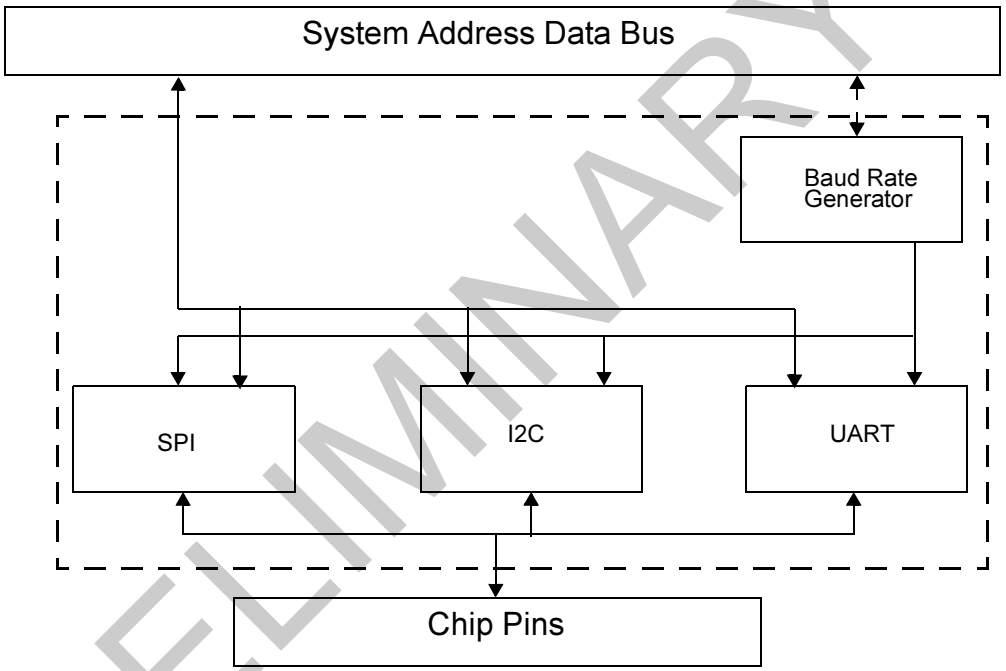


Figure 5. UZI Block Diagram

UZI Control Register (UZI_CTL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	M1	M0
Reset	0	0	0	0	0	0	0	0
CPU access	R	R	R	R	R	R	R/W	R/W

The UZI control register determines which of the three serial devices is active.

Bits 7-2 Reserved, read as 0.

Bits 1, 0 **M1-0** - Mode select bits are initialized to 0 at reset. These bits select the UZI mode.

00 - Standby

01 - UART

10 - SPI

11 - I2C

Baud Rate Generator (BRG)

The Baud Rate Generator is located outside the three serial communication blocks. The Baud Rate Generator output is used by the SPI, I2C and UART.

The Baud Rate Generator core is used to generate a lower frequency clock from a high frequency system clock provided at the input. This module consists of a 16-bit counter, two 8-bit pre-load registers and associated decoding logic.

Baud Rate Generator Functional Description

The Baud Rate Generator module implements two 8-bit divisor registers which can be read/written to by the application. It also consists of a 16-bit count down counter that counts down from the value loaded into the divisor registers to 0. Before the count reaches 0 the module generates one SYS_CLK wide pulse at the BDOUT output and reloads the count from the divisor latches. Loading a count of 1 or 0 in the divisor latches is prohibited. The count loaded in the divisor latches must be from 2 to 65535. The counter is either reloaded when the count reaches 0, or it is loaded when a processor write to the divisor registers is detected.

The divisor registers can only be accessed if the DLAB bit (bit 7 of the **UART_LCTL** register) is set to 1. After reset, this bit is set to 0.

Baud Rate Generator Registers

The BRG registers are described in the following sections.

BRG Divisor Latch Low Register (BRG_DLRL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	DIVL7	DIVL6	DIVL5	DIVL4	DIVL3	DIVL2	DIVL1	DIVL0
Reset	0	0	0	0	0	0	1	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

This register holds the lower byte of the 16-bit divisor count loaded by the processor for baud rate generation. Bit 7 of UART_LCTL register must be 1 to access this register, see UART Line Control Register (UART_LCTL) on page 38.

BRG Divisor Latch High Register (BRG_DLRH)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	DIVH7	DIVH6	DIVH5	DIVH4	DIVH3	DIVH2	DIVH1	DIVH0
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

This register holds the upper byte of 16-bit divisor count loaded by the processor for baud rate generation. Bit 7 of the UART_LCTL register must be 1 to access this register, see UART Line Control Register (UART_LCTL) on page 38.

UART Line Control Register (UART_LCTL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	DLAB	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Reset	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A
CPU access	R/W	N/A	N/A	N/A	N/A	N/A	N/A	N/A

This register is addressed at the same address as the actual register (UART_LCTL) in UART. However, the BRG only uses bit 7 of this register as a DLAB (divisor latch access bit). Set the DLAB bit to 1 to enable the processor to access the divisor registers.

BRG Design Considerations

Upon assertion of RESET, the DLAB (bit 7 of the UART_LCTL register) is set to 0, the BRG_DLRL register is set to 02h and the BRG_DLRH register is set to 00h as the default values and counter is loaded with 0002h.

The following is the normal sequence of operations that occur within the core after the system is powered on.

- RESET assertion and deassertion.
- Programming the DLAB for access of divisor registers.
- Programming the BRG_DLRL and BRG_DLRH registers.
- Programming the DLAB bit for locking access to divisor register.

The DLAB bit is programmed as 1 to access the divisor registers. The actual value is determined from the output clock frequency required by the enabled communication controller (UART, SPI or I2C) and the input SYS_CLK frequency. Once the divisor registers are programmed, the DLAB bit should be set to 0. A 16-bit divisor value of 0 or 1 is invalid and should not be programmed. The BDOUT output is undefined if a count of 0 or 1 is loaded into the divisor registers.

Universal Asynchronous Receiver Transmitter (UART)

The UART module implements all the logic required to support asynchronous communications protocol. The module also implements 16-byte deep FIFOs for both transmit and receive direction.

The UART module provides the following asynchronous communication protocol related features/functions.

- 5, 6, 7 or 8-bit data transmission.
- Even/odd or no parity bit generation and detection.
- Start and stop bit generation and detection (supports up to two stop bits)
- Line break detection and generation.
- Receiver overrun and framing errors detection.

In addition the module also implements logic and associated IOs to provide modem handshake capability. The core uses an externally provided clock from the Baud Rate Generator for the serial transmit/receive function.

Functional Description

The UART module supports all the various options in asynchronous transmission and reception protocol including:

- 5 to 8-bit transmit/receive
- Start bit generation and detection
- Parity generation and detection
- Stop bit generation and detection
- The module also supports break generation and detection

The UART contains 16-byte deep FIFOs in either direction. The FIFOs can be enabled or disabled by the application. The receive FIFO has trigger level detection logic, which enables the processor to block transfer data bytes from the receive FIFO. Similar logic for the transmit FIFO is not available.

Interrupts

There are four different sources of interrupts from the module. All four sources of the interrupt can be individually enabled or disabled from generating an interrupt to the processor. The four sources of interrupts are:

- transmitter
- receiver (two interrupts)
- modem status

The transmitter interrupt is generated if there is no data available for transmission. This interrupt can be disabled using the individual interrupt enable bit or cleared by writing data into the UART_THR register. The modem status interrupt is generated if there is any change in state of modem status inputs to the UART. This interrupt is cleared when the process reads the UART_MSR register.

A receiver interrupt can be generated by three possible events. The first event, a receiver data ready interrupt event, indicates that one or more data bytes were received and are ready to be read. If the FIFO is enabled, and the trigger level is set, then this interrupt is generated if the receiver FIFO has bytes equal to or more than the set trigger level. This interrupt is cleared by reading the UART_RBR.

The second interrupt source is the receiver timeout interrupt. A receiver timeout interrupt is generated when there are less data bytes in the receiver FIFO than there are in the trigger level. There are no reads and writes to or from the receiver FIFO for four consecutive byte times. Once the receiver timeout interrupt is generated, it is cleared only after emptying the entire receive FIFO.

For the above two interrupt sources from the receiver, there is only one interrupt enable bit. The third source of the receiver interrupt is an error in byte reception. This may result from:

- Received parity being incorrect
- Incorrect framing, i.e., the stop bit was not detected by receiver at the end of the byte
- Receiver over run condition
- A break condition being detected on the receive data input

An interrupt due to one of the above conditions is cleared once the UART_LSR register is read. In case of FIFO mode, a line status interrupt is generated only after the received byte with an error reaches the top of the FIFO and is ready to be read.

A line status interrupt is activated (provided this interrupt is enabled) as long as the read pointer of the receiver FIFO points to the location of the FIFO that contains a wrong byte. The interrupt is immediately cleared when the UART_LSR register is read. The ERR bit of the UART_LSR register is active as long as an erroneous byte is present in the receiver FIFO.

UART Function

The UART function implements:

- The transmitter and associated control logic
- The receiver and associated control logic
- The modem interface and associated logic

The transmitter block controls the data transmitted on the TXD output. It implements the FIFO, accessed through the UART_THR register, the transmit shift register, the parity generator, and control logic for the transmitter to control parameters for the asynchronous communication's protocol.

The UART_THR is a write only register of the module. The processor writes the data byte to be transmitted into this register. In the FIFO mode, up to 16 data bytes can be written through the UART_THR register. The data byte from FIFO is transferred to the transmit shift register at the appropriate time and transmitted out on TXD output. After SYNC_RESET, the UART_THR register is empty so the THRE bit (Bit 5 of **UART_LSR** register) is 1 and an interrupt is sent to the processor (if interrupts are enabled). The processor can reset this interrupt by loading data into UART_THR register, which disables the transmitter interrupt.

The transmit shift register places the byte to be transmitted on the TXD signal serially. The least significant bit of the byte to be transmitted is shifted out first and the most significant bit is shifted out last. The control logic within the block adds the asynchronous communication protocol bits to the data byte being transmitted. The transmitter block obtains the parameters for the protocol from the bits programmed through UART_LCTL register. The TXD output is set to 1 if the transmitter is idle, meaning it does not have any data to be transmitted.

The transmitter operates with the BDIN clock. The data bits are placed on the TXD output once every 16 BDIN clock cycles. The transmitter block also implements a parity generator and attaches the parity bit with the byte if programmed to do so.

The receiver block controls the data reception from the RXD signal. The receiver block implements a receiver shift register, receiver line error condition monitoring logic and receiver data ready logic. It also implements parity checker.

The UART_RBR is a read only register of the module. The processor reads received data from this register. The condition of the UART_RBR register is monitored by the DR bit (bit 0 of the UART_LSR register). The DR bit is 1 when a data byte is received and transferred to the UART_RBR register from the receiver shift register. The DR bit is reset only when the processor reads all the received data bytes. If the number of bits received is less than eight, the unused most significant bits of the data byte read are 0.

The receiver uses the clock from the BDIN input of the megacell for receiving the data. This clock must be 16 times the desired baud rate. The receiver synchronizes the shift clock on the falling edge of RXD input start bit. It then receives a complete byte according to the set parameters. The receiver also implements logic to detect framing error, parity error and break signal. It also implements logic to detect overrun error.

The modem control logic provides two outputs and four inputs for handshaking with the modem. Any change in the modem status inputs, except NRI, is detected and an interrupt can be generated. For NRI, an interrupt is generated only when the trailing edge of the NRI is detected. The module also provides loop mode for self diagnostics.

UART Registers

There are 10-byte command and status registers which are accessible to the application. After SYNC_RESET goes active, all registers are set to their default values. Any writes to unused registers or register bits are ignored and reads return a value of 0. For compatibility with future revisions, unused bits within a register should always be written with a value of 0. Read, write attributes of all the registers are provided in the table below.

UART Transmit Holding Register (UART_THR)

(CPU Write only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	TXD7	TXD6	TXD5	TXD4	TXD3	TXD2	TXD1	TXD0
Reset	0	0	0	0	0	0	0	0
CPU access	W	W	W	W	W	W	W	W

If less than eight bits are programmed for transmission, the lower bits of the byte written to this register are selected for transmission. The transmit FIFO is mapped at this address. The user can write up to 16 bytes for transmission at one time to

this address if the FIFO is enabled by the application. If the FIFO is disabled, this buffer is only one byte deep.

UART Receive Buffer Register (UART_RBR)

(CPU Read only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	RXD7	RXD6	RXD5	RXD4	RXD3	RXD2	RXD1	RXD0
Reset	0	0	0	0	0	0	0	0
CPU access	R	R	R	R	R	R	R	R

The bits in this register reflect the data received. If less than eight bits are programmed for receive, the lower bits of the byte reflect the bits received whereas upper unused bits are 0. The receive FIFO is mapped at this address. If the FIFO is disabled, this buffer is only one byte deep.

UART Interrupt Enable Register (UART_IER)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	Reserved	Reserved	Reserved	Reserved	MIE	LSIE	TIE	RIE
Reset	N/A	N/A	N/A	N/A	0	0	0	0
CPU access	N/A	N/A	N/A	N/A	R/W	R/W	R/W	R/W

This register is used to define and set interrupts for the UART.

Bits 7-4 Reserved

Bit 3 **MIE** - (Modem Interrupt Enable) Set this bit to 1 to enable an interrupt to be generated as a result of appropriate edge detection on the modem status inputs. See UART Modem Status Register (UART_MSR) on page 52 for more information on modem signals that cause an interrupt.

Bit 2 **LSIE** - (Line Status Interrupt Enable) Set this bit to 1 to enable an interrupt to be generated as a result of a receive data error. The error could be because of an incorrect parity bit received, framing error, overrun error or break detection.

Bit 1 **TIE** - (Transmit Interrupt Enable) Set this bit to 1 to enable an interrupt to be generated when the transmit FIFO/buffer is empty and no more bytes are available for transmission.

Bit 0 **RIE** - (Receive Interrupt Enable) Set this bit to 1 to generate an interrupt if the receive buffer contains the byte (if FIFO is not enabled) or receive data FIFO (if enabled) contains one or more data bytes to be read by the processor. The exact number depends on the programmed threshold level in the UART_FCTL register. This bit also enables the receiver timeout interrupt.

UART Interrupt Identification Register (UART_IIR)

(CPU Read only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	FSTS1	FSTS0	Reserved	Reserved	INTSTS2	INTSTS1	INTSTS0	INTBIT
Reset	0	0	x	x	0	0	0	1
CPU access	R	R	N/A	N/A	R	R	R	R

This register allows the user to check whether the FIFO is enabled and the status of interrupts.

Bits 7, 6 **FSTS 1-0** - (FIFO Status) These two bits reflect the status of FIFO. If the FIFO has been enabled, the two bits are both 1. If the FIFO has not been enabled, the two bits are 0.

Bits 5, 4 Reserved.

Bits 3-1 **INTSTS** - (Interrupt Status Codes) The code indicated in these three bits is valid only if the INTBIT (Bit 0) is 1. If two internal interrupt sources are active and their respective enable bits are high, only the higher priority interrupt is seen by the application. The lower priority interrupt code is indicated only after the higher priority interrupt has been serviced. Table 8 shows the interrupt status codes.

Bit 0 **INTBIT** - A 0 indicates an active interrupt source within the module if the respective enable is 1.

Table 8.UART Interrupt Status Codes

INTSTS Value	Priority	Interrupt Type
011	Highest	Receiver Line Status Interrupt Code
010	Second	Receive Data Ready or Trigger Level Interrupt Code

Table 8.UART Interrupt Status Codes (Continued)

INTSTS Value	Priority	Interrupt Type
110	Third	Character Timeout Interrupt Code
001	Fourth	Transmit Buffer Empty Interrupt Code
000	Lowest	Modem Status Interrupt Code

UART FIFO Control Register (UART_FCTL)

(CPU Write only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	TRIG1	TRIG0	Reserved	Reserved	Reserved	CLRTXF	CLRRXF	FIFOEN
Reset	0	0	x	x	X	0	0	0
CPU access	W	W	N/A	N/A	N/A	W	W	W

This register is used to monitor trigger levels, clear FIFO pointers, and enable or disable FIFOs.

Bits 7, 6 **TRIG 0-1** - (Receive FIFO Trigger Level) These two bits indicate the receive FIFO trigger levels. A combination of 00 indicates a trigger level of one, meaning that if there is one byte in the receive FIFO, a receive data ready interrupt is generated. A 01 combination indicates a trigger level of 4. A 10 combination indicates a trigger level of eight and a 11 combination indicates a trigger level of 14. Valid only if FIFO is enabled through setting of FIFOEN bit.

Bits 5-3 Reserved.

Bit 2 **CLRTXF** - (Clear Transmit FIFO) When a 1 is written to this bit, the transmit FIFO pointers are reset and the transmit FIFO is emptied. Valid only if FIFO is enabled by setting Bit 0 (FIFOEN).

Bit 1 **CLRRXF** - (Clear Receive FIFO) When a 1 is written to this bit, the receive FIFO pointers are reset and the receive FIFO is emptied along with the receive error FIFO. Valid only if FIFO is enabled by setting Bit 0 (FIFOEN).

Bit 0 **FIFOEN** - (FIFO Enable) When set to 1, both the transmit and receive FIFOs are enabled. If 0, FIFOs are disabled and both the transmit and receive buffers are one byte deep.

UART Line Control Register (UART_LCTL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	DLAB	SB	FPE	EPS	PEN	CHAR2	CHAR1	CHAR0
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

This register is used to control the communication control parameters.

Bit 7 **DLAB** - (Divisor Latch Access Bit) When 1, this bit enables the application to access the Baud Rate Generator registers for the standard UART. It must be set to 0 to access transmit/receive registers.

Bit 6 **SB** -(Send Break) This bit, when 1, sends continuous 0 on the transmit output from the next following bit boundary. The transmit data in the transmit shift register is ignored. After making this bit high, the TXD output is made 0 only after the bit boundary is reached. Just before making the TXD 0, it clears the transmit FIFO once. Any new data written to the transmit FIFO during a break should be written only after the THRE bit of UART_LSR register becomes high. See UART Line Status Register (UART_LSR) on page 50. This new data is transmitted after the UART recovers from the break. After the break is removed, the UART recovers from break for the next BDIN edge.

Bit 5 **FPE** - (Force Parity Error) When this bit and the enable parity bit (PEN) are high, the bit forces an incorrect parity bit to be transmitted with the data byte.

Bit 4 **EPS** - (Even Parity Set) If this bit is programmed as 1, the total number of 1 in the transmitted data bits and parity bit is even. If programmed as 0, the total number of 1 in the transmitted data bits and parity bit is odd.

Bit 3 **PEN** - (Parity Enable) This bit enables the generation and transmission of a parity bit in the transmitter and reception and checking of parity bits in the receiver block of the module.

Bits 2-0 **CHAR 2-0** - (Character Parameters) These bits control the number of data and stop bits to be transmitted/received. Table 9 shows the details of this bit. The receiver always checks for only one stop bit - regardless of the programmed number of stop bits.

Table 9.UART Character Length, Stop Bits

CHAR[2:0]	Character Length, Stop Bits
000	5 data bits, 1 stop bit
001	6 data bits, 1 stop bit
010	7 data bits, 1 stop bit
011	8 data bits, 1 stop bit
100	5 data bits, 2 stop bits
101	6 data bits, 2 stop bits
110	7 data bits, 2 stop bits
111	8 data bits, 2 stop bits

UART Modem Control Register (UART_MCTL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	Reserved	Reserved	Reserved	LOOP	OUT2	OUT1	RTS	DTR
Reset	x	x	x	0	0	0	0	0
CPU access	N/A	N/A	N/A	R/W	R/W	R/W	R/W	R/W

This register is used to control and check the modem status.

Bits 7-5 Reserved.

Bit 4 **LOOP** - (Loop-back Enable) When 1, this bit puts the UART in internal loop back mode. In this mode, the transmit data output port is disconnected from the internal transmit data output and set to 1. The receive data input port is disconnected and internal receive data is connected to internal transmit data. The modem status input ports are disconnected and the four bits of the modem control register are connected as modem status inputs. The two modem control output ports (OUT1&2) are set to their inactive state.

Bit 3 **OUT2** - (Output 2) This bit has no function under normal mode. In LOOP mode, this bit is connected to DCD bit in the modem status register.

Bit 2 **OUT1** - (Output 1) This bit has no function under normal mode. In LOOP mode, this bit is connected to RI bit in the modem status register.

Bit 1 **RTS** - (Request To Send) This bit controls the NRTS output port. The NRTS output reflects the inverted state of this bit. In LOOP mode, this bit is connected to the CTS bit of the modem status register.

Bit 0 **DTR** - (Data Terminal Ready) This bit controls the NDTR output port. The NDTR output reflects the inverted state of this bit. In LOOP mode, this bit is connected to the DSR bit of the modem status register.

UART Line Status Register (UART_LSR)

(CPU Read)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	ERR	TEMT	THRE	BI	FE	PE	OE	DR
Reset	0	1	1	0	0	0	0	0
CPU access	R	R	R	R	R	R	R	R

This register is used to show the status of UART interrupts and registers.

Bit 7 **ERR** - (Receive Error) This bit is 0 in the non FIFO mode of operation. In FIFO mode of operation, this bit is 1 when there is at least one parity, framing or break indication in the FIFO. This signal is reset when the **UART_LSR** register is read and there are no more bytes with error status in the FIFO.

Bit 6 **TEMT** - (Transmit Empty) This bit, when 1, indicates that the transmit holding register/FIFO and the transmit shift register are both empty and the transmitter is idle. This bit cannot change to 1 during the break condition. This bit becomes 1 only when the break command is removed.

Bit 5 **THRE** - (Transmit Holding Register Empty) This bit, when 1, indicates that the transmit FIFO/holding register is empty. This bit is the source of transmitter interrupt. This bit cannot change to 1 during the break condition. This bit becomes 1 only when the break command is removed.

Bit 4 **BI** - (Break Indicator) This bit, when high, indicates that the receiver has detected a break condition on the receive input line. This bit is 1 if the duration of break condition on the receive data is longer than one character transmission time, the time depends on the programming of **UART_LCR** register, see UART Line Control Register (UART_LCTL) on page 47. In case of FIFO only one null character is loaded into the receiver FIFO with the framing error. The framing error is revealed to the CPU whenever that particular data is read from the receiver FIFO. This bit can be reset when **UART_LSR** register is read by the processor.

Bit 3 **FE** - (Framing Error) This bit is 1 whenever the stop bit following that data/parity bit is logic 0. The bit is reset whenever the CPU/system reads the **UART_LSR** register. In the FIFO mode the framing error is stored for the particular byte received. This error is revealed to the

CPU when the associated character is at the top of the FIFO. This bit can be reset when the **UART_LSR** register is read by the processor.

Bit 2 **PE** - (Parity Error) This bit indicates that the received character does not have a correct parity. This bit is 1 upon detection of a parity error and resets when the CPU/system reads the contents of the **UART_LSR** register. In the FIFO mode of operation, this error is associated with the particular character received and revealed to the CPU when the byte is at the top of the FIFO.

Bit 1 **OE** - (Overrun Error) This bit indicates that data in the receiver buffer register was not read by the CPU before the next character was transferred into the receiver buffer register. This bit is 1 when the overrun condition occurs and resets when the CPU reads the **UART_LSR** register. In the FIFO mode, this bit is 1 when the FIFO is full and the next character is received by the receiver shift register. The character in the receiver shift register is not put into the receiver FIFO.

Bit 0 **DR** - (Data Ready) This bit is 1 when a complete incoming character is transferred into the receiver buffer register from the receiver shift register. The bit is reset by reading the receiver buffer register. In the FIFO mode, this bit is set whenever a character is received and transferred to the receiver FIFO. This bit is reset after reading all the bytes from the receiver FIFO.

UART Modem Status Register (UART_MSR)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS
Reset	x	x	x	x	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

This register is used to show the status of modem signals.

Bit 7 **DCD** - (Data Carrier Detect) This bit reflects the inverted state of the NDCD input signal. In loop back mode, this reflects the OUT2 (Bit 3) of the **UART_MCTL** register. See UART Modem Control Register (UART_MCTL) on page 49.

Bit 6 **RI** - (Ring Indicate) This bit reflects the inverted state of NRI input signal. In loop back mode, this reflects the OUT1 bit (Bit 2) of the **UART_MCTL** register. See UART Modem Control Register (UART_MCTL) on page 49.

Bit 5 **DSR** - (Data Set Ready) This bit reflects the inverted state of NDSR input signal. In loop back mode, this reflects the DTR bit (Bit 0) of the **UART_MCTL** register. See UART Modem Control Register (UART_MCTL) on page 49.

Bit 4 **CTS** - (Clear To Send) This bit reflects the inverted state of NCTS input signal. In loop back mode, this reflects the RTS bit (Bit 1) of the **UART_MCTL** register. See UART Modem Control Register (UART_MCTL) on page 49.

Bit 3 **DDCD** - (Delta Status Change of DCD) This bit is the Delta DCD bit. It is 1 whenever NDCD signal changes state. The bit is reset when the processor reads the UART_MSR register.

Bit 2 **TERI** - (Trailing Edge Change of RI) This bit indicates occurrence of trailing edge of the NRI signal. It is 1 whenever a falling edge on NRI signal is detected. The bit is reset when the processor reads the UART_MSR register.

Bit 1 **DDSR** - (Delta Status Change of DSR) This bit is the Delta DSR bit. It is 1 whenever NDSR signal changes state. The bit is reset when the processor reads the UART_MSR register.

Bit 0 **DCTS** - (Delta Status Change of CTS) This bit is the Delta CTS bit. It is 1 whenever NCTS signal changes state. The bit is reset when the processor reads the UART_MSR register.

UART Scratch Pad Register (UART_SPR)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	SPR7	SPR6	SPR5	SPR4	SPR3	SPR2	SPR1	SPR0
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

This register has no significance to the functionality of the UART. It can be used by the system as a general purpose read/write register.

Design Considerations

The following is the standard sequence of events that would occur in the system using this module.

- Module reset
- Control transfers. In this phase, the UART operation is configured
- Data transfers. Data transfers between processor and UART take place

Module Reset

Upon reset all internal registers are set to their default values. All command status registers are programmed with their default values and FIFOs are flushed.

Control Transfers

Based on the application requirement, the data transfer baud rate is determined and a 16X frequency is provided at the BDIN signal input. Interrupts are disabled and the communication control parameters are programmed in the UART_LCTL register. The FIFO configuration is determined and the UART_FCTL register is programmed according to the included receive trigger levels. The status registers, UART_LSR and UART_MSR are read and ensure that none of the interrupt sources are active. The interrupts are enabled (except for the transmit interrupt) and the application is ready to use the module for transmission/reception.

Data Transfers - Transmit

To transmit data, the application enables the transmit interrupt. An interrupt is immediately expected in response to this interrupt. The application reads the UART_IIR register and determines that the interrupt is because of an empty UART_THR register. Once the application ascertains this, the application writes the transmit data bytes to the UART_THR register. The number of bytes that the application writes depends on enabling the FIFO. If the FIFO is enabled, the application can write 16 bytes at a time. If not, the application can write one byte at a time. As a result of the first write, the interrupt is deactivated. The processor then waits for the next interrupt. When the interrupt is raised by the UART module, the processor repeats the same process until it has exhausted all the data for transmission.

To control and check the modem status, the application sets up the modem by writing to the UART_MCTL register and reading the UART_MCTL register before starting the above process.

Data Transfers - Receive

The receiver is always enabled and keeps on checking for the start bit on the RXD input signal. When an interrupt is raised by the UART module, the application reads the UART_IIR register and determines the cause for the interrupt. If the cause is a line status interrupt, the application reads the UART_LSR register, reads the data byte and then discards the byte or it can also take appropriate action. If the interrupt is caused by a receive-data-ready condition, the application alternately reads the UART_LSR and UART_RBR registers and removes all the received data bytes. It reads the UART_LSR register before reading the UART_RBR register to determine that there is no error in the received data.

To control and check modem status, the application sets up the modem by writing to the UART_MCTL register and reading the UART_MSR register before starting the above mentioned process.

Poll Mode Transfers

The application has to continuously poll the UART_LSR register to transmit or receive data without enabling the interrupts. The same holds true for the UART_MSR register. If the interrupts are not enabled, the data in the UART_IIR register cannot be used to determine the cause of interrupt.

Serial Peripheral Interface

The serial peripheral interface (SPI) is a synchronous interface allowing several SPI-type devices to be interconnected. In a serial peripheral interface, separate signals are required for data and clock. In the SPI format, the clock is not included in the data stream and must be furnished as a separate signal. The SPI may be configured as either a master or a slave.

The four basic SPI signals (MISO, MOSI, SCK, and $\overline{SS}$) are discussed in the following paragraphs. Each signal is described in both master and slave modes.

Master In Slave Out (MISO)

The MISO signal is configured as an input in a master device and as an output in a slave device. It is one of the two lines that transfer serial data in one direction, with the most significant bit sent first. The MISO of a slave device is placed in high-impedance state if the slave is not selected. When SPI is disabled or not selected in UZI, this signal is in a high-impedance state.

Master Out Slave In (MOSI)

The MOSI signal is configured as an output in a master device and as an input in a slave device. It is one of the two lines that transfer serial data in one direction, with the most significant bit sent first. When SPI is disabled or not selected in UZI, this signal is in a high-impedance state.

Serial Clock (SCK)

The serial clock is used to synchronize data movement both in and out of the device through its MOSI and MISO signals. The master and slave are capable of exchanging a byte of data during a sequence of eight clock cycles. Since SCK is generated by the master, this line becomes an input on a slave device.

As shown in Figure 6, four possible timing relations may be chosen by using control bits CPOL and CPHA in the **SPI_CTL** register. See SPI Control Register (SPI_CTL) on page 59. Both the master and slave must operate with the same timing. The master device always places data on the MOSI line a half-cycle before the clock edge (SCK signal), in order for the slave device to latch the data.

Two bits (SPR0 and SPR1) in the master device select the clock rate. In a slave device, SPR0 and SPR1 have no effect on the operation of the SPI.

Slave Select ($\overline{SS}$)

The slave select ($\overline{SS}$) input signal is used to select a slave device. It has to be low prior to data transactions and must stay low for the duration of the transaction.

The $\overline{SS}$ signal on the master must be high. If the $\overline{SS}$ signal goes low, a Mode Fault error flag (MODF bit) is set in the **SPI_SR** register. See SPI Status Register (SPI_SR) on page 60.

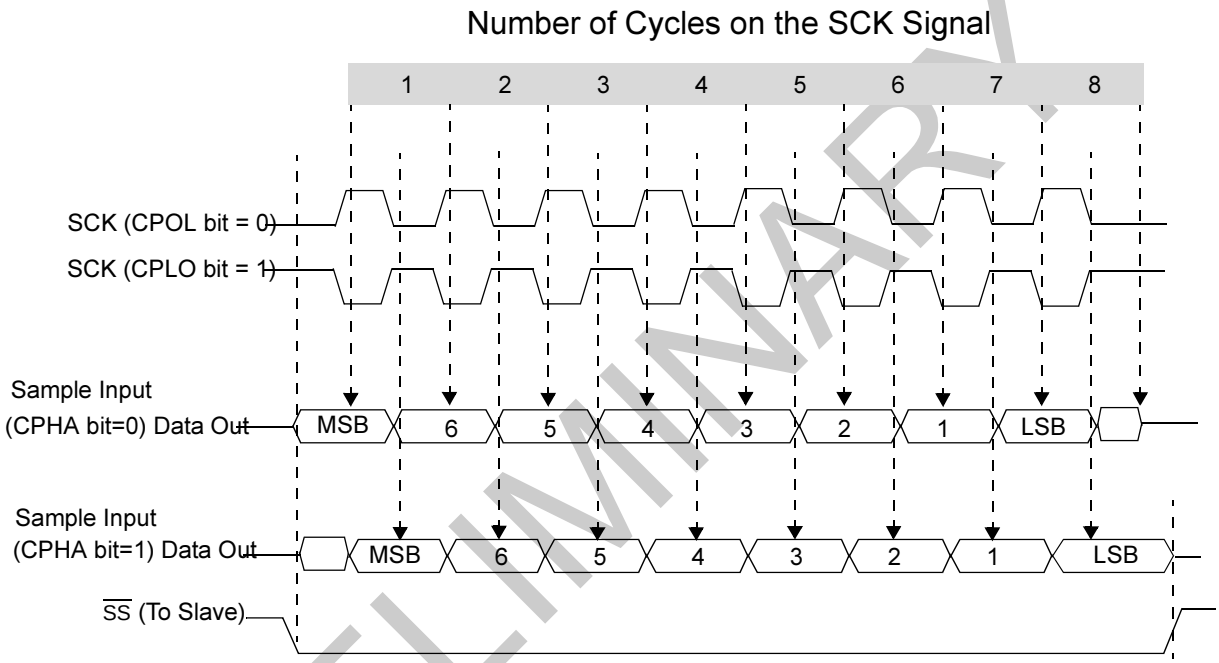


Figure 6. SPI Timing Diagram

When the CPHA bit = 0, the shift clock is the OR of $\overline{SS}$ with SCK. In this clock phase mode, $\overline{SS}$ must go high between successive characters in an SPI message. When CPHA = 1, $\overline{SS}$ may be left low for several SPI characters. In cases where there is only one SPI slave MCU, its $\overline{SS}$ line could be tied low as long as CPHA = 1 clock mode is used. See SPI Control Register (SPI_CTL) on page 59 for more information on the CPHA bit.

Functional Description

Figure 7 shows a block diagram of the serial peripheral interface circuitry. When a master transmits to a slave device via the MOSI signal, the slave device responds by sending data to the master via the master's MISO signal. This implies full duplex transmission with both data out and data in synchronized with the same clock signal. Thus the byte transmitted is replaced by the byte received and eliminates the need for separate transmit-empty and receive-full status bits. A single status bit is used to signify that the I/O operation has been completed, see SPI Status Register (SPI_SR) on page 60.

The SPI is double buffered on read, but not on write. If a write is performed during data transfer, the transfer occurs uninterrupted, and the write is unsuccessful. This condition causes the write collision (WCOL) status bit in the SPI_SR register to be set. After a data byte is shifted, the SPIF flag of the SPI_SR register is set.

In the master mode, the SCK pin is an output. It idles high or low, depending on the CPOL bit in the SPI_CTL register, until data is written to the shift register. When data is written to the shift register eight clocks are generated to shift the eight bits of data in both directions and then the SCK signal goes idle.

In slave mode, the start logic receives a logic low from the $\overline{SS}$ pin and a clock input at the SCK pin, and the slave is synchronized to the master. Data from the master is received serially from the slave MOSI signal and loads the 8-bit shift register. After the 8-bit shift register is loaded, its data is parallel transferred to the read buffer. During a write cycle data is written into the shift register, then the slave waits for a clock train from the master to shift the data out on the slave's MISO signal.

Figure 8 illustrates the MOSI, MISO, SCK, and $\overline{SS}$ master-slave interconnections.

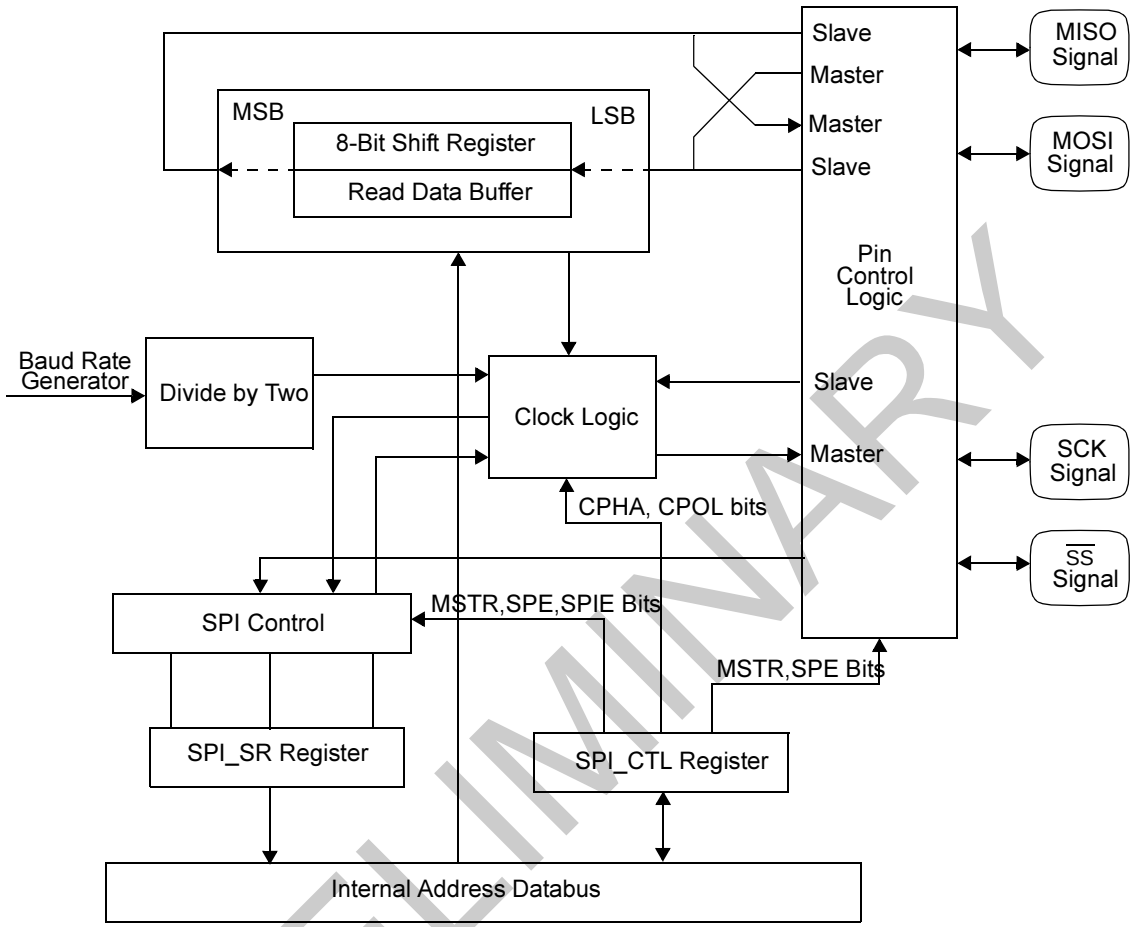


Figure 7. SPI Block Diagram

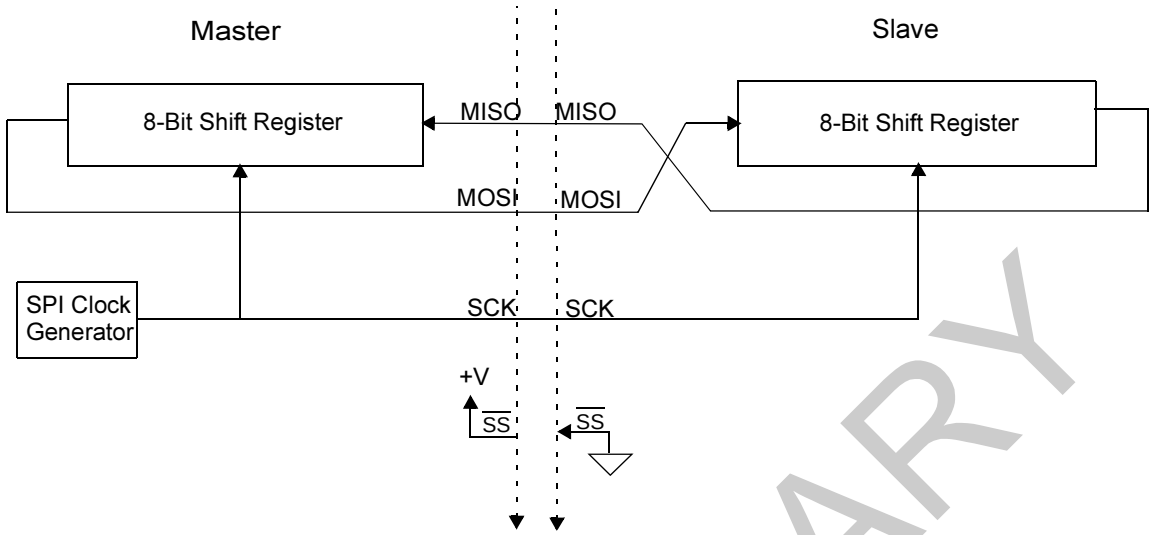


Figure 8. SPI Master/Slave Connections

SPI Registers

There are four registers in the Serial Peripheral Interface which provide control, status, and data storage functions. These registers are called the SPI Control Register (SPI_CTL), SPI Status Register (SPI_SR), SPI Receive Buffer Register (SPI_RBR), and SPI Transmit Shift Register (**SPI_TSR**), and are described in the following paragraphs.

SPI Control Register (SPI_CTL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	SPIE	Reserved	SPE	MSTR	CPOL	CPHA	Reserved	Reserved
Reset	0	x	0	0	0	1	x	x
CPU access	R/W	N/A	R/W	R/W	R/W	R/W	N/A	N/A

This register is used to control and setup the serial peripheral interface.

Bit 7 SPIE - (SPI Interrupt Enable) Setting this bit to 1 requests a hardware interrupt sequence each time the SPIF or MODF status flag is set.

0 = SPI system interrupt disabled

1 = SPI system interrupt enabled

Bit 6 Reserved.

Bit 5 SPE - (SPI Enable) When this bit is set to 1, the SPI is enabled.

0 = SPI system disabled

1 = SPI system enabled

Bit 4 MSTR (Master Mode Select) This bit is used to select master or slave mode.

0 = Slave mode.

1 = Master mode.

Bit 3 CPOL - (Clock Polarity) When the clock polarity bit is cleared and data is not being transferred, a steady state low value is produced at the SCK pin of the master device. If this bit is 1, the SCK signal idles high. This bit is also used in conjunction with the clock phase control bit to produce the desired clock-data relationship between master and slave. See Figure 6.

Bit 2 **CPHA** - (Clock Phase) This bit in conjunction with the CPOL bit controls the clock-data relationship between master and slave. The CPOL bit can be thought of as simply inserting an inverter in series with the SCK line. The CPHA bit selects one of two fundamentally different clocking protocols. When CPHA = 0, the shift clock is the OR of SCK with $\overline{SS}$. As soon as $\overline{SS}$ goes low the transaction begins and the first edge on the SCK signal invokes the first data sample. When CPHA = 1, the $\overline{SS}$ signal may be thought of as a simple output enable control. Refer to Figure 6.

Bits 1, 0 Reserved.

SPI Status Register (SPI_SR)

(CPU Read only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	SPIF	WCOL	Reserved	MODF	Reserved	Reserved	Reserved	Reserved
Reset	0	0	0	0	0	0	0	0
CPU access	R	N/A	N/A	R	N/A	N/A	N/A	N/A

This register is used to show the status of data transmitted using the serial peripheral interface.

Bit 7 SPIF - (SPI Transfer Complete Flag) The serial peripheral data transfer flag bit is 1 upon completion of data transfer between the processor and external device. When the SPIF bit is high, a serial peripheral interrupt is generated if enabled through SPIE bit in the **SPI_CTL** register. Clearing the SPIF bit is accomplished by reading the SPI_SR register (with SPIF set).

Bit 6 WCOL - (Write Collision) This bit is 1 when an attempt is made to write to the serial peripheral data register while data transfer is taking place. If the CPHA bit in the **SPI_CTL** register is 0, a transfer begins when $\overline{SS}$ pin signal goes low and the transfer ends when $\overline{SS}$ goes high after eight clock cycles on SCK. When the CPHA bit is 1, a transfer begins the first time SCK becomes active while $\overline{SS}$ is low and the transfer ends when the SPIF flag gets set. Clearing the WCOL bit is accomplished by reading SPI_SR (with the WCOL bit set).

Bit 5 Reserved.

Bit 4 MODF - (Mode Fault) The mode fault flag indicates that there may have been a multi-master conflict for system control and allows a proper exit from system operation to a reset or default system state. The MODF bit is normally clear, and is only set when the master device has its $\overline{SS}$ pin pulled low. Setting the MODF bit affects the internal serial peripheral interface in the following ways:

1. The SPE bit is cleared, disabling the SPI.
2. The MSTR bit is cleared and forces the device into slave mode.
3. SPI interrupt is generated if the SPIE is 1.

Clearing the MODF bit is accomplished by reading the SPI_SR register (with MODF set). Control bits SPE and MSTR are restored to their original state by the user software after the MODF bit has been cleared.

Bits 3-0 Reserved.

SPI Transmit Shift Register (SPI_TSR)

(CPU Write Only)

The SPI Transmit Shift Register (SPI_TSR) is used to transmit data onto the serial bus. Only a write to this register initiates transmission of another byte, and this only occurs in the master device. At the completion of transmitting a byte of data,

the SPIF status bit is 1 in both the master and slave devices. A write to the SPI_TSR register places data directly into the shift register for transmission.

SPI Receive Buffer Register (SPI_RBR)

(CPU Access Read Only)

The SPI Receive Buffer Register **(SPI_RBR)** is used to receive data from the serial bus. A write to the (SPI_TSR) register initiates reception of another byte, and only occurs in the master device. When the user reads the SPI_RBR register, a buffer is being read. The first SPIF bit must be cleared by the time a second transfer of data from the shift register is initiated or an overrun condition exists. In cases of overrun the byte that caused the overrun is lost.

> **Note:** The **SPI_TSR** register and the **SPI_RBR** register both exist in the same address space.

I2C Overview

General Characteristics

The I2C-bus serial I/O operates in four modes:

- Master Transmitter
- Master Receiver
- Slave Transmitter
- Slave Receiver

Both SDA and SCL are bidirectional lines, connected to a positive supply voltage via a pull-up resistor. When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain in order to perform the wired-AND function. Data on the I2C-bus can be transferred at a rate of up to 100 kbit/s in the standard-mode, or up to 400 kbit/s in the fast-mode.

One clock pulse is generated for each data bit transferred.

Data Validity

The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW as shown in Figure 9.

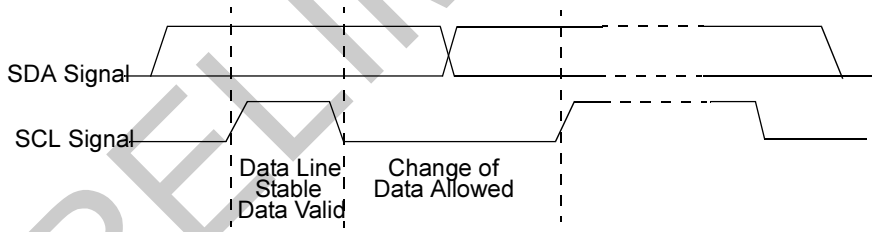


Figure 9. I2C Clock and Data Relationship

START and STOP Conditions

Within the I2C-bus procedure, unique situations arise which are defined as START and STOP conditions (see Figure 10). A HIGH to LOW transition on the SDA line while SCL is HIGH is one such unique case. This situation indicates a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.

START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free a certain time after the STOP condition.

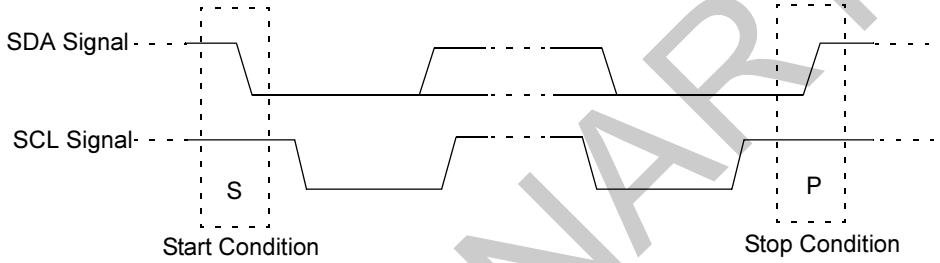


Figure 10. Start and Stop Conditions In I2C Protocol

Transferring Data

Byte format

Every byte put on the SDA line must be 8 bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first (see Figure 11). If a receiver cannot receive another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL (LOW) to force the transmitter into a wait state. Data transfer then continues when the receiver is ready for another byte of data and releases clock line SCL.

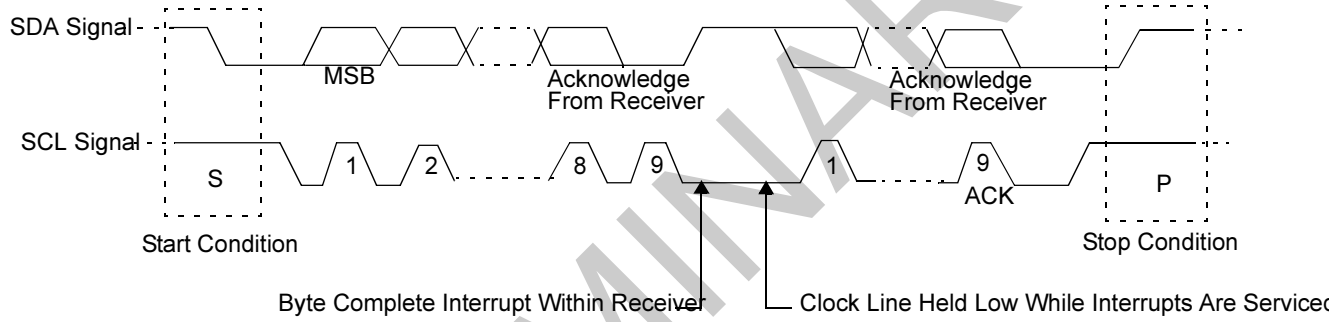


Figure 11. I2C Frame Structure

Acknowledge

Data transfer with acknowledge is obligatory. The acknowledge-related clock pulse is generated by the master. The transmitter releases the SDA line (HIGH) during the acknowledge clock pulse. The receiver must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse (see Figure 12).

A receiver that has been addressed is obliged to generate an acknowledge after each byte has been received. When a slave-receiver doesn't acknowledge the slave address (for example, unable to receive because it's performing some real-time function), the data line must be left HIGH by the slave. The master then generates a STOP condition to abort the transfer.

If a slave-receiver acknowledges the slave address, but cannot receive any more data bytes, the master must abort the transfer. The abort is indicated by the slave generating the not acknowledge on the first byte to follow. The slave leaves the data line HIGH and the master generates the STOP condition.

If a master-receiver is involved in a transfer, it must signal the end of data to the slave-transmitter by not generating an acknowledge on the last byte that was clocked out of the slave. The slave-transmitter must release the data line to allow the master to generate a STOP or repeated START condition.

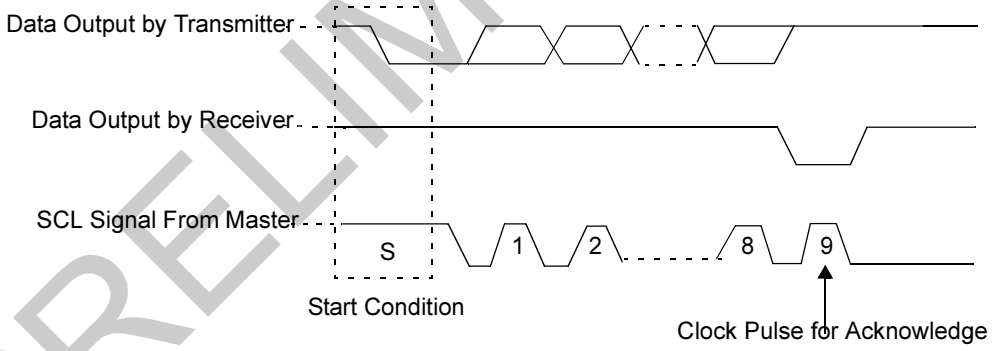


Figure 12. I2C Acknowledge

Arbitration And Clock Generation

Synchronization

All masters generate their own clock on the SCL line to transfer messages on the I2C-bus. Data is only valid during the HIGH period of the clock.

Clock synchronization is performed using the wired-AND connection of I2C interfaces to the SCL line. This means that a HIGH to LOW transition on the SCL line causes the relevant devices to start counting off their LOW period. Once a device clock has gone LOW, it holds the SCL line in that state until the clock HIGH state is reached (see Figure 13). However, the LOW to HIGH transition of this clock may not change the state of the SCL line if another clock is still within its LOW period. The SCL line is held LOW by the device with the longest LOW period. Devices with shorter LOW periods enter a HIGH wait-state during this time.

When all devices concerned have counted off their LOW period, the clock line is released and goes HIGH. There is no difference between the device clocks and the state of the SCL line, and all the devices start counting their HIGH periods. The first device to complete its HIGH period again pulls the SCL line LOW. In this way, a synchronized SCL clock is generated with its LOW period determined by the device with the longest clock LOW period, and its HIGH period determined by the one with the shortest clock HIGH period.

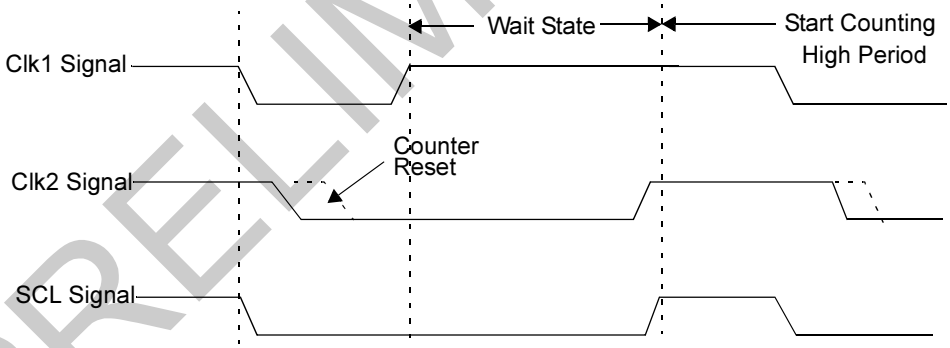


Figure 13. Clock Synchronization In I2C Protocol

Arbitration

A master may start a transfer only if the bus is free. Two or more masters may generate a START condition within the minimum hold time of the START condition which results in a defined START condition to the bus. Arbitration takes place on the SDA line, while the SCL line is at the HIGH level, in such a way that the master which transmits a HIGH level, while another master is transmitting a LOW level will switch off its DATA output stage because the level on the bus doesn't correspond to its own level.

Arbitration can continue for many bits. Its first stage is comparison of the address bits. If the masters are each trying to address the same device, arbitration continues with comparison of the data. Because address and data information on the I2C-bus is used for arbitration, no information is lost during this process. A master which loses the arbitration can generate clock pulses until the end of the byte in which it loses the arbitration.

If a master also incorporates a slave function and it loses arbitration during the addressing stage, it's possible that the winning master is trying to address it. The losing master must switch over immediately to its slave-receiver mode. Figure 13 shows the arbitration procedure for two masters. Of course, more may be involved (depending on how many masters are connected to the bus). The moment there is a difference between the internal data level of the master generating DATA 1 and the actual level on the SDA line, its data output is switched off, which means that a HIGH output level is then connected to the bus. This will not affect the data transfer initiated by the winning master. Since control of the I2C-bus is decided solely on the address and data sent by competing masters, there is no central master, nor any order of priority on the bus.

Special attention must be paid if, during a serial transfer, the arbitration procedure is still in progress at the moment when a repeated START condition or a STOP condition is transmitted to the I2C-bus. If it's possible for such a situation to occur, the masters involved must send this repeated START condition or STOP condition at the same position in the format frame. In other words, arbitration is not allowed between:

- A repeated START condition and a data bit.
- A STOP condition and a data bit.
- A repeated START condition and a STOP condition.

Use of the Clock Synchronizing Mechanism as a Handshake

The clock synchronization mechanism also enables receivers to cope with fast data transfers, on either a byte or bit level. The byte level allows a device to receive bytes of data at a fast rate, but needs more time to store a receive byte or prepare another byte for transmission. Slaves hold the SCL line LOW after reception and acknowledge the byte, forcing the master into a wait state until the slave is ready for the next byte transfer in a handshake procedure.

On the bit level, a microcontroller with or without a limited hardware on-chip I2C interface can slow down the bus clock by extending each clock LOW period. The speed of any master is adapted to the internal operating rate of this device.

I2C Registers

Addressing

The processor interface provides access to six 8-bit registers; four read/write registers, one read only register and one write only register. See Table 11.

Table 10.I2C Register Descriptions

Register	Description
I2C_SAR	Slave address
I2C_XSAR	Extended slave address
I2C_DR	Data byte
I2C_CTL	Control register
I2C_SR	Status register (read only)
I2C_SRR	Software reset

Resetting the I2C Registers

Hardware reset When the MI2C is reset using the RST pin, the **I2C_SAR**, **I2C_XSAR**, **I2C_DR** and **I2C_CTL** registers are cleared to 00h; **I2C_SR** register is set to F8h.

Software reset Perform a software reset by writing any value to the I2C Software Reset Register (**I2C_SRR**) mapped to DDh. A software reset sets the MI2C back to idle and the STP, STA and IFLG bits of the **I2C_CTL** register to 0.

I2C 10-Bit Addressing

To perform 10-bit addressing, set Bits 7 - 4 in the **I2C_SAR** register to 1 and Bit 3 - 0. See Table 11.

Table 11. Enabling I2C 10-Bit Extended Addressing

I2C_SAR Bit Name	Bit	Value
SLA 6	7	1
SLA 5	6	1
SLA 4	5	1
SLA 3	4	1
SLA2	3	0

The bits in the above table enable 10-bit extended addressing using the **I2C_SAR** and **I2C_XSAR** registers. See Table 12 for the location of the bits. When the **IC_SAR** register receives an address that starts with F7 to F0h MI2C recognizes that Bits 1 and 2 are used for an extended address and sends an ACK (the device does not generate an interrupt at this point.) After the next byte of the address has been received, the MI2C generates an interrupt and goes into slave mode.

Table 12. I2C 10-Bit Extended Addressing

Name	Register	Bit
SLAX 9	I2C_SAR	2
SLAX 8	I2C_SAR	1
SLAX 7	I2C_XSAR	7
SLAX 6	I2C_XSAR	6
SLAX 5	I2C_XSAR	5
SLAX 4	I2C_XSAR	4
SLAX 3	I2C_XSAR	3
SLAX 2	I2C_XSAR	2
SLAX 1	I2C_XSAR	1
SLAX 0	I2C_XSAR	0

I2C Slave Address Register (I2C_SAR)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	SLA6	SLA5	SLA4	SLA3	SLA2	SLA1	SLA0	GCE
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The I2C_SAR register provides the 7-bit address of the MI2C when in slave mode and allows 10-bit addressing using the **I2C_XSAR** register. See I2C 10-Bit Addressing on page 70.

Bits 7-1 **SLA6-0** - (Slave Address Bits) SLA6-SLA0 is the 7-bit address of the MI2C when in slave mode. When the MI2C receives this address after a START condition, it enters slave mode (SLA6 corresponds to the first bit received from the I2C bus.) When the register receives an address starting with F7 to F0h, MI2C recognizes that Bits 1 and 2 are used for an extended address. See I2C 10-Bit Addressing on page 70.

Bit 0 **GCE** - (General Call Address Enable) If GCE is set to 1, the MI2C recognizes the general call address (00h).

I2C Extended Slave Address Register (I2C_XSAR)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	SLAX7	SLAX6	SLAX5	SLAX4	SLAX3	SLAX2	SLAX1	SLAX0
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The I2C_XSAR register is used in conjunction with the I2C_SAR register to provide 10-bit addressing of the MI2C when in slave mode. See I2C 10-Bit Addressing on page 70.

Bits 7-0 **SLA7-0** - (Slave Address Bits) SLA7-SLA0 is seven bits of the 10-bit address of the MI2C when in slave mode. See Table 12 on page 70 for more information for all bits used in 10-bit addressing.

I2C DATA Register (I2C_DR) CPU Access (Read/Write)

This register contains the data byte/slave address to be transmitted or the data byte that has just been received. In transmit mode, the byte is first sent to MSB. In receive mode, the first bit received is placed in the MSB of the register.

After each byte is transmitted, the I2C_DR register contains the byte that was present on the bus in case a lost arbitration event occurs.

I2C Control Register (I2C_CTL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	IEN	ENAB	STA	STP	IFLG	AAK	Unused	Unused
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	R/W	R/W	R/W	R/W	R	R

The **I2C_CTL** register is a control register that is used to control the interrupts and the master slave relationships on the MI2C bus.

Bit 7 **IEN** - (Interrupt Enable) When IEN is set to 1, the interrupt line (INTR) goes high when the IFLG bit is 1. When IEN is cleared to 0, the interrupt line always remains low.

Bit 6 **ENAB** - (Bus Enable)

If ENAB = 0, the I2C bus ISDA/ISCL inputs are ignored and the MI2C does not respond to any address on the bus.

If ENAB = 1, the MI2C responds to calls to its slave address and to the general call address if the GCE bit in the **I2C_SAR** register is set.

Bit 5 **STA** - (Master Mode Start) If STA is set to1, the MI2C enters master mode and sends a START condition on the bus when the bus is free.

If the STA bit is set to 1 when the MI2C is in master mode and one or more bytes have been transmitted, then a repeated START condition is sent.

If the STA bit is set to 1 when the MI2C is being accessed in slave mode, the MI2C completes the data transfer in slave mode and then enters the master mode after the bus is released. The STA bit is cleared automatically after a START condition is sent and writing a 0 to this bit has no effect.

Bit 4 **STP** - (Master Mode Stop) If STP is set to 1 in master mode, a STOP condition is transmitted on the I2C bus.

If the STP bit is set to 1 in slave mode, the MI2C functions as if a STOP condition has been received, but no STOP condition is transmitted on the I2C bus.

If both STA and STP bits are set, the MI2C first transmits the STOP condition (if in master mode) then transmits the START condition. The STP bit is automatically cleared and writing a 0 to this bit has no effect.

Bit 3 **IFLG** - (Interrupt Flag) The IFLG is automatically set to 1 when any of 30 of the possible 31 MI2C states is entered. The only state that does not set IFLG is state F8h (see Table 13 on page 75).

If IFLG is set to 1 and the IEN bit is set, the interrupt line goes high. When IFLG is set by the MI2C, the low period of the I2C bus clock signal (SCL) is stretched and data transfer is suspended. If 0 is written to IFLG, the interrupt signal goes low and the I2C clock signal is released.

Bit 2 **AAK** - (Assert Acknowledge) When AAK is set to 1, an acknowledge (low level on SDA) will be sent during the acknowledge clock pulse on the I2C bus if one of the following conditions exist:

- Either the whole of a 7-bit slave address or the first or the second byte of a 10-bit slave address have been received.
- The general call address has been received and the GCE bit in the **I2C_SAR** register is set to 1.
- A data byte has been received in the master or slave mode.

When AAK is cleared to 0, a Not Acknowledge (high level on SDA) will be sent when a data byte is received in master or slave mode. If AAK is set to 0 in the slave transmitter mode, the byte in the **I2C_DR** register is assumed to be the last byte. After this byte has been transmitted, the MI2C will enter state C8h and then return to the idle state. The MI2Cdoes not respond as a slave unless AAK is set.

I2C Status Register (I2C_SR) CPU Access (Read only)

The I2C_SR register is a read only register that contains a 5-bit status code in the five MSBs: the three LSBs are always 0.

There are 31 possible status codes, shown below in Table 13. When the I2C_SR register contains the status code F8h, no relevant status information is available, no interrupt is generated and the IFLG bit in the I2C_CTL register is not set. All other status codes correspond to a defined state of the MI2C.

When each of these states is entered, the corresponding status code appears in this register and the IFLG bit in the I2C_CTL register is set. When the IFLG bit is cleared, the status code returns to F8h. For more information on the processor's response to some status codes see Table 14 on page 78.

Table 13.MI2C Status Register

Code	Status
00h	Bus error
08h	START condition transmitted
10h	Repeated START condition transmitted
18h	Address + write bit transmitted, ACK received
20h	Address + write bit transmitted, ACK not received
28h	Data byte transmitted in master mode, ACK received
30h	Data byte transmitted in master mode, ACK not received
38h	Arbitration lost in address or data byte
40h	Address + read bit transmitted, ACK received
48h	Address + read bit transmitted, ACK not received
50h	Data byte received in master mode, ACK transmitted
58h	Data byte received in master mode, not ACK transmitted
60h	Slave address + write bit received, ACK transmitted
68h	Arbitration lost in address as master, slave address + write bit received, ACK transmitted
70h	General Call address received, ACK transmitted
78h	Arbitration lost in address as master, General Call address received, ACK transmitted
80h	Data byte received after slave address received, ACK transmitted
88h	Data byte received after slave address received, not ACK transmitted

Table 13.MI2C Status Register (Continued)

Code	Status
90h	Data byte received after General Call received, ACK transmitted
98h	Data byte received after General Call received, not ACK transmitted
A0h	STOP or repeated START condition received in slave mode
A8h	Slave address + read bit received, ACK transmitted
B0h	Arbitration lost in address as master, slave address + read bit received, ACK transmitted
B8h	Data byte transmitted in slave mode, ACK received
C0h	Data byte transmitted in slave mode, ACK not received
C8h	Last byte transmitted in slave mode, ACK received
D0h	Second Address byte + write bit transmitted, ACK received
D8h	Second Address byte + write bit transmitted, ACK not received
E0h	Second Address byte + read bit transmitted, ACK received
E8h	Second Address byte + read bit transmitted, ACK not received
F8h	No relevant status information, IFLG=0

If an illegal condition occurs on the I2C bus, the bus error state is entered (status code 00h). To recover from this state, the STP bit in the I2C_CTL register must be set and the IFLG bit cleared. The MI2C then returns to the idle state. No STOP condition is transmitted on the I2C bus.

> **Note:** The STP and STA bits may be set to 1 at the same time to recover from the bus error. The MI2C then sends a START.

Bus Clock Speed

The I2C bus is defined for bus clock speeds up to 100 kbits/s (400 kbits/s in fast-mode).

The bus clock speed generated by the MI2C in master mode is one tenth of the frequency of the BDOUT signal generated by UZI Baud Rate Generator. See Baud Rate Generator (BRG) on page 36.

Clock Synchronization

If another device on the I2C bus drives the clock line when the MI2C is in master mode, the MI2C synchronizes its clock to the I2C bus clock. The high period of the clock is determined by the device that generates the shortest high clock period. The low period of the clock is determined by the device that generates the longest low clock period.

A slave may stretch the low period of the clock to slow down the bus master. The low period may also be stretched for handshaking purposes. This can be done after each bit transfer or each byte transfer. The MI2C stretches the clock after each byte transfer until the IFLG bit in the I2C_CTL register is cleared.

Bus Arbitration

In master mode, the MI2C checks that each transmitted logic 1 appears on the I2C bus as a logic 1. If another device on the bus overrules and pulls the SDA signal LOW, arbitration is lost. If arbitration is lost during the transmission of a data byte or a Not-Acknowledge bit, the MI2C returns to the idle state. If arbitration is lost during the transmission of an address, the MI2C switches to slave mode so that it can recognize its own slave address or the general call address.

Operating Modes

Master Transmit

In the master transmit mode, the MI2C transmits a number of bytes to a slave receiver.

The master transmit mode is entered by setting the STA bit in the I2C_CTL register to 1. The MI2C then tests the I2C bus and transmits a START condition when the bus is free. When a START condition has been transmitted, the IFLG bit is 1 and the status code in the I2C_SR register is 08h. Before this interrupt is serviced, the I2C_DR register must be loaded with either a 7-bit slave address or the first part of a 10-bit slave address, with the LSB cleared to 0 to specify transmit mode. The IFLG bit should now be cleared to 0 to prompt the transfer to continue.

After the 7-bit slave address (or the first part of a 10-bit address) plus the write bit have been transmitted, the IFLG is set again. A number of status codes are possible in the I2C_SR register. See Table 14 on page 78.

Table 14.MI2C Master Transmit Status Codes

Code	MI2C State	Microprocessor Response	Next MI2C Action
18h	Addr+W transmitted, ACK received	**For a 7-bit address** Write byte to DATA, clear IFLG Or set STA, clear IFLG Or set STP, clear IFLG Or set STA & STP, clear IFLG **For a 10-bit address** Write extended address byte to DATA, clear IFLG	Transmit data byte, receive ACK Transmit repeated START Transmit STOP Transmit STOP then START Transmit extended address byte
20h	Addr+W transmitted, ACK not received	Same as code 18h	Same as code18h
38h	Arbitration lost	Clear IFLG Or set STA, clear IFLG	Return to idle Transmit START when bus is free

Table 14.MI2C Master Transmit Status Codes (Continued)

Code	MI2C State	Microprocessor Response	Next MI2C Action
68h	Arbitration lost, SLA+W received, ACK transmitted	Clear IFLG, AAK=0	Receive data byte, transmit not ACK
		Or clear IFLG, AAK=1	Receive data byte, transmit ACK
78h	Arbitration lost, General call addr received, ACK transmitted	Same as code 68h	Same as code 68h
80h	Arbitration lost, SLA+R received, ACK transmitted	Write byte to DATA, clear IFLG, AAK=0 Or write byte to DATA, clear IFLG, AAK=1	Transmit last byte, receive ACK Transmit data byte, receive ACK

If 10-bit addressing is being used, then the status code is 18h or 20h after the first part of a 10-bit address plus the write bit are successfully transmitted.

After this interrupt is serviced and the second part of the 10-bit address is transmitted, the I2C_SR register contains one of the codes in Table 15 on page 79.

Table 15.MI2C 10-Bit Master Transmit Status Codes

Code	MI2C State	Microprocessor Response	Next MI2C Action
38h	Arbitration lost	Clear IFLG Or set STA, clear IFLG	Return to idle Transmit START when bus free
68h	Arbitration lost, SLA+W received, ACK transmitted	Clear IFLG, AAK=0 Or clear IFLG, AAK=1	Receive data byte, transmit not ACK Receive data byte, transmit ACK

Table 15.MI2C 10-Bit Master Transmit Status Codes (Continued)

Code	MI2C State	Microprocessor Response	Next MI2C Action
B0h	Arbitration lost, SLA+R received, ACK transmitted	Write byte to DATA, clear IFLG, AAK=0 Or write byte to DATA, clear IFLG, AAK=1	Transmit last byte, receive ACK Transmit data byte, receive ACK
D0h	Second Address byte + W transmitted, ACK received	Write byte to DATA, clear IFLG Or set STA, clear IFLG Or set STP, clear IFLG Or set STA & STP, clear IFLG	Transmit data byte, receive ACK Transmit repeated START Transmit STOP Transmit STOP then START
D8h	Second Address byte + W transmitted, ACK not received	As code D0h	As code D0h

If a repeated START condition has been transmitted, the status code is 10h instead of 08h.

After each data byte is transmitted, the IFLG is 1 and one of the status codes listed in Table 16 on page 80 is in the I2C_SR register.

Table 16.MI2C Master Transmit Status Codes For Data Bytes

Code	MI2C State	Microprocessor Response	Next MI2C Action
28h	Data byte transmitted, ACK received	Write byte to DATA, clear IFLG Or set STA, clear IFLG Or set STP, clear IFLG Or set STA & STP, clear IFLG	Transmit data byte, receive ACK Transmit repeated START Transmit STOP Transmit START then STOP
30h	Data byte transmitted, ACK not received	Same as code 28h	Same as code 28h
38h	Arbitration lost	Clear IFLG Or set STA, clear IFLG	Return to idle Transmit START when bus free

When all bytes have been transmitted, the STP bit in the I2C_CTL register should have a 1 written to it. The MI2C then transmits a STOP condition, clears the STP bit and returns to the idle state.

Master Receive

In the master receive mode, the MI2C receives a number of bytes from a slave transmitter.

After the START condition is transmitted, the IFLG bit is 1 and the status code 08h is loaded in the I2C_SR register. The I2C_DR register should be loaded with the slave address (or the first part of a 10-bit slave address), with the LSB set to 1 to signify a read. The IFLG bit should be cleared to 0 as a prompt for the transfer to continue.

When the 7-bit slave address (or the first part of a 10-bit address) and the read bit have been transmitted, the IFLG bit is set and one of the status codes listed in Table 17 on page 81 is in the I2C_SR register.

Table 17.MI2C Master Receive Status Codes

Code	MI2C State	Microprocessor Response	Next MI2C Action
40h	Addr + R transmitted, ACK received	**For a 7-bit address** Clear IFLG, AAK = 0 Or clear IFLG, AAK = 1 **For a 10-bit address** Write extended address byte to DATA, clear IFLG	Receive data byte, transmit not ACK Receive data byte, transmit ACK Transmit extended address byte
48h	Addr + R transmitted, ACK not received	For a 7-bit address: Set STA, clear IFLG Or set STP, clear IFLG Or set STA & STP, clear IFLG For a 10-bit address: Write extended address byte to DATA, clear IFLG	Transmit repeated START Transmit STOP Transmit STOP then START Transmit extended address byte
38h	Arbitration lost	Clear IFLG Or set STA, clear IFLG	Return to idle Transmit START when bus is free
68h	Arbitration lost, SLA+W received, ACK transmitted	Clear IFLG, AAK=0	Receive data byte, transmit not ACK
		Or clear IFLG, AAK=1	Receive data byte, transmit ACK
78h	Arbitration lost, General call addr received, ACK transmitted	Same as code 68h	Same as code 68h

If 10-bit addressing is being used, the status code is 40h or 48h after the first part of a 10-bit address and the read bit have been successfully transmitted. After this interrupt is serviced and the second part of the address transmitted, the I2C_SR register contains one of the codes listed in Table 18 on page 82.

Table 18.MI2C 10-Bit Master Receive Status Codes

Code	MI2C State	Microprocessor Response	Next MI2C Action
38h	Arbitration lost	Clear IFLG Or set STA, clear IFLG	Return to idle Transmit START when bus is free
68h	Arbitration lost, SLA+W received, ACK transmitted	Clear IFLG, AAK=0	Receive data byte, transmit not ACK
		Or clear IFLG, AAK=1	Receive data byte, transmit ACK
78h	Arbitration lost, General call addr received, ACK transmitted	Same as code 68h	Same as code 68h
B0h	Arbitration lost, SLA+R received, ACK transmitted	Write byte to DATA, clear IFLG, AAK=0 Or write byte to DATA, clear IFLG, AAK=1	Transmit last byte, receive ACK Transmit data byte, receive ACK
E0h	Second Address byte+ R transmitted, ACK received	Clear IFLG, AAK = 0 Or clear IFLG, AAK = 1	Receive data byte, transmit not ACK Receive data byte, transmit ACK
E8h	Second Address byte + R transmitted, ACK not received	As for code E0h	As for code E0h

If a repeated START condition is received, the status code is 10h instead of 08h.

After each data byte has been received, the IFLG is set and one of the status codes listed in Table 19 on page 83 is in the I2C_SR register.

Table 19.MI2C Master Receive Status Codes For Data Bytes

Code	MI2C State	Microprocessor Response	Next MI2C Action
50h	Data byte received, ACK transmitted	Read DATA, clear IFLG, AAK = 0 Or read DATA, clear IFLG, AAK = 1	Receive data byte, transmit not ACK Receive data byte, transmit ACK
58h	Data byte received, not ACK transmitted	Read DATA, set STA, clear IFLG Or read DATA, set STP, clear IFLG Or read DATA, set STA & STP, clear IFLG	Transmit repeated START Transmit STOP Transmit STOP then START
38h	Arbitration lost in not ACK bit	As for master transmit	As for master transmit

When all bytes have been received, the STP bit in the **I2C**_CTL register should have a 1 written to it. The MI2C then transmits a STOP condition, clears the STP bit and returns to the idle state.

Slave Transmit

In the slave transmit mode a number of bytes are transmitted to a master receiver.

The MI2C enters the slave transmit mode when it receives its own slave address and a read bit after a START condition. The MI2C then transmits an acknowledge bit and sets the IFLG bit in the I2C_CTL register and the I2C_SR register contains the status code A8h.

> **Note:** When MI2C has a 10-bit slave address (signified by F0h-F7h in the I2C_SAR register), it transmits an acknowledge after the first address byte is received but an interrupt is not generated, IFLG does not set and the status does not change. Only after the second address byte has been received does the MI2C generate an interrupt, set the IFLG bit and load the status code as described above.

MI2C goes from a master mode to slave transmit mode when arbitration is lost during the transmission of an address, and the slave address and read bit are received. This action is represented by the status code B0h in the **I2C_SR** register.

The data byte to be transmitted is loaded into the **I2C_DR** register and the IFLG bit cleared. After the MI2C has transmitted the byte and received an acknowledge, the IFLG bit is set and the **I2C_SR** register contains B8h. Once the last byte to be transmitted is loaded into the **I2C_DR** register, the AAK bit is cleared when the IFLG is cleared. After the last byte has been transmitted, the IFLG is set and the **I2C_SR** register will contain C8h and the MI2C returns to the idle state. The AAK bit must be set to 1 before the slave mode can be re-entered.

If no acknowledge is received after transmitting a byte, the IFLG is set and the **I2C_SR** register contains C0h. The MI2C then returns to the idle state.

If a STOP condition is detected after an acknowledge bit, the MI2C returns to the idle state.

Slave Receive

In the slave receive mode, a number of data bytes are received from a master transmitter.

The MI2C enters the slave receive mode when it receives its own slave address and a write bit (LSB=0) after a START condition. The MI2C transmits an acknowledge bit and sets the IFLG bit in the **I2C_CTL** register and the **I2C_SR** register contains the status code 60h. The MI2C also enters slave receive mode when it receives the general call address 00h (if the GCE bit in the **I2C_SAR** register is set). The status code is then 70h.

> **Note:** When the MI2C has a 10-bit slave address (signified by F0h-F7h in the **I2C_SAR** register), it transmits an acknowledge after the first address byte is received but no interrupt is generated. IFLG does not be set and the statusdoes not change. Only after the second address byte has been received does the MI2C generate an interrupt, set the IFLG bit and load the status code as described above.

MI2C goes from the master mode to the slave receive mode when arbitration is lost during the transmission of an address, and the slave address and write bit (or the general call address if the CGE bit in the **I2C_SAR** register is set to 1) are received. The status code in the **I2C_SR** register is 68h if the slave address was received or 78h if the general call address was received. The IFLG bit must be cleared to 0 to allow data transfer to continue.

If the AAK bit in the **I2C_CTL register** is set to 1 then an acknowledge bit (low level on SDA) is transmitted and the IFLG bit is set after each byte is received. The **I2C_SR** register contains the status code 80h or 90h if slave receive mode was entered with the general call address. The received data byte can be read from the **I2C_DR** register and the IFLG bit must be cleared to allow the transfer to continue. If a STOP condition or a repeated START condition is detected after the acknowledge bit, the IFLG bit is set and the **I2C_SR** register contains status code A0h.

If the AAK bit is cleared to 0 during a transfer, the MI2C transmits a not-acknowledge bit (high level on SDA) after the next byte is received, and set the IFLG bit. The **I2C_SR** register contains the status code 88h or 98h if the slave receive mode was entered with the general call address. The MI2C returns to the idle state when the IFLG bit is cleared to 0.

Multiply Accumulator

The most significant process in digital signal processing is the multiply-accumulate function, which forms a sum of products.

$$\sum_{i=0}^{n} X_i * Y_i$$

where $\mathbf{X}$ and $\mathbf{Y}$ are vectors (tables of values, one-dimensional arrays) in memory

The Multiply Accumulator block on the eZ80 Webserver performs DSP functions without incurring the control overhead costs associated with a separate DSP.

Feature include:

- A 16x16-bit multiplier whose 32-bit product output goes to one input of an adder. The other input of the adder is the currently selected one of two 40-bit accumulator registers. The output of the adder is the write side of that accumulator.
- Two dual-port RAMs called X and Y. One port of each RAM is 16-bit read-only and feeds one side of the multiplier. The second port is 8-bit read/write and is connected to the microprocessor bus allowing the RAMs to be part of the microprocessor's memory space.
- A set of registers in the microprocessor's I/O space start Multiply Accumulator operation, determine when the Multiply Accumulator has completed a calculation, and retrieve the resulting accumulation. Software can provide calculation parameters to these registers.

Multiply Accumulator Functional Description

The Multiply Accumulator is designed so software can write to its register block using OTI2R instructions and read it using INI2R instructions. INI2R is similar to INIR except that it increments the I/O address in the C register during each cycle. These instructions drive the values in the B register onto the A15-8 lines. The B register shows number of bytes remaining to transfer. The Multiply Accumulator decodes its I/O addresses from the A7-0 lines and detects the last transfer of a block by decoding the A15-8 lines for all 0.

If signals A15-8 write 01H to the **MAC_CTL** register or any of the **MAC _AC0- 3** registers (indicating the last byte to be written) or if the **MAC_ AC4** register is written to regardless of signals A15-8 then one of the following conditions occur:

- If the other bank is in progress, the Multiply Accumulator changes the state of the current bank from empty to ready.
- If the other bank is empty or done, the Multiply Accumulator changes the state of the current bank to in progress and then swaps the banks.

If any of the **MAC_ AC0-3** registers is read with the signals A15-8 equal to 0116 (indicating the last byte to be read) or if the **MAC_ AC4** register is read regardless of signals A15-8, the Multiply Accumulator changes the state of the current bank from done to empty. The Multiply Accumulator swaps the banks when the state of the other bank is done.

Whenever the Multiply Accumulator completes a calculation, it changes the state of the other bank from in progress to done. If the current bank is ready, the Multiply Accumulator swaps the banks, changes the new other bank's state to in progress, and starts performing the new calculation.

If the software reads the state ‘other done, current empty’ from the status register, and it has no next calculation to program, it can write 80h to the status register. This swaps the banks to other empty current done. Software can then access the accumulator result.

Block Diagram

Figure 14 shows the Multiply Accumulator block diagram.

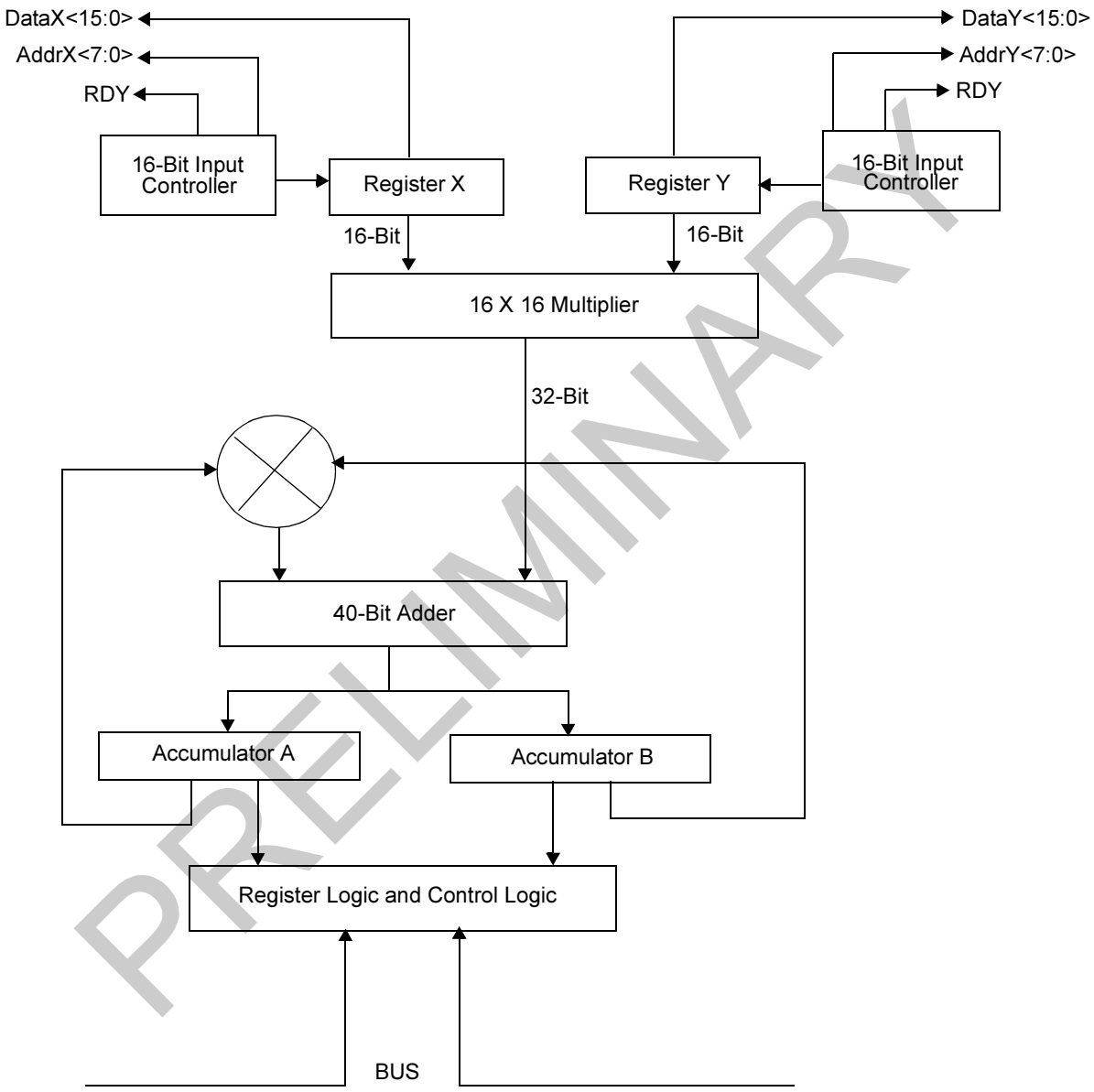


Figure 14. Multiply Accumulator Block Diagram

Register Blocks

The Multiply Accumulator registers are in the eZ80 Webserver's I/O space and can be block loaded using the new OTI2R instruction. OTI2R is similar to OTIR except that it increments the I/O address in the C register as part of each cycle. There are two register blocks in the Multiply Accumulator, one of which is accessible to the processor and the other of which can be used by the Multiply Accumulator, in a ping-pong fashion. Each register block includes the following values. Base indicates comparison for lines A7-4, that are supplied to the Multiply Accumulator by other on-chip logic.

Table 20.Multiply Accumulator Register Descriptions

Register	Function
MAC_XSTART	The address in the X RAM, of the first value to be multiplied.
MACX_XEND	The address in the X RAM, of the end of the circular buffer.
MAC_XRELOAD	The address in the X RAM, of the beginning of the circular buffer. As the Multiply Accumulator performs the calculation, it increments its working addresses to get to the next pair of values. If the X working address matches the **MAC_XEND** register, instead of incrementing, the Multiply Accumulator loads the contents of this register into the X working address.
MAC_YSTART	The address in the Y RAM, of the first value to be multiplied.
MAC_YEND	The address in the Y RAM, of the end of the circular buffer.
MAC_YRELOAD	The address in the Y RAM, of the beginning of the circular buffer. As the Multiply Accumulator performs the calculation, it increments its working addresses to get to the next pair of values. If the Y working address matches the **MAC_YEND** register, instead of incrementing, the Multiply Accumulator loads the contents of this register into the Y working address.
MAC_LENGTH	The number of pairs of values to be multiplied and accumulated.

Table 20.Multiply Accumulator Register Descriptions (Continued)

Register	Function
MAC_CTL	The accumulator is cleared to 0 when this register is written to. This register has the following values: **In Shift:** A 3-bit value that specifies the number of low-order bits that should be appended below the value written to the **MAC_AC0-4** registers. If this field is not zero, and the **MAC_AC4** register is written to, then this number of high-order bits is ignored. **Out Shift:** A 3-bit value that specifies the number of low-order bits of the accumulation that should be discarded by shifting, when the **MAC_AC0-4** registers are read. If this field is non-zero, reading the **MAC_AC4** register includes this number of high-order 0. **Noise:** If In Shift is non-zero, the bit(s) that should be appended below the value written to the **MAC_AC0-4** registers. **Interrupt Enable:** If this bit is 1, the Multiply Accumulator requests an interrupt when it completes the calculation specified in this bank.
MAC_AC0-4	A starting value for the accumulator can be written to these locations. The accumulated value can be read from these locations, using an INIMR instruction.
MAC_SR	Bits 3-2 of this register contain the status of the register bank that's not currently accessible to the eZ80 Webserver processor. Bits 1-0 of this register contain the status of the register bank that is accessible. Both fields are encoded as: 00 Empty 01 Ready 10 In Progress 11 Done Bit 4 of this register identifies the absolute identity of the current register bank. Properly written software has no need to know this, but this facility is included for diagnostics and to allow software to be appropriately written. **Note:** See Table 21 on page 91 for more information on the Status register's non-transient bits.

Table 21.Common States of Bits 3-0 In The Multiply Accumulator Status Register

Bits 3-2: Other Bank	Bits 1-0: Current Bank	Multiply Accumulator Status
00	00	Other Empty, Current Empty. These fields reset to this state. If software writes a new calculation spec to the register bank, the Multiply Accumulator state goes to 10,00.
10	00	Other In Progress, Current Empty. The Multiply Accumulator is working on a calculation. The processor has read the result of any previous calculation, but has not yet loaded the specification of the next calculation into the current register bank. If it does so before the Multiply Accumulator finishes, the state goes to 10,01. If the Multiply Accumulator finishes before the processor specs the next calculation, the state goes to 11,00.
11	00	Other Done, Current Empty. The Multiply Accumulator has completed a calculation. The software has read the result of any previous calculation, but has not yet loaded the specification of the next calculation into the current register bank. If software does so, the state goes to 10,11. If there is no next calculation to be done and software wants to read the result accumulation, it can flip the banks to state 00,11 by writing 80h to the Status Register.
00	11	Other Empty, Current Done. The Multiply Accumulator has completed the calculation specified in this register bank, and the result accumulation can be read from the **MAC AC0-4** registers. When software does so, the state goes to 00,00.

Table 21.Common States of Bits 3-0 In The Multiply Accumulator Status Register

Bits 3-2: Other Bank	Bits 1-0: Current Bank	Multiply Accumulator Status
10	01	Other In Progress, Current Ready. The Multiply Accumulator is working on a calculation, and the microprocessor has loaded the specification of the next calculation into the current register bank. When the Multiply Accumulator finishes with its current calculation, it goes on to the next calculation and the state changes to 10,11.
10	11	Other In Progress, Current Done. The Multiply Accumulator is working on a calculation, and software has not yet read out the result of the previous calculation. If software does this before the Multiply Accumulator finishes, the state changes to 10,00. If the Multiply Accumulator finishes before software reads out the previous result, the state changes to 11,11.
11	11	Other Done, Current Done. The Multiply Accumulator has completed two calculations, but software has not read out the result of either. When it reads out the first result, the state changes to 00,11.

Software Overview

Set up a new calculation

Follow the below procedures to set up a new calculation:

1. Read the status register. If the current status is ready, the Multiply Accumulator has not started the previous calculation, and software has to wait until it does at which time the current status changes to done.
2. If the current status is done, software reads the result accumulation from as many of the **MAC_AC0-3** registers as are needed, as described in the following procedure. (This changes the current status to empty.)
3. If the current status is empty, software sets up the **HL** register pair to point to the block describing the new calculation, the **C** register to the corresponding starting I/O address, and the **B** register to the number of bytes in the block. Then the software should perform an OTI2R instruction.

Retrieve a calculation

Follow the below procedures to retrieve the results of a calculation:

1. Read the status register. If the current status is ready, the Multiply Accumulator hasn't started the last calculation that the software provided, and software has to wait until

the Multiplu Accumulator starts the calculation, at which time the current status changes to done.

2. If the current status is empty and the other status is done, write the 8016 to the status register. This swaps the register banks so that the current status is now done.
3. If both status fields indicate empty, there is no result to retrieve.
4. If the current status is done, software reads as many of the **MAC_AC0-3** registers as are needed. Since the Multiply Accumulator decodes the A15-8 lines to tell when such a transfer is complete, this can be done with an INI2R instruction. Software can also use IN0 instructions to read all five bytes into registers (**MAC_AC4** register last), or can use IN0 to read the first 0-3 bytes into registers, and one of the following two sequences to read the last byte needed:

```
LD BC,base+n+100h or LDA,

INr,c)INA,(base+n)
```

5. Reading the last byte of the result changes the current status to empty unless there is another result to retrieve, in which case the other status is empty and the current status is done.

Interrupt Controller

The interrupt controller takes the interrupts from the internal and external devices and routes them to the eZ80's vectored interrupt function.

A vectored interrupt consists of the following operations.

1. Push the PC (Program Counter) onto the stack.
2. Set the lower 8 bits of the address to the value being provided by the IVS bus and the upper 8 bits within the **I** register.
3. Fetch two bytes from the data bus.
4. Load these two bytes into the program counter and begin executing from this new address.

IRQ Operation

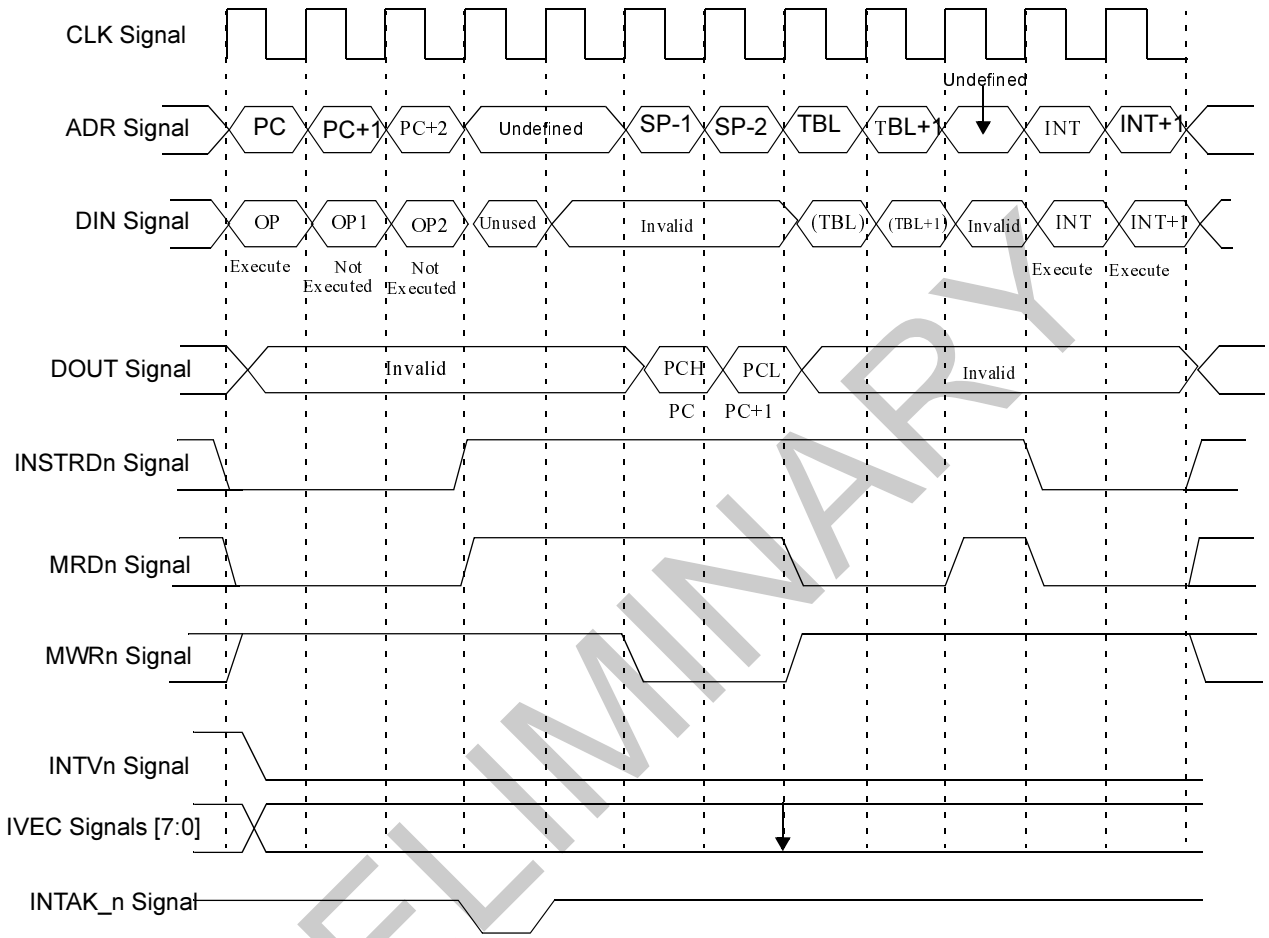
When any of the interrupt request inputs becomes active, an interrupt request to the CPU is generated and a corresponding 8 bit interrupt vector for the highest priority interrupt is placed on vector 7-0. The CPU, at a defined time instance after recognizing irqreqn, reads the vector data applied on its dedicated vector bus. The irqreqn from this block is connected to the intv_n interrupt of the CPU which does

not require an interrupt acknowledge sequence. Figure 15 shows the timing for interupts.

Table 22 lists the vector for each of the interrupt sources. The interrupts are in order of priority with vector 00H being the highest priority interrupt.

Table 22.Interrupt Vector Sources

Vector	Source	Vector	Source
00H	MACC	30H	Port B 5
02H	DMA 0	32H	Port B 6
04H	DMA 1	34H	Port B 7
06H	Timer 0	36H	Port C 0
08H	Timer 1	38H	Port C 1
0AH	Timer 2	3AH	Port C 2
0CH	Timer 3	3CH	Port C 3
0EH	Timer 4	3EH	Port C 4
10H	Timer 5	40H	Port C 5
12H	UZI 0	42H	Port C 6
14H	UZI 1	44H	Port C 7
16H	Port A 0	46H	Port D 0
18H	Port A 1	48H	Port D 1
1AH	Port A 2	4AH	Port D 2
1CH	Port A 3	4CH	Port D 3
1EH	Port A 4	4EH	Port D 4
20H	Port A 5	50H	Port D 5
22H	Port A 6	52H	Port D 6
24H	Port A 7	54H	Port D 7
26H	Port B 0	56H	Reserved
28H	Port B 1	58H	Reserved
2AH	Port B 2	5AH	Reserved
2CH	Port B 3	5CH	Reserved
2EH	Port B 4	5EH	Reserved



Note: No external access for previous command of OP and processing OP.

VCT=Interrupt Vector sampled at the instance shown.

TBL=[(I reg), VCT]

INT=[(TBL),(TBL+1)]

Arrow on ivec[7:0] indicates the instance of sampling of vector data by CPU

Figure 15. Interrupt Timing

Interrupt Logic has two clock latency internally. Interrupt sources are required to be active till the Interrupt Service Routine (ISR) starts.

DMA Controller

The eZ80 Webserver features two DMA channels.

DMA Controller Operation

The DMA controller can be used for direct memory to memory data transfers without CPU intervention. There are two DMA channels, channel 0 and channel 1, each having independent registers. Transfers can be either in burst mode or cycle steal mode.

In burst mode, after the DMA controller gains access to the bus, it maintains control of the bus until the transfer is complete for that channel.

In cycle steal mode, the DMA gains access and transfers only one byte and returns control of the bus to the CPU for eight clock cycles. The DMA then requests the bus and gains access to transfer the next byte. This process continues until the programmed number of bytes are transferred.

When both channels are configured for cycle steal mode, they alternate between each other. Channel 0 performs a byte transfer then releases the bus so that the channel 1 DMA controller can request the bus. The channel 1 DMA controller performs the bus request for eight clock cycles after the bus is released by channel 0. The channel 1 DMA controller performs a byte transfer after taking control of the bus. Figure 16 shows the timing for the DMA.

In all other mode combinations, channel 0 has a higher priority than channel 1.

If DMA channel 0 is programmed in cycle steal mode and DMA channel 1 is programmed in burst mode, DMA channel 1 is not allowed to transfer its data until DMA channel 0 is complete.

NOTE: The DMA channel cannot be used with internal I/O. However, it can be used with external memory-mapped I/O.

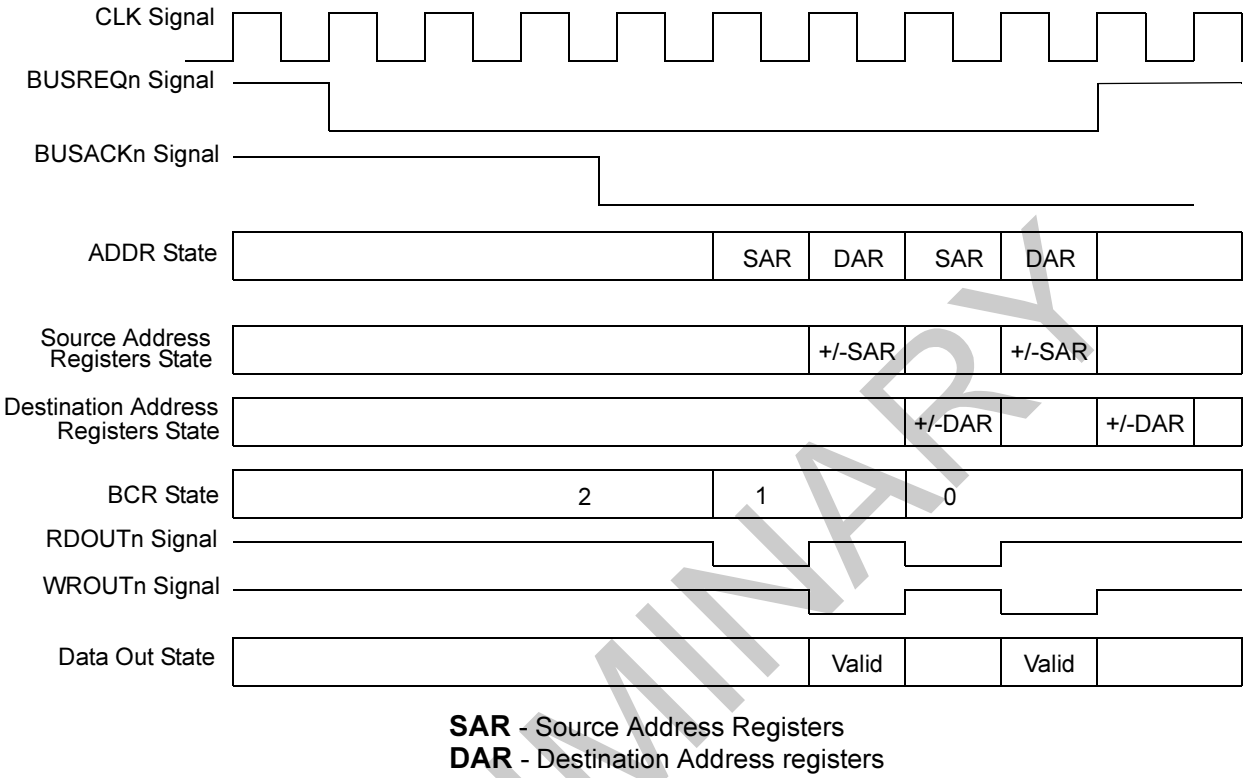


Figure 16. DMA Transfer Without Wait States

The wait_n input signal extends the current memory access from DMA. During memory read/write conditions, wait_n signal is sampled and the current bus access is extended until the wait_n signal becomes inactive (high).

DMA Controller Registers

Table 23 shows the registers used by the DMA controller.

Table 23.DMA Registers

Name	CPU Access	Reset Value	Description
DMA_SARL	R/W	XX	Source Address [7:0]
DMA_SARM	R/W	XX	Source Address [15:8]
DMA_SARH	R/W	XX	Source Address [23:16]
DMA_DARL	R/W	XX	Destination Address [7:0]

Table 23.DMA Registers (Continued)

Name	CPU Access	Reset Value	Description
DMA_DARM	R/W	XX	Destination Address[15:8]
DMA_DARH	R/W	XX	Destination Address[23:16]
DMA_BCL	R/W	00h	Byte Count[7:0]
DMA_BCH	R/W	00h	Byte Count[15:8]
DMA_CTL	R/W	00h	DMA Control

Source Address Register Group

(**DMA_SARL** + **DMA_SARM** + **DMA_SARH**) (CPU Read/Write)

This group of registers holds the 24-bit address of the current source memory location. The register group can automatically be incremented, decremented, or unchanged.

Destination Address Register Group

(**DMA_DARL**+ **DMA_DARM** + **DMA_DARH**) (CPU Read/Write)

This group of registers holds the 24-bit address of the current destination memory location. The register group can automatically be incremented, decremented, or unchanged.

Byte Count Register Pair

(**DMA_BCL** + **DMA_BCH**) (CPU Read/Write)

This pair of registers contains the number of bytes to be transferred. The register pair is decremented after each transfer. One to 65535 bytes can be transferred at a time.

DMA Control Register (DMA_CTL)

(CPU Read/Write)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	DEN	ITEN	Reserved	BRST	DC1	DC0	SC1	SC0
Reset	0	0	0	0	0	0	0	0
CPU access	R/W	R/W	N/A	R/W	R/W	R/W	R/W	R/W

This register is used to control the DMA channels.

Bit 7 **DEN** - DMA Enable

1 - DMA Enabled

0 - DMA Disabled.

This register must be cleared by software when access is completed.

Bit 6 **INTEN** - Interrupt Enable

1 - Interrupt Enabled

0 - Interrupt Disabled

Bit 4 **BRST**- Burst Mode

1 - Burst Mode

0 - Cycle Steal Mode

Bits 3, 2 **DC** - Destination Address Registers Control

00 - Destination Address Registers Unchanged

01 - Destination Address Registers Increments With Each Transfer

10 - Destination Address Registers Decrements With Each Transfer

11 - Reserved

Bits 1, 0 **SC** - Source Address Registers Control

00 - Source Address Registers Unchanged

01 - Source Address Registers Increments With Each Transfer

10 - Source Address Registers Decrements With Each Transfer

11 - Reserved

ZiLOG Debug Interface (ZDI)

Introduction

The ZiLOG Debug Interface (ZDI) provides a built-in debugging interface to the eZ80 CPU. ZDI provides basic in-circuit emulation features such as:

- Examining and modifying internal registers
- Examining and modifying memory
- Starting/stopping the user program
- Setting program and data breakpoints
- Single stepping the user program
- Debugging final product with inclusion of one small connector
- Downloading code into SRAM
- C-Source-Level Debugging using ZiLOG Developer Studio

The above features are built into the silicon and control is provided via a two-wire interface that is connected to the ZPAK emulator. Figure 17 shows a typical setup using a a target board ZPAK and the host PC using ZiLOG Developer Studio.

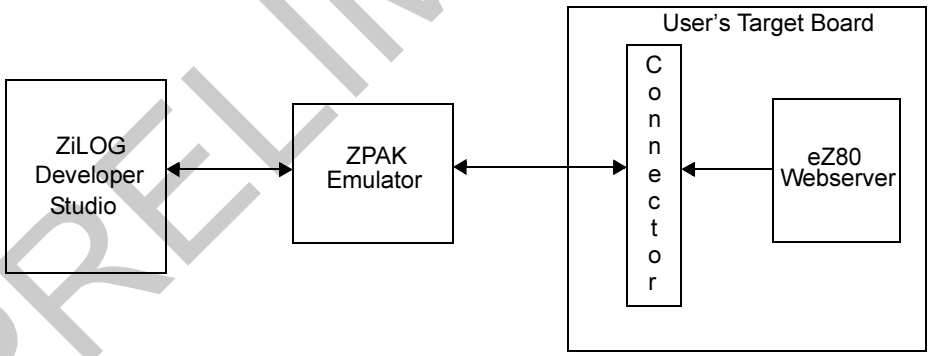


Figure 17. Typical ZDI Debug Setup

ZDI General Description

The ZDI block for the eZ80 Webserver provides increased functionality from the previous versions. ZDI allows reading and writing of most of the internal registers without disturbing the state of the machine. New features allow reads and writes to memory to occur as fast as the ZDI can down/up load data.

> **Note:** The interfaces maximum baud rate is the eZ80 clk divided by 2. The Write Data, Instruction Store and Write Memory Registers are shared. Since each of these functions are independent of each other to save space register sharing was used.

ZDI Interface

ZDI supports a bi-directional serial protocol. The protocol defines any device that sends data as a transmitter and the receiving device as the receiver. The device controlling the transfer is the master and the device being controlled is the slave. The master always initiates the data transfers, and provides the clock for both transmit and receive operations. ZDI is considered a slave in all applications.

Figure 18 shows the schematic for building a connector on a target board. This connector allows the user to connect directly to the ZPAK emulator using a six-pin header.

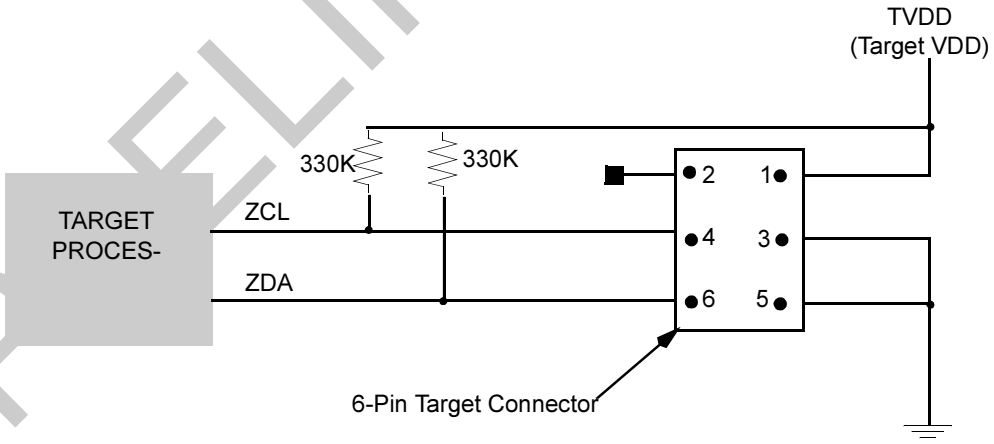


Figure 18. Schematic For Building a Target Board ZPAK Connector

ZDI Clock and Data Conventions

Data states on the ZDA line can change only during ZCL LOW, ZDA state changes during ZCL HIGH is reserved for indicating the start condition.

Data is shifted into and out of ZDI with the MSB (Bit 7) of each byte being first in time, and the LSB (Bit 0) the last in time. Data is shifted out on the falling edge of ZCL and shifted in on the rising edge of ZCL, as shown in Figure 19.

All information is passed between the master and slave in 8-bit (byte) units. Each byte is transferred with nine clock cycles, eight to shift the data and the ninth for internal operations.

When an operationis completed, the master stops during the ninth cycle with the ZCL signal HIGH.

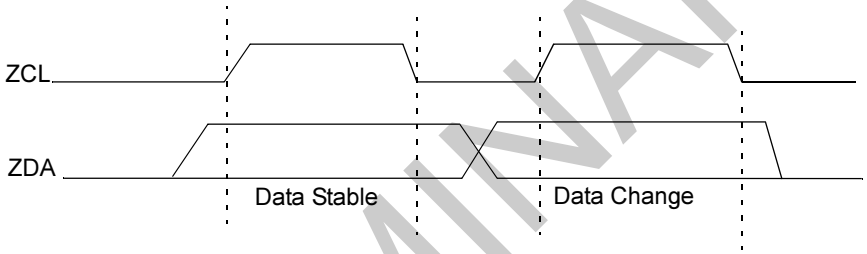


Figure 19. Data Validity

ZDI Start Condition

All commands are preceded by the start condition, which is a HIGH to LOW transition of ZDA when ZCL is HIGH. The ZDI continuously monitors the ZDA and ZCL lines for the start condition and does not respond to any command until this condition has been met. The master pulls ZDA LOW, with ZCL HIGH, to indicate start condition. A LOW to HIGH transition of ZDA while the ZCL is HIGH has an on effect. Figure 20 shows a diagram of the ZDI start condition.

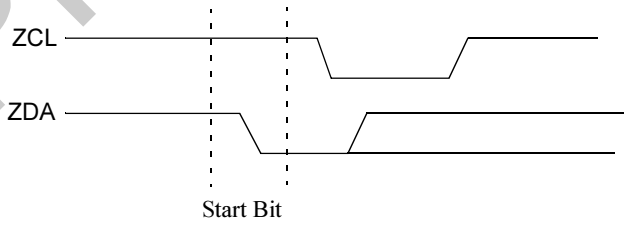


Figure 20. ZDI Definition of Start

Device Addressing

Following a start condition the master must output the ZDI register address. The format for address is seven bits of address, followed by one bit for read or write control. ZDI executes a read or write operation depending on the state of the R/W bit. The register address is available from the ZDI at the end of the seventh cycle. Figures 21 and 22 shows the address table and timing for ZDI.

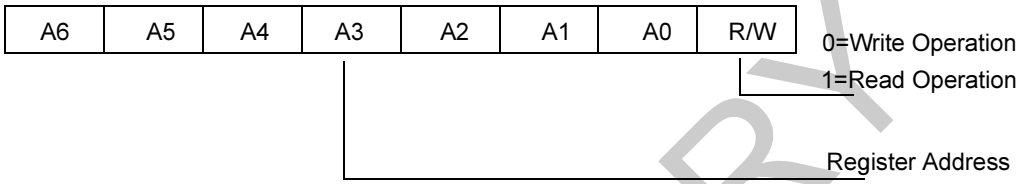


Figure 21. Address Table

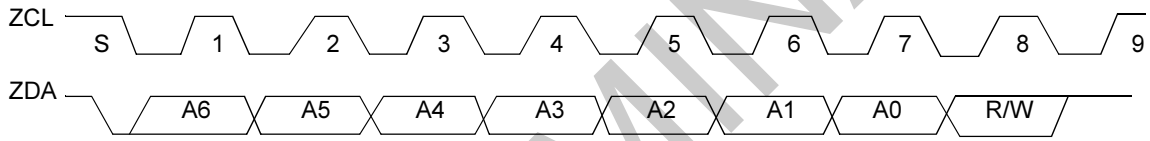


Figure 22. Address Timing

ZDI Write Operations

Byte Write

Upon receipt of the byte address and write control, ZDI shifts the next byte on the next eight ZCL cycles. ZDI internal write occurs during the last half of cycle eight. The master terminates activity after eight clock cycles.

Block Write

The block write operation is initiated the same as the byte write operation, but instead of terminating the write operation after the first data byte is transferred, the master can continue to transmit additional bytes of data.

After the receipt of each byte the register address increments by one. At the end of the address space the address rolls over to 0. Figure 23 shows the ZDI write timing.

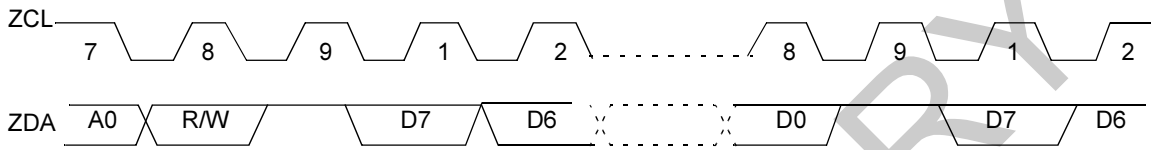


Figure 23. ZDI Write Timing

ZDI Read Operations

Byte Read

Read operations are initiated in the same manner as write operations with the exception that the R/W bit of the slave address is set to 1. Upon receipt of a slave address with the R/W bit set to 1, ZDI loads the selected data into the shifter at the beginning of the first cycle. The MSB of the shifter drives the ZDA pin. The read is terminated by the master halting the ZCL after the eighth data bit clock.

Block read is initiated the same as a byte read, however the master now continues to clock in the next byte from the ZDI and the ZDI continues to output data. The address counter increments with each read, and rolls over to 1 when it reaches the end of its address space. Figure 24 shows a diagram of ZDI's read timing.

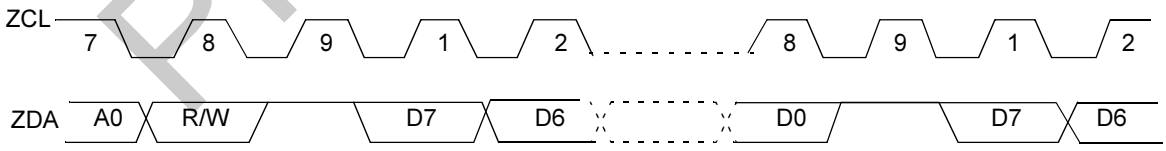


Figure 24. ZDI Read Timing

eZ80 Webserver Internal ZDI Registers

ZDI Read Registers

Table 24 lists the registers which can be read by the ZDI interface.

Table 24.ZDI Read Registers

Address	Register	Reset Value
00h	eZ80 Webserver ID Register Low	05h
01h	eZ80 Webserver ID Register High	00h
02h	eZ80 Webserver Rev Register	00h
03h	Status Register	00h
04-0Fh	Reserved	
10h	Read Register Low	XXh
11h	Read Register High	XXh
12h	Read Register Upper	XXh
20h	Read Memory Value	XXh

ZDI Register ID and Rev Registers

The first three registers are used to identify the revision and the part to which the interface is communicating. The first two registers contain 0005h value to signify that this is an eZ80 Webserver. The third register changes from revision to revision.

ZDI Status Register

(CPU Read only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	ZDI Active	Reset Pending	Low Power Mode	ADL	Mixed ADL	IEF1 Flag	Reserved	Reserved
Reset	0	0	0	0	0	0	0	0
CPU access	R	R	R	R	R	R	R	R

The ZDI Status register at address 03H shows the current state of the eZ80 Webserver processor.

Bit 7 **ZDI Active** - A 1 in this bit signifies that the eZ80 Webserver is now in ZDI Mode.

Bit 6 **Reset Pending** - A 1 in this bit means that the eZ80 Webserver is going through a system reset.

Bit 5 **Low Power Mode** - A 1 in this bit means that the eZ80 Webserver is in a low power mode such as halt, SLP or STNBY.

Bit 4 **ADL** - This bit reflects the current state of the ADL bit.

Bit 3 **Mixed ADL** - This bit reflects the current state of the Mixed ADL bit.

Bit 2 **IEF1 flag** - This bit reflects the state of the IEF1 flag. If a wait to the eZ80 Webserver is asserted then the last value of the IEF1 flag is latched until the current wait is released.

Bits 1, 0 These bits are reserved for future use.

ZDI Read Memory Register

When a read is executed from the Read Memory Register, located at 20h, the eZ80 Webserver fetches the data from the memory address currently pointed to by the program counter and the program counter is incremented. The ZDI internal address increment is disabled and the programmer can continue to read bytes without resetting the address. To execute this procedure perform the following steps:

1. Write, Read/Write Control Register with a 0bh. This fills the one byte pipeline.
2. Read as many bytes as you want to download from address 20h. Write the address once, since there is no auto increment when address 20h is read.

ZDI Read Register Low, High and Upper

These three registers contain the value requested by the Read Control Register. This data is only valid while in ZDI break mode and if the instruction has just been read by a request from the Read Control Register.

ZDI Write Registers

Table 25 lists the registers which can be written to by the ZDI interface.

Table 25.ZDI Write Registers

Address	Register	Register Value
00h	Address Match Low 0	XXh
01h	Address Match High 0	XXh
02h	Address Match Upper 0	XXh
04h	Address Match Low 1	XXh
05h	Address Match High 1	XXh
06h	Address Match Upper 1	XXh
08h	Address Match Low 2	XXh
09h	Address Match High 2	XXh
0Ah	Address Match Upper 2	XXh
0Ch	Address Match Low 3	XXh
0Dh	Address Match High 3	XXh
0Eh	Address Match Upper 3	XXh
10h	Break Control Register	00h
11h	Master Control Register	00h

Table 25.ZDI Write Registers (Continued)

Address	Register	Register Value
13h	Write Data Low	XXh
14h	Write Data High	XXh
15h	Write Data Upper	XXh
16h	Read/Write Control Register	00h
21h	Instruction Store 4	XXh
22h	Instruction Store 3	XXh
23h	Instruction Store 2	XXh
24h	Instruction Store 1	XXh
25h	Instruction Store 0	XXh
30h	Write Memory Register	XXh

ZDI Address Match Registers

The address match registers (see Table 25 on page 107) are used for setting breakpoints. When the Break On Address bit is set in the ZDI control register, the addresses in the Address Match Upper, High and Low registers are compared with the current eZ80 Webserver address. If a match is found, ZDI issues a break to the eZ80 Webserver placing the processor in a ZDI mode pending further instructions.

> **Note:** Address match does not check to see if the current address is the first op-code fetch. If the address is not the first op-code fetch the ZDI break is executed at the end of the instruction in which it is executed. There are four sets of address match registers. They can be used in conjunction with each other to breakpoint on branching instructions.

ZDI Break Control Register

(CPU Write only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Break on Next Instruction	Break On Address Match 3	Break On Address Match 2	Break On Address Match 1	Break On Address Match 0	Ignore Low On Address Break 1	Ignore Low On Address Break 0	Single Step
Reset	0	0	0	0	0	0	0	0
CPU access	W	W	W	W	W	W	W	W

The control register, located at 10h, is used to set breakpoints, or cause the part to be reset by the ZDI.

Bit 7 **Break On Next Instruction** - If this bit is 1, ZDI asserts its break function at the start of the next instruction. Since the eZ80 Webserver has instructions which have multiple op-codes, this function only breaks on the first opcode. This bit stays 1 until a 0 is written to it by an additional Control Register Write. If both the SCL and SDA signals are held low during RESET then this bit is 1 and breaks on the first instruction.

Bit 6 **Break On Address 3** - If this bit is 1, ZDI asserts its break function when the eZ80 Webserver address matches the **Address Match 3** Registers. Breaks only happen on an instruction boundary. If the address is not the beginning of an instruction, then the break occurs at the end of the current instruction. The break is implemented by setting the Break On Next Instruction bit. The programmer has to write the Break On Next Instruction bit down to release it from the break.

Bit 5 **Break On Address 2** - If this bit is 1, ZDI asserts its break function when the eZ80 Webserver address matches the **Address Match 2** Registers. Breaks only happen on an instruction boundary. If the address is not at the beginning of an instruction, then the break occurs at the end of the current instruction. The break is implemented by setting the Break On Next Instruction bit. The programmer has to write the Break On Next Instruction bit down to release it from the break.

Bit 4 **Break On Address 1** - If this bit is 1, ZDI asserts its break function when the eZ80 Webserver address matches the **Address Match 1** Registers. If the Ignore low on address-break-one is 1 then the

lower eight bits are not used in the address compare. Breaks only happen on an instruction boundary. If the address is not at the beginning of an instruction, then the break occurs at the end of the current instruction. The break is implemented by setting the Break On Next Instruction bit. The programmer has to write the Break On Next Instruction bit down to release it from the break.

Bit 3 **Break On Address 0** - If this bit is 1, ZDI asserts its break function when the eZ80 Web ServereZ80 Webserver address matches the **Address Match 0** Registers. If the Ignore low on address break 0 is 1, the lower eight bits are not used in the address compare. Breaks only happen on an instruction boundary. If the address is not at the beginning of an instruction, then the break occurs at the end of the current instruction. The break is implemented by setting the Break On Next Instruction bit. The programmer has to write the Break On Next Instruction bit down to release it from the break.

Bit 2 **Ignore Low On Address Break I** - If this bit is 1 the compare for Break On Address 1 only checks the upper 16 bits. This allows a whole 256 page address space to cause a breakpoint.

Bit 1 **Ignore Low On Address Break 0** - If this bit is 1 the compare for Break On Address 0 only checks the upper 16 bits. This allows a whole 256 page address space to cause a breakpoint.

Bit 0 **Single Step** - Setting this bit causes the processor to execute one instruction and then return to the ZDI break state. The Operation Complete status bit resets when this bit is 1. When the instruction is completed, the Operation Complete bit is 1.

ZDI Read/Write Control Register

This register, located at 0Dh, is used to select certain functions from the eZ80 Webserver ZDI interface. When this register is written, the eZ80 Webserver immediately executes the instruction. You can check the Operation Complete bit in the status register to verify the operation has been completed. When a read operation is done, the requested values can be read out of the Read Data registers. When a write operation is requested the data is taken from the Write Data registers. Table 26 lists the instructions that can be written to this register.

Table 26.ZDI Read/Write Control Register

Hex Value	Command	Hex Value	Command
00	Read AF MBASEA Low Byte F High Byte MBASE Upper	80	Write AF A Low Byte F High Byte
01	Read BC	81	Write BC
02	Read DE	82	Write DE
03	Read HL	83	Write HL
04	Read IX	84	Write IX
05	Read IY	85	Write IY
06	Read SP	86	Write SP
07	Read PC	87	Write PC
08	Set ADL	88	Reserved
09	Reset ADL	89	Reserved
0A	EXX All Including AF	8A	Reserved
0B	Read Memory From Current PC Value Increment PC	8B	Write Memory From Current PC Value Increment PC

Note: The prime register cannot be read directly. The ZDI programmer must execute the exchange instruction before reading the additional values.

Master Control Register

(CPU Write only)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Reset eZ80	Wake up eZ80	Generate BUSREQ	Reserved	Reserved	Reserved	Reserved	Reserved
Reset	0	0	0	0	0	0	0	0
CPU access	W	W	W	W	W	W	W	W

This register, located at 11h, is used to control the operation of the eZ80 Webserver. It has the capability of forcing a reset, waking up the eZ80 Webserver if it is in one of the power down modes or it can force a BUSREQ to have the eZ80 Webserver release the bus. These functions are currently not implemented. To implement the reset and wake functions you need a power on reset and some type of clock control circuitry.

Bit 7 **Reset eZ80 Webserver** - If this bit is 1 ZDI issues a reset to the eZ80 Webserver. During this time the ZDI reset is blocked. This bit automatically resets at the end of the reset period.

Bit 6 **Wake Up eZ80** - If this bit is 1, it causes the eZ80 Webserver to wake up from sleep mode by asserting an interrupt to the eZ80 CPU. If interrupts are enabled, the eZ80 responds by performing an interrupt acknowledge cycle. The eZ80 goes to the address which is in the two bytes pointed to by the **I** register plus **I low** register plus vector. This bit automatically resets after the eZ80 Webserver exits sleep mode.

Bit 5 **Generate BUSREQ** - If this bit is 1 it generates a BUSREQ to the eZ80 until it is reset by a write to this register or a reset occurs.

Bits 4-0 Reserved.

Write Data Low, High and Upper Addr Registers

These three registers (see Table 25 on page 107) are used to store the data that will be written when a write instruction is sent to the Read/Write Control Register. The Read/Write Control register has been strategically placed at address 16 so the programmer can write the data and the command in one transfer.

Instruction Store 4-0 Registers

These registers (see Table 25 on page 107) can be written with instruction data. When **Instruction Store 0** is written the eZ80 Webserver, exits ZDI break state and execute a single instruction. The information for that instruction comes from the **Instruction Store** registers. The **Instruction Store Register 0** is the first byte fetched, followed by Store Register 1, 2, 3 and 4. Only the bytes the processor requires to execute the instruction need to be stored in these registers.

> **Note:** The **Instruction Store 0** register is at a higher address than the other **Instruction Store** registers. This feature allows the use of the ZDI auto-address increment function to load up and execute an instruction with a single data stream from the ZDI Loader.

Write Memory Register

A write to this register, located at 30h, causes the eZ80 Webserver to write the data in this register to the current address in the program counter. The program counter increments after the write. The ZDI internal address increment is disabled when this register is accessed. This allows the programmer to write any number of bytes by writing to this address and then writing any number of data bytes.

Perform the following steps for downloading:

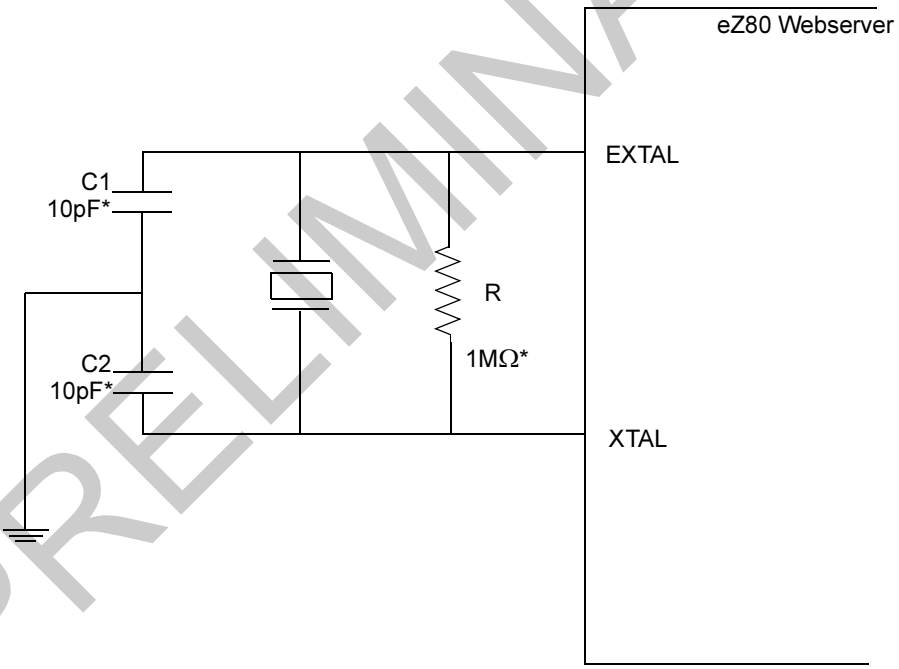
1. Load PC with start value.
2. Write data to address 30h. Write as many bytes as needed without changing the address since the auto increment function of the ZDI is disabled during a write to this register.

Onchip Crystal Oscillator

The eZ80 Webserver has an on-chip crystal oscillator that supplies clocks to both the internal eZ80 CPU core and peripherals and to the external pin. The clock circuitry uses three dedicated pins: EXTAL. XTAL and PHI.

The external clock/crystal (EXTAL) input provides two clock generation options. EXTAL may be used to interface the internal oscillator to an external crystal (see Figure 25). Typical circuit parameters are C1 = C2 = 10 pF and R = 1MΩ using a parallel resonant crystal.

EXTAL can also accept a CMOS-level clock input. The crystal output (XTAL) connects the internal crystal oscillator to an external crystal. If an external clock is used, XTAL should be left unconnected. The PHI pin, which drives the high-speed system clock, may be used to synchronize other peripherals to the eZ80 Webserver system clock.



Note: *:The values given need to be tuned for the crystal and the frequency of operation. These are typical values.

Figure 25. Crystal Oscillator

Electrical Characteristics

Absolute Maximum Ratings

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This rating is a stress rating only. Operation of the device at any condition outside those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Table 27 shows the absolute maximum ratings.

Table 27. Absolute Maximum Ratings

Parameter	Min	Max	Units	Notes
Ambient Temperature under Bias	TBD	TBD	TBD	
Storage Temperature	TBD	TBD	TBD	
Voltage on any Pin with Respect to V_{SS}	TBD	TBD	TBD	
Voltage on V_{DD} Pin with Respect to V_{SS}	TBD	TBD	TBD	
Total Power Dissipation	TBD	TBD	TBD	
Maximum Current out of V_{SS}	TBD	TBD	TBD	
Maximum Current into V_{DD}	TBD	TBD	TBD	
Maximum Current on Input and/or Inactive Output Pin	TBD	TBD	TBD	
Maximum Output Current	TBD	TBD	TBD	

Notes:
1. Operating temperature is specified in DC Characteristics
2. This applies to all pins except where noted otherwise. Maximum current through a pin is specified below.

DC Characteristics

Table 28 shows the DC characteristics for the eZ80 Webserver.

Table 28.DC Characteristics

		0^0C	85^0C	Conditions	
Symbol	**Parameter**	**Minimum**	**Maximum**	**V_{DD}**	
V_{IL}	Low Level Input Voltage	-0.5V	0.8V	2.7V - 3.6V	Guaranteed Input Low Voltage
V_{IH}	High Level Input Voltage	2.0V	V_{DD}+0.5V	2.7V - 3.6V	Guaranteed Input High Voltage
V_{OL}	Low Level Output Voltage		0.4 V	2.7V	I_{OL}, 2 to 8 mA Depending on the Pin Drive
V_{OH}	High Level Output Voltage	2.4V		2.7V	$I_{OH,}$ 2 to 8 mA Depending on the Pin Drive
I_{IL}	Input Leakage Current	TBD	TBD	TBD	TBD
I_{DD}	Supply Current (Normal Operation)	TBD		55	TBD

AC Characteristics

All the timings correspond to a load of 50 pF on all outputs.

External Reads Timing

Figure 26 and Table 29 shows the timing for external reads.

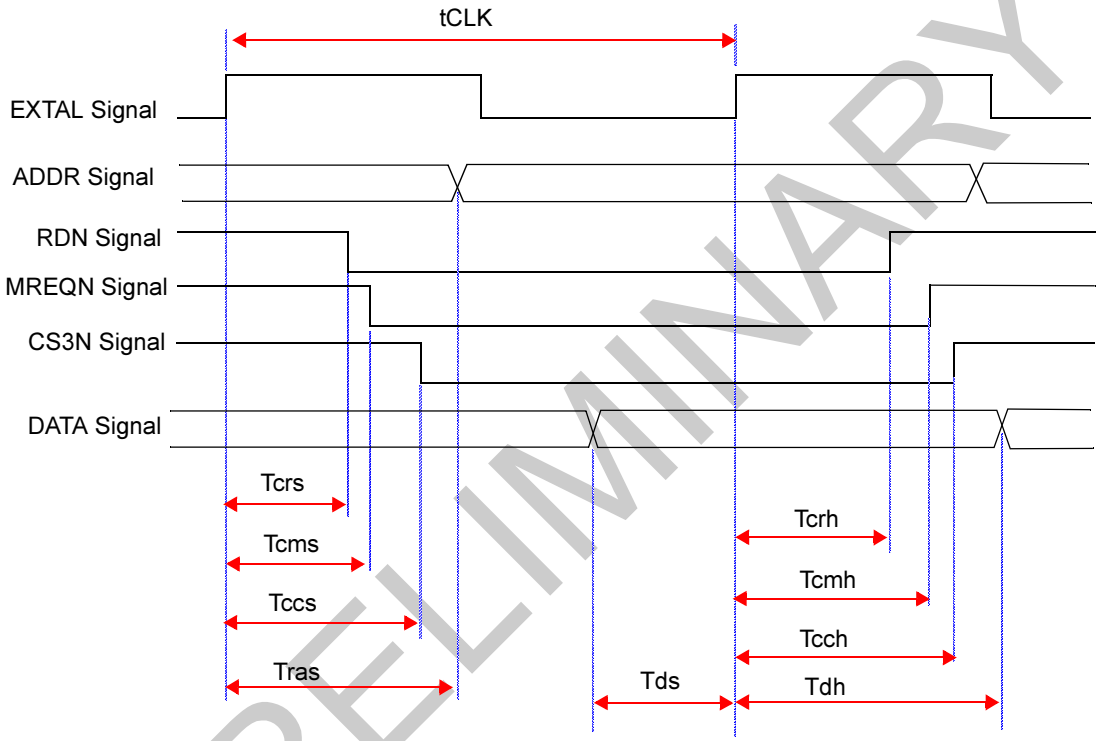


Figure 26. eZ80 Webserver External Memory Read Timing Diagram

Table 29.eZ80 Webserver External Read Timings

		Delay (ns)	
Parameter	**Abbreviation**	**Min.**	**Max.**
Tcrs	Clock to RDN Assertion Delay	3.03	10.07
Tcms	Clock to MREQN Assertion Delay	3.08	10.14
Tccs	Clock to CSnN Assertion Delay	2.3	16.06

Table 29.eZ80 Webserver External Read Timings (Continued)

		Delay (ns)	
Parameter	**Abbreviation**	**Min.**	**Max.**
Tcrh	Clock to RDN Deassertion Delay	2.9	9.65
Tcmh	Clock to MREQN Deassertion Delay	2.96	9.71
Tcch	Clock to CSnN Deassertion Delay	1.95	15.47
Tras	Read Address Stabilization Time	2.85	11.0
Tds	Data Setup Required With Respect To Clock	-0.17	-0.52
Tdh	Data Hold Time Required With Respect To Clock	0.58	1.05

External Writes Timing

Figure 27 and Table 30 shows the timing for external writes.

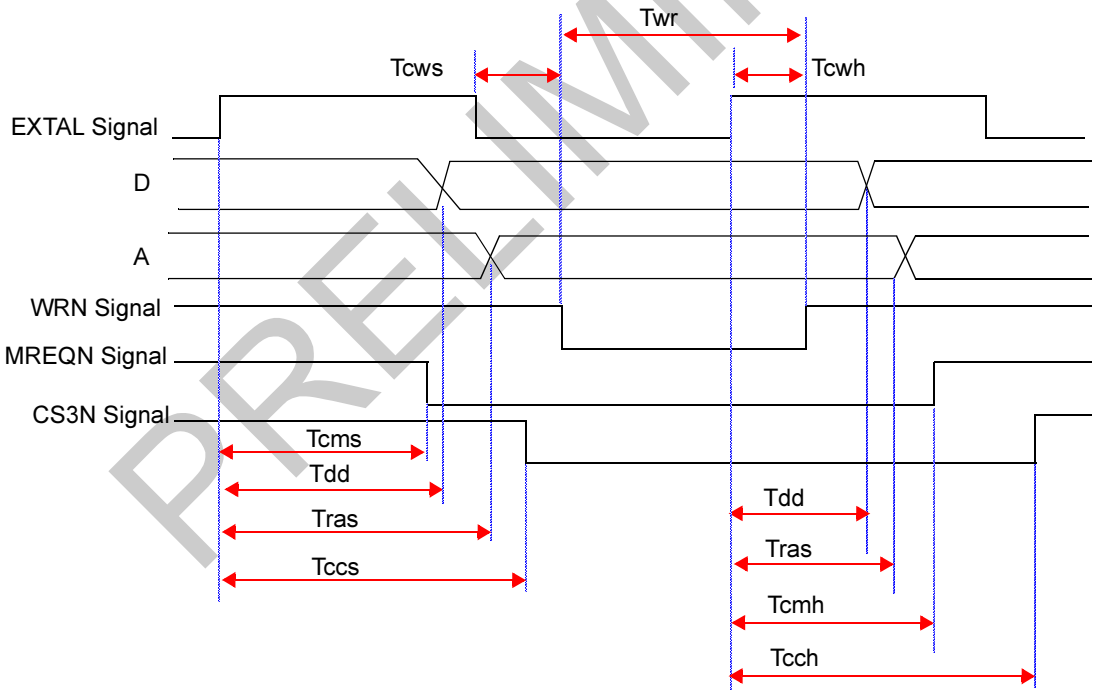


Figure 27. eZ80 Webserver External Memory Write Timing Diagrams

Table 30.eZ80 Webserver External Write Timings

		Delay (ns)	
Parameter	**Abbreviation**	**Min.**	**Max.**
Tcws	Clock to WRN Assertion Delay	2.0	4.78
Tcwh	Clock to WRN Deassertion Delay	1.73	3.95
Tds	Data Setting Delay From Clock	3.18	13.34
Tcms	Clock to MREQN Assertion Delay	3.08	10.14
Tccs	Clock to CSnN Assertion Delay	2.3	16.06
Tdd	Data Delay With Respect to Clock.	3.18	13.34
Tras	Read Address Stabilization Time	2.85	11.0
Tcmh	Clock to MREQN Deassertion Delay	2.96	9.71
Tcch	Clock to CSnN Deassertion Delay	1.95	15.47

External I/O Reads Timing

Figure 28 and Table 31 shows the timing for external I/O reads.

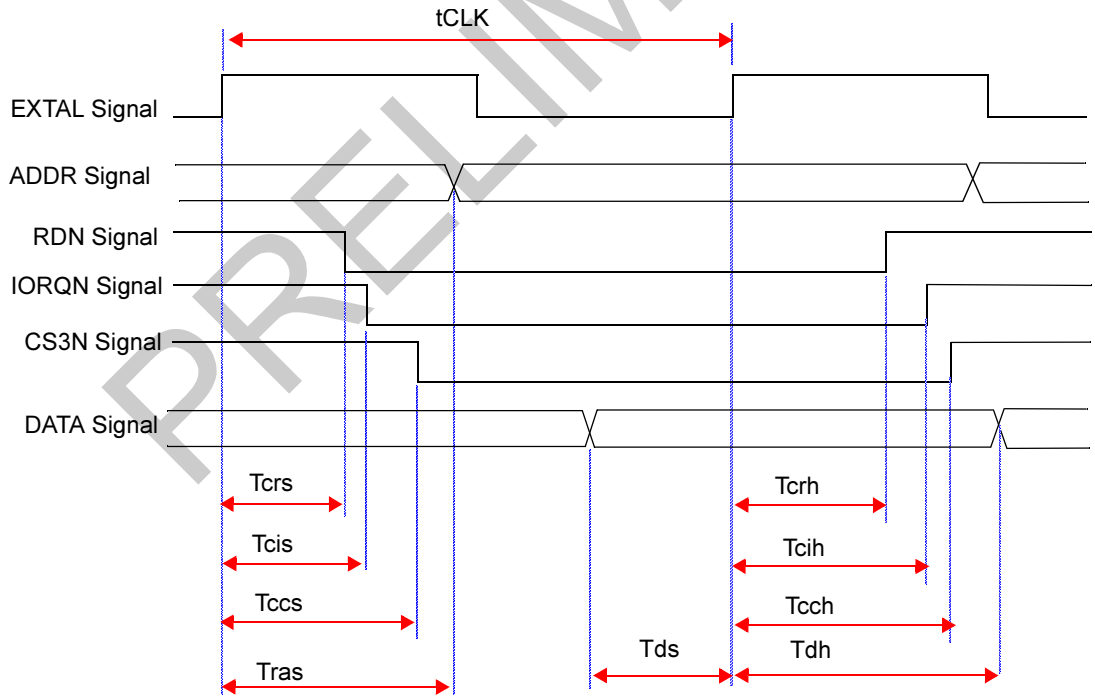


Figure 28. eZ80 Webserver External I/O Read Timing Diagram

Table 31.eZ80 Webserver External I/O Read Timings

		Delay (ns)	
Parameter	**Abbreviation**	**Min**	**Max**
Tcrs	Clock to RDN Assertion Delay	3.03	10.07
Tcms	Clock to IORQN Assertion Delay	3.08	10.14
Tccs	Clock to CSnN Assertion Delay	2.3	16.06
Tcrh	Clock to RDN Deassertion Delay	2.9	9.65
Tcih	Clock to IORQN Deassertion Delay	2.96	9.71
Tcch	Clock to CSnN Deassertion Delay	1.95	15.47
Tras	Read Address Stabilization Time	2.85	11.0
Tds	Data Setup Required With Respect to Clock	-0.17	-0.52
Tdh	Data Hold Time Required With Respect to Clock	0.58	1.05

External I/O Writes Timing

Figure 29 and Table 32 shows the timing for external I/O writes.

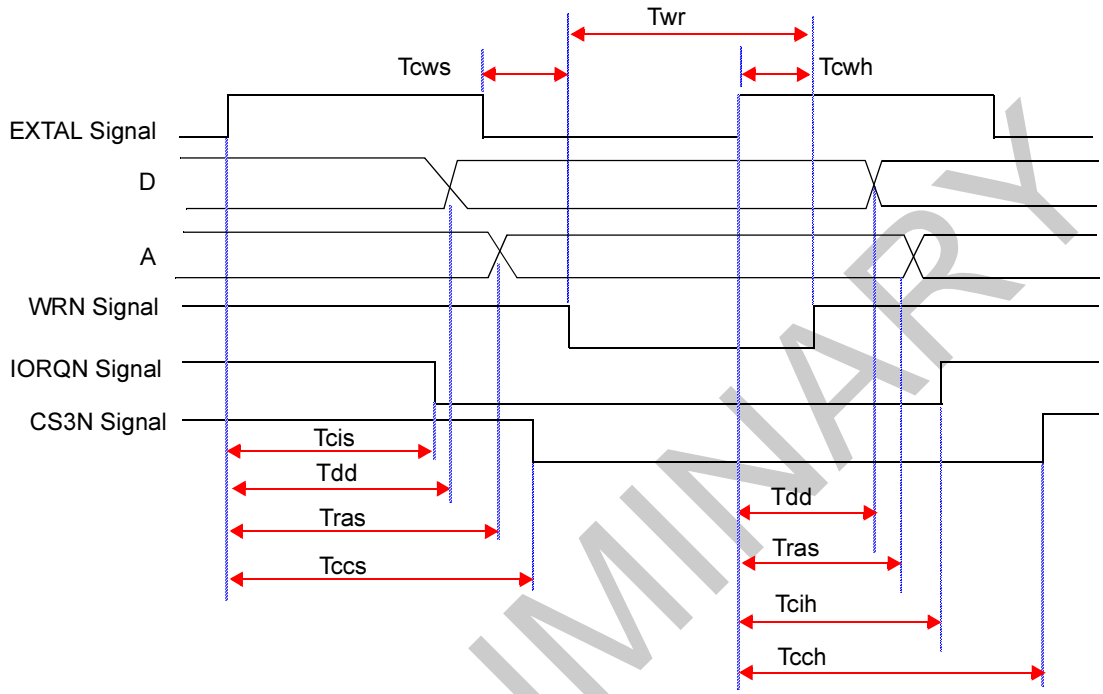


Figure 29. eZ80 Webserver External I/O Write Timing Diagram

Table 32.eZ80 Webserver External I/O Write Timings

		Delay (ns)	
Parameter	**Abbreviation**	**Min**	**Max**
Tcws	Clock to WRN Assertion Delay	2.0	4.78
Tcwh	Clock to WRN Deassertion Delay	1.73	3.95
Tds	Data Setting Delay Grom Clock	3.18	13.34
Tcis	Clock to IORQN Assertion Delay	3.08	10.14
Tccs	Clock to CSnN Assertion Delay	2.3	16.06
Tdd	Data Delay With Respect to Clock.	3.18	13.34

Table 32.eZ80 Webserver External I/O Write Timings (Continued)

		Delay (ns)	
Parameter	**Abbreviation**	**Min**	**Max**
Tras	Read Address Stabilization Time	2.85	11.0
Tcih	Clock to IORQN Deassertion Delay	2.96	9.71
Tcch	Clock to CSnN Deassertion Delay	1.95	15.47

SPI Timing

Figures 30 and 31 and Table 33 shows the timing for the SPI reads.

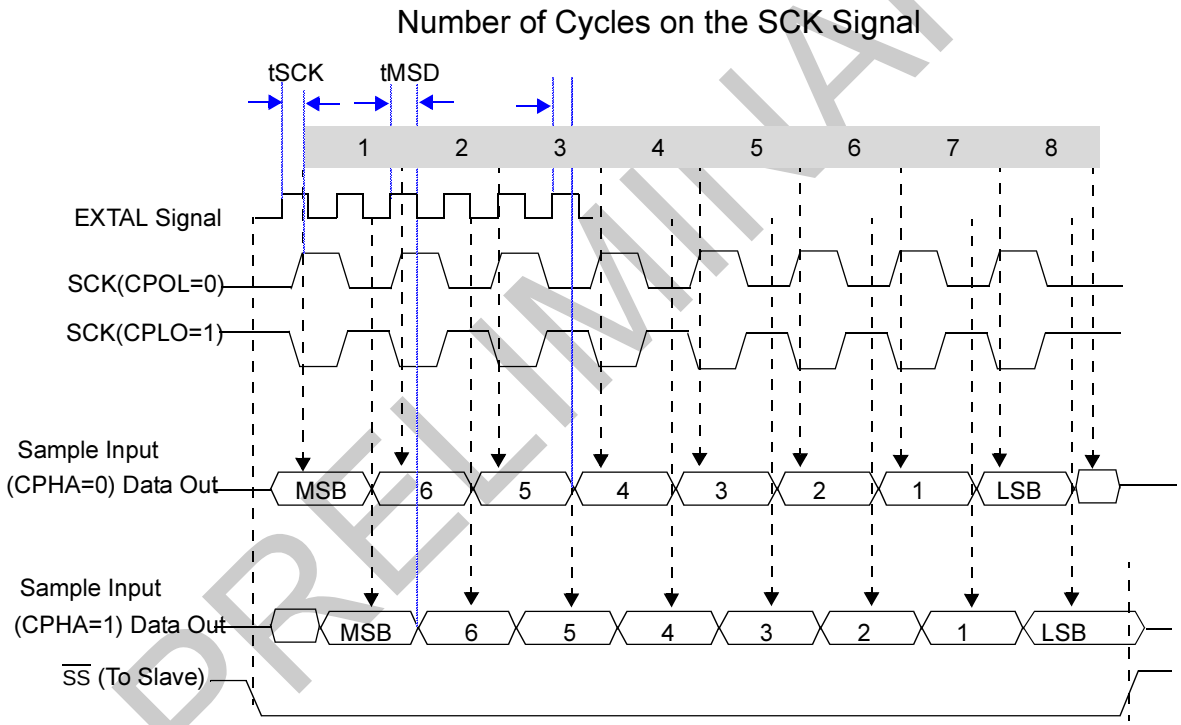


Figure 30. SPI Timing Diagram (1)

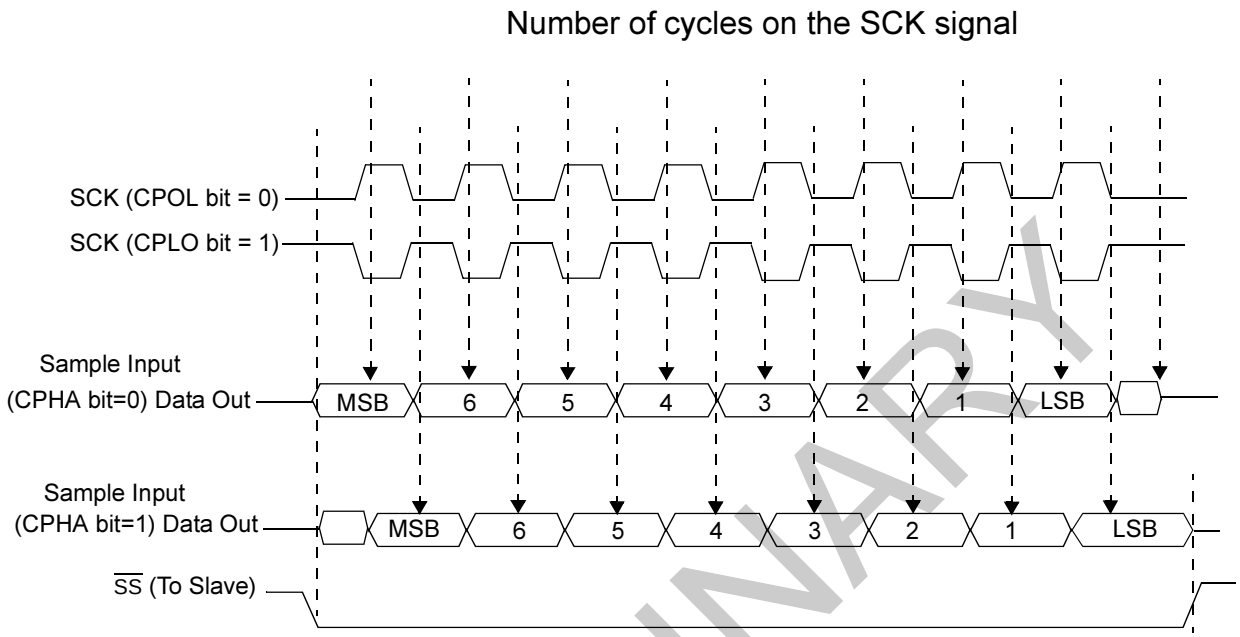


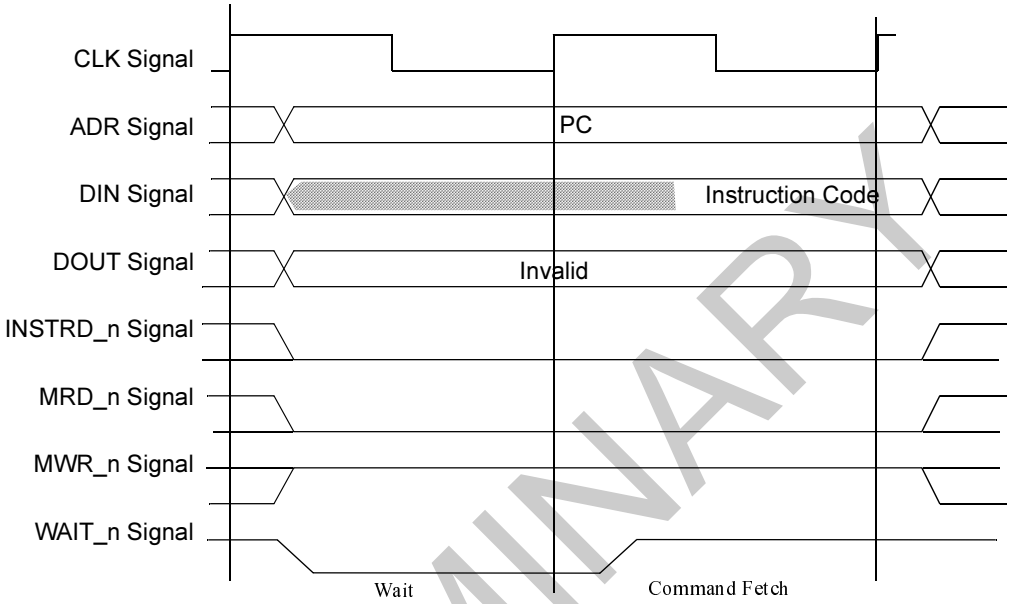
Figure 31. SPI Timing Diagram (2)

Table 33.SPI Timings (ns)

Parameter	Abbreviation	Delay
tSCK	Clock to SCK Delay in Master Mode.	11.9(max)
tMSD	Clock to MOSI Delay in Master Mode.	11.9(max)
tSSD	Clock to MISO Delay in Slave Mode.	11.9(max)

Wait State Timing

Figure 32 shows the timing for the wait state generator.

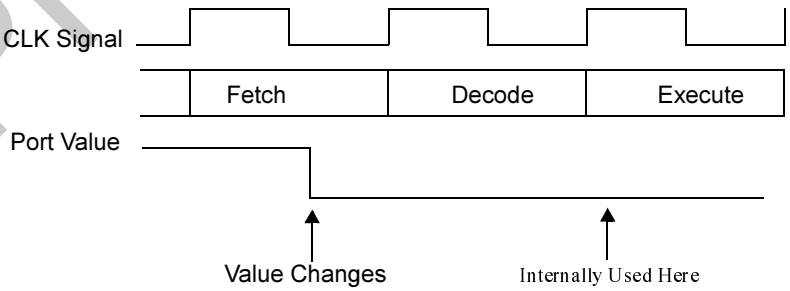


Note: This diagram indicates the RD/WR strobe signal extension using one wait state. Refer to separate timings for external memory timings

Figure 32. Wait State Timing Diagram

Port Timings

Figure 33 shows port sampling timings.



Note: This diagram shows when the external port signal is used.

Figure 33. Port sampling timings

Table 34 shows external bus request timings.

Table 34.External Bus Request Timings

		Delay (ns)	
Parameter	**Abbreviation**	**Min.**	**Max.**
Tba	Clock to Bus Acknowledge Delay	3.09	8.52
Tbrs	Busreqn Setup Time With Respect to Clock	-0.94	-1.97
Tbrh	Busreqn Hold Time With Respect to Clock	1.16	2.86
Tas	Address Setup Time With Respect to Clock	0.15	2.8
Tah	Address Hold Time With Respect to Clock	0.84	0
Tmrs	Mreqn Setup Time With Respect to Clock	0.2	1.08
Tmrh	Mreqn Hold Time With Respect to Clock	-0.02	-0.58
Tirs	Iorqn Setup Time With Respect to Clock	0.79	2.74
Tirh	Iorqn Hold Time With Respect to Clock	-0.72	2.08
Tacs	Address To Chip Select Delay	3.41	8.53

Figure 34 shows port input and output timings.

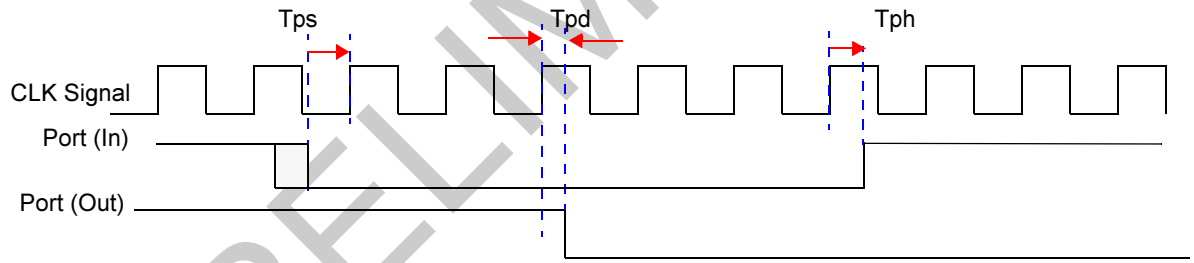


Figure 34. Port Input and Output Timings

Table 35 shows port timings.

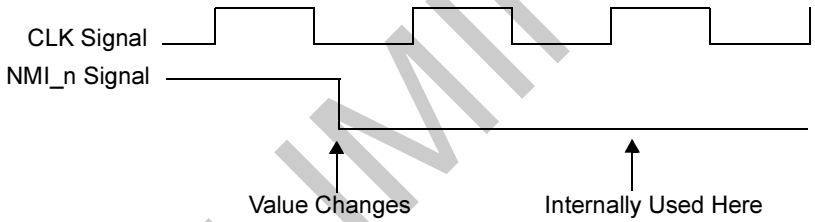
Table 35.Port Timings

		Delay (ns)	
Parameter	**Abbrevation**	**Min.**	**Max.**
Tpad	Port A Output Delay	2.44	9.66
Tpbd	Port B Output Delay	2.38	10.16
Tpcd	Port C Output Delay	2.43	11.31
Tpdd	Port D Output Delay	2.43	12.08

Table 35.Port Timings (Continued)

		Delay (ns)	
Parameter	**Abbrevation**	**Min.**	**Max.**
Tpas	Port A Input Setup Delay	-0.73	-1.85
Tpah	Port A Input Hold Delay	1.02	2.53
Tpbs	Port B Input Setup Delay	-0.81	-2.03
Tpbh	Port B Input Hold Delay	1.13	2.66
Tpcs	Port C Input Setup Delay	-0.76	-1.66
Tpch	Port C Input Hold Delay	1.14	2.69
Tpds	Port D Input Setup Delay	-0.66	-1.5
Tpdh	Port D Input Hold Delay	1.12	2.67

Figure 35 shows port input and output timings.



Note: This diagram shows when exactly external NMI_n signal is used.

Figure 35. NMI_n Timing

Drain Diagram

Figure 36 shows the drain diagram for the eZ80 Webserver.

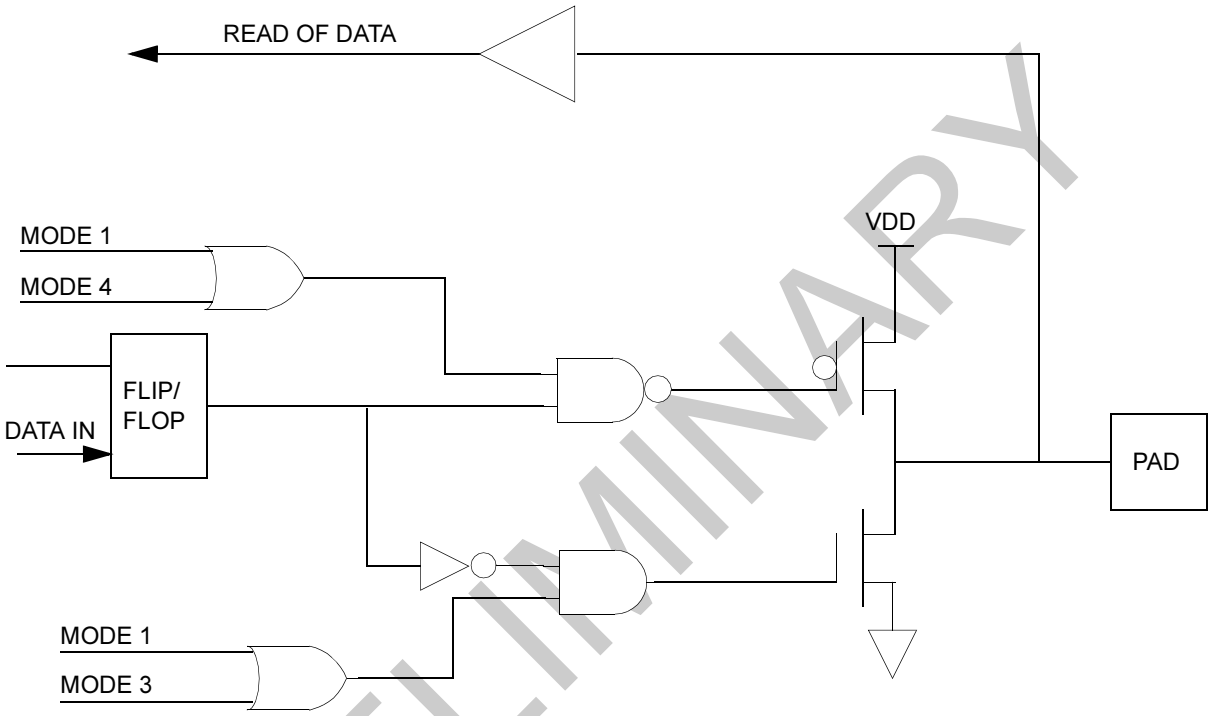


Figure 36. Drain diagram for the eZ80 Webserver

Instruction Set Summary

Tables 37-46 show the instructions used by the eZ80 Webserver processor. The instructions are grouped by class. More detailed information is available in the eZ80 CPU User's Manual.

Table 36.Load Instructions

Mnemonic	Operands	Instruction
LD	dst,src	Load
LEA	qq,IX/Y±d	Load Effective Address
PEA	IX/Y±d	Push Effective Address
POP	dst	Pop
PUSH	src	Push

Table 37.Arithmetic Instructions

Mnemonic	Operands	Instruction
ADC	dst,src	Add with Carry
ADD	dst,src	Add
CP	A,src	Compare
CPD(R)		Block Scan, Decrementing (and Repeat)
CPI(R)		Block Scan, Incrementing (and Repeat)
DAA		Decimal Adjust Accumulator
DEC	dst	Decrement
INC	dst	Increment
MLT	rr	Multiply
NEG		Negate Accumulator
SBC	dst,src	Subtract with Carry
SUB	A,src	Subtract

Table 38.Logical Instructions

Mnemonic	Operands	Instruction
AND	A,src	Logical AND
CPL		Complement Accumulator
OR	A,src	Logical OR
TST	A,src	Test Accumulator
XOR	A,src	Logical Exclusive OR

Table 39.Exchange Instructions

Mnemonic	Operands	Instruction
EX	AF,AF’	Exchange Accumulator and Flags
EX	DE,HL	Exchange DE and HL
EX	(SP),rr	Exchange Register and Top of Stack
EXX		Exchange Register Banks

Table 40.Program Control Instructions

Mnemonic	Operands	Instruction
CALL	cc,dst	Conditional Call
CALL	dst	Call
DJNZ	dst	Decrement and Jump if Non-Zero
JP	cc,dst	Conditional Jump
JP	dst	Jump
JR	cc’,dst	Conditional Jump Relative
JR	dst	Jump Relative
RET	cc	Conditional Return
RET		Return
RETI		Return from Interrupt
RETN		Return from Nonmaskable Interrupt
RST	dst	Restart

Table 41.Bit Manipulation Instructions

Mnemonic	Operands	Instruction
BIT	n,src	Bit Test
RES	n,dst	Reset Bit
SET	n,dst	Set Bit

Table 42.Block Transfer Instructions

Mnemonic	Operands	Instruction
LDD(R)		Block Move, Decrementing (and Repeat)
LDI(R)		Block Move, Incrementing (and Repeat)

Table 43.Rotate and Shift Instructions

Mnemonic	Operands	Instruction
RL	dst	Rotate Left
RLA		Rotate Left Accumulator
RLC	dst	Rotate Left Through Carry
RLCA		Rotate Left Through Carry Accumulator
RLD		Rotate Left Decimal
RR	dst	Rotate Right
RRA		Rotate Right Accumulator
RRC	dst	Rotate Right Through Carry
RRCA		Rotate Right Through Carry Accumulator
RRD		Rotate Right Decimal
SLA	dst	Shift Left
SRA	dst	Shift Right Arithmetic
SRL	dst	Shift Right Logical

Table 44.Input/Output Instructions

Mnemonic	Operands	Instruction
IN	A, (n)	Input to A from Port n
IN	r, (C)	Input to Register from Port in BC
IN0	r, (n)	Input to r from Port n in Page 0
IND(R)		Block Input, Decrement HL (and Repeat)
IND2(R)		Block Input, Decrement Both (and Repeat)
INDM(R)		Block Input, Page 0, Decrement Both (and Repeat)
INI(R)		Block Input, Increment HL (and Repeat)
INI2(R)		Block Input, Decrement Both (and Repeat)
INIM(R)		Block Input, Page 0, Increment Both (and Repeat)
OTDM(R)		Block Output, Page 0, Decrement Both (and Repeat)
OTIM(R)		Block Output, Page 0, Increment Both (and Repeat)
OUT	(n), A	Output from A to Port n
OUT	(C), r	Output from Register to Port in BC
OUT0	(n), r	Output from Register to Port n in Page 0
OUTD (OTDR)		Block Output, Decrement HL (and Repeat)
OUTD2 (OTD2R)		Block Output, Decrement Both (and Repeat)
OUTI (OTIR)		Block Output, Increment HL (and Repeat)
OUTI2 (OTI2R)		Block Output. Decrement Both (and Repeat)
TSTIO	n	Test Port (0,C) under mask

Table 45.Processor Control Instructions

Mnemonic	Operands	Instruction
CCF		Complement Carry Flag
DI		Disable Interrupts
EI		Enable Interrupts

Table 45.Processor Control Instructions (Continued)

Mnemonic	Operands	Instruction
HALT		Halt
IM	0/1/2	Interrupt Mode
NOP		No Operation
RSMIX		Reset Mix Flag
SCF		Set Carry Flag
SLP		Sleep
STMIX		Set Mix Flag

Op Code Map

Tables 47 through 52 show the hex values for each of the eZ80 instructions.

Table 46. Op Code Map (First Op Code)

UPPER NIBBLE (HEX) \ LOWER NIBBLE (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA	EX AF,AF′	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	DJNZ d	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA	JR d	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ,d	LD HL,nn	LD (nn),HL	INC HL	INC H	DEC H	LD H,n	DAA	JR Z,d	ADD HL,HL	LD (HL),nn	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC,d	LD SP,nn	LD (nn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF	JR C,d	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	.16.i16 prefix	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD C,A	LD C,B	.24.i16 prefix	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD D,B	LD D,C	.16.i24 prefix	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A	LD E,B	LD E,C	LD E,D	.24.i24 prefix	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADC A,A	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SUB A,B	SUB A,C	SUB A,D	SUB A,E	SUB A,H	SUB A,L	SUB A,(HL)	SUB A,A	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	AND A,B	AND A,C	AND A,D	AND A,E	AND A,H	AND A,L	AND A,(HL)	AND A,A	XOR A,B	XOR A,C	XOR A,D	XOR A,E	XOR A,H	XOR A,L	XOR A,(HL)	XOR A,A
B	OR A,B	OR A,C	OR A,D	OR A,E	OR A,H	OR A,L	OR A,(HL)	OR A,A	CP A,B	CP A,C	CP A,D	CP A,E	CP A,H	CP A,L	CP A,(HL)	CP A,A
C	RET NZ	POP BC	JP NZ,nn	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST 0	RET Z	RET	JP Z,nn	(Table 21)	CALL Z,nn	CALL nn	ADC A,n	RST 8
D	RET NZ	POP DE	JP NC,nn	OUT (n),A	CALL NC,nn	PUSH DE	SUB A,n	RST 10H	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn	(Table 48)	SBC A,n	RST 18H
E	RET PO	POP HL	JP PO,nn	EX (SP),HL	CALL PO,nn	PUSH HL	AND A,n	RST 20	RET PE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn	(Table 49)	XOR A,n	RST 28H
F	RET P	POP AF	JP P,nn	DI	CALL P,nn	PUSH AF	OR A,n	RST 30H	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn	(Table 50)	CP A,n	RST 38H

Notes:
n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

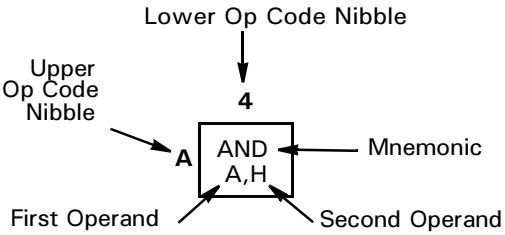


Table 47. Op-code Map (Second Op-code after 0CBH)

UPPER NIBBLE (HEX) \ LOWER NIBBLE (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC RRCA	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3									SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
4	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

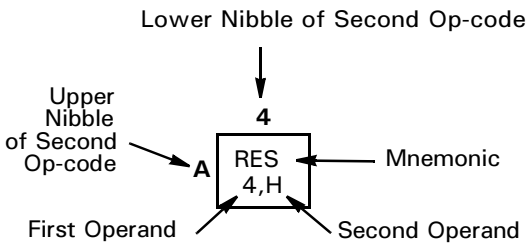


Table 48. Op-code Map (Second Op-code After 0DDH)

UPPER NIBBLE (HEX) \ LOWER NIBBLE (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0								LD BC, (IX ± d)		ADD IX,BC						LD (IX ± d),BC
1								LD DE, (IX ± d)		ADD IX,DE						LD (IX ± d),DE
2		LD IX,nn	LD (nn),IX	INC IX	INC IXH	DEC IXH	LD IXH,n	LD HL, (IX ± d)		ADD IX,IX	LD IX,(nn)	DEC IX	INC IXL	DEC IXL	LD IXL,n	LD (IX ± d),HL
3		LD IY, (IX ± d)			INC (IX ± d)	DEC (IX ± d)	LD (IX ± d),n	LD IX, (IX ± d)		ADD IX,SP					LD (IX ± d),IY	LD (IX ± d),IX
4					LD B,IXH	LD B,IXL	LD B, (IX ± d)						LD C,IXH	LD C,IXL	LD C, (IX ± d)	
5					LD D,IXH	LD D,IXL	LD D, (IX ± d)						LD E,IXH	LD E,IXL	LD E, (IX ± d)	
6	LD IXH,B	LD IXH,C	LD IXH,D	LD IXH,E	LD IXH,H	LD IXH,L	LD H, (IX ± d)	LD IXH,A	LD IXL,B	LD IXL,C	LD IXL,D	LD IXL,E	LD IXL,H	LD IXL,L	LD L, (IX ± d)	LD IXL,A
7	LD (IX ± d),B	LD (IX ± d),C	LD (IX ± d),D	LD (IX ± d),E	LD (IX ± d),H	LD (IX ± d),L		LD (IX ± d),A					LD A,IXH	LD A,IXL	LD A, (IX ± d)	
8					ADD A,IXH	ADD A,IXL	ADD A, (IX ± d)						ADC A,IXH	ADC A,IXL	ADC A, (IX ± d)	
9					SUB A,IXH	SUB A,IXL	SUB A, (IX ± d)						SBC A,IXH	SBC A,IXL	SBC A, (IX ± d)	
A					AND A,IXH	AND A,IXL	AND A, (IX ± d)						XOR A,IXH	XOR A,IXL	XOR A, (IX ± d)	
B					OR A,IXH	OR A,IXL	OR A, (IX ± d)						CP A,IXH	CP A,IXL	CP A, (IX ± d)	
C												(Table 51)				
D																
E		POP IX		EX (SP),IX		PUSH IX				JP (IX)						
F										LD SP,IX						

Notes:
n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

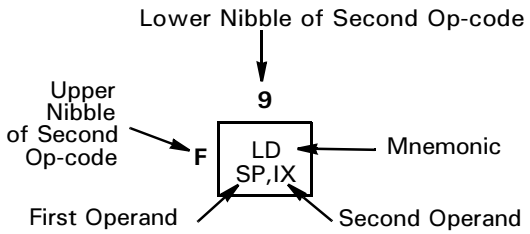


Table 49. Op-code Map (Second Op-code After 0EDH)

UPPER NIBBLE (HEX) \ LOWER NIBBLE (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	IN0 B,(n)	OUT0 (n),B	LEA BC ,IX ± d	LEA BC ,IY ± d	TST A,B			LD BC, (HL)	IN0 C,(n)	OUT0 (n),C			TST A,C			LD (HL) ,BC
1	IN0 D,(n)	OUT0 (n),D	LEA DE ,IX ± d	LEA DE ,IY ± d	TST A,D			LD DE, (HL)	IN0 E,(n)	OUT0 (n),E			TST A,E			LD (HL) ,DE
2	IN0 H,(n)	OUT0 (n),H	LEA HL ,IX ± d	LEA HL ,IY ± d	TST A,H			LD HL, (HL)	IN0 L,(n)	OUT0 (n),L			TST A,L			LD (HL) ,HL
3	IN0 F,(n)	LD IY, (HL)	LEA IX ,IX ± d	LEA IY ,IY ± d	TST A,(HL)			LD IX, (HL)	IN0 A,(n)	OUT0 (n),A			TST A,A		LD (HL) ,IY	LD (HL) ,IX
4	IN B,(C)	OUT (C),B	SBC HL,BC	LD (nn),BC	NEG	RETN	IM 0	LD I,A	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(nn)	MLT BC	RETI		LD R,A
5	IN D,(C)	OUT (C),D	SBC HL,DE	LD (nn),DE	LEA IX ,IY ± d	LEA IY ,IX ± d	IM 1	LD A,I	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(nn)	MLT DE		IM 2	LD A,R
6	IN H,(C)	OUT (C),H	SBC HL,HL	LD (nn),HL	TST A,n	PEA IX ± d	PEA IY ± d	RRD	IN L,(C)	OUT (C),L	ADC HL,HL	LD HL,(nn)	MLT HL	LD MB,A	LD A,MB	RLD
7	IN F,(C)		SBC HL,SP	LD (nn),SP	TSTIO n		SLP		IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(nn)	MLT SP	STMIX	RSMIX	
8			INIM	OTIM	INI2						INDM	OTDM	IND2			
9			INIMR	OTIMR	INI2R						INDMR	OTDMR	IND2R			
A	LDI	CPI	INI	OUTI	OUTI2				LDD	CPD	IND	OUTD	OUTD2			
B	LDIR	CPIR	INIR	OTIR	OTI2R				LDDR	CPDR	INDR	OTDR	OTD2R			
C																
D																
E																
F																

Notes:
n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

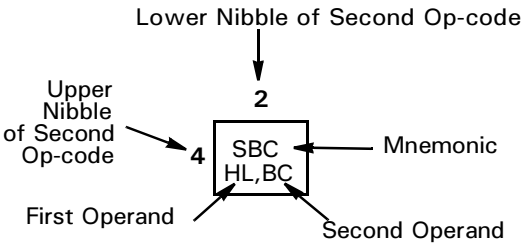


Table 50. Op-code Map (Second Op-code After 0FDH)

UPPER NIBBLE (HEX) \ LOWER NIBBLE (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0								LD BC, (IY ± d)		ADD IY,BC						LD (IY ± d),BC
1								LD DE, (IY ± d)		ADD IY,DE						LD (IY ± d),DE
2		LD IY,nn	LD (nn),IY	INC IY	INC IYH	DEC IYH	LD IYH,n	LD HL, (IY ± d)		ADD IY,IY	LD IY,(nn)	DEC IY	INC IYL	DEC IYL	LD IYL,n	LD (IY ± d),HL
3		LD IX, (IY ± d)			INC (IY ± d)	DEC (IY ± d)	LD (IY ± d),n	LD IY, (IY ± d)		ADD IY,SP					LD (IY ± d),IX	LD (IY ± d),IY
4					LD B,IYH	LD B,IYL	LD B, (IY ± d)						LD C,IYH	LD C,IYL	LD C, (IY ± d)	
5					LD D,IYH	LD D,IYL	LD D, (IY ± d)						LD E,IYH	LD E,IYL	LD E, (IY ± d)	
6	LD IYH,B	LD IYH,C	LD IYH,D	LD IYH,E	LD IYH,H	LD IYH,L	LD H, (IY ± d)	LD IYH,A	LD IYL,B	LD IYL,C	LD IYL,D	LD IYL,E	LD IYL,H	LD IYL,L	LD L, (IY ± d)	LD IYL,A
7	LD (IY ± d),B	LD (IY ± d),C	LD (IY ± d),D	LD (IY ± d),E	LD (IY ± d),H	LD (IY ± d),L		LD (IY ± d),A					LD A,IYH	LD A,IYL	LD A, (IY ± d)	
8					ADD A,IYH	ADD A,IYL	ADD A, (IY ± d)						ADC A,IYH	ADC A,IYL	ADC A, (IY ± d)	
9					SUB A,IYH	SUB A,IYL	SUB A, (IY ± d)						SBC A,IYH	SBC A,IYL	SBC A, (IY ± d)	
A					AND A,IYH	AND A,IYL	AND A, (IY ± d)						XOR A,IYH	XOR A,IYL	XOR A, (IY ± d)	
B					OR A,IYH	OR A,IYL	OR A, (IY ± d)						CP A,IYH	CP A,IYL	CP A, (IY ± d)	
C												(Table 52)				
D																
E		POP IY		EX (SP),IY		PUSH IY				JP (IY)						
F										LD SP,IY						

Notes:
n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

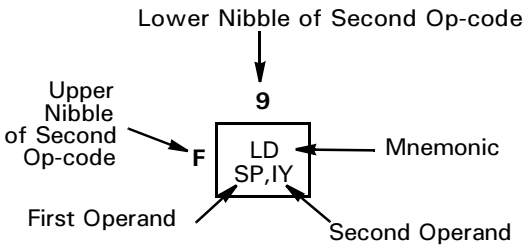


Table 51. Op-code Map (4th Byte, after 0DDH, 0CBH, and d)

UPPER NIBBLE (HEX) \ LOWER NIBBLE (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0							RLC (IX ± d)								RRC (IX ± d)	
1							RL (IX ± d)								RR (IX ± d)	
2							SLA (IX ± d)								SRA (IX ± d)	
3															SRL (IX ± d)	
4							BIT 0, (IX ± d)								BIT 1, (IX ± d)	
5							BIT 2, (IX ± d)								BIT 3, (IX ± d)	
6							BIT 4, (IX ± d)								BIT 5, (IX ± d)	
7							BIT 6, (IX ± d)								BIT 7, (IX ± d)	
8							RES 0, (IX ± d)								RES 1, (IX ± d)	
9							RES 2, (IX ± d)								RES 3, (IX ± d)	
A							RES 4, (IX ± d)								RES 5, (IX ± d)	
B							RES 6, (IX ± d)								RES 7, (IX ± d)	
C							SET 0, (IX ± d)								SET 1, (IX ± d)	
D							SET 2, (IX ± d)								SET 3, (IX ± d)	
E							SET 4, (IX ± d)								SET 5, (IX ± d)	
F							SET 6, (IX ± d)								SET 7, (IX ± d)	

Notes:
d = signed 8-bit displacement

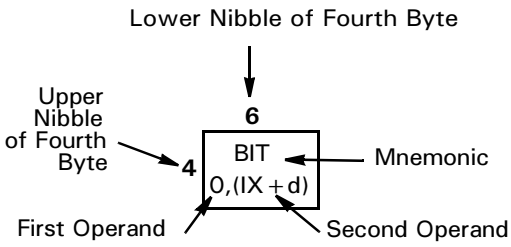
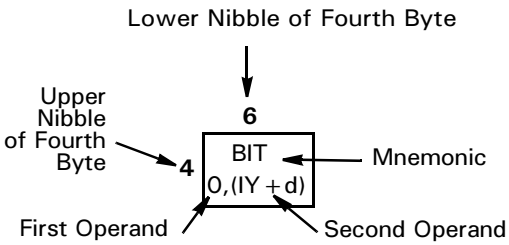


Table 52. Op-code Map (4th Byte, After 0FDH, 0CBH, and d)

UPPER NIBBLE (HEX) \ LOWER NIBBLE (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0							RLC (IY ± d)								RRC (IY ± d)	
1							RL (IY ± d)								RR (IY ± d)	
2							SLA (IY ± d)								SRA (IY ± d)	
3															SRL (IY ± d)	
4							BIT 0, (IY ± d)								BIT 1, (IY ± d)	
5							BIT 2, (IY ± d)								BIT 3, (IY ± d)	
6							BIT 4, (IY ± d)								BIT 5, (IY ± d)	
7							BIT 6, (IY ± d)								BIT 7, (IY ± d)	
8							RES 0, (IY ± d)								RES 1, (IY ± d)	
9							RES 2, (IY ± d)								RES 3, (IY ± d)	
A							RES 4, (IY ± d)								RES 5, (IY ± d)	
B							RES 6, (IY ± d)								RES 7, (IY ± d)	
C							SET 0, (IY ± d)								SET 1, (IY ± d)	
D							SET 2, (IY ± d)								SET 3, (IY ± d)	
E							SET 4, (IY ± d)								SET 5, (IY ± d)	
F							SET 6, (IY ± d)								SET 7, (IY ± d)	

Notes:
d = signed 8-bit displacement



Ordering Information

Contact ZILOG worldwide customer support center for more information on ordering the eZ80 Webserver. The customer support center is open from 7 a.m. to 7 p.m. Central Time.

The customer support toll-free number for the United States and Canada is 1-877-ZiLOGCS (1-877-945-6427). For calls outside of the United States and Canada dial 512-306-4169. The FAX number to the customer support center is 512-306-4072. Customers can also access customer support via the website at:

- For customer service - http://register.zilog.com/login.asp?login=servicelogin
- For technical support- http://register.zilog.com/login.asp?login=supportlogin

For valuable information about hardware and software development tools go to ZiLOG home page at http://www.zilog.com. The latest released version of the ZDS can be downloaded from this site.

Errata Information

The following errata information is for Revision AB of the eZ80 Webserver processor.

- Use of the core registers AF,BC,DE or HL immediately after a POP instruction can give an incorrect result. Inserting a NOP after POP solves the problem. This problem is automatically taken care of by both ZiLOG's C-Compiler and assembler.
- The overflow flag is set incorrectly when SUBC instruction has the second operand as the most negative number. This happens for both 24 and 16-bit operations. To fix this problem use 8-bit math to perform 16 and 24 SUBC operations or detect that the second operand is the most negative number and complement the overflow flag. This problem is automatically taken care of in ZILOG's C-compiler.
- The POP instruction functions differently than the POP BC, DE or HL instruction. For the POP AF instruction, first A is pushed, then F, and finally a dummy byte is pushed.
- The JP.IL instruction doesn't work if used in non-ADL mode with non-zero mbase to jump to the ADL mode. This has been adjusted for in the ZiLOG C-Compiler. The instruction encoding has been fixed to set the ADL one cycle later so that the third byte is read in. To properly transfer the 24-bit operand to the PC, the instruction must be specified with the .LIL suffix. JP.LIL nnn. To fix in the assembly code make jump to ADL mode in the first page and before PC23-16 becomes unknown, or use the CALL.IL instruction.

- The RETN instruction doesn't copy IEF2 to IEF1. For NMI routines which need to return to the application, test the state of IFF2 with LD A,R or LD A,I instruction. If IFF2 is set, enable interrupts just before return.

Index