Application Note

An Interrupt-Driven UART for Z8 Encore! XP® and Z8 Encore! MC™ MCUs

AN033001-0511

Abstract

This application note demonstrates the implementation of an interrupt-driven UART, an on-chip peripheral block featured in Zilog's Z8 Encore! XP and Z8 Encore! MC families of microcontrollers. This document contains sample codes to initialize the UART and to manage UART interrupts for devices in each of these two MCU families. A circular buffer implementation is also introduced to facilitate the buffering of UART data streams.

For ease of discussion, the terms Z8 Encore! and Z8 Encore! devices will be used in this document to refer to both Z8 Encore! XP and Z8 Encore! MC devices.

Note: The source code file associated with this application note, AN0330-SC01.zip, is available for download on zilog.com. This source code has been tested with version 5.0.0 of ZDS II for Z8 Encore! XP- and Z8 Encore! MC-powered MCUs. Subsequent releases of ZDS II may require you to modify the code supplied with this application note.

Overview of the UART Peripheral in Z8 Encore! Devices

The Universal Asynchronous Receiver/Transmitter (UART) is a full-duplex communication channel capable of handling asynchronous data transfers. The UART uses a single 8-bit data mode with selectable parity. Features of the UART include:

- 8-bit asynchronous data transfer
- Selectable even- or odd-parity generation and checking
- Option of one or two stop bits
- Separate transmit and receive interrupts
- Separate transmit and receive enables
- Framing, parity, overrun and break detection
- 16-bit Baud Rate Generators (BRG)
- Selectable MULTIPROCESSOR (9-bit) mode with three configurable interrupt schemes
- Baud Rate Generator timer mode
- Driver enable output for external bus transceivers

The UART consists of three primary functional blocks: transmitter, receiver, and baud rate generator. The UART's transmitter and receiver each function independently but use the same baud rate and data format. Figure 1 shows the UART architecture.

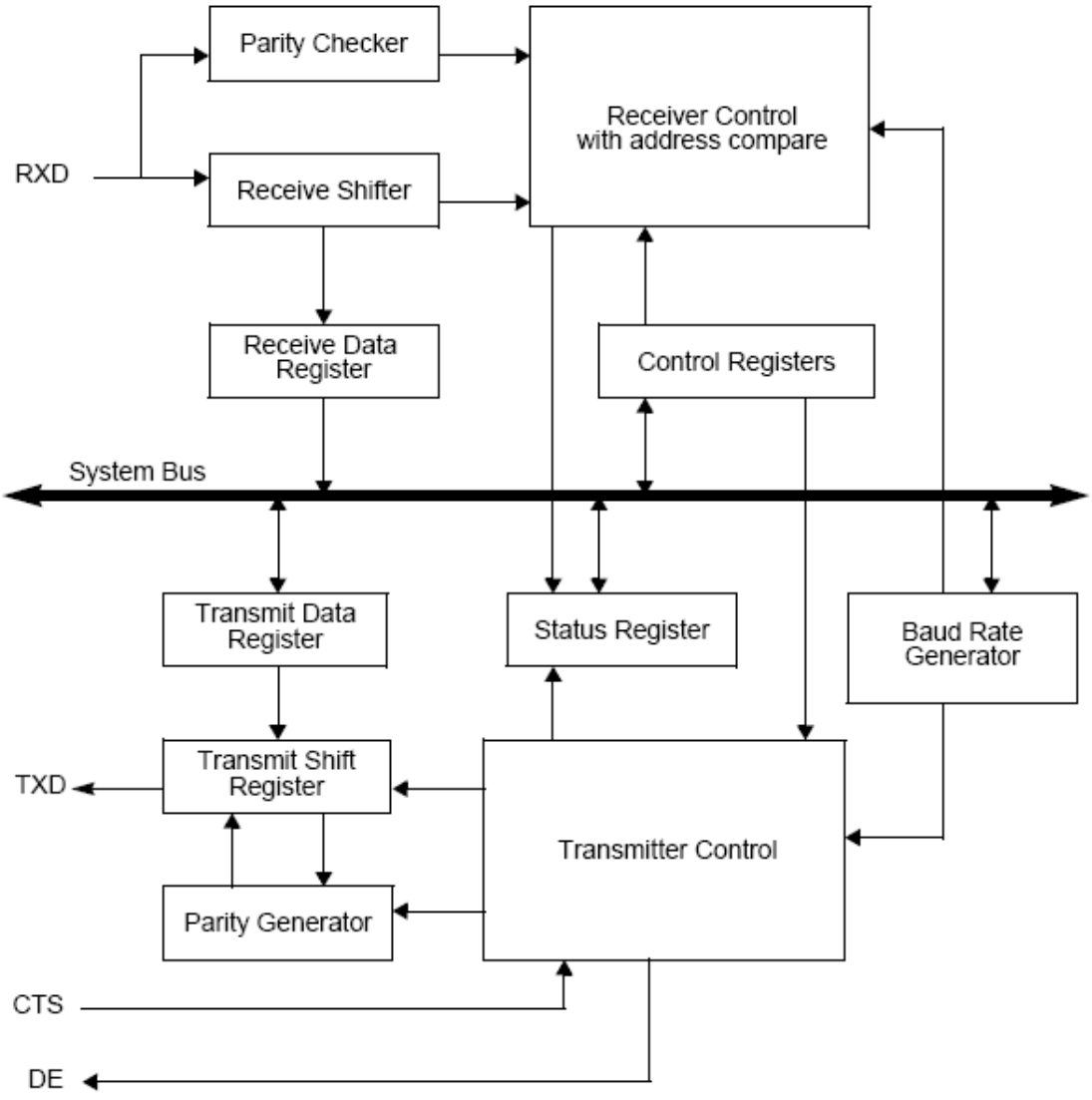

Figure 1. Diagram of the Z8 Encore! XP UART Block

Z8 Encore! UART Register Description

The Z8 Encore! UART registers are briefly discussed in this section.

UART Control Registers

The UART Control 0 and 1 registers configure the properties of the UART's transmit and receive operations. These registers must not be written to while the UART is enabled. See Tables 1 and 2.

Table 1. UART Control 0 Register (UxCTL0)

Bits	7	6	5	4	3	2	1	0
Field	TEN	REN	CTSE	PEN	PSEL	SBRK	STOP	LBEN
Reset	0							
R/W	R/W							
ADDR	F42H and F4AH							

Bit Position	Description
[7] TEN	**Transmit Enable** Enables or disables the transmitter. Transmit enable may also be used in conjunction with CTS signal and CTSE bit.
[6] REN	**Receive Enable** Enables or disables the receiver.
[5] CTSE	**CTS Enable** Defines if CTS signal has no effect on the transmitter or if UART recognizes the CTS signal as an enable control for the transmitter.
[4] PEN	**Parity Enable** Enables or disables the parity bit.
[3] PSEL	**Parity Select** If parity is enabled, this bit specifies if odd- or even-parity will be used.
[2] SBRK	**Send Break** Pauses or breaks data transmission.
[1] **STOP**	**Stop Bit Select** Defines the number of stop bits (1 or 2 stop bits) the transmitter should sent.
[0] LBEN	**Loop Back Enable** Determines if the transmitted data should be looped back to the receiver or not.

Table 2. UART Control 1 Register (UxCTL1)

Bits	7	6	5	4	3	2	1	0
Field	MPMD[1]	MPEN	MPMD[0]	MPBT	DEPOL	BRGCTL	RDAIRQ	IREN
Reset	0							
R/W	R/W							
ADDR	F43H and F4BH							

Bit Position	Description
[7,5] MPMD[1:0]	**Multiprocessor Mode** If multiprocessor mode (MPEN) is enabled, these bits selects the interrupt scheme to be used.
[6] MPEN	**Multiprocessor Enable** Enables or disables the multiprocessor (9-bit) mode.
[4] MPBT	**Multiprocessor Bit Transmit** If multiprocessor mode (MPEN) is enabled, this bit determines what data to send at the multiprocessor bit location (9th bit) of the data stream.
[3] DEPOL	**Driver Enable Polarity** Determines if DE signal is active low or active high.
[2] BRGCTL	**Baud Rate Control** This bit causes different UART behavior depending on whether UART receiver is enabled or disabled. Generally, this bit defines whether the BRG generates an interrupt or not.
[1] RDAIRQ	**Receive Data Interrupt Enable** Determines whether the receiver generates an interrupt on (1) data receive and/or receiver errors, or (2) receiver errors only.
[0] IREN	**Infrared Encoder/Decoder Enable** Enables or disables infrared encoder/decoder.

UART Status Registers

The UART Status 0 and 1 registers identify the current UART operating configuration and status. See Tables 3 and 4.

Table 3. UART Status 0 Register (UxSTAT0)

Bits	7	6	5	4	3	2	1	0
Field	RDA	PE	OE	FE	BRKD	TDRE	TXE	CTS
Reset			0			1	1	X
R/W	R							
ADDR	F41H and F49H							

Bit Position	Description
[7] RDA	**Receive Data Available** Indicates if new data is received. Reading the UART Receive Data Register clears this bit.
[6] PE	**Parity Error** Indicates that a parity error has occurred. Reading the UART Receive Data Register clears this bit.
[5] OE	**Overrun Error** Indicates that an overrun error has occurred. Reading the UART Receive Data Register clears this bit.
[4] FE	**Framing Error** Indicates that a framing error occurred (no stop bit following data reception was detected). Reading the UART Receive Data Register clears this bit.
[3] BRKD	**Break Detect** Indicates that a break occurred.
[2] TDRE	**Transmit Data Register Empty** Indicates that the UART Transmit Data Register is empty and is ready for additional data. Writing to the UART Transmit Data Register clears this bit.
[1] TXE	**Transmitter Empty** Indicates that the Transmit Shift Register is empty and that character transmission is finished.
[0] CTS	**CTS Signal** Reading this bit returns the level of the CTS signal.

Table 4. UART Status 1 Register (UxSTAT1)

Bits	7	6	5	4	3	2	1	0
Field	Reserved						NEWFRM	MPRX
Reset	0							
R/W	R				R/W		R	
ADDR	F44H and F4CH							

Bit Position	Description
[7:2]	Reserved.
[1] NEWFRM	**New Frame** Indicates if the current byte is the first data byte of a new frame.
[0] MPRX	**Multiprocessor Receive** Returns the value of the last multiprocessor bit received.

UART Baud Rate High and Low Byte Registers

The UART Baud Rate High and Low Byte registers combine to create a 16-bit baud rate divisor, BRG[15:0], that sets the baud rate of the UART. The 16-bit baud rate divisor value is {BRH[7:0], BRL[7:0]}. See Tables 5 and 6.

Table 5. UART Baud Rate High Byte Register (UxBRH)

Bits	7	6	5	4	3	2	1	0
Field	BRH							
Reset	1							
R/W	R/W							
ADDR	F46H and F4EH							

Bit Position	Description
[7:0] BRH	**Baud Rate High Byte** Sets the high byte of the UART baud rate.

Table 6. UART Baud Rate Low Byte Register (UxBRL)

Bits	7	6	5	4	3	2	1	0
Field	BRL							
Reset	1							
R/W	R/W							
ADDR	F47H and F4FH							

Bit Position	Description
[7:0] BRL	**Baud Rate Low Byte** Sets the low byte of the UART baud rate.

The baud rate divisor for a given UART data rate can be calculated using the following equation:

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

The baud rate error relative to the desired baud rate is calculated using the equation below. To ensure a reliable communication, the UART baud rate must never exceed 5%.

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

UART Transmit Data Register

Data bytes written to the UART Transmit Data Register (UxTXD) are shifted out on the TXD pin. This register shares a register file address with the read-only UART Receive Data Register. See Table 7.

Table 7. UART Transmit Data Register (UxTXD)

Bits	7	6	5	4	3	2	1	0
Field	TXD							
Reset	X							
R/W	W							
ADDR	F40H and F48H							

Bit Position	Description
[7:0] TXD	**Transmit Data** Data byte to be shifted out through the TXD pin.

UART Receive Data Register

Data bytes received through the RXD pin are stored in the UART Receive Data Register (UxRXD). This register shares a register file with the write-only UART Transmit Data Register. See Table 8.

Table 8. UART Receive Data Register (UxRXD)

Bits	7	6	5	4	3	2	1	0
Field	RXD							
Reset	X							
R/W	R							
ADDR	F40H and F48H							

Bit Position	Description
[7:0] RXD	**Receive Data** Data byte received from the RXD pin.

UART Address Compare Register

The UART Address Compare Register (UxADDR) stores the multi-node network address of the UART when multiprocessor mode is enabled. When the MPMD[1] bit of the UART Control 0 Register is set, all incoming address bytes are compared to the value stored in this register. Receive interrupts and RDA assertions only occur in the event of a match. See Table 9.

Table 9. UART Address Compare Register (UxADDR)

Bits	7	6	5	4	3	2	1	0
Field	COMP_ADDR							
Reset	0							
R/W	R/W							
ADDR	F45H and F4DH							

Bit Position	Description
[7:0] COMP_ADDR	**Compare Address** Defines the 8-bit address value to which the incoming address bytes should be compared to.

Circular Buffer Implementation

This section describes the implementation of a circular buffer. The routines presented here can be used in any queuing or buffering applications.

A buffer is generally used as temporary data storage, usually for streaming data. Similarly, a circular buffer (or ring buffer) is a temporary data storage with a memory allocation scheme where the buffer can be of a fixed size and each memory location can be reused when the index pointer has returned back to the starting location. This buffering scheme is widely used and has several existing versions, each of which varies depending on application requirements. This section describes a simple buffering mechanism.

To initialize circular buffers, a memory segment or an array of predefined length is initialized. This is where the buffered data will be stored.

```
#define RBUF_IN_BUFFERSIZE      ((UINT8)64)
UINT8 RBUF_InBuff[RBUF_IN_BUFFERSIZE];
```

To facilitate how the circular buffer is managed, two index pointers and a data counter is initialized.

```
UINT8 RBUF_InRdPtr;     // Pointer to the next read location
UINT8 RBUF_InWrPtr;     // Pointer to the next write location
UINT8 RBUF_InLength;    // Buffer length
```

Upon initialization, the buffer does not contain anything and the pointers are at the beginning of the buffer, as shown in Figure 2.

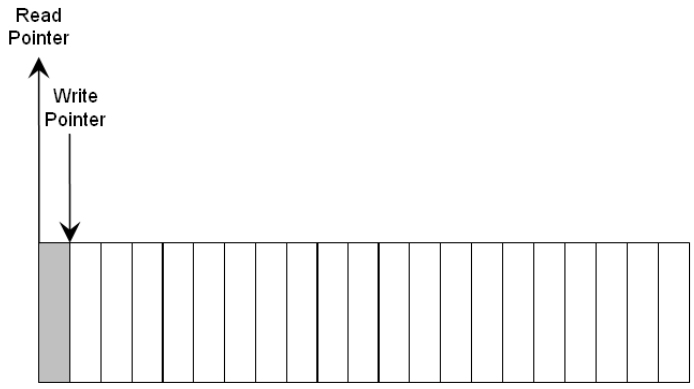

Figure 2. Initializing the Buffer

While data is being written to the buffer, the write pointer increments and the data counter also increments. Similarly, while data is being read from the buffer, the read pointer increments and the data counter decrements. See Figure 3.

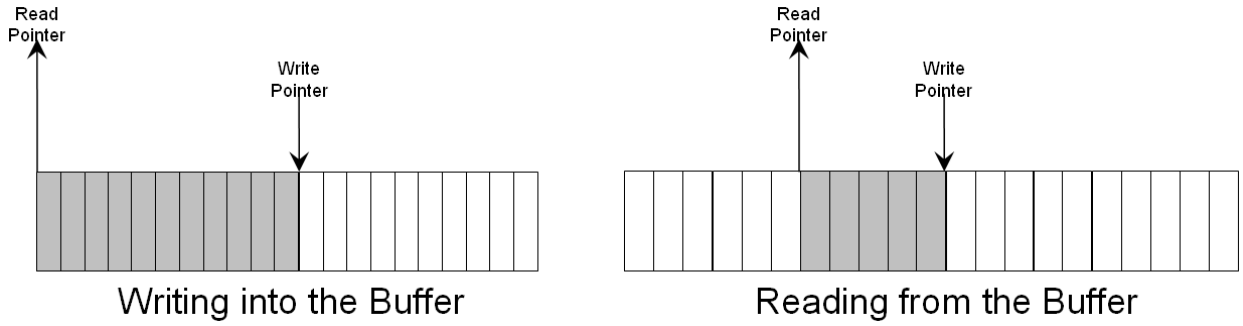

Figure 3. Read/Write Operations of the Buffer

```
void RBUF_AddByteToInBuffer(UINT8 data)// Write
{
  if( ((RBUF_InWrPtr + 1) % RBUF_IN_BUFFERSIZE) != RBUF_InRdPtr )
  {
      RBUF_InBuff[RBUF_InWrPtr] = data;
      RBUF_InWrPtr = (RBUF_InWrPtr + 1) % RBUF_IN_BUFFERSIZE;
      RBUF_InLength++;
   }
}
UINT8 RBUF_GetByteFromInBuffer(void)// Read
{
  UINT8 data = RBUF_InBuff[RBUF_InRdPtr];
  RBUF_InRdPtr = (RBUF_InRdPtr + 1) % RBUF_IN_BUFFERSIZE;
  RBUF_InLength--;

  return data;
}
```

When the read or write pointer reaches the end of the buffer, it will jump back to the start, causing a wrap-around effect. As a result, the data that has been previously fetched using the read operation will be overwritten. See Figure 4.

Figure 4. Wrap Around

Interrupt-Driven UART Implementation

This section describes the methods for initializing, transmitting and receiving data via the UART peripheral of the Z8 Encore! device.

The software implementation for UART presented in this document supports the basic format, which is 8 data bits, no parity, and 1 stop bit. The program waits to receive a string (terminated by newline) and then echoes back the input string.

To facilitate a data input/output via UART, the circular buffer discussed in the previous section is used for storing data. Separate buffers are used for handling the transmit and receive data. The buffer size, RBUF_IN_BUFFERSIZE and RBUF_OUT_BUFFERSIZE, can be changed as per user needs.

```
UINT8 RBUF_InBuff[RBUF_IN_BUFFERSIZE]; // Input buffer
UINT8 RBUF_OutBuff[RBUF_OUT_BUFFERSIZE];// Output buffer
```

Initialization

The Z8 Encore! UART is a full-duplex communication channel capable of handling asynchronous data transfers. A reliable UART communication is affected by two factors - system clock speed, and desired baud rate. The user should ensure that UART baud rate error should never exceed 5%. For a given UART data rate, the integer BRG value can be calculated using the following equation:

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

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

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

The source code listed below demonstrates how to configure the UART using the 8-N-1 format. The value of BAUDRATE varies depending on processor type, and is provided in the accompanied source code.

```
////////////////////////////////////////////////////////////////
void UART_Init(void)
{
  PADD |= 0x30;             // Setup ports for alternate function
  PAAF |= 0x30;
  #ifdef _Z8ENCORE_F1680
  PAAFS1 &= ~0x30;
  #endif // _Z8ENCORE_F1680

  U0BRH = (UINT8)((BAUDRATE & 0xFF00) >> 8);   // Setup baud rate
  U0BRL = (UINT8)((BAUDRATE & 0x00FF) & 0x00FF);
  IRQ0ENH |= 0x18;          // Enable UART Tx&Rx interrupts
  IRQ0ENL |= 0x18;
  IRQ0 &= ~0x18;            // Clear any pending interrupts
  U0CTL0 = 0xC0;            // Receive En, No Parity, 1 Stop bit
}
////////////////////////////////////////////////////////////////
```

UART Rx Data Handling

The code provided below demonstrates how to handle data received from the UART Receive Data Register. The data received from this register is transferred to the buffer. It is up to the user how to get and interpret the data from the input buffer, `RBUF_InBuff`.

```
////////////////////////////////////////////////////////////////
void interrupt UART0_RxIsr(void) _At UART0_RX
{
  UINT8 temp = U0RXD;
  if((U0STAT0 & 0x78) == 0x78)
    return;                 // ERROR detected!!!
                            // Data is read to clear this bit
  RBUF_AddByteToInBuffer(temp);

  #ifndef _Z8ENCORE_F1680
  IRQ0 &= ~0x10;            // Clear interrupt flag
  #endif                    // _Z8ENCORE_F1680
}
////////////////////////////////////////////////////////////////
```

UART Tx Data Handling

Similarly, the succeeding code demonstrates how to use the buffer for handling data to be transmitted via the UART Transmit Data Register. Data needs to be placed into the buffer before starting the transmission.

```
RBUF_AddStrToOutBuffer(strData, len);    // place data into buffer
UART_StartTx();                          // start transmission


////////////////////////////////////////////////////////////////
void interrupt UART0_TxIsr(void) _At UART0_TX
{
  if( RBUF_GetLengthOutBuffer() > 0 )   // If there is data to tx
    U0TXD = RBUF_GetByteFromOutBuffer();

  #ifndef _Z8ENCORE_F1680
  IRQ0 &= ~0x08;                        // Clear interrupt flag
  #endif                                // _Z8ENCORE_F1680
}

void UART_StartTx(void)
{
  if( RBUF_GetLengthOutBuffer() )  // If there is data to be tx'ed
  {
              // Trigger Tx interrupt to start loading buffer data
    IRQ0 |= 0x08;
  }
}
////////////////////////////////////////////////////////////////
```

Hardware Setup

Figure 5 shows the hardware setup of Z8 Encore! development kit connected to a PC via HyperTerminal. HyperTerminal setting is 8-N-1, with flow control set to none. The default baud rate settings in the source code along with this document uses 115kbps.The user can change this setting as desired.

Figure 5. Z8 Encore to PC Connection via the RS-232 Port

Summary

This application note describes a basic UART implementation, including how to configure the Z8 Encore! MCU's ports for an interrupt-driven UART application. A brief discussion about the circular buffer is also introduced to develop an understanding of how to administer data input/output.

References

The following documents associated with the Z8 Encore! XP Series of MCUs are available on www.zilog.com.

- Z8 Encore! XP F1680 Series Product Specification (PS0250)
- Z8 Encore! XP F1680 28-Pin Series Development Kit User Manual (UM0203)
- Software UART for the Z8 Encore! XP MCU Application Note (AN0147)
- An OLED Interface Using Z8 Encore! XP Series MCUs Application Note (AN0329)

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.