Application Note

Software UART for the Z8 Encore! XP® MCU

AN014705-0208

Abstract

This Application Note describes the implementation of a software-emulated universal asynchronous receiver/transmitter (UART) for Zilog's Z8 Encore! XP® 8-bit microcontrollers. Software UART implementation is useful for applications in which an extra UART is required in addition to the hardware UART(s) available with Z8 Encore! XP devices. The hardware UARTs operate in full-duplex mode, where as the software UART implementation is half-duplex. The software UART is also an event-driven and supports an 8-N-1 protocol using an RS-232 interface.

Data transfer is achievable at baud rates from 300 to 57600. The software features APIs for basic operations such as an initialization, and data reception/transmission. Source code for the software UART implementation is provided in Assembly and C languages, with an exception of Z8F642x support, for which code is provided only in C.

Note: *The following source codes associated with this Application Note are available for download at www.zilog.com.*

- AN0147-SC01 Assembly code for the 8 KB Z8 Encore! XP MCU (Z8F082x).
- AN0147-SC02 C code for the 8 KB Z8 Encore! XP MCU (Z8F082x).
- AN0147-SC03 Assembly code for the 64 KB Z8 Encore! XP MCU (Z8F640x).
- AN0147-SC04 C code for the 64 KB Z8 Encore! XP MCU (Z8F640x).
- AN0147-SC05 C code for the 64 KB Z8 Encore! XP MCU (Z8F642x).

Z8 Encore! XP Flash Microcontrollers

Zilog's Z8 Encore! XP products are based on the new eZ8 CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows for faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with Z8® MCU.

Z8 Encore! XP microcontrollers combine a 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of on-chip peripherals. These peripherals make the Z8 Encore! XP suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Discussion

The UART protocol is based on the EIA RS-232C standard, published in the year 1969. The standard was popular with the introduction of personal computers and it is one of the most commonly used serial interfaces.

Originally defined as a 25-pin interface with several modem handshake and control signals, the basic UART interface requires only three lines: Receiver (R_X), Transmitter (T_X) and Ground (GND). The handshake is executed in software by transmitting special XON and XOFF characters. In most of the MCU applications, half-duplex communication is sufficient, that is each side is either a receiver or a transmitter at any given time.

Copyright ©2008 by Zilog®, Inc. All rights reserved.
www.zilog.com

In an asynchronous serial data communication, data is transmitted sequentially, one bit at a time. The T_X idle state of the UART is High. A High-to-Low transition of the Start bit initiates the transmission. Eight data bits follow before the Stop bit pulls High again.

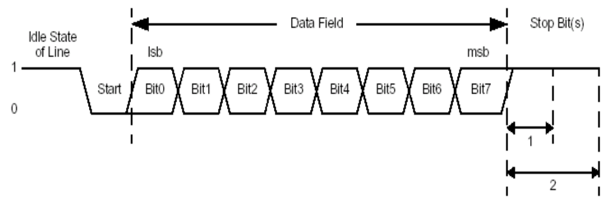


Figure 1. Basic 8-bit UART Protocol

In an asynchronous operation, the clock is not transmitted. The receiver must operate with the same baud rate as the transmitter, and the data rate is usually derived from a local oscillator. The receiver must also synchronize the baud rate to the falling edge of the Start bit, and sample the incoming data in middle of a bit.

Developing a Software UART for the Z8 Encore! XP® MCU

The Z8 Encore! XP software UART supports the basic 8-N-1 format, which is 8 data bits, no parity, and 1 stop bit. It communicates in half-duplex mode. In R_X mode, the program waits to receive a character, then stores it in the R_X Data buffer. In T_X mode, the program sends the character that is stored in the T_X Data buffer.

Options

Several options at assembly time can be selected to adapt the program to the appropriate operation.

Table 1contains a list of these options.

Table 1. Options at Assembly Time

Variable Name	Description
BAUD	Baud rate: 300 to 57600. When BAUD is specified, the program selects all the appropriate timings. Default is 9600.
RAM_TOP	Top of RAM; default is EFF.
ROM_TOP	Top of Flash; default is FFFF.
MODE	R_X or T_X.

Hardware Architecture

The Figure 2 displays the hardware setup of Z8 Encore! XP® MCU connected by a MAX-232A line driver to a PC, running a HyperTerminal program with a setting of *no handshake*. Only R_X and T_X lines are used.

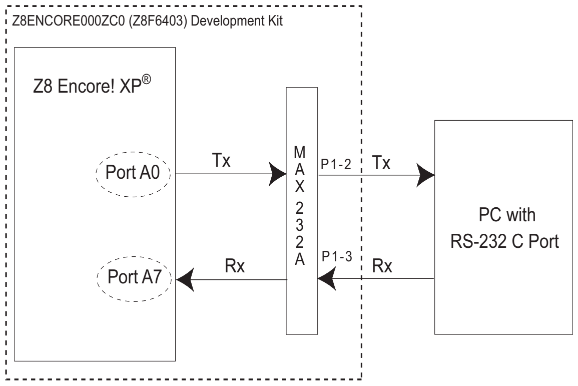


Figure 2. Block Diagram of Z8 Encore! XP Connected to PC

Both the communicating UARTs (in this case, the PC and the Z8 Encore! XP® MCU) must be programmed with the same baud rate—one as transmitter and one as receiver; the R_X and T_X lines are crossed. Pin PA0 is used for T_X and pin PA1 is used for R_X.

Software Implementation

The software UART implementation comprises of three basic operations: initialization, receiving data, and transmitting data. The implementation of these operations is common for the software UART, which is coded in Assembly and C languages.

Initialization

In this implementation, PA0 is the T_X pin and PA1 is the R_X pin. The following operations are performed sequentially during initialization:

1. Pin PA0 is set to OUTPUT mode and a High is output at PA0.
2. PA1 is set to INPUT mode and initialized to generate interrupts at the falling edge of the signal.
3. The timer, which is used as a counter and is set in the CONTINUOUS mode of operation to generate interrupts upon reaching a set counter value. To sample the received data at the center of the bit, the start value of the timer register is set to half the value of the reload register.
4. The timer is enabled during the T_X mode, and disabled otherwise. Port A interrupts are enabled during R_X mode and disabled otherwise.

Figure 7 on page 9 displays the flowchart of the Main Software UART Routine.

Receiving Data

To receive data, the Start bit must be detected. The Start bit is detected by the falling edge of the signal at pin PA1 when a port interrupt is generated. This interrupt is handled by the Port ISR (see Figure 8 on page 9), wherein the timer is enabled, and the port interrupts are disabled to receive the remainder of the data bits.The timer reload value is set to generate interrupts at one-bit intervals. The start value of the R_X Sampling when Receiving Data timer register is set to half the reload value to ensure that the first interrupt generated, after enabling the timer, is at the middle of the Start bit. See Figure 3.

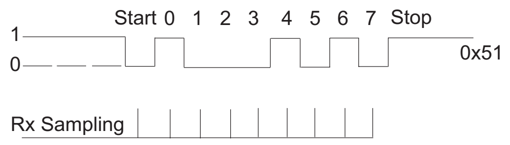


Figure 3. R_X Sampling when Receiving Data

If the middle of the first bit is a zero, it indicates that the received bit is a valid Start bit and not a glitch.

The `serial_in` function performs the following tasks:

1. To receive transmission, the `serial_in` function ensures that the received bit is not a glitch, sets the `valid_data` flag to TRUE, and sets the receive bit counter, Rx_COUNT, to 0.
2. The data is sampled in the middle of the next bit.
3. The data bit is stored in the appropriate bit position in the R_X Data register, and Rx_COUNT is incremented.When Rx_COUNT = 8, then Rx_COUNT is reset, and the timer is disabled.
4. The timer start value is reloaded. The `valid_data` flag is set to FALSE and the Port A interrupts are enabled.

Figure 11 on page 11 displays the flowchart for the Data Reception Routine.

Transmitting Data

To transmit data, the start value of the timer register is set to zero and the timer reload value is set to generate interrupts at one-bit intervals. See Figure 4 on page 4.

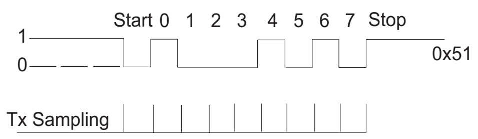


Figure 4. T_X Sampling when Transmitting Data

The function `serial_out` performs the following tasks:

1. To start transmission, the transmit bit counter, Tx_COUNT, is set to 0 and the T_X line (PA0) is pulled Low to send the Start bit. Tx_COUNT is incremented.
2. The Tx_DATA register contains the valid data to be transmitted. It is read, loaded into an intermediate buffer, and shifted out to Port A0, one bit at a time, from bit 0 to bit 7. Tx_COUNT is incremented after each bit is transmitted (see Figure 4).
3. To stop transmission, the T_X line (PA0) is pulled High to signal the end of transmission; the timer is disabled.

Figure 10 on page 10 displays the flowchart for the data transmission routine.

Features Specific to C implementation of Software UART

Two additional features are provided with the software UART implemented in C language. They are described below.

Flow Control

Two functions/APIs are provided to control the flow of data while receiving and transmitting data. These APIs are:

1. `Z_Start_Commu()`
2. `Z_Stop_Commu()`

Calling the `Z_Start_Commu()` API resumes communication. Calling `Z_Stop_Commu()` stops the communication between the communicating devices.

Overrun Error

A data overrun error occurs when the R_X Data buffer receives data before previously-received data is read and emptied. An `Over_Run` flag is set when an overrun occurs, and it is reset when the data in the R_X Data buffer is read and emptied.

Setting the Port and Pin for Software UART T_X/R_X

The software UART implementation, by default, uses Port A, Pin 0 and Pin 1 as the T_X and R_X data pins respectively. However, the pins and the port can be changed as require. This is achieved by modifying the code in the `sio.h` file provided in the source code zip file.

It can be modified the T_X pin/port, R_X pin/ port, and the baud rate for software UART by changing the definitions provided in the `sio.h` file, as detailed below.

- To modify the crystal frequency and baud rate, the following settings are modified:

```
//*Set the crystal frequency and BAUD
rate*//
#define     XTAL      20000000ul
            // Crystal frequency
#define     BAUD      9600ul
            // Baud rate
            //(300,600,1200,
            // 2400,4800,9600,19200,
            // 38400,57600)
```

- The following settings are modified to alter the R_X pin and port:

Note: *Use only those ports and pins on which interrupts can be set as the R_X data line.*

```
//**Select the port/pin for Rx bit**/
#define SW_UART_RX_PORTPAADDR
            // select port
#define SW_UART_RX_PORT_CTLPACTL
            // select  port control
#define SW_UART_RX_PORT_INPAIN
            // select port data input
            // register
#define SW_UART_RX_PORT_OUTPAOUT
            //select port data output
            // register
#define SW_UART_RX_PIN_POS1
            // set pin position
            // (no 0-7) for Rx bit
#define SW_UART_RX_DATA_BITBIT1
            //set bit (Bit 0-7) for Rx
            // data
```

- To set the interrupt functionality for the R_X bit, the following configuration settings are modified:

```
/** Set interrupt functionality for Rx
bit **/

#define SW_UART_RX_PORT_INTP1AD
            //set vector for
            //ISR(only for Port
            //A/D(4-lower 4 bit
            //only for port C))
#define SW_UART_RX_CLR_INTIRQ1
            //clear interrupt IRQ1
            //for port A/D and IRQ2
            //for Port C
#define SW_UART_RX_SET_INTIRQ1ENH
            // set priority interrupt
            // IRQ1ENH for port A/D
            //and IRQ2ENH for Port C
#define SW_UART_RX_PORT_ENBIT1
            // Here Port pin is used /
            /to enable the port
            //interrupt
#define SW_UART_RX_PORT_DIS~(BIT1)
            // Here Port pin is used /
            /to disable the port
            //interrupt
#define SW_UART_RX_PORT_SLCIRQPS
#define SW_UART_RX_PORT_AD0x00
            //select 0x00 for port A
            //and 0xFF for port D to
            //generate the interrupt
```

- The following settings are modified to alter the T_X pin/port position:

```
/**Select the port/pin for Tx bit**/
#define SW_UART_TX_PORTPAADDR
            // select port
#define SW_UART_TX_PORT_CTLPACTL
            // select port control
#define SW_UART_TX_PORT_INPAIN
            // select port data
            // input register
#define SW_UART_TX_PORT_OUTPAOUT
            // select port data
            // output register
#define SW_UART_TX_PIN_POS0
            // set pin position
            // (no. 0-7) for Tx bit
#define SW_UART_TX_DATA_BIT BIT0
            // set bit (Bit 0-7) for
            // Tx data
```

Note: *The T_X bit and the R_X bit must not be set for the same pin.*

Testing the Software UART Application

This section describes the equipment and procedure to test the software UART developed for the Z8 Encore! XP® MCU.

Equipment Used

The following equipments are used for testing**:**

- Z8 Encore! XP Development Kit (Z8ENCORE000ZC0-D) featuring the Z8F640x MCU.
- Z8 Encore! XP Development Kit (Z8F08200100) featuring the Z8F082x MCU.
- Z8 Encore! XP Development Kit (Z8F64200100KIT) featuring the Z8F642x MCU.
- ZDSII IDE for the Z8F08xx, Z8F640x, and the Z8F642x, MCUs.
- HyperTerminal application on the PC.

Note: *The XTAL is 20 MHz and the default baud rate is 9600 for R_X/T_X mode.*

The software UART test setup for Z8 Encore! XP software UART with the Z8ENCORE000ZC0 Development Kit (Z8F640x MCU) is displayed in Figure 5.

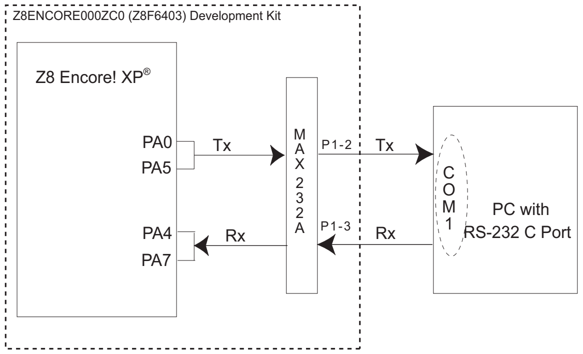


Figure 5. Test Setup for the Z8 Encore! XP® MCU Software UART

Appendix A—Schematics on page 8 displays the test setup schematic for the Z8 Encore! XP® software UART with the Z8F08200100 Development Kit (Z8F082x MCU).

Procedure

Follow the steps below to test the software UART application:

1. Connect the PA0 pin to the PA5 pin. Connect the PA1 pin to the PA4 pin. The available serial driver MAX 232A on the Z8 Encore! XP Development Board, is displayed in Figure 5.
2. Connect the Z8 Encore! XP Development Board with a 9-pin serial cable to a standard PC running Windows NT.
3. Launch the **HyperTerminal** application. Go to **File** → **Properties,** and in the **Properties** dialog box, under the **Connect to** tab, select the **COM2** port in the **Connect using…** text field.
4. Click the **Configure** button and in the **Port Settings** dialog box, enter the following in the text fields:

Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

5. Click **OK** button to get back to the **Connect to** tab. Click the **Settings** tab and click the **ASCII Setup** button. In the **ASCII Setup** dialog box, check the **Echo typed characters locally** option. Click **OK** button until the **Properties** dialog box closes.
6. Download the test application program using ZDSII[1].
7. The default mode is R_X. When a character is entered in HyperTerminal, it is echoed back to the screen.

After a brief delay, the same character is displayed again in the HyperTerminal window, indicating that the character was received by the software UART's R_X Data buffer, transferred to the T_X Data buffer, and transmitted back to HyperTerminal to be displayed on the screen.

8. The C-coded software UART can be used to test for data overrun conditions when an overrun error is generated. To test overrun error generation, disable the `Set_Tx_Mode` call by commenting the call in the `Test_Uart` function before downloading the code once again.
9. When a character is entered in HyperTerminal, an OVERRUN ERROR message appears, indicating that a data overrun occurred.

1. For the Assembly-coded software UART, no prompt appears in the HyperTerminal window upon downloading the code. The HyperTerminal prompt, Z8 Encore! XP, appears only when the C-coded software UART is downloaded.

10. This procedure is repeated for all the baud rates indicated in Table 2. To change the baud rates, the Constants definition, BAUD, is used (see Table 1 on page 2).

Note: *The ZDSII Debugger checks the value of the R_X Data buffer. This value must be equivalent to the* hex *value of the entered character.*

Results

The following results are obtained:

1. In the Table 2, testing with the baud rates and clock frequency settings are indicated.
2. In R_X mode, the software UART samples the data bit in the middle (50% of the bit cell) for all baud rates mentioned in the Table 2.
3. The received data is unaffected by jitter because the test program validates the Start bit before receiving any data.

Table 2. Baud Rates Tested for R_X and T_X Modes at 20 MHz Clock Frequency

	Baud Rates Tested								
Mode	**300**	**600***	**1200**	**2400**	**4800**	**9600**	**19200**	**38400**	**57600**
R_X	P	—	P	P	P	P	P	P	P
T_X	P	—	P	P	P	P	P	P	P
*baud rate option is not available with the HyperTerminal.									

Summary

The software UART implemented in this Application Note supports the most common UART protocol, 8-N-1. The Assembly and C language codes achieve data transfer at baud rates as high as 57600 at 20 MHz clock frequency.

The size of the Assembly code is approximately 160 bytes, which is ideal for small Z8 Encore! XP® modules such as the Z8F08xx. Conversely, the C language code is approximately 1 KB in size and is suitable for larger Z8 Encore! XP modules, such as the Z8F64xx.

Although the software UART can operate only in the half duplex mode, it is otherwise functionally comparable to the hardware UARTs on the Z8 Encore! XP MCU, and can provide an additional UART when required.

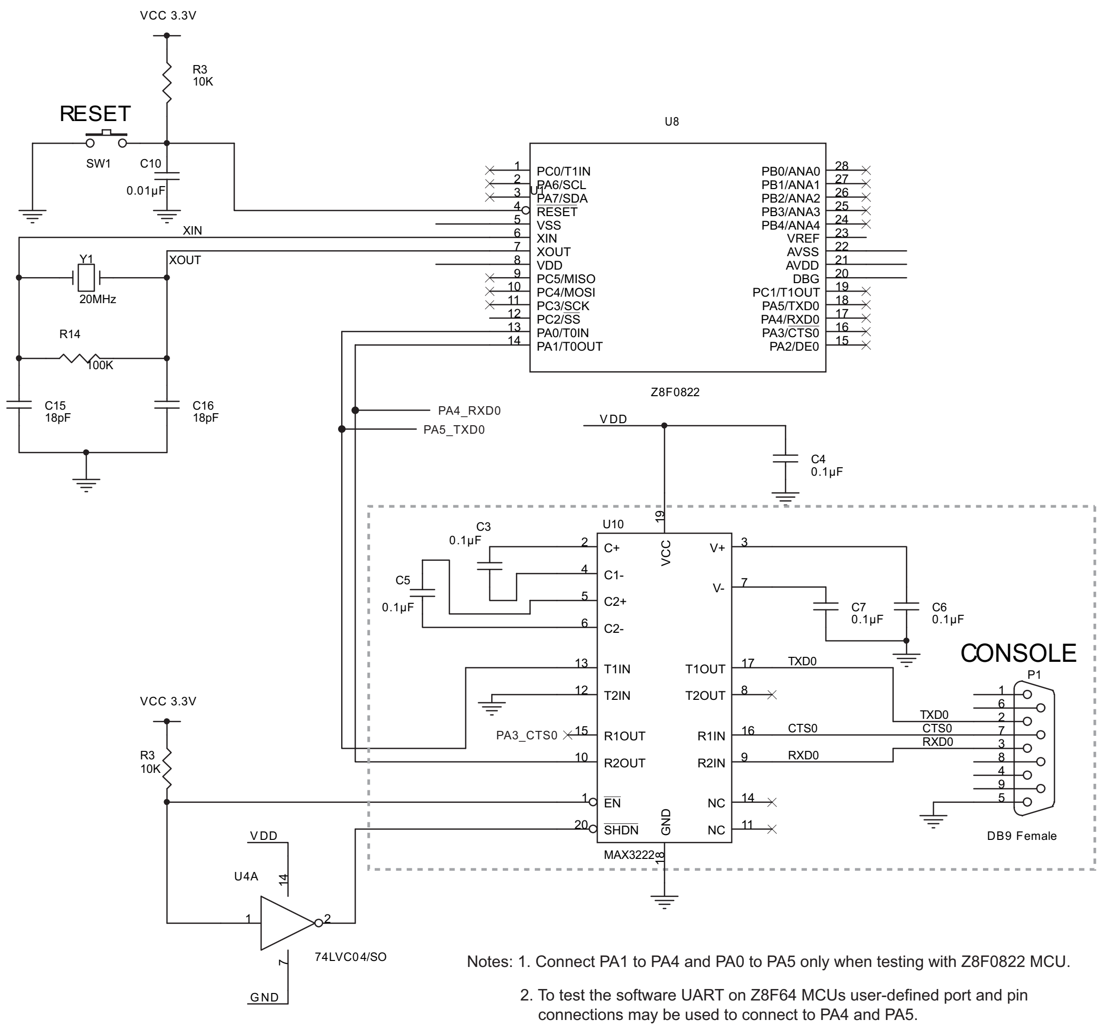
Reference

The documents associated with Z8 Encore! XP MCU available at www.zilog.com are provided below:

- eZ8 CPU User Manual (UM0128).
- Z8 Encore! XP 64K Series Flash Microcontrollers Product (PS0199).
- Z8 Encore! XP 8K/4K Series Development Kit User Manual (UM0150).
- Z8 Encore! XP Flash Microcontroller Development Kit User Manual (UM0146).

Appendix A—Schematics

Figure 6 displays the software UART implementation using the Z8 Encore! XP® MCU.



Notes: 1. Connect PA1 to PA4 and PA0 to PA5 only when testing with Z8F0822 MCU.

2. To test the software UART on Z8F64 MCUs user-defined port and pin connections may be used to connect to PA4 and PA5.

3. When testing the Software UART application, enter the crystal frequency according to the crystal available on user development board.

Figure 6. Schematic for Software UART Implementation Using the Z8 Encore! XP MCU

Appendix B—Flowcharts

Figure 7 displays the main software UART routine.

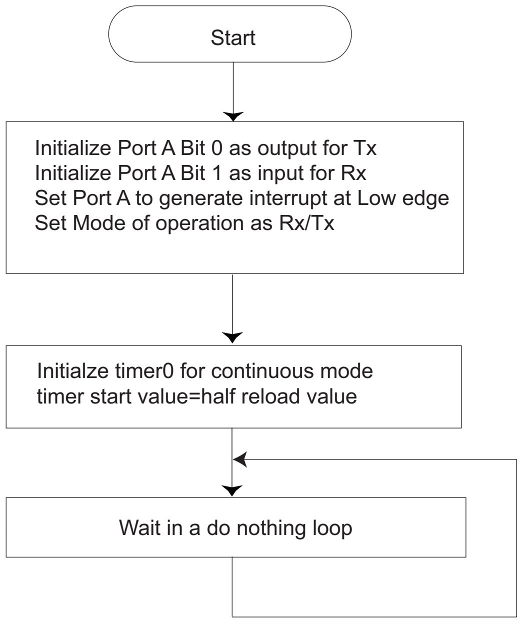

Start
Initialize Port A Bit 0 as output for Tx
Initialize Port A Bit 1 as input for Rx
Set Port A to generate interrupt at Low edge
Set Mode of operation as Rx/Tx
Initialze timer0 for continuous mode
timer start value=half reload value
Wait in a do nothing loop

Figure 7. Main Software UART Routine

Figure 8 displays the port interrupt service routines.

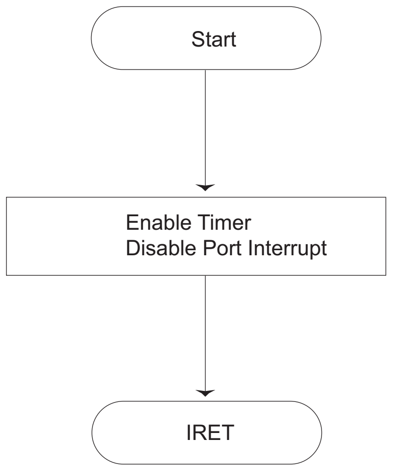


Figure 8. Port ISR Flow

Figure 9 displays the timer interrupt service routines.

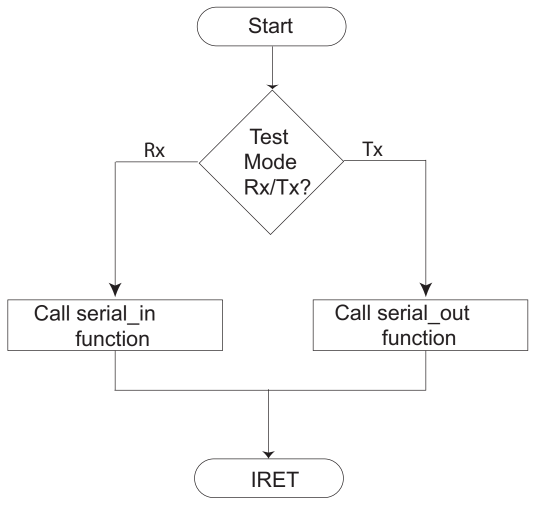


Figure 9. Timer ISR Flow

Figure 10 displays the flow of the transmitting data (Serial Data Out).

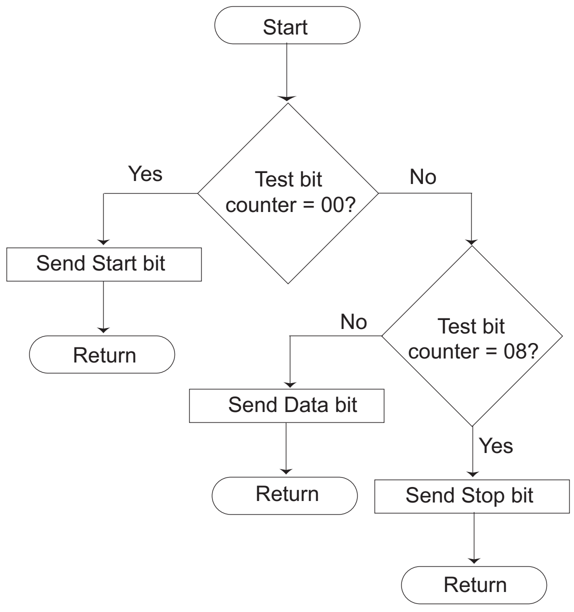


Figure 10. Transmitting Data Flow

Figure 11 displays the flow of the receiving data (Serial Data In).

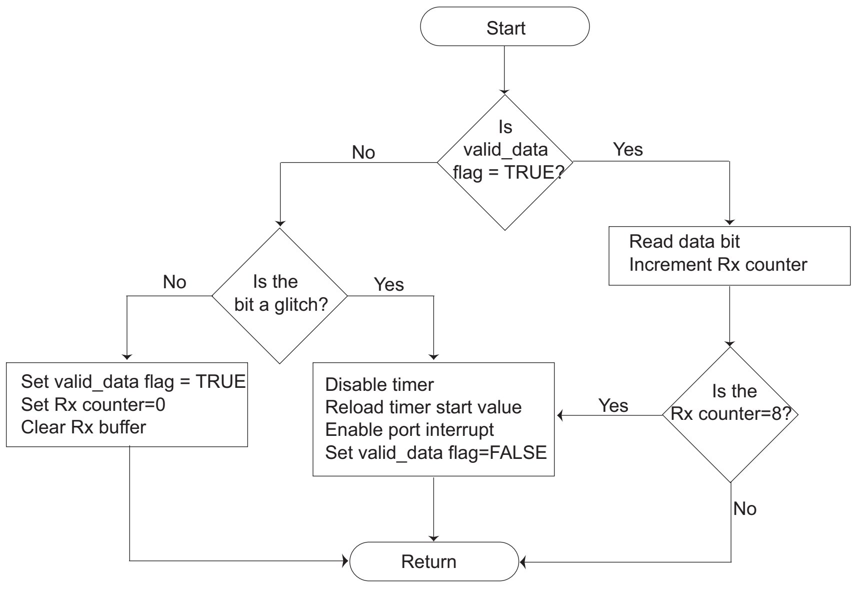


Figure 11. Receiving Data Flow

Warning: DO NOT USE IN LIFE SUPPORT.

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

©2008 by Zilog, Inc. All rights reserved. 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.

Z8 and Z8 Encore! XP® are registered trademark of Zilog, Inc. All other product or service names are the property of their respective owners.