Application Note

Software UART for the Z8 Encore XP!® MCU

AN024302-0608

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 unit (MCU). The software UART implementation is useful for applications in which a UART is required in addition to the hardware UART(s) available with Z8 Encore! XP devices. The hardware UART operates in full duplex mode, whereas the software UART implementation is in half duplex mode. The software UART is also 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 (using external oscillator) and 300 to 4800 (using internal precision oscillator). The software features APIs for basic operations such as initialization, data reception, and data transmission. The source code for the software UART implementation is provided in C language.

Note: *The source code file associated with this document,* `AN0243-SC01.zip` *is available for download at* www.zilog.com.

Z8 Encore! XP® Series Flash Microcontrollers

Zilog's Z8 Encore! products are based on the new eZ8™ CPU and introduce Flash memory to Zilog's extensive line of 8-bit MCUs. 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 Zilog's popular Z8® MCU. Z8 Encore! MCUs combine a 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of on-chip peripherals.

The Z8 Encore! series of devices support up to 8 KB of Flash program memory and 1 KB register RAM. An on-chip temperature sensor allows temperature measurement over a range of –40 ºC to +105 ºC. These devices include two enhanced 16-bit timer blocks featuring PWM and Capture and Compare capabilities. An on-chip internal precision oscillator (5 MHz/32 KHz) can be used as a trimmable clock source requiring no external components. The Z8 Encore! XP devices include 128 bytes of non volatile data storage (NVDS) memory where individual bytes can be written or read. The full-duplex UART, in addition to providing serial communications and IrDA encoding and decoding capability, also supports multidrop address processing in hardware.

The rich set of on-chip peripherals make the Z8 Encore! XP MCUs 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 1969). This 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 three lines: Receive (Rx), Transmit (Tx), and GND. The handshake is executed in software by transmitting special XON and XOFF characters.

In most MCU applications, half-duplex communication is sufficient, which means that each side is either a receiver or a transmitter at any given time. In asynchronous serial data communication, data is transmitted sequentially, one bit at a time. The Tx 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. 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.

Features of software UART are listed below:

- Configurable baud rates such as 300, 600, 1200, 2400, and 4800 using internal precision oscillator (IPO)
- Supports half-duplex mode of communication
- Higher baud rates up to 57600 can be achieved using external oscillator
- Supports 8-N-1 mode
- Works in parallel with hardware UART
- Hardware flow control can be achieved by configuring any of the pins of Port A or Port C

Developing a Software UART for the Z8 Encore! XP MCU

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

Hardware Architecture

The hardware setup for the software UART is displayed in Figure 1. The setup displays the Z8 Encore! XP MCU connected by a MAX3222 line driver to a PC, running a HyperTerminal program with a setting of no handshake. Only Rx and Tx lines are used. 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 Rx and Tx lines are crossed. Pin PA5 is used for Tx and pin PA4 is used for Rx.

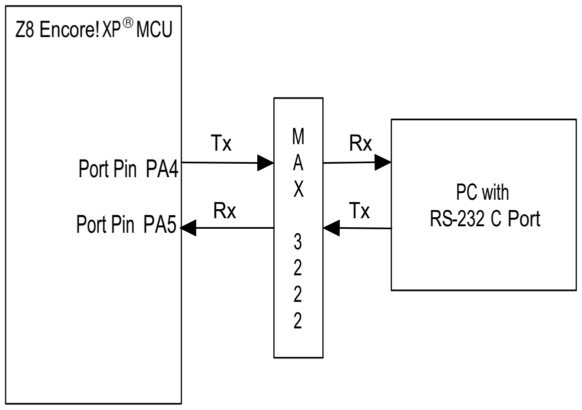

Figure 1. Block Diagram of Z8 Encore XP 4K connected to PC using MAX3222

Default settings for software UART are as follows:

- Port A pin 4 (PA4) is used as Rx pin
- Port A pin 5 (PA5) is used as Tx pin
- Baud rate of operation is 4800

Software Implementation

The software UART implementation comprises three basic operations:

- Initializing software UART
- Receiving data
- Transmitting data

The implementation of these operations is common for the software UART coded in C language.

Initializing Software UART

The software UART implementation, by default, uses Port A pin 4 and pin 5 as the Rx and Tx data pins respectively. However, you can configure any pin of Port A or Port C as Rx/Tx data pin by modifying the code in `global_configuration.h` file as described below:

```
//-----------------------------------
//User Configurable area
//------------------------------------
#define Rx_Port_Pin PA4 // User need
//to select port pin over here
#define Tx_Port_Pin PA5 // User need
//to select port pin over here



#define Hardware_Control_Port_Pin PA2
// User need to select port pin over
//here
#define Port_Pin_Number 4 // User need
//to enter port pin number (0..7)
#define Enable_Hardware_Control True
// If hardware control is required or
//not
```

The port pin numbers in the above code can be changed as required. For example; if you need to change the Rx pin then change the PA4 to the required pin number. `Port_Pin_Number` is used for getting the port pin interrupt vector address. You can enter values as per the port pin. For example; for Port A pin 0 value is 0 and Port A pin 7, the value is 7. This is used only for Rx as Rx is interrupt driven.

Following operations are performed sequentially during initialization:

1. Pin PA5 is set to OUTPUT mode and a High is output at PA5.
2. Pin PA4 is set to INPUT mode and initialized to generate interrupts at the falling edge of the signal.
3. The Timer0, which is used as a counter, is set in the CONTINUOUS mode of operation to generate interrupts upon reaching a set counter value.
4. The start value of the timer register is set to half the value of the reload register to sample the received data at the center of the bit.
5. The Timer0 is enabled during the Tx mode, and disabled otherwise. Port interrupts are enabled during Rx mode and disabled otherwise.

The flowchart of the main routine is displayed in Figure 7.

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 PA4 when a port interrupt is generated. This interrupt is handled by the `isr_gpio`, where the timer is enabled, and the port interrupts are disabled to receive the remaining data bits. The timer reload value is set to generate interrupts at one-bit intervals. The start value of the 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. If the middle of the first bit is zero, it indicates that the received bit is a valid Start bit and not a glitch.

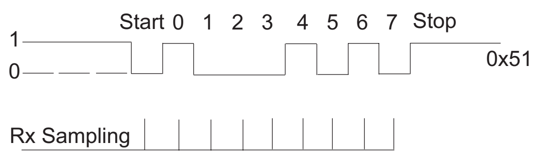

Figure 2. Rx Sampling while Receiving Data

The function `isr_timer0` performs the following tasks:

1. To receive transmission, the function ensures that the received bit is not a glitch and sets the receive bit counter, `Bit_Counter` 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 `Byte_Received register`, and `Bit_Counter` is incremented.
4. If `Bit_Counter` = 8, then `Bit_Counter` is reset and the timer is disabled.
5. The timer start value is reloaded. The Port A interrupts are enabled.

The flowchart for the timer interrupt service routine is displayed in Figure 10.

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.

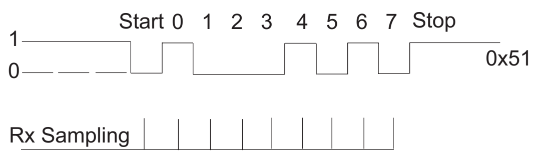

Figure 3. Tx Sampling while Transmitting Data

The function `Transmitt` performs the following tasks:

1. To start transmission, the transmit bit counter, `Bit_Counter`, is set to 0 and the Tx line (PA5) is pulled Low to send the Start bit. `Bit_Counter` is incremented.
2. The `Byte_To_Transmit` register contains the valid data to be transmitted. It is read, loaded into an intermediate buffer, and shifted out to PA5, one bit at a time, from bit 0 to bit 7. `Bit_Counter` is incremented after each bit is transmitted.
3. To stop transmission, the Tx line (PA5) is pulled High to signal the end of transmission; the timer is disabled.

API Definitions

This section describes APIs used in the implementation of software UART.

Init_SWUART()

This API initializes GPIO and Timer0 for the software UART using the following functions:

- `Init_GPIO()`—In this function, Port A or Port C pin PA4 is configured as input (Rx) and Port A or Port C pin PA5 is configured as output (Tx). The port pin settings are configurable. Any of the pin can be defined as output (Tx), but for input, pins which can generate interrupt on falling edge can be defined as input (Rx) pins.
- `Init_Timer0()`—Timer0 is used to generate timing for baud rate. Timer0 is configured in CONTINUOUS mode, prescalar 1, and interrupt on reload event. Timer0 is activated in port pin interrupt service routine and is disabled when byte is received or transmitted.

Receive()

In this API, if stop bit is set then it checks for stop bit condition. Otherwise, it checks for start bit condition. In start bit condition, level of port pin is checked to ensure the baud rate is unchanged. Then a timer is loaded with count of one bit time. Start bit is set, which indicates that start bit is received. Next time onwards data is shifted to a byte and timer is loaded with one bit time. After eight bits are received the API sets the stop bit. Stop bit condition is checked and once it is met, timer interrupt is disabled and port pin interrupt is enabled. Received byte is transferred to another byte from where you can access the byte.

Transmitt()

In this API, if start bit is sent then timer is loaded for a bit count and then data bits are shifted out on Tx pin and then finally stop bit is shifted. Once data is transmitted, byte transmitted bit will be set for you. This helps you to send another byte. Timer is again disabled when byte is transmitted.

Testing the Software UART Application

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

Equipment Used

The following equipments were used for testing:

- Z8 Encore! XP Development Kit (Z8F04A08100KIT) featuring Z8F042A - 28-pin
- ZDS II version 4.9.6 IDE for Z8 Encore! XP
- HyperTerminal application on the PC
- 9-pin serial cable

Test Setup

The software UART test setup for Z8 Encore XP! is displayed in Figure 4, Figure 5, and Figure 6.

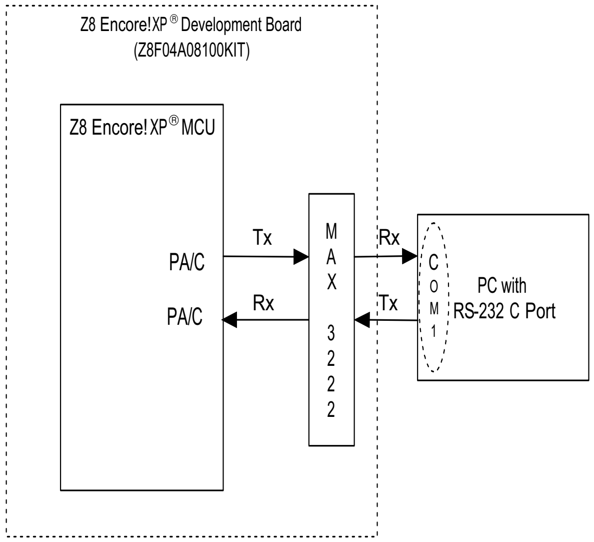

Figure 4. Communication between PC and Z8 Encore! XP on Software UART-Software Tx/Rx shorted with Hardware Tx/Rx

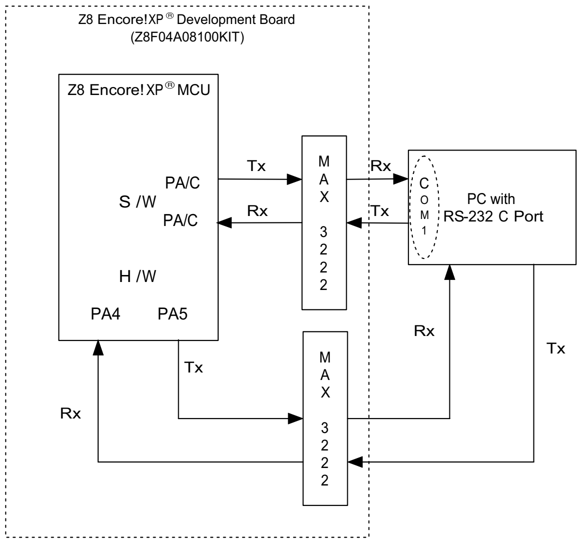

Figure 5. Communication between PC and Z8 Encore! XP® on Software UART as well as Hardware UART

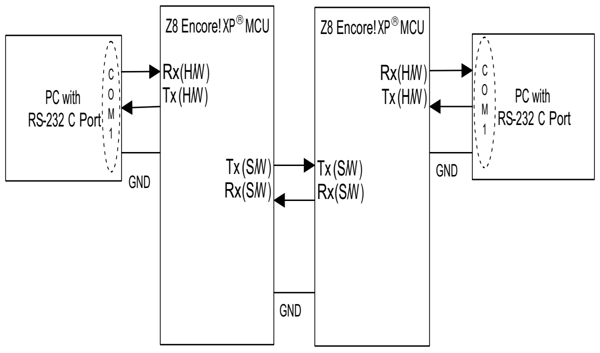

Figure 6. Communication between PC and Z8 on Software UART - Software Tx/Rx shorted with Hardware Tx/Rx

Procedure

The software UART application was tested using the following procedure:

1. Connect the software-configured Port A/C pin (Tx) to PA5 (Tx hardware pin). Connect the software-configured Port A/C pin (Rx) to PA4 (Rx hardware pin). The available serial driver MAX 3222 on the Z8 Encore! XP Development Board, described in Figure 4, is used for testing purposes.
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 **COM2** port in the **Connect using…** text field.
4. Click **Configure** button and in the **Port Settings** dialog box, enter the following in the text fields:
   - Bits per second: 4800
   - Data bits: 8
   - Parity: None
   - Stop bits: 1
   - Flow control: None
5. Click **OK** to get back to the **Connect to** tab. Click **Settings** tab and click **ASCII Setup** button. In the **ASCII Setup** dialog box, check the **Echo typed characters locally** option. Click **OK** to close the **Properties** dialog box.
6. Download the test application program using ZDS II.
7. The default mode is Rx. Upon entering a character in HyperTerminal, it is echoed back to the screen.
8. After a brief delay, the same character is displayed again in the HyperTerminal window. This indicates that the character was received by the software UART's Rx Data buffer, transferred to the Tx Data buffer, and transmitted back to HyperTerminal to be displayed on the screen.

This test procedure is repeated for all the baud rates listed in Table 1. To change the baud rates, the constants definition, BAUD, is used (see Table 1).

Results

The following results were obtained:

1. Testing with the baud rates and clock frequency settings indicated in Table 1 was successful.
2. In Rx mode, the software UART samples the data bit in the middle (50% of the bit cell) for all baud rates mentioned in the Table 1.
3. The received data is unaffected by jitter because the test program validates the Start bit before receiving any data.
4. Time taken by APIs for execution is less than 20 - 25% of the CPU time.

Table 1. Baud Rates Tested for R_X and T_X Modes at 5.5296 MHz Clock Frequency

Mode	Baud Rates Tested				
	300	600	1200	2400	4800
Rx	P	P	P	P	P
Tx	P	P	P	P	P

Summary

The software UART implemented in this Application Note supports the most common UART protocol, 8-N-1. The C language codes achieve data transfer at baud rates as high as 57600 at 20 MHz clock frequency. The size of the C language code is 832 bytes and is more suitable for larger Z8 Encore! modules, such as the Z8F04xx. Although the software UART operates only in the half-duplex mode, it is functionally comparable to the hardware UARTs on the Z8 Encore! MCU, and provides you with an additional UART when required. In addition, software UART works in parallel with hardware UART taking only 20 - 25% of CPU time at 1200 baud rate. Any of the pins of port A or port C can be configured as Rx and Tx. Hardware flow control can also be configured using these two port pins.

References

The documents associated with eZ8, Z8 Encore! XP, and ZDS II available on www.zilog.com are provided below:

- eZ8™ CPU User Manual (UM0128)
- Z8 Encore! XP® F082A Series Product Specification (PS0228)
- Z8 Encore! XP® F042A Series 8-Pin Development Kit User Manual (UM0187)
- Zilog Developer Studio II—Z8 Encore! User Manual (UM0130)

Appendix A—Glossary

Definitions for terms and abbreviations used in this Application Note that are not commonly used are listed in Table 2.

Table 2. Glossary

Term/Abbreviation	Definition
ISR	Interrupt service routine
IPO	Internal precision oscillator
Tx	Transmit
Rx	Receive
API	Application programming interface

Appendix B—Flowcharts

The basic Software UART process is displayed in Figure 7.

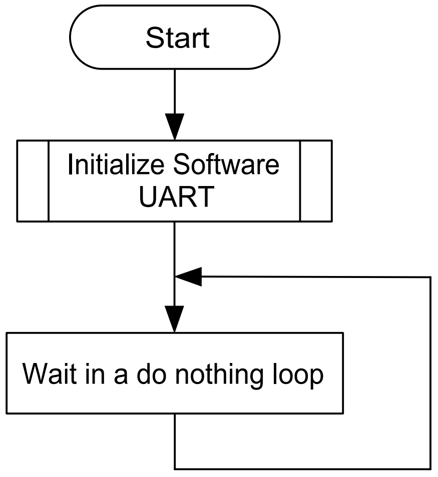

Figure 7. Basic Process Flow of Software UART

Figure 8 displays the software UART Initialization routine.

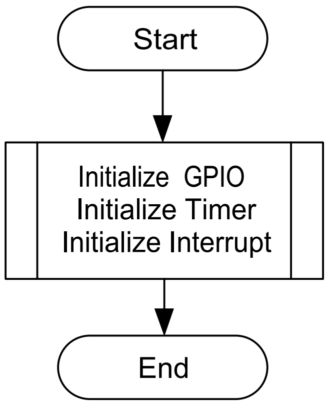

Figure 8. Software UART Initialization Routine

Figure 9 displays Port A Interrupt Service Routine.

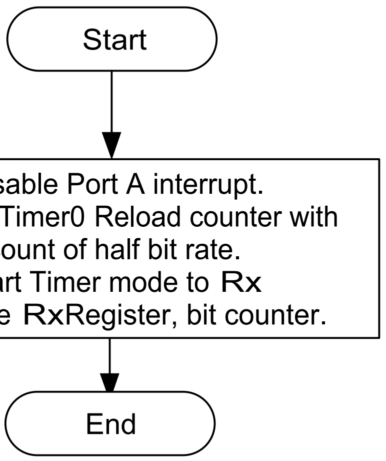
Start
Disable Port A interrupt.
Load the Timer0 Reload counter with count of half bit rate.
Start Timer mode to Rx
Clear the RxRegister, bit counter.
End

Figure 9. Port A Interrupt Service Routine

Figure 10 displays the Timer0 Interrupt Service Routine.

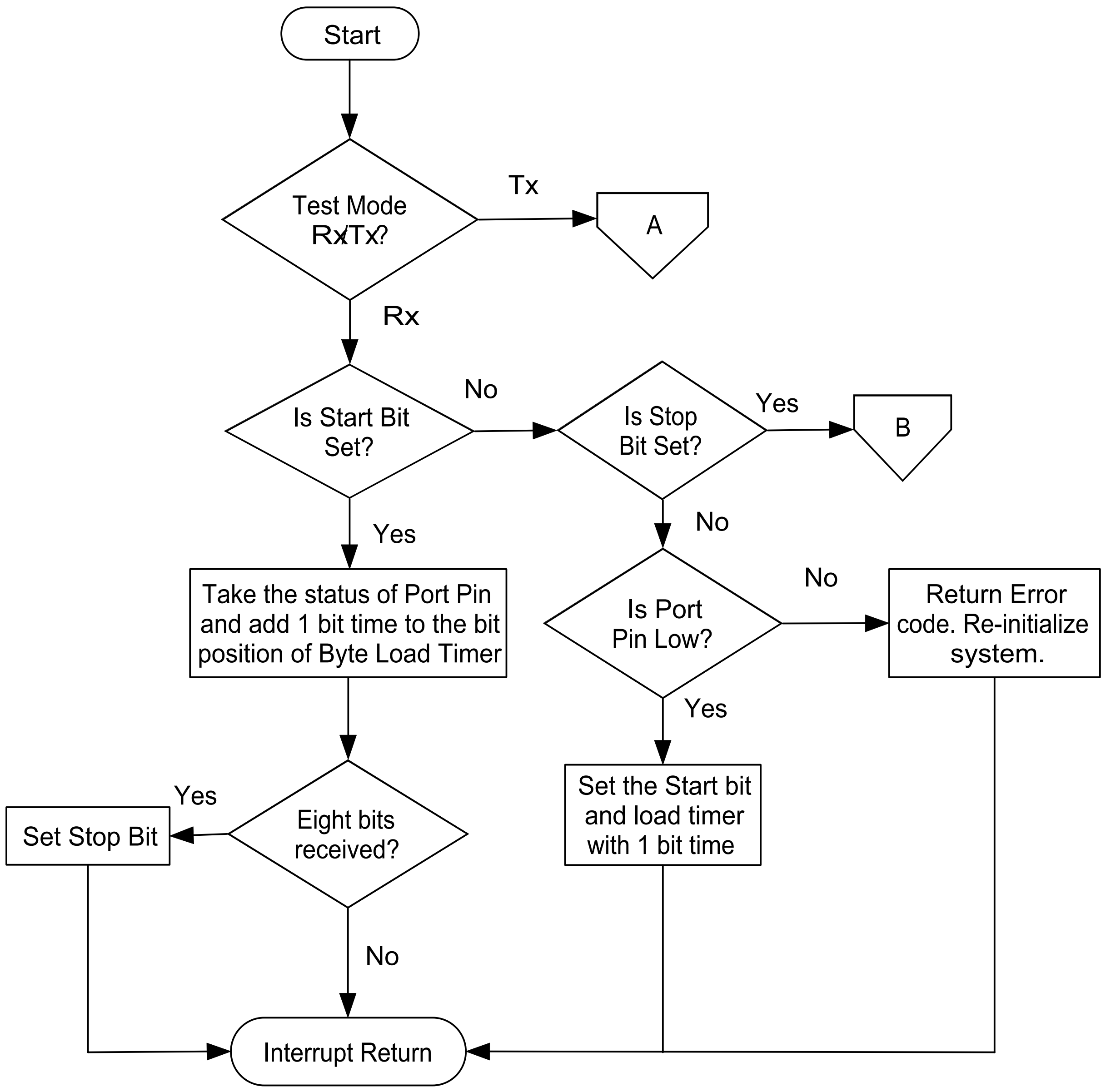
Start
Test Mode
Rx/Tx?
Tx
A
Rx
Is Start Bit
Set?
No
Is Stop
Bit Set?
Yes
B
Yes
No
Take the status of Port Pin and add 1 bit time to the bit position of Byte Load Timer
Is Port
Pin Low?
No
Return Error code. Re-initialize system.
Yes
Yes
Set Stop Bit
Eight bits
received?
Set the Start bit and load timer with 1 bit time
No
Interrupt Return

Figure 10. Timer0 Interrupt Service Routine

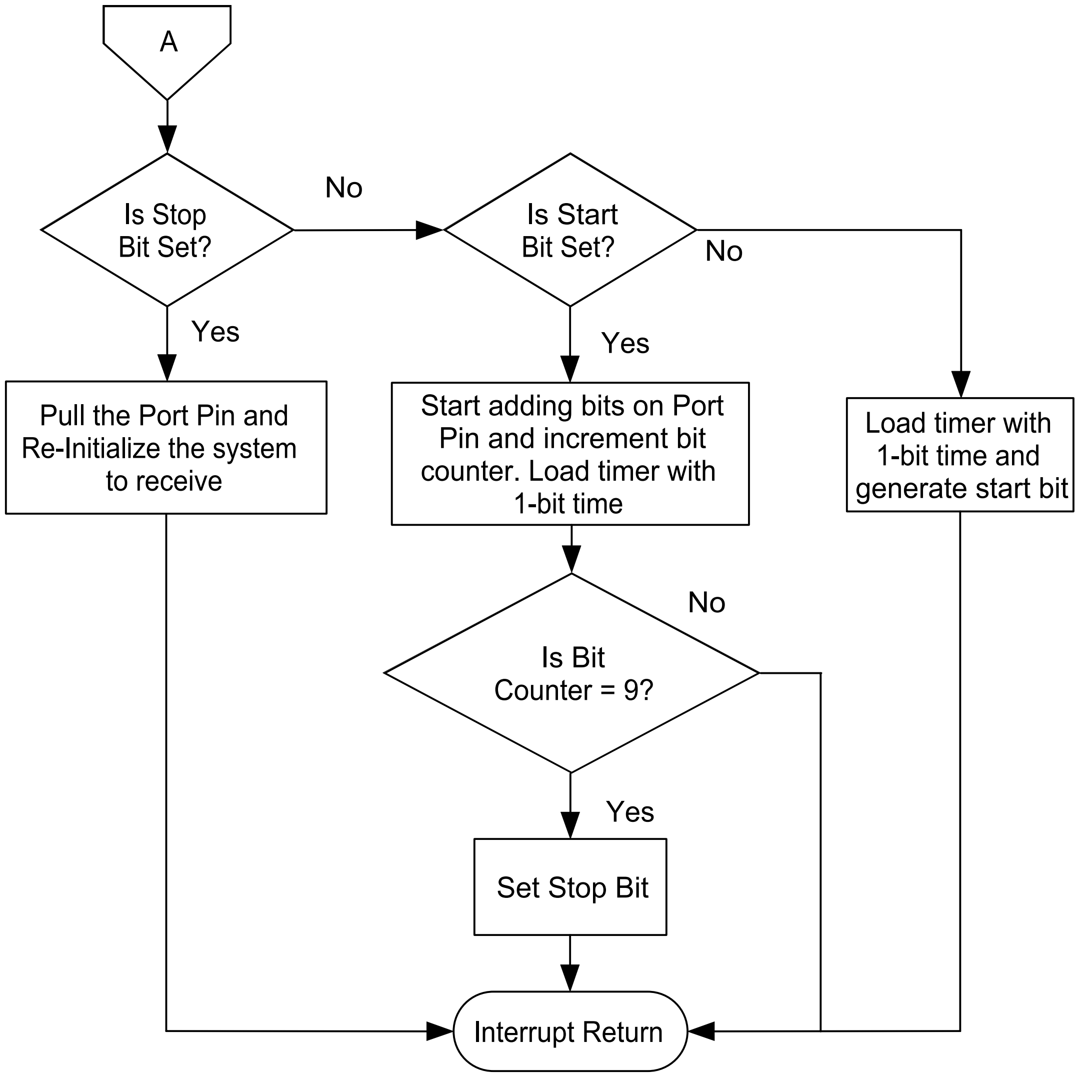
A
Is Stop Bit Set?
No
Yes
Is Start Bit Set?
No
Yes
Pull the Port Pin and Re-Initialize the system to receive
Start adding bits on Port Pin and increment bit counter. Load timer with 1-bit time
Load timer with 1-bit time and generate start bit
Is Bit Counter = 9?
No
Yes
Set Stop Bit
Interrupt Return

Figure 10. Timer0 Interrupt Service Routine (Contd)

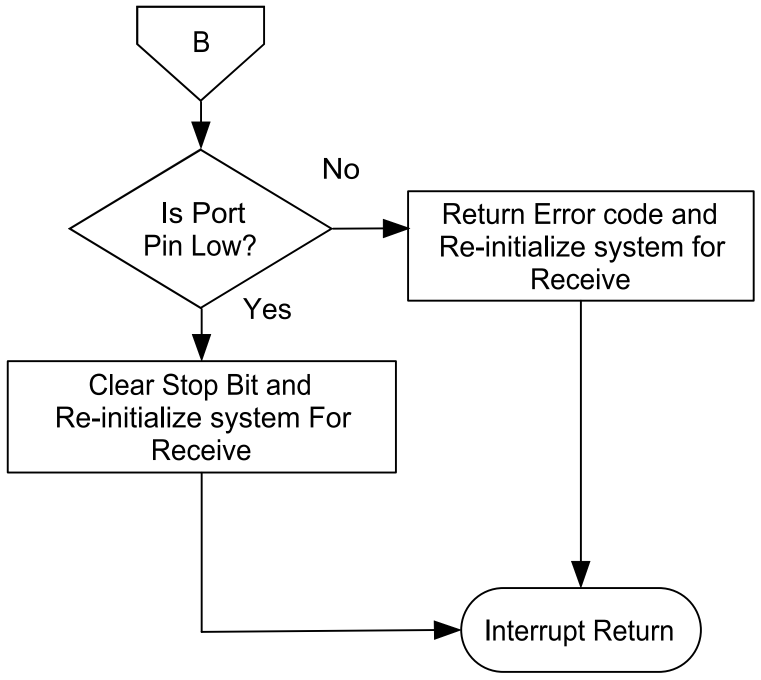
B
Is Port Pin Low?
No
Yes
Return Error code and Re-initialize system for Receive
Clear Stop Bit and Re-initialize system For Receive
Interrupt Return

Figure 10. Timer0 Interrupt Service Routine (Contd)

Warning: DO NOT USE IN LIFE SUPPORT