Application Note

Software UART for the eZ80F91 MCU

AN015303-0608

Abstract

This Application Note describes an implementation of a software-emulated Universal Asynchronous Receiver/Transmitter (UART) for Zilog's eZ80F91 8-bit microcontroller unit (MCU). Software UART implementation is useful for applications where an extra UART is required in addition to the hardware UART(s) available with eZ80Acclaim*Plus!*™ MCUs. These hardware UARTs are full-duplex, whereas the software UART implementation is half-duplex. The software UART is also event-driven and supports an 8-N-1 protocol, using an RS-232 interface.

Note: *The source code file associated with this application note, AN0153-SC01 is available for download at* www.zilog.com.

Data transfer at baud rates from 300 bps to 9600 bps for the RAM version and from 300 bps to 4800 bps for the Flash version, is achieved at different clock frequencies (32 MHz to 50 MHz). The software provides APIs for basic operations such as initialization, and receiving and transmitting data.

eZ80Acclaim*Plus!* Flash Microcontrollers

eZ80Acclaim*Plus!* on-chip Flash Microcontrollers are an exceptional value for customers designing high performance, 8-bit MCU-based systems. With speeds up to 50 MHz and an on-chip Ethernet MAC (eZ80F91 only), you have the performance necessary to execute complex applications quickly and efficiently. Combining Flash and SRAM, eZ80Acclaim*Plus!* devices provide the memory required to implement communication protocol stacks and achieve flexibility when performing in-system updates of application firmware.

The eZ80Acclaim*Plus!* Flash MCU can operate in full 24-bit linear mode addressing 16 MB without a Memory Management Unit. Additionally, support for the Z80-compatible mode allows Z80/Z180 customers to execute legacy code within multiple 64KB memory blocks with minimum modification. With an external bus supporting eZ80®, Z80®, Intel®, and Motorola® bus modes and a rich set of serial communications peripherals, you have several options when interfacing to external devices.

Some of the many applications suitable for eZ80Acclaim*Plus!* devices include vending machines, point-of-sale terminals, security systems, automation, communications, industrial control and facility monitoring, and remote control.

Discussion

The UART protocol is based on the EIA RS-232C standard, published in 1969. That 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, Receive (Rx), Transmit (Tx), and GND. The handshake is executed in software by transmitting Xon and Xoff characters. In most MCU applications, the 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 as characters intermittently. Each character is made up of a sequence of bits. 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.

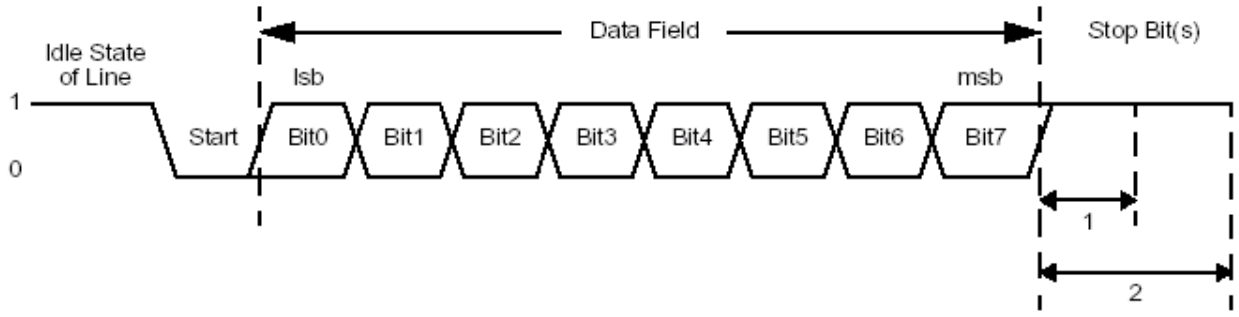

Figure 1. Basic 8-bit UART Protocol

In an asynchronous operation, the clock is not transmitted. The receiver must operate with the same baud rate as the transmitter, which 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 the Software UART for eZ80F91

The eZ80F91 Software UART supports the basic format 8-N-1, which is 8 data bits, no parity and 1 stop bit. It communicates in the half-duplex mode. In the Rx mode, the program waits to receive data bytes and stores them in the Rx Data buffer. In the Tx mode, the program sends out the data byte that was stored in the Tx Data buffer.

Assembly Options

Several options at assembly time can be selected to adapt the program to the required operation. Table 1 list these options.

Table 1. Options at Assembly Time

Variable Name	Description
BAUD	Baud rate: 300 to 9600. When BAUD is specified, the program selects all of the appropriate timings. Default is 4800.
MODE	Rx or Tx. Default is Rx.
TEST_CONTROL	CHAR/FILE: Option to select character transfer or file transfer. Default is CHAR.
BUFF_SIZE	Specifies the buffer size. To avoid an overrun, the BUFF_SIZE must be larger than file size.

Hardware Architecture

Figure 2 displays the hardware setup to accommodate the Software UART. The setup consists of the eZ80F91 MCU connected via a MAX-3245CAI serial line driver to a PC running the HyperTerminal application configured with a setting of *software flow control*. Only Rx and Tx lines are used for the Software UART communication.

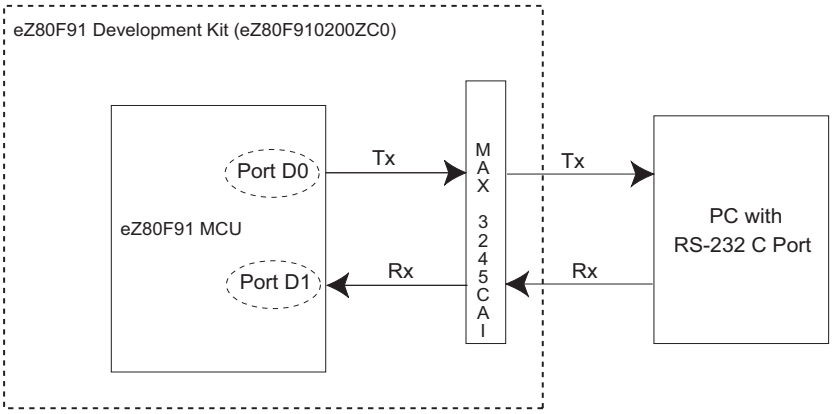

Figure 2. The eZ80F91 MCU Connected to a PC

In this case, both the HyperTerminal application on the PC and the Software UART application loaded on the eZ80F91 MCU are programmed with the same baud rate, with one as transmitter and the other as receiver; the Rx and Tx lines are crossed. Pin PD0 is used for transmitting (Tx line) and Pin PD1 is used for receiving (Rx line).

Software Implementation

Figure 3 displays the eZ80F91 memory map. The Software UART code occupies only 2.5 KB of memory space.

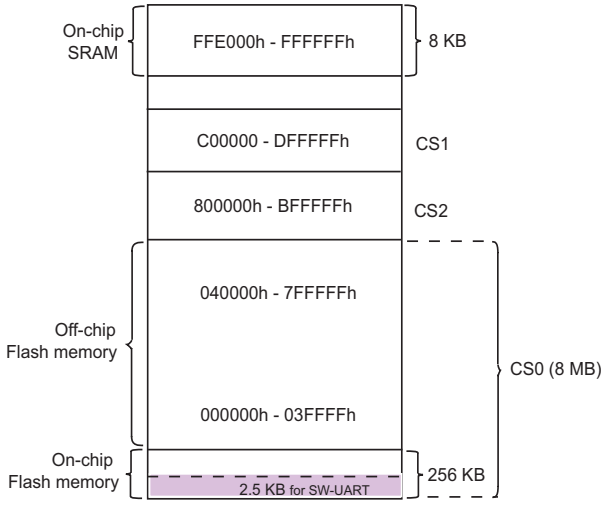

Figure 3. Memory Map for Software UART on eZ80F91 MCU

The Software UART implementation consists of three basic functionalities:

- Initialization
- Transmitting a Character
- Receiving a Character

Initialization

In this implementation, PD0 is the Tx pin and PD1 is the Rx pin on the eZ80F91 MCU. Initialization performs the following operations:

1. Sets pin PD0 to the output mode and outputs a High at PD0.
2. Sets PD1 to the input mode and initializes it to generate interrupts at the falling edge of the signal.
3. The timer, which is used as a counter, is set in the continuous mode of operation to generate interrupts on reaching the 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 Tx mode, and disabled otherwise. Port-A interrupt is enabled and the timer is disabled during the Rx mode.

Figure 8 on page 9 displays the flowchart of the `main()` routine.

Receiving a Character

To receive a character, the Start bit must be detected by the falling edge of the signal at pin PD1(Rx) to generate an interrupt. This interrupt is handled by the Port ISR (see Figure 9 on page 10), wherein 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, occurs at the middle of the Start bit (see Figure 4).

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

The receive character API, `Z_Get_Char()`, continually calls the `Z_Get_Bit()` function to get the received character. The `Z_Get_Bit()` function performs the following tasks:

1. To receive transmission, the function ensures that the received bit is not a glitch, sets the `valid_data` flag to `TRUE`, and sets the receive bit counter `Rx_Bit_Count` to 0. The data is sampled in the middle of the next bit.
2. The function stores the data bit in the appropriate bit position and increments `Rx_Bit_Count`.
3. When `Rx_Bit_Count = 8`, the `Z_Get_Bit()` function stops receiving bits. The timer start value is reloaded, and the `valid_data` flag is set to `FALSE`. The Port A interrupts are enabled, and the timer is disabled.
4. If the previous value of the `Current_Rx_Data` buffer is read by the user application, the `Z_Get_Bit()` function copies the current data into the `Current_Rx_Data` buffer. If the `Current_Rx_Data` buffer is not read by the user application, the `Z_Get_Bit()` function sets the `Over_Run` flag to indicate that a data overrun has occurred.

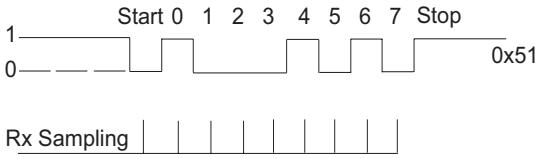

Figure 4. Rx Sampling While Receiving Data

Figure 12 on page 12 displays the flow of the transmitting data routine.

Transmitting a Character

To transmit a character, 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 5 on page 5).

The transmit character API, `Z_Put_Char()`, continually calls the `Z_Put_Bit()` function to send the character to be transmitted.
The `Z_Put_Bit()` function performs the following tasks:

1. To start the transmission, the `Z_Put_Bit()` function sets the transmit bit counter `Tx_Bit_Count` to 0 and pulls the Tx line (PD0) Low to send the Start bit.
   The `Z_Put_Bit()` function also increments the `Tx_Bit_Count`. The `TX_DATA` register contains the valid data to be transmitted.
2. The `Z_Put_Bit()` function reads the `TX_DATA` register, loads the data into an intermediate buffer, and shifts out the data to Port A0 (one bit at a time) from bit 0 to bit 7. The `Z_Put_Bit()` function also increments the `Tx_Bit_Count` after each bit is transmitted (see Figure 5 on page 5).

3. To stop the transmission, the `Z_Put_Bit()` function pulls the Tx line (PD0) High and disables the timer.

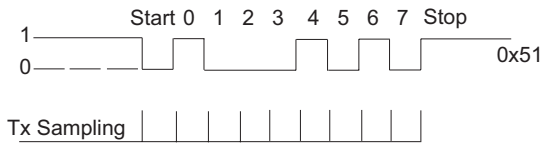

Figure 5. Tx Sampling While Transmitting Data

Figure 11 on page 11 displays the flowchart for transmitting data routine.

Note: *Interrupt handling with the eZ80F91 MCU is described in the Setting Interrupts with the eZ80® CPU Application Note (AN0170) available for download at www.zilog.com.*

Flow Control—Following two APIs are provided to control the flow of data while receiving and transmitting data:

- `Z_Start_Commu()`
- `Z_Stop_Commu()`

Calling the `Z_Start_Commu()` API resumes the communication, while calling `Z_Stop_Commu()` API stops the communication between the communicating devices. This process is called *software flow control*.

Overrun Error—A data overrun error occurs when the Rx Data buffer receives data before the previously-received data is read and emptied. The data overrun flag is set when an overrun occurs, and the flag is reset when the data in the Rx Data buffer is read and emptied.

Testing the Software UART Application

This section describes the equipment and procedure used to test the Software UART demonstration developed for the eZ80F91 MCU.

Note: *The test procedure provided in this Application Note is for the RAM version of the Software UART application. To test the Flash version of the Software UART application, you must use the Flash Loader to burn the application into Flash memory.*

Equipment Used

The equipment to test this demonstration include:

- eZ80F91 Development Kit (eZ80F910200ZC0)
- eZ80Acclaim*Plus!* Development Kit (eZ80 ASSP)
- PC with Windows 95/98/NT/XP
- ZDS II for eZ80Acclaim!® v4.10.0
- HyperTerminal application on the PC

Note: *The XTAL used with the eZ80F91 Development Kit ranges from 32 MHz to 50 MHz and the default baud rate is 4800 for Rx/Tx mode. The RAM version of the Software UART application operates at speeds up to 9600 baud, while the Flash version operates at speeds up to 4800 baud.*

Figure 6 on page 6 displays the test setup for the Software UART demonstration.

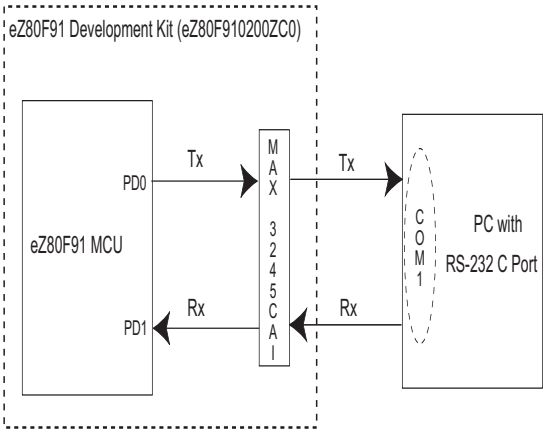

Figure 6. Test Setup for the eZ80F91 MCU Software UART

Procedure

This section describes the procedure to test the Software UART demonstration. Specifically, two test procedures are outlined for the RAM version of the Software UART—to test transmission and reception of a single character, and to test transmission and reception of a file that includes testing for overrun.

The initial steps which are common for both the types of tests are described below. In addition to these test procedures, the steps required to test the Flash version of the Software UART are described in Testing the Flash Version of the Software UART Application on page 7.

Note: *The console port of the eZ80® Development Platform is used to test UART functionality.*

Common Initial Procedure for Character and File Transfer

The common initial steps to be performed before testing character or file transfers are described below:

The pins PD0 (to transmit data) and PD1 (to receive data) are connected to the available serial driver, MAX 3245CAI, on the eZ80® Development Platform. The PD0 and PD1 pins are also used as the UART Tx and Rx data pins when the alternate function is selected for these pins (see Figure 6).

1. Connect the console port of the eZ80® Development Platform to the standard PC running on Windows NT with a 9-pin straight cable.
2. Launch the HyperTerminal application and set it up for direct communication on the COM2 port with the following settings:
   - In the **Port Settings** dialog box: 9600 baud, 8-bit, no parity and 1 stop bit
   - In the **ASCII Setup** dialog box: check the *Echo typed characters locally* option to compare the entered character with the echo received from the Software UART application. This setting is not mandatory.
3. Launch ZDS II and open the Software UART application project, `sw_uart.zdsproj`.
4. Select **eZ80F91–RAM** in the **Select Active Configuration** text field on the ZDS II menu bar.

Character Transfer Test Procedure

Follow the steps below to test for character transfer using the Software UART application (after completing Steps 1 to 4 described in Common Initial Procedure for Character and File Transfer on page 6):

1. From within ZDS II, open the `main.c` file. Specify CHARACTER TRANSFER mode using `#define TEST_CONTROL`, as shown below:

```
#define TEST_CONTROL  CHAR
// select either CHAR/FILE
```

2. Build the `sw_uart.zdsproj` project file and download the output file to the target device.

3. Run the program. The `eZ80Acclaim!>` prompt appears in the HyperTerminal window.
4. The default mode is Rx. Enter a keyboard character. The HyperTerminal window echoes the character back to the screen if the **echo back** option is checked.

   After a brief delay, the same character is displayed again in the HyperTerminal window, indicating that the character is received by the Software UART's Rx Data buffer, transferred to the Tx Data buffer, and routed to the HyperTerminal application to be displayed on the screen.
5. The procedure is repeated for all baud rates at different XTAL frequencies as indicated in Table 2 on page 8. To change the baud rates, the constant definition BAUD (see Table 1 on page 2) is modified in the `sio.h` file. Correspondingly, use the same baud rate as a HyperTerminal setting.

File Transfer Test Procedure

Follow the steps below to test for file transfer using the Software UART application (after completing Steps 1 to 4 described in Common Initial Procedure for Character and File Transfer on page 6):

1. From within ZDS II, open the `main.c` file. Specify FILE TRANSFER mode using the `#define TEST_CONTROL` string as shown below, and select the buffer size using the `#define BUFF_SIZE` string:

```
#define TEST_CONTROL  FILE
// select either CHAR/FILE
#define BUFF_SIZE  10
// select a buffer size for
// the file
```

2. Build the project file, `sw_uart.zdsproj`, and download the output file to the target device.
3. Run the program. The `eZ80Acclaim!>` prompt appears in the HyperTerminal window.
4. In the HyperTerminal window, navigate to **Transfer → Send Text File**. Select the appropriate text file in the **Send Text File** dialog box and click **Open**. The size of the file must be greater than the size of the buffer (in this case, greater than 10).

   On filling the buffer, the program displays a *Buffer full* message in the HyperTerminal window and prints the information stored in the buffer. If the processor is busy performing another task, the message `OVERRUN ERROR` is displayed via HyperTerminal. The entire process continues until the file is transferred completely.
5. The procedure is repeated for all baud rates at different XTAL frequencies as indicated in Table 2 on page 8. To change the baud rates, the constant definition BAUD (see Table 1 on page 2) is modified in the `sio.h` file. Correspondingly, use the same baud rate as a HyperTerminal setting.

Testing the Flash Version of the Software UART Application

Follow the steps below to test the Flash version of the demonstration (after completing Step 1 to Step 3 in the Common Initial Procedure for Character and File Transfer on page 6):

1. Select **eZ80F91–Flash** in the **Select Active Configuration** text field on the ZDS II menu bar.
2. Execute Step 5 of the Character Transfer Test Procedure or the File Transfer Test Procedure.
3. Execute a **Build** to generate the `sw_uart.hex` file.
4. Use the Flash Loader to burn the `sw_uart.hex` file into eZ80F91 MCU Flash memory.

Note: *The sequence for burning the program into Flash is **Erase → Program → Verify.***

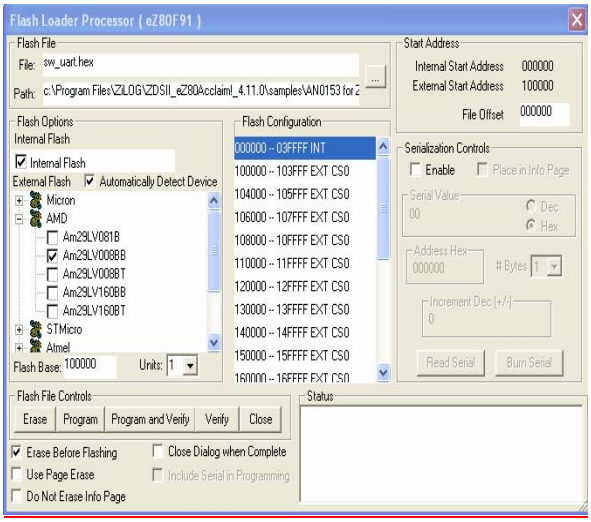

Figure 7. Flash Loader Processor

5. Reset the eZ80® Development Platform and continue the test procedure from Step 4 in the File Transfer Test Procedure on page 7.

Results

Testing with baud rates and clock frequency settings listed in Table 2 was successful. In Rx mode, the Software UART demonstration samples the data bit in the middle (50% of the bit cell) for all baud rates listed in Table 2. The received data is unaffected by jitter as the test program validates the Start bit before receiving any data.

Table 2. Baud Rates Tested at Different Clock Frequencies (RAM Version)

	Baud Rates Tested for Rx and Tx				
XTAL	**300**	**1200**	**2400**	**4800**	**9600**
32 MHz	P	P	P	P	P
40 MHz	P	P	P	P	P
50 MHz	P	P	P	P	P

Summary

The Software UART demonstration implemented in this Application Note supports the most common UART protocol, 8-N-1. The code, written in C language, achieves data transfer at speed as high as 9600 baud for the RAM version, and 4800 baud for the Flash version, at a clock frequency of 50 MHz.

The Software UART application can operate only in half-duplex mode. However, it features the functionality to provide designers with an additional UART when required.

References

The documents associated with eZ80F91, eZ80Acclaim*Plus!*, and ZDS II available on www.zilog.com are provided below:

- eZ80F91 MCU Product Specification (PS0192)
- eZ80F92/eZ80F93 Product Specification (PS0153)
- eZ80Acclaim*Plus!* Connectivity ASSP eZ80F91 ASSP Product Specification (PS0270)
- Zilog Developer Studio II –eZ80Acclaim!® User Manual (UM0144)

Appendix A—Flowcharts

Figure 8 displays the flowchart of the main routine of the Software UART demonstration.

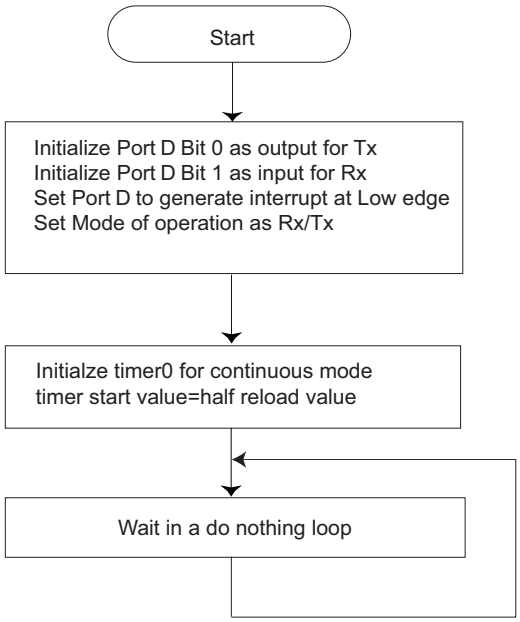

Figure 8. Flowchart for Main Routine

Figure 9 and Figure 10 display the flowcharts of the Port and Timer interrupt service routines, respectively.

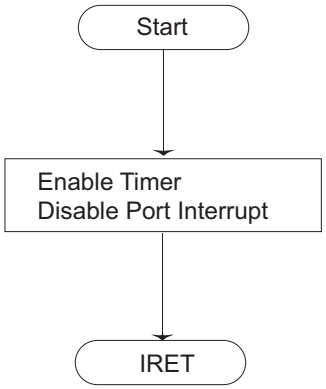

Figure 9. Flowchart for Port ISR

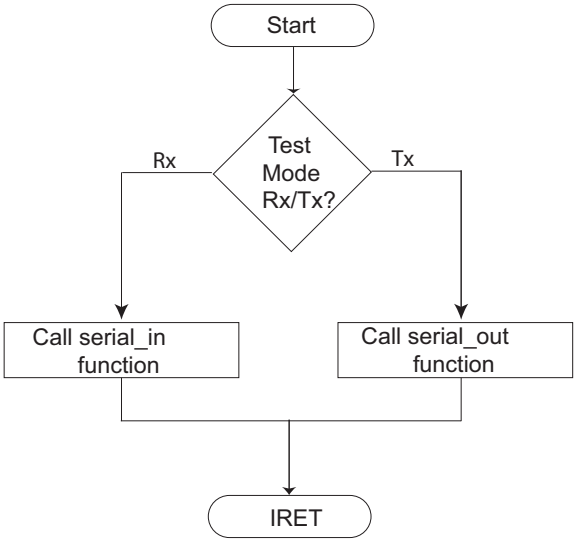

Figure 10. Flowchart for Timer ISR

Figure 11 displays the flowchart for transmitting data (Serial Data Out).

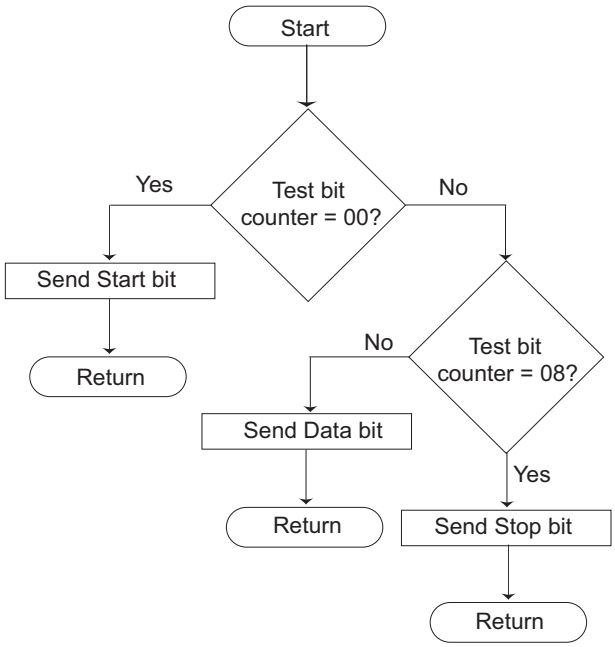

Figure 11. Flowchart for Transmitting Data

Figure 12 displays the flowchart for receiving transmitted data (Serial Data In).

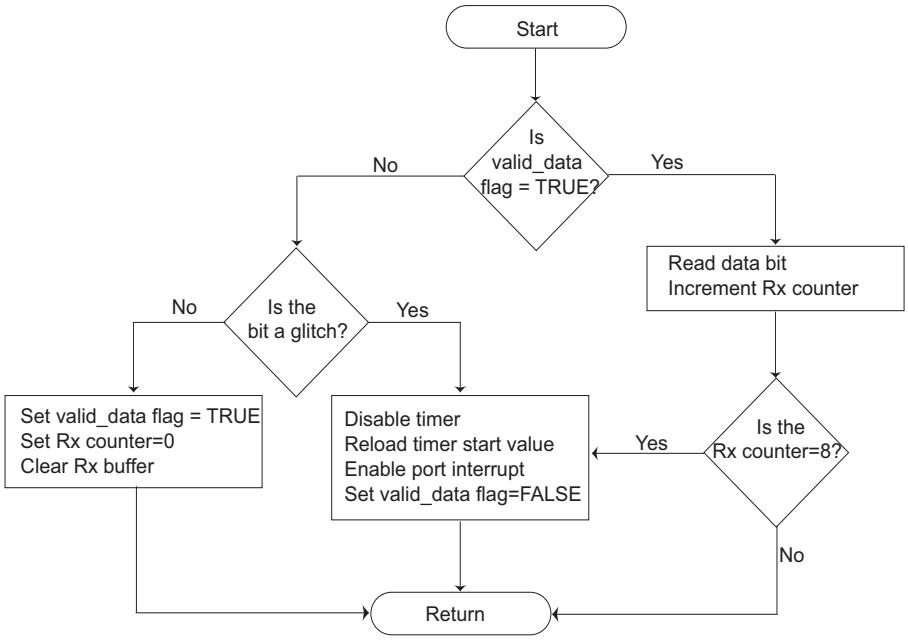

Figure 12. Flowchart for Receiving Data

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