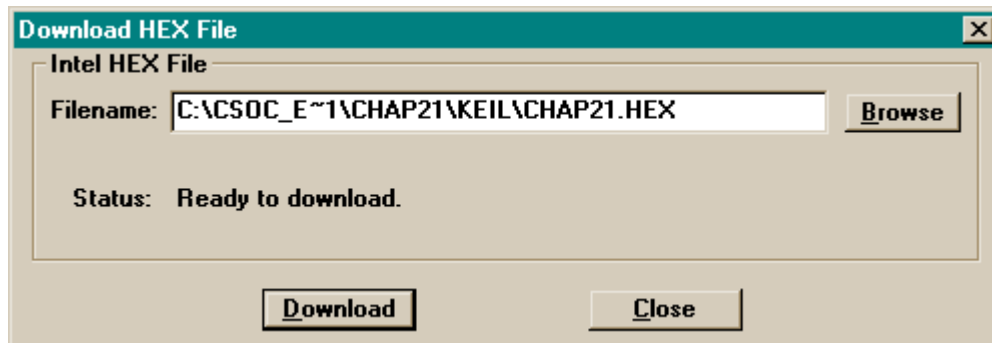


This brings up a **Download HEX file** window where you can specify the name or browse to the location of the updated HEX file. Click on Download to reload the CSoC with the updated application code.



After the CSoC has been updated, you still need to load the debugging info for the new source program into dScope. Do this by clicking on the File⇒Load Object File... menu item and select the chap21 OMF file.

Once the updated debugging information is loaded, click on the Reset button in the **Toolbox** window and then click on Go! in the **Module** window. With the program running, you should see that the DIP switch buttons have the opposite effect on the activation of the LED digit segments.

After verifying the operation of your modified program, click Stop! to halt the program. Then select File⇒Exit in the dScope window to close the debugger. You can also exit from the Keil IDE and the FastChip program.

Design 2.2 - UART Loopback

Your next MCU-based CSoC design will use the UART in the 8032 to transmit and receive bytes of test data (Figure 14). The MCU will continuously load the UART transmitter and poll the receiver to see if the received byte is identical to the transmitted byte. An AND gate will be placed in the loopback path so the transmitted byte can be blocked or passed through based on the setting of DIP switch 1. The MCU will display an O on the LED digit as long as the loopback path is not broken (**gate** = 1). When the path is broken (**gate** = 0), the transmitted and received bytes will no longer match and the MCU will display an E.

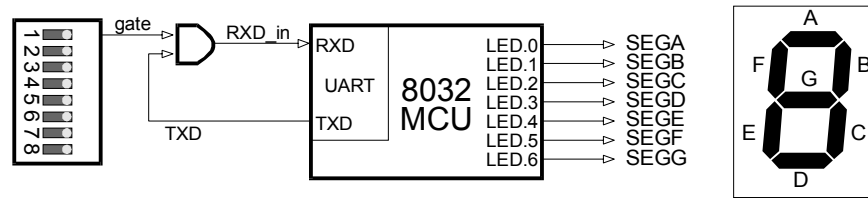


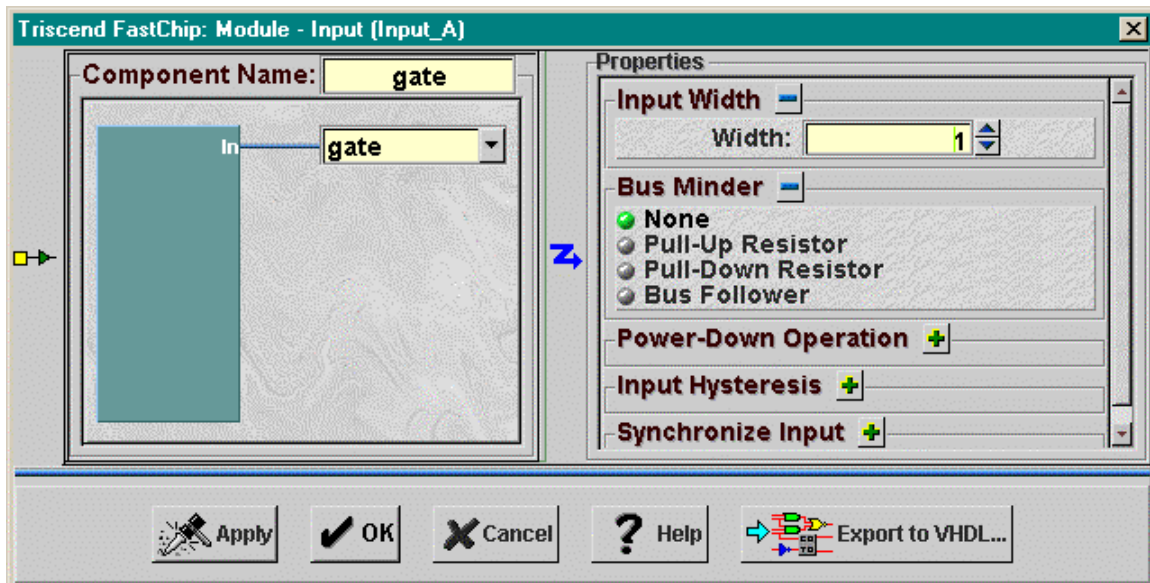
Figure 14: Block diagram of a UART loopback circuit.

This design will use the LED drive circuit from the previous example and will add the loopback circuit in the CSL. Then you have to write some code to make the 8032 initialize the UART and transmit and receive bytes in a polled fashion. You will create the hardware in the next section and then go on to write the software.

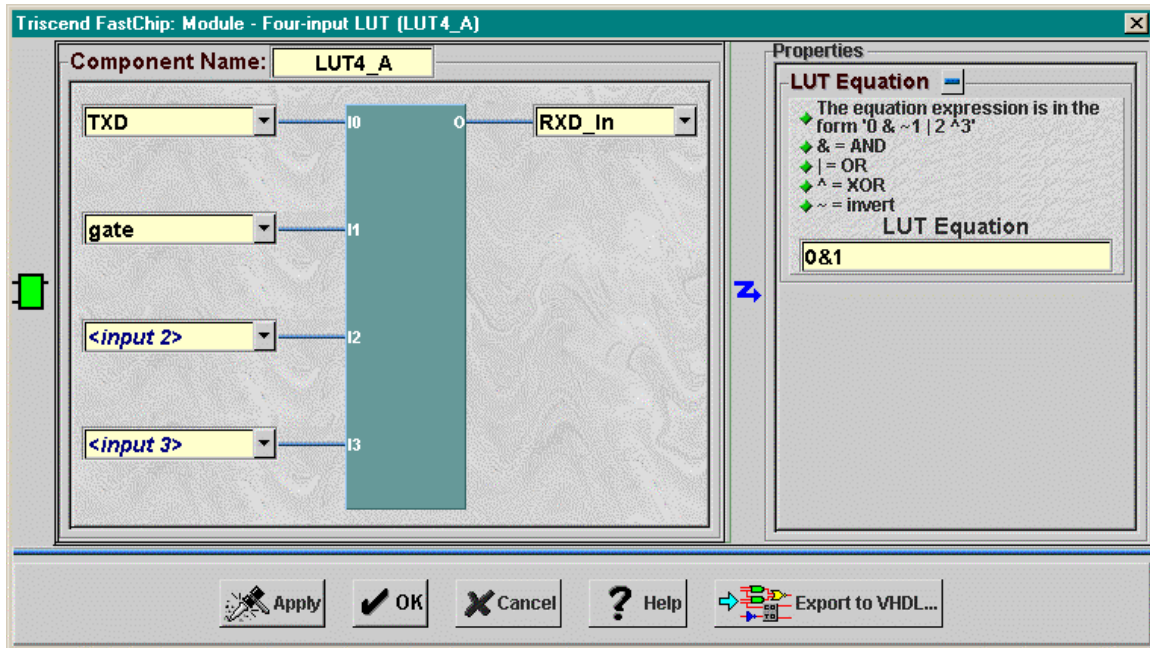
Building the Loopback Circuit

You can start this design in FastChip by opening the **Chap21** design from the previous example and then saving it under the name **Chap22**. Then drag the **P0** module in the Programmable I/O Pins area to the wastebasket because you won't need this MCU I/O port.

Next, drag an **Input** module from the I/O section of the Triscend Library area and drop it into the Programmable I/O Pins area. Click on the **Input** module to bring up the **Module** window. Set the input width to one bit and rename the input net from **<input>** to **gate**. Also rename the component as **gate**. Your input module should appear as in the following window. Click on OK to finalize your changes.

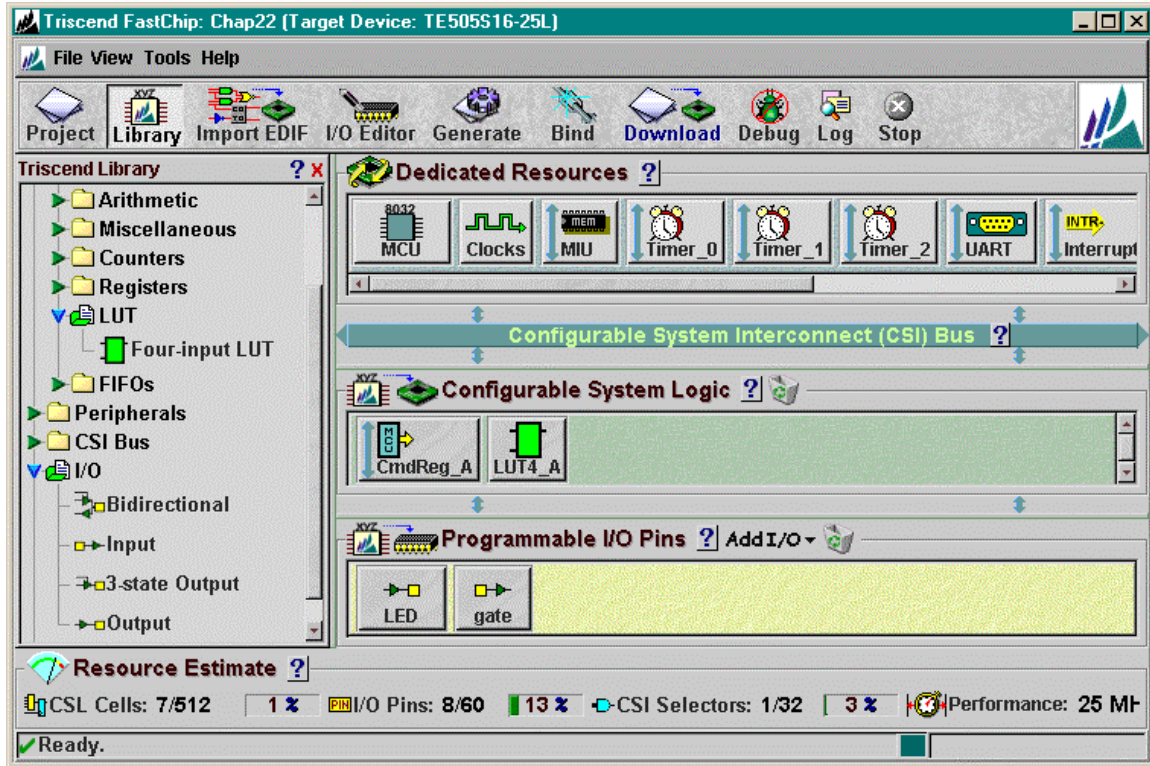


Now drag an **LUT** module from the Logic Modules section of the Triscend Library and drop it into the Configurable System Logic area. Edit the LUT information so it looks like the following:

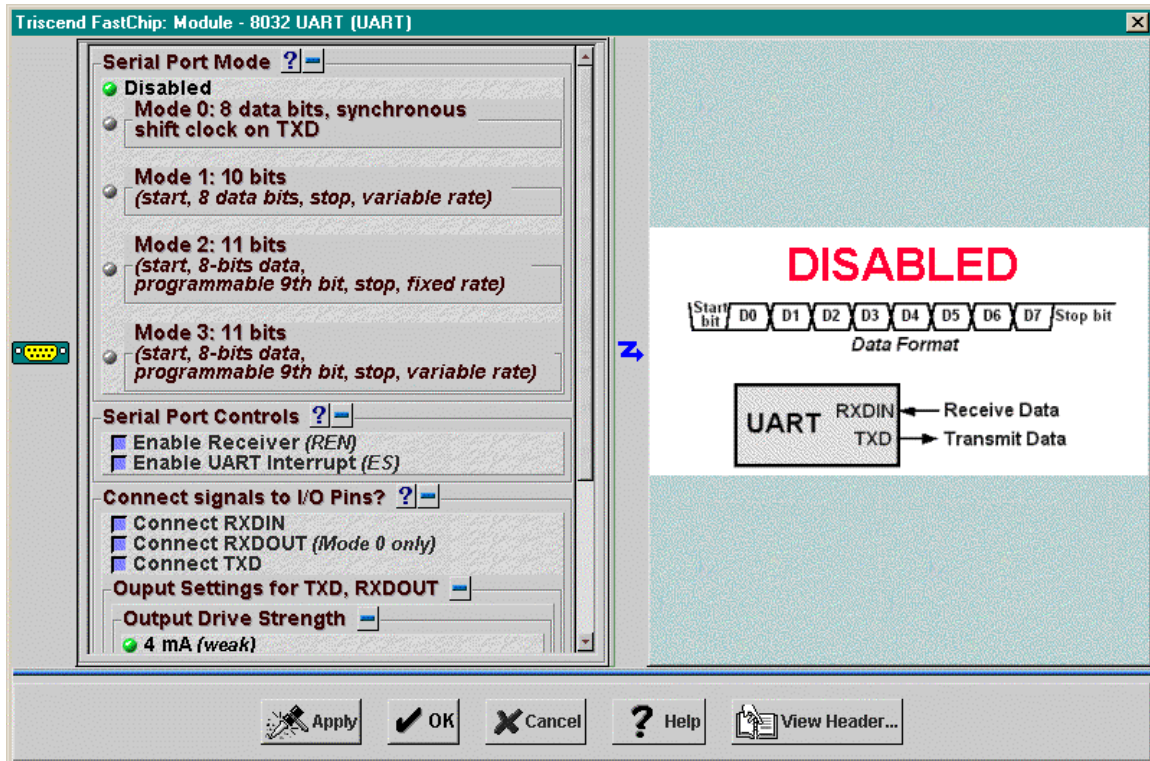


Input **I0** of the LUT is connected to the transmitter output of the 8032 UART (**TXD**). The **I1** input to the LUT is driven by the **gate** input net. The LUT is programmed with an equation that logically ANDs these two inputs. The output of the AND operation is connected to the input to the UART (**RXD_in**).

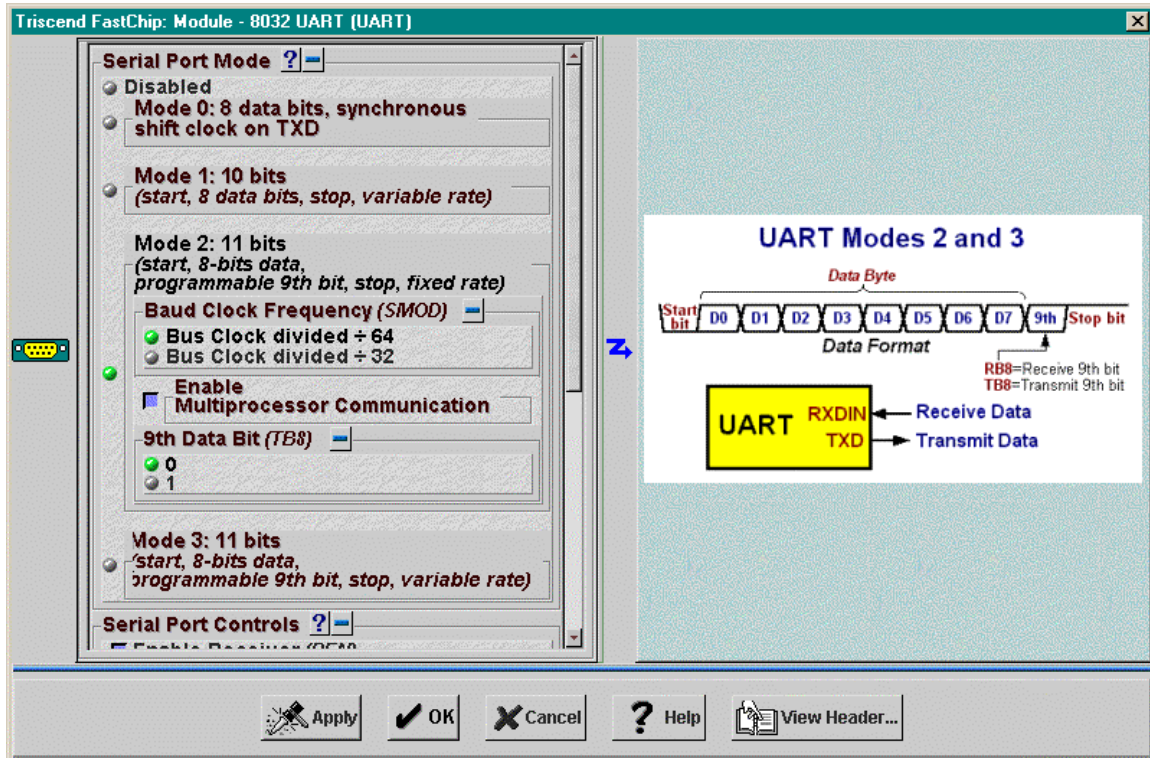
Once the changes to the LUT are completed, your FastChip project window should look like the one below.



Now you can set-up the UART. Click on the UART button and the **8032 UART** window appears. The UART transmitter and receiver signals loopback through the LUT in the CSL so these signals never appear on the pins of the CSoC. Therefore, you should uncheck all the boxes in the Connect signals to I/O Pins? area. (You would check the RXDIN and TXD boxes if you ever wanted to interface the UART to an external device. You could then assign **RXDIN** and **TXD** to pins of the CSoC using the I/O Editor.)



Next you need to select an operational mode for the UART. The UART is disabled by default. I picked Mode 2 for this example because it is easy to set-up. Mode 2 transmits eight data bits and a programmable ninth bit along with the start and stop bits required for serial communication. The bits are transmitted at 1/64 or 1/32 of the 25 MHz bus clock rate (390 or 781 Kbps, respectively). The ninth bit can be set to always be a logic 1 or 0. You can pick either transmission speed and ninth-bit setting for this example. This completes the set-up for the UART so click on OK.



Once you have the hardware modules instantiated and connected, you can use the I/O Editor window to make the pin assignments shown in Table 7.

Table 7: Pin assignments and functions for the UART loopback design.

| Signal | Pin | CSoC Board Resource |
|--------|-----|------------------------|
| gate | 53 | DIP switch position #1 |
| LED.0 | 35 | LED digit segment A |
| LED.1 | 39 | LED digit segment B |
| LED.2 | 43 | LED digit segment C |
| LED.3 | 41 | LED digit segment D |
| LED.4 | 40 | LED digit segment E |
| LED.5 | 34 | LED digit segment F |
| LED.6 | 36 | LED digit segment G |

Writing the Loopback Application Code

Your next step is to press the Generate button on the FastChip project window toolbar to make the Chap22.h header file that links the CSoC hardware and software. Then begin to build the C application code for the loopback design by creating a folder named keil within the Chap22 FastChip project folder. Start the Keil IDE, open a file window, and enter the C code shown in Listing 3.

The C code works as follows. Line 1 includes the definitions and initialization subroutines for the CSoC hardware. The main initialization subroutine is called on line 10. Then the UART receiver is enabled on line 11 by setting the REN flag bit (REN is defined in the Chap22.h file).

An infinite for loop starts on line 12 by writing an 8-bit counter value to the UART transmitter buffer (SBUF) on line 14. Then the program polls the transmitter interrupt flag (TI) until it is set (line 15). This indicates the transmission is complete, so the code clears the transmitter interrupt flag (line 16).

On line 17, the program checks for the reception of the transmitted value. The transmitted value has entered the receiver buffer when the receiver interrupt (RI) is set. The receiver interrupt is clear on line 18.

On line 22 the program reads the received value from the UART receive buffer (which is at the same address, SBUF) and compares it to the counter value. If they match, then the program displays the letter O on the LED display. If they don't match, the letter E is

displayed (line 23). Then the counter value is incremented and the loop repeats. The counter value rolls over to zero after every 256 iterations of the loop.

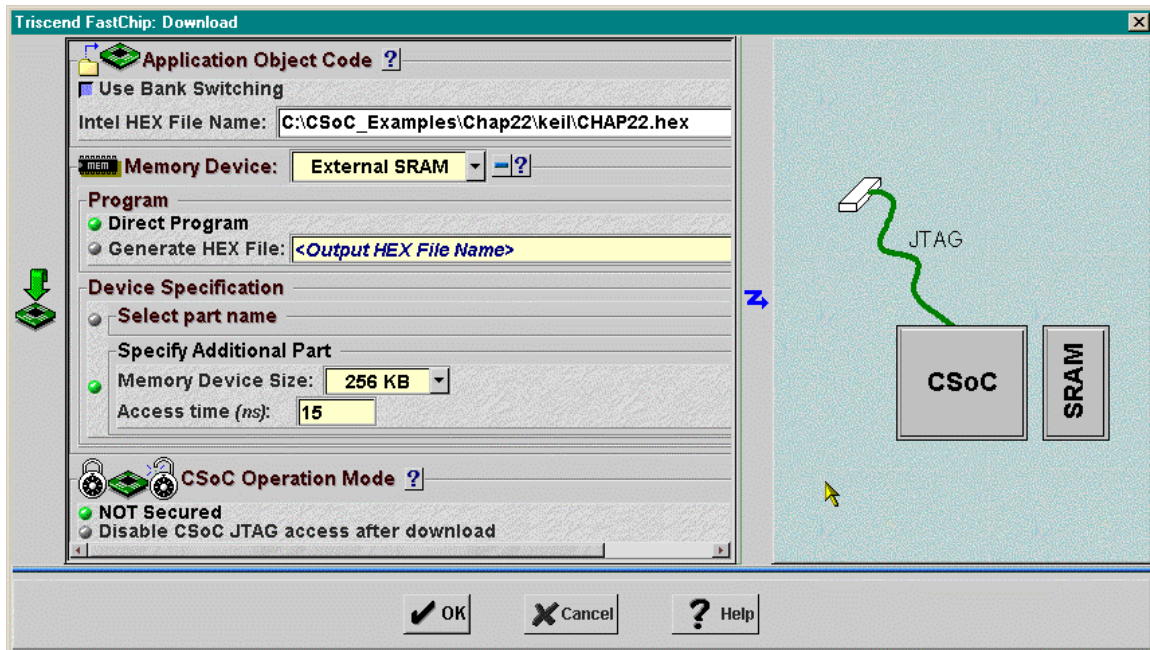
Once the C code is entered, store it in a file named chap22.c in the keil folder. Then create a new project named **chap22** and add the chap22.c file to it. Set-up the same compiler and linker options that you did in the previous design example and build the chap22.hex file.

Listing 3: C application code for the UART loopback design.


```
1 #include "..\Chap22.h"
2
3 #define LETTERE 0x79
4 #define LETTERO 0x3F
5
6 main()
7 {
8     unsigned char cnt;
9
10    Chap22_INIT(); // initialize on-chip resources
11    REN = 1;      // enable UART receiver
12    for(cnt=0; 1; cnt++)
13    {
14        SBUF = cnt; // transmit counter value
15        while(TI==0) ; // wait for transmit to complete
16        TI = 0;     // clear transmitter flag
17        while(RI==0) ; // wait for reception
18        RI = 0;    // clear receiver flag
19        // compare received value to transmitted value
20        // and display an "O" if they match and
21        // display an "E" if they don't.
22        if(cnt==SBUF) ledPort=LETTERO; // they match!
23        else ledPort=LETTERE;         // they don't match!
24    }
25 }
```

Testing the Loopback Application

At this point, return to the FastChip project window and initiate the binding operation. Then download the loopback hardware and the chap22.hex application code into the external SRAM of the CSoc Board.



Now click on the Run⇒dScope Debugger... menu item in the Keil IDE to start the debugger. Tell the debugger of the type of MCU by selecting the 8032-te5.dll entry in the drop-down list near the top of the **dScope** window. Then establish the debugging connection with the CSoc Board by clicking on OK in the **TCP/IP Configuration** window that appears. After the connection is established, you should reset the 8032 MCU in the CSoc by clicking on the Reset button in the **Toolbox** window. Then load the debugging information from the chap22 OMF file into dScope with the File⇒Load object file... menu item.

Before testing the UART loopback design, lower DIP switch #1 into its OFF position. This applies a logic 1 to the **gate** input of the AND gate and allows the bits from the UART transmitter to reach the receiver. Then click on Go! or the  button in the **Module** window and the program starts executing. You should see a O displayed on the LED digit. This indicates that the receiver is getting the bits sent by the transmitter.

Now if you raise DIP switch #1 to its ON position, this applies a logic 0 to the AND gate and breaks the connection between the transmitter and receiver. This should cause an E to be displayed on the LED digit.

Once the E is displayed, will the O reappear if you return the DIP switch to its OFF position and re-establish the loopback path? Usually not. When you break the loopback path, it will most likely truncate the string of eleven bits comprised of the start bit, eight data bits forming the counter value, the programmable ninth data bit, and the stop bit. If the UART receiver doesn't receive a complete transmission, it will not set its interrupt flag (RI). If RI is never set, then the program will loop forever on line 17 waiting for the interrupt flag to be set. Therefore, no more transmissions are initiated and the display updates cease.

Design 2.3 - UART Loopback with Interrupts

The previous design has the drawback that the program gets stuck in a polling loop once the loopback path is interrupted. The example design in this section fixes that problem by using interrupts from the UART.