Introduction

This lab has two goals: the development of a simple register transfer design (a serial transmitter) using Verilog and the creation of a sequential testbench to verify that the design operates properly. We will build on this experience next week by designing a more complex circuit (an asynchronous receiver).

Background

Asynchronous serial data transmission protocols such as RS-232 involve sending data serially at a known transmission rate called the baud rate. The RS-232 protocol is used to transmit data between computers and terminals. Data is transmitted in the following form:

When no characters are being transmitted, the active low DATA signal remains high. When a character is transmitted, it is sent one bit at a time at the baud rate but is preceded by a start bit that is always asserted low. In addition, the data bits are followed by a stop bit that is always asserted high. After the complete transmission of a character and its start and stop bits, additional characters may be sent immediately if desired following the same protocol. If an additional character is not sent immediately after the stop bit, the data signal remains high until a new character is sent an indefinite time later. The protocol is asynchronous because the transmitter sending the data and the receiver that receives the data operate using different clocks (they may be separated by a large distance, so distributing the clock is impractical).

An asynchronous serial transmitter is simple to implement using a 10-bit shift register and a little extra hardware. When a character is ready to be transmitted, it is loaded in parallel into the shift register along with start and stop bits. Data is then shifted out at the baud rate until the shift register has a value of zero (since the stop bit is always 1, the shift register will not be zero until the stop bit is shifted out of the shift register).

An asynchronous receiver is harder to implement. Because the transmitter and receiver are asynchronous with respect to each other, a new character may start arriving at any time.
Some method is needed to synchronize the receiver with the incoming data. The typical method for synchronizing the receiver with incoming data (which you will use in your Experiment 8) is to clock the receiver circuit at a multiple of the baud rate (usually 16 or 64) and sampling the incoming data to "find" the falling edge of the stop bit. Given the approximate location of the falling edge of the stop bit and the rate of transmission with respect to the clock rate, the "center" of each transmitted bit can be found using a delay which can be computed using a counter (remember that the clock rate is a multiple of the baud rate). Following this scheme, the receiver must sample the data input as shown below:

![Diagram of data sampling](START_BIT LS explosive STOP)

Search for Falling Edge

---

Sample Start Bit

Sample LSB

Sample MSB

Sample Stop Bit

It is necessary to sample the start bit to see if the falling edge detected by the receiver was a real start bit and not just line noise. If the falling edge was caused by line noise, the start bit would not still be asserted at the center of the spurious start bit and the receiver should go back to searching for another falling edge. If the start bit is in fact valid, then each of the following eight bits should be sampled at the center of the transmitted value and shifted into a shift register to reassemble the 8-bit character. Finally, the stop bit should be sampled to test whether the complete character was received correctly. If the sampled stop bit is not the high value that its should be, then the "framing" of the received character is incorrect (probably because the transmitter and receiver baud rates are set to different values). This condition is referred to as a framing error.

**Lab Preparation and Design**

Design a serial transmitter assuming a clock rate that is equal to the baud rate. The transmitter circuit must implement the following steps:

1. Accept an 8-bit parallel value as input on command.
2. Output serial data transmitted at the baud rate as a start bit followed by 8 data bits followed by a stop bit.

Your asynchronous serial transmitter should have the following inputs and outputs:
When your circuit is initialized, it should assert the RDY output to signal that it is ready to accept data to be transmitted and wait for the START input to be asserted. It should then deassert the RDY signal, input 8 bits of data using the DATA output and serially transmit the data over the TxD output by sending a start bit, the data bits, and a stop bit as outputs. When the transmission is complete, it should assert the RDY signal again to signal that it can accept data. The CLOCK input will provide a clock signal that is equal to the baud rate at which data should be transmitted. Your design should be able to transmit successive bytes with only a single STOP and START bit separating the two values.

1. Create a block diagram of your transmitter design in terms of primitive building blocks (shift registers, logic blocks, etc.) and bring it with you to lab.

2. Write Verilog code for your transmitter design. Follow the coding guidelines distributed in class – include a title block, use one file for each module, and use the indentation style described in the guidelines. Bring your code with you to Lab.

In the Lab

1. Compile your Verilog code and use the simulator to test its basic operation.

2. Use the techniques discussed in class, write a sequential testbench that generates the proper sequence of signals to exercise your transmitter design and checks that its outputs are transmitted correctly. It should test the transmission of several different byte values, including 8'b01010101, 8'b00110011, and 8'b00001111. In addition, it should check that transmission occurs correctly both when successive bytes transmit immediately and when they are separated by several clock cycles. Your testbench should use the $display system task to print out any errors which are found. In addition, it should print a message at the end indicating that it completed and then pause simulation using the $stop system task. Use this testbench to check your circuit fully.

3. Synthesize your circuit using the Synopsys tools and then use db2mag to generate a layout. Plot the resulting layout and note its size.
Report

For your lab report hand in the following:

1. A short technical memorandum which describes (a) what was done, (b) what you learned, and (c) what difficulties you encountered.
2. The block diagram of your serial transmitter design.
3. Verilog listings for your serial transmitter design.
4. A Verilloger timing diagram printout showing that it works for a single byte.
5. A Verilog listing of your testbench file.
6. A printout of the “verilog.log” output log showing the messages printed by your testbench.