Chapter 8

UART Receiver

The purpose of this laboratory is to design a Universal Asynchronous Receiver (a UART without the transmitter). This receiver will be used to receive ASCII characters from the PC workstation and display them in HEX on your board. You will use this receiver later in the course.

Exercises

Exercise 1 – Overview and State Machine Design

The receiver circuit you will design is very similar to the transmitter. If you successfully completed the transmitter, you should have a good background on the operation of the receiver. While the circuits for the receiver and transmitter are similar, there are a few important differences between the receiver circuit and the transmitter circuit. The first important difference between the two circuits is the way in which the receiver circuit leaves the "idle" state and begins acquisition of an incoming byte. Unlike the transmitter, the receiver continually monitors the incoming single-bit "`RX`" input and will leave its IDLE state and begin sampling data when the `RX` signal is low indicating a "START" bit.

Another important difference between the receiver and the transmitter is the timing of the state machine and the sequencing of the transmitted bits. A timer was used in the transmitter to measure a single bit period and this timer was used to transition between states. In the receiver, a timer is needed to measure the time for *half* of a bit period as well as a full bit period. As the the bits are sampled on the RX line, this sampling will need to occur as close to the middle of the bit period as possible. This will allow the receiver to tolerate slight variations between the baud rate of the transmitter and that of the receiver.

Figure 8.1 demonstrates how this sampling should occur. When the start bit is first sampled low, the receiver recognizes that a transfer is starting. The receiver will use this event to synchronize its internal timers. The receiver expects a start bit (`RX` = low) for one full bit period. Next, the receiver expects the first bit of the transmitted byte during the next bit period. To safely sample the first bit, the receiver should wait 1.5 bit periods as shown in Figure 8.1. At this point in time (1.5 bit periods from the beginning of the start bit), bit D0 of the byte will be sampled into a shift register from the `RX` input.

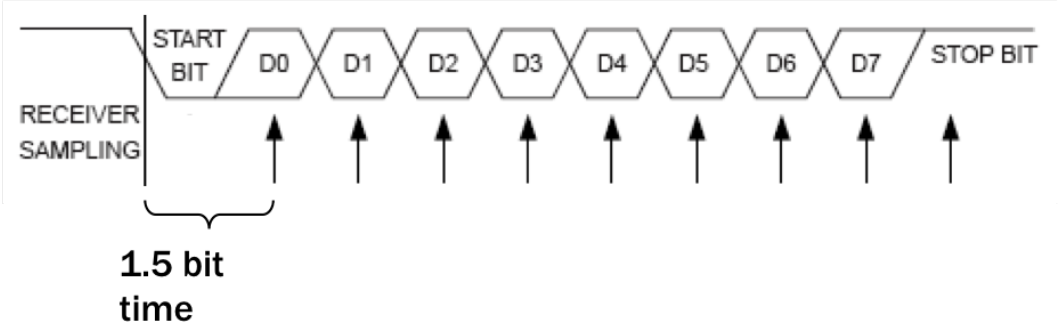

Figure 8.1: UART Receiver Timing.

After the first bit, D0, has been sampled, the next bit (D1) can be sampled by waiting another full bit period. When sampling, the `RX` signal can be shifted into a shift register to preserve the previous value of D0. This process of sampling a bit every baud period continues until all 9 bits from the message have been sampled (8 data bits and the stop bit).

Answer the following questions to test your understanding of the receiver operation.

Question: How do you detect the start of a new byte?

Question: Why is it important to sample the incoming signal in the middle of the bit period?

Question: What can happen if the baud rate of the transmitter does not match the baud rate of the receiver?

Question: If the transmitter transmits a message at a baud rate that is much faster than the baud rate expected by the receiver it is possible that the receiver will sample the wrong message. Determine the maximum transmitter baud rate that can be tolerated by the receiver when the receiver is expecting a 19,200 baud rate?

Question: Determine the minimum transmitter baud rate that can be tolerated by the receiver when the receiver is expecting a message with a 19,200 baud rate?

Unlike the transmitter lab, you will not be given a state machine or a datapath example to complete this lab. You will need to create your own state machine and data path to guide you in your VHDL design. Note that the book contains the description of a UART Receiver in Section 12.5. Create an ASMD diagram for your UART Receivier. Consider the following issues when designing your diagram:

- The delay from the falling edge of rx_in (the front edge of the start bit) to when you sample rx_in for the first data bit should be as close to 1.5 bits periods as possible. After this 1.5 bit period wait, you will need to sample rx_in every 1 bit period until the data byte has been fully received. One way to do this is to have a timer (similar to the Bit Timer in the transmitter) that has both an rx_bit output and a rx_half_bit output — one indicates when a full bit time has elapsed, the other tells when a half bit time has elapsed.
- Create a state machine output signal called rx_busy that is asserted when the receiver is not idle and waiting for a new byte. This signal is used by higher-level circuitry to query the status of the receiver.
- Create an output signal named “data_strobe” that is asserted for one clock cycle when a valid new byte has been received. A valid byte has been received when you sample a valid stop bit (i.e., you sample rx_in = ’1’ during the stop-bit time period). Do not data_strobe if you sample rx_in = ’0’ during the stop-bit time period (i.e., the received byte will be ignored). Wait until the rx_in returns to ‘1’ before returning to an idle state
- Some UART transmitters may power-up with its TX line set to ‘0’. After its initial power-up sequence is over, its TX line will be set to ‘1’ indicating no transfer. If your UART receiver is connected to such a UART transmitter, your receiver may think that the initial ‘0’ is a start bit and assume a character is being transmitted. You will need to design your state machine to ignore the case when the transmitter powers up with a default ‘0’ on the TX line. Consider these suggestions for ignoring an initial ‘0’:
  - Create an initial “power-up” state that will remain in the power-up state as long as RX = ‘0’. Once RX = ‘1’, proceed out of the power-up state into a regular “Idle” state.
  - To make sure your “power-up” state is the initial state of your state machine when the FPGA is configured, make sure the power-up state is the first state listed in your state encoding:

```
type state_type is (power_up, idle, ...
```

Begin your lab by creating an ASMD diagram of your receiver circuit. This ASMD diagram should include all of the states and all of the RTL statements necessary to implement a complete UART receiver. Provide sufficient detail in your ASMD diagram that another person could implement the system without assistance.

Upload: Upload a detailed ASMD of your receiver state machine.

After completing the ASMD diagram, create a conceptual diagram of all of the registers used in your UART receiver (see Figure 11.12 (b) for an example). This diagram should include all registers (except the state register) and a conceptual diagram of the next state logic for each of these registers.

Upload: Upload your conceptual diagram of your receiver state machine.

Exercise 2 – Receiver Design (rx.vhd)

The first design part of this lab is to create a VHDL file that contains all of the logic for your receiver. Create a VHDL entity named '`rx`' and include the following ports:

Port Name	Direction	Width	Purpose
clk	Input	1	50 MHz clock
rst	Input	1	Asynchronous reset
rx_in	Input	1	The input serial signal (connected to the rx input from the serial port)
data_out	Output	8	The byte that was received. This is only valid when data_strobe is asserted
data_strobe	Output	1	Asserted high for one cycle when a byte has been received and data_out is valid
rx_busy	Output	1	Asserted when the receiver is not in the idle state. When zero, the circuit is waiting for a new byte

Generic	Type	Purpose
CLK_RATE	NATURAL	Indicates the frequency of the input clock (Default=50_000_000)
BAUD_RATE	NATURAL	Baud rate of the transmitter (Default=19_200)

Like the transmitter lab, use generics to determine the period of each bit time. Supporting these generics will allow you to use your receiver with different input clocks and to receive characters with a different baud rate.

After implementing your state machine and datapath, perform some simple tests to verify that your circuit is operating as you expect. Remove any obvious bugs before proceeding to the next exercise. Once your transmitter compiles without errors and performs a few basic tests, proceed to the next exercise.

Exercise 3 – Testbench

Download the testbench for this design exercise. This testbench will test your receiver under a number of conditions and see if your receiver conforms to the specifications described in this lab. This testbench will simulate the sending of bytes to your receiver and perform a number of checks to make sure you received the data correctly.

Download: rx_testbench.vhd

Simulate your receiver until the "DONE" message is printed on the console. Continue to edit and debug your design until it passes all of the tests without any errors.

Upload: Submit a copy of your working rx.vhd receiver.

Exercise 4 – Top-Level Receiver Design

Once your receiver successfully passes all of the testbench test cases, create a new top-level design that includes your receiver. This top-level design should also contain your seven-segment display controller so you can display the hex value of the characters you receive onto the seven-segment display. There are several considerations you need to make when creating this top-level design. Each of these will be described below.

Synchronizers

The `rx_in` input signal arrives asynchronously to the Nexys2 clock. Your synchronous UART receiver circuit may experience problems if it samples this asynchronous signal (we will discuss the details of this problem in Chapter 16 of the textbook). To synchronize the asynchronous `rx_in` input to your system, add two flip-flops after the input `rx_in` signal as shown in Figure 8.2. Use the output of the second synchronizer flip-flop as the input signal for the `rx_in` signal of your UART Receiver.

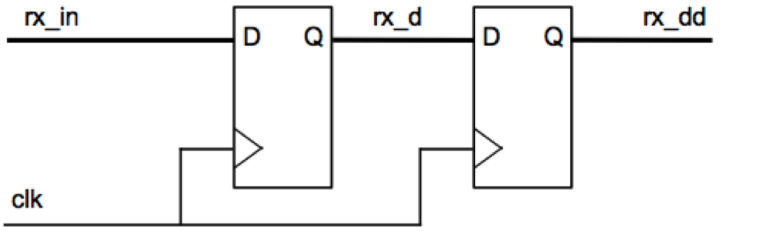

Figure 8.2: Synchronizers for rx_in signal.

Design your top-level logic so that incoming bytes received on the UART receiver are displayed in hex on the right two digits on the seven segment display. When a new byte is received, shift the byte currently displayed on the right most digits to the left two digits and display the new byte on the right most digits. You will need to use the data_strobe signal to control when the data is loaded into the display and shifted.

Seven-Segment Display Input

To drive the display, create two eight bit registers as shown in Figure 8.3. One of the 8 bit registers is used for the right two digit display and the other 8 bit register is used for the left two digit display. Add the appropriate logic to load a new byte into the right digit display using the data_strobe signal from your receiver module

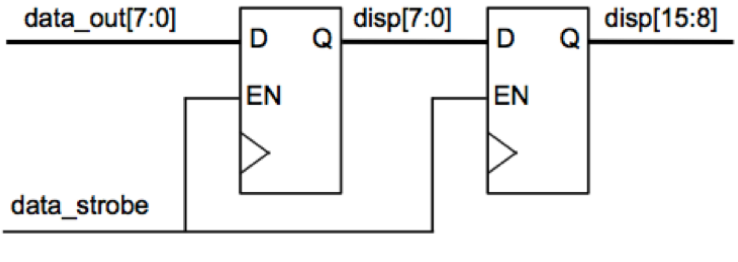

Figure 8.3: Seven Segment Display Registers for UART Receiver.

Create a reset signal on one of the buttons for an asynchronous reset to your design. Simulate the top-level design carefully to make sure your display logic is working properly. After verifying that your top-level design works, create a UCF file for your top-level design and synthesize your design.

Upload: Submit your top-level VHDL file.

Question: Review the synthesis log to determine the state encoding of your transmitter state machine. Cut and past the encoding of your synthesized state machine for this question.

Question: Summarize and justify all of the synthesis warnings you received when synthesizing this circuit.

Exercise 5 – Download

Once you have created a bitfile, test your receiver by opening a putty terminal on your desktop terminal and setup the terminal with the proper UART settings. Press a few keys on the putty terminal and verify that when a key is pressed the corresponding ASCII value is displayed on your seven segment display.

Personal Exploration

For your personal exploration, modify the top-level design to add some additional features. Ideas include:

- Modify the top-level design so that it includes both a receiver and transmitter. Verify that you can both transmit and receive

- Modify your top-level design to include your VGA text display circuit. Display characters received from the UART on your display
- Try a different baud rate and see if your receiver works at a different speed

Pass Off

Demonstrate to the TA a working circuit on your board. The TA will experiment with your circuit to see if it operates correctly under a number of circumstances (i.e. the TA will test the transmission of several different characters).