EECE 359 – Computer Communications and Networking **Spring 2020**

Lab #1 –Asynchronous Communication

Objectives

This lab will introduce concepts for a basic asynchronous digital communications device that is typically used in conjunction with the RS-232 standard to provide a serial data link between two pieces of electronic equipment. See the Wikipedia article at http://en.wikipedia.org/wiki/RS-232 for background information on the RS-232 standard. This standard defines the physical and electrical characteristic of the serial interface.

The device that is used to generate the protocol for an asynchronous serial interface is called a **Universal Asynchronous Receiver/Transmitter** (UART). In this lab you will analyze a VHDL implementation of a UART. See the Wikipedia article at http://en.wikipedia.org/wiki/UART for background information on the UART. We will not actually apply the RS-232 standard, but will just use the signals as provided at the FPGA board I/O connectors.

You will create an ISE project called "Lab1" and add the provided source code for the UART. You will use the simulator to analyze the operation of the UART. For this lab, the hardware will not be implemented on the FPGA board. However, it will be in Lab2.

Design Hierarchy

- Test_UART.vhd – behavior (Test_UART.vhd)
  - uut – UART – Behavioral (UART.vhd)
    - Inst_UART_Tx – UART_Tx – Behavioral (UART_Tx.vhd)
    - Inst_UART_Rx – UART_Rx – Behavioral (UART_Rx.vhd)

The VHDL source code for the UART is provided.

You will develop the test bench file as part of this lab.

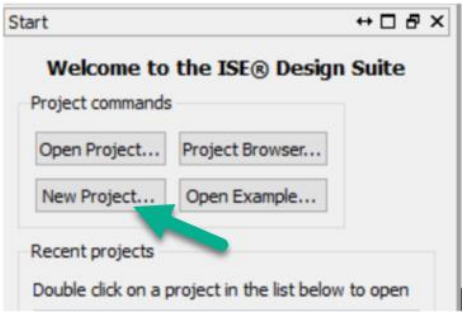
Procedure

- **Log-in to the Windows client**
  - Select the **kings.binghamton.edu** server and click "Connect"
  - Log-in using your PODs ID and password
  - Connect to the **Monster Lab** desktop
  - After the desktop loads, right click on the Windows **"Start"** icon and select **"File Explorer"** (or click on the folder icon in the task bar)
    - Navigate to your **U-Drive**. If you don't have a U-Drive, search the binghamton.edu website for "BUShare U-Drive"
  - On your U-Drive, create a new folder named **EECE359Labs**
  - **IMPORTANT: It is your responsibility to safeguard your lab files. You will have to recreate any files or data that you lose no matter what the cause.**
  - Also in the **Documents** folder, create a new folder named **"working"**

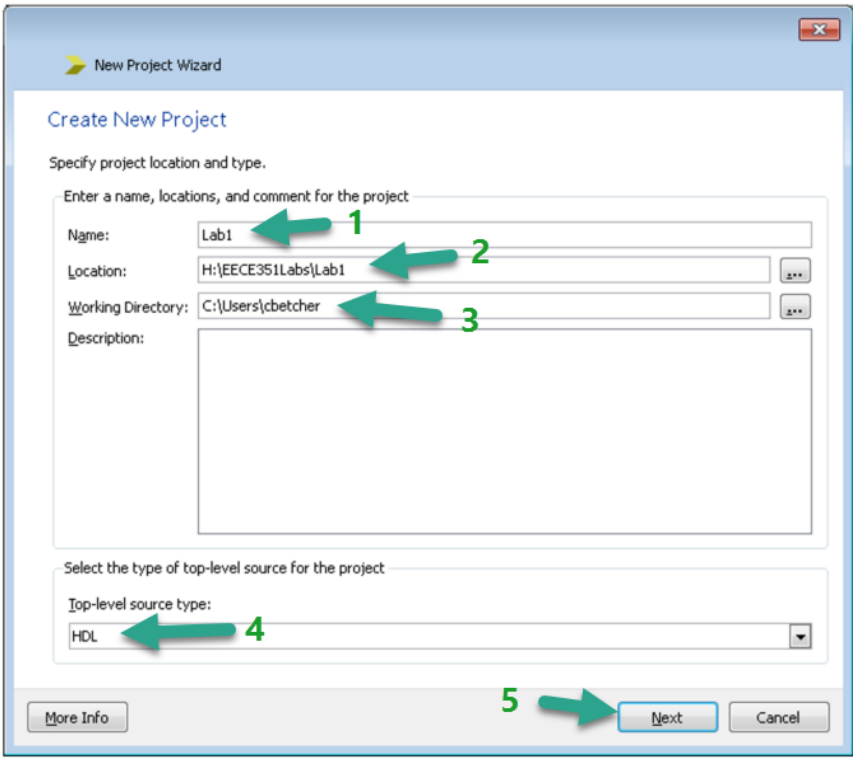
Create a Project

- Open the **ISE Design Suite 14.7** software by double clicking the desktop icon 

- Add a new project: Click on the **"New Project…"** project command button, or you can click on **File**, then select **"New Project…"**



- In the **New Project Wizard (see figure below)**,
  1. set the **Name** field to **"Lab1",**
  2. set the **Location** field to the folder you just created on your U-Drive,
  3. set the **Working Directory** field to your user directory on the C-drive of the Windows client (this is important to prevent the ISE software from cluttering your U-Drive with temporary working files and cause longer process times when running the Xilinx ISE software tools),
  4. select **HDL** for the **Top-level source type**, and
  5. click ***Next***.



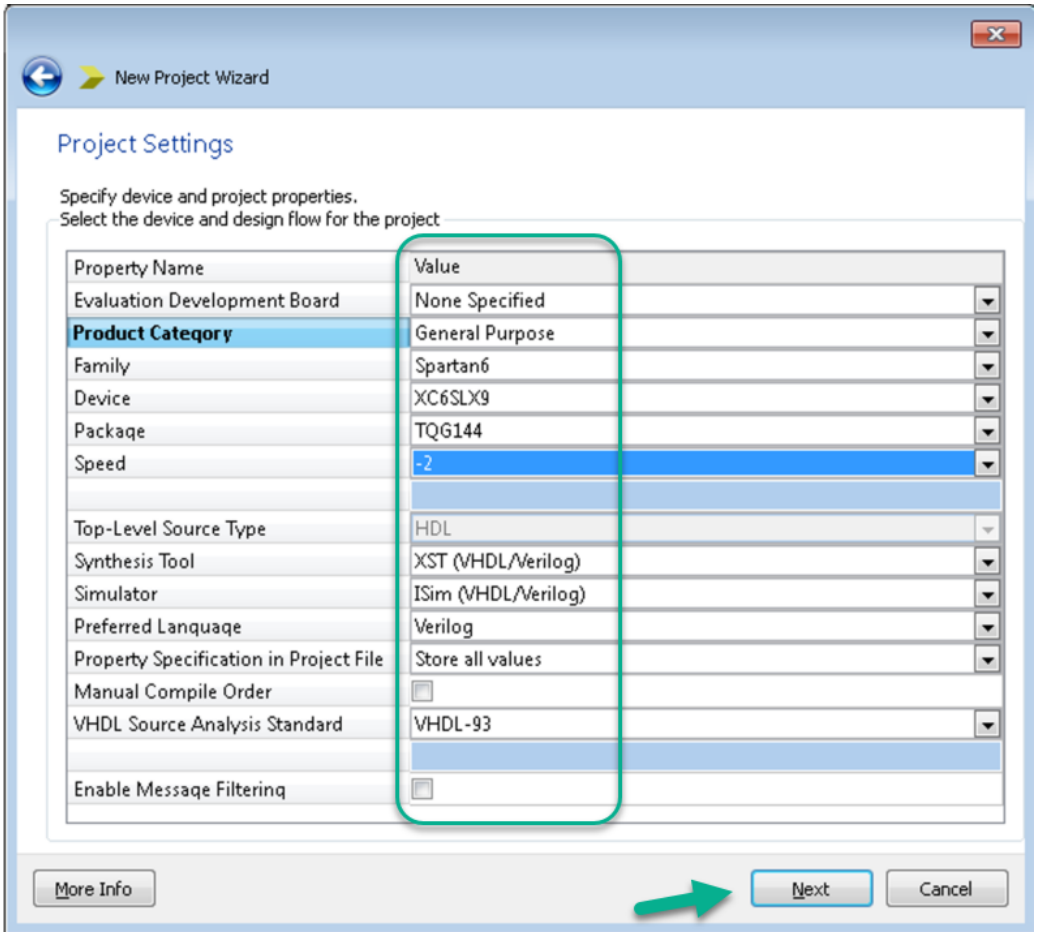
- Set the **Project Settings (see figure below)**.

  In this dialog box, you will select the settings for the FPGA on your FPGA Board.

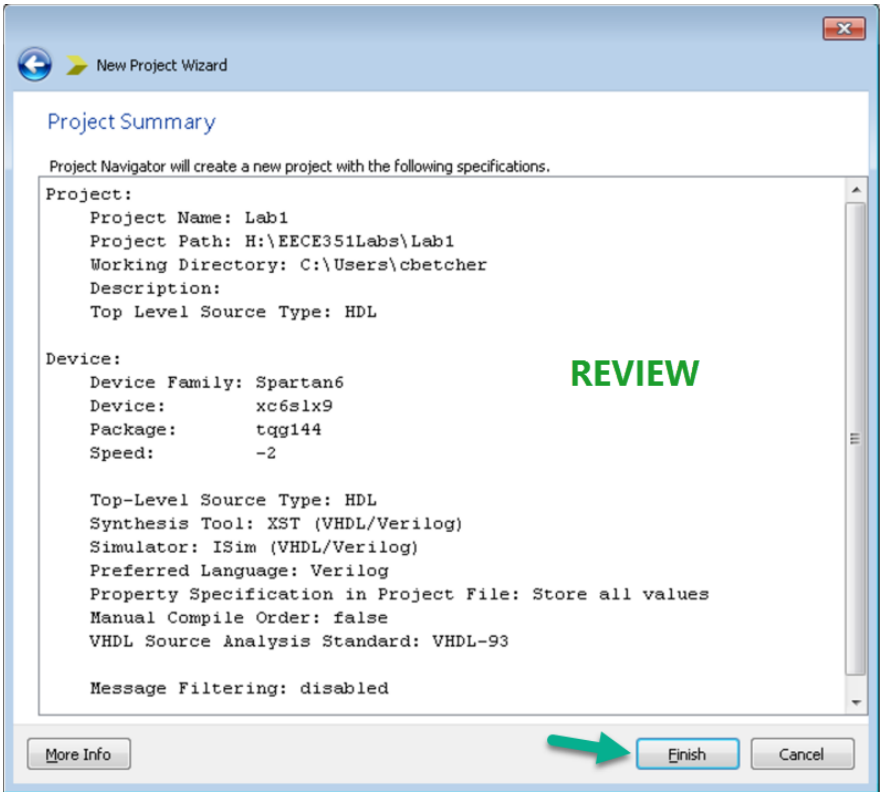
  Set the following properties:

  1. Evaluation Development Board: ***None Specified***
  2. Product Category: ***General Purpose***
  3. Family: ***Spartan 6***
  4. Device: ***XC6SLX9***
  5. Package: ***TQG144***
  6. Speed: ***-2***
  7. Synthesis Tool: ***XST(VHDL/Verilog)***
  8. Simulator: ***Isim (VHDL/Verilog)***
  9. Preferred Language: ***VHDL***
  10. Property Specification in Project File: ***Store all values***
  11. Manual Compile Order: ***box unchecked***
  12. VHDL Source Analysis Standard: ***VHDL-93***
  13. Enable Message Filtering: ***box unchecked***

  Click ***Next***>



- Review the summary and click ***Finish***



- **Add the following source files to your project (provided on myCourses):**
  - UART.vhd
  - UART_Tx.vhd
  - UART_Rx.vhd

- **Develop the test bench and simulate the design (Parts A-C)**

Part A:

In this first part of the lab, you will create a test bench with stimuli which will cause the UART to transmit three bytes of data. From the simulation, you will analyze the operation of the UART for the transmission of data.

Create the Simulation Test Bench

- Create a simulation test bench file named “Test_UART.vhd” (Hint: Use ISE Project Navigator menu option **Project >> New Source >> VHDL Test Bench**
- Set the clock period to be the period of the system clock on your FPGA board (e.g. 20ns for a 50 MHz clock; 31.25ns for a 32 MHz clock).
- Write a VHDL *procedure* named “transmit_data” to provide 8 bits of data to *DBIN* and generate a *WR* pulse. The code for the procedure will be part of the “stim_proc” process in the test bench. Place it before the “*begin*” statement of that process. This procedure defines a set of statements that can be “called” multiple times in the stimulus process without having to rewrite all of the statements each time.

- Use the following VHDL syntax:

  procedure procedure_name (parameter_list) **is**

  begin

  <sequencial statements>

  end procedure;

- Use a parameter named “data” to pass an 8-bit vector of data to the procedure. The parameter must include the “type” and “mode” of the parameter in the form of

  name : mode type

  where mode would be in, out, or inout, and type would typically be std_logic or std_logic_vector.

- Write VHDL statements in the procedure to do the following:
  - Use a *while loop* to wait for the TBE signal to change from ‘0’ to ‘1’
  - Add a statement in the *while loop* to wait for 1 CLK_period of time (necessary to cause time to advance while in the *while loop*)
  - Assign the 8 bits of “data” to the input signal *DBIN*
  - Wait for 1 CLK_period of time
  - Set *WR* to ‘1’
  - Wait for 40 µs
  - Set *WR* to ‘0’
  - Add this statement at the end of this procedure:

    report "TEST BENCH WROTE DATA TO DBIN";

  When this procedure is called, the 8 bits of “data” will be written to a register in the UART which will then shift the data out serially on the TXD output of the UART.

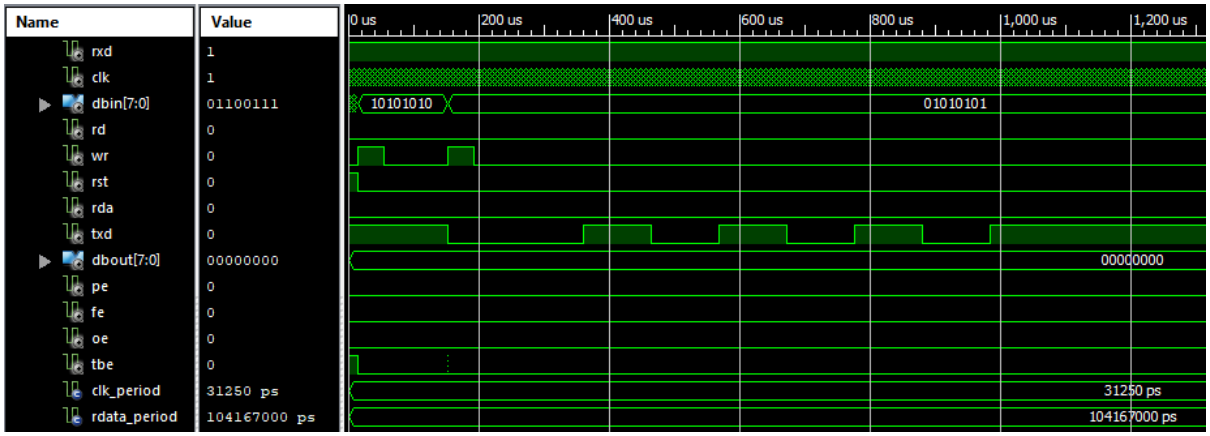
- Modify/write the code in the statement section of the stimulus process which is after the “begin”, to do the following:
  - Set the *RXD* input to ‘1’ which is the normal level of this signal when no data is being transmitted to the receiver.
  - Generate a 15 µs long reset signal (*RST*)
  - Then, wait for 10 clock periods
  - Now generate the code to apply a data pattern of “10101010” to *DBIN* and pulse the *WR* signal, using the procedure you just wrote. The syntax is as follows:

    procedure_name (parameter_value);

  - Use the “transmit_data” procedure to send two more data patterns: “01010101” and “01100111”.
- Check for syntax errors

Run the Behavioral Simulation

- Run a behavioral simulation for 5 milliseconds (5000 microseconds) using iSim.
- Zoom to full view.
- Put the cursor at time 0 and then zoom in until you can see the 15 µs “rst” signal.
- Almost immediately after the reset you should see a pulse on the “wr” signal.
- Zooming back out, you should be able to see a serial data pattern on signal “txd” as seen in the waveform view shown below.



- Group the signals appearing in the waveform viewer and name it “TEST BENCH”.
- Add the signals of the “uut” instance to the waveform viewer and group them with the name “UUT”.
- Add the signals of the “UART_Tx” instance to the waveform viewer and group them with the name “TRANSMITTER”
- Save the waveform configuration using **File → Save As** and save to a file in your project folder.
- Rerun the simulation for 5 milliseconds to capture the additional signal waveforms
- The Console window should display the following messages:

  ISim>

  # run 5.00ms

  Simulator is doing circuit initialization process.

  Finished circuit initialization process.

  at 55343750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).

  at 192843750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).

  at 1752843750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).

  ISim>

- Save your waveforms, showing all of the signals in the TEST BENCH and TRANSMITTER groups, to a PDF by printing with the “Microsoft Print to PDF” printer option. Include this PDF with your lab submission on myCourses.

Analysis of the Transmitter Section of the UART

- Referring to the source code and the simulation results, analyze the operation of the transmitter portion of the UART. You will need to zoom in and out to determine the clock relationships. Answer the following questions and submit your answers with the Lab1 assignment in myCourses as a PDF or Word document:
    1. What is the period of "tClk"? (measured using both cursors in the simulator waveform viewer)
    2. Describe how "rClk" and "tClk" are generated from the system clock "clk" (hint: change the radix of the signals "clkdiv", "rclkdiv", and constant "bauddivide" to *unsigned decimal*):
    3. What is the function of the signal tdSReg [10:0]?
    4. What bits in tdSReg [10:0] are loaded with the data from DBIN [7:0]?
    5. What are the other bits in tdSReg [10:0] used for?
    6. What is the maximum frame rate that data can be transmitted (frames per second)? Hint: In the simulator, measure the time from the beginning of the start bit of one frame to the beginning of the start bit of the next frame. Then calculate the frame rate.
    7. Using the frame rate determined in the previous question, what is the bit rate of the data that is achieved? Note that the start, stop, and parity bits are not counted as data bits.
- Draw the state diagram for the transmitter FSM (txState). Show all states, outputs, and transitions. Label all state names and transition conditions. Include this diagram in the file with your answers, or as a separate image or PDF file, or as a hardcopy submitted to the TA.
- Describe the operation of the transmitter state machine, where the current state is represented by "txState"

Part B:

In this part of the lab, you will modify your test bench to provide a signal to the RXD input of the UART so it receives a frame containing a byte of data. You will then use the simulation to analyze the operation of the UART for receiving data.

Modify the Simulation Test Bench

- Declare a **constant** named "rdata_period" of type **time** and set it to a value of 104167 ns. This constant will be used to establish a data rate of 9600 baud (that is, 9600 symbols per second where a symbol in this case has a binary value of 0 or 1).
- Declare a **shared variable** named "DELAY" of type integer and set it to a default value of 5. This variable will be used as a parameter by the **RD_process** described below to set a time spacing between consecutive transmissions to the receiver. The spacing helps when viewing the waveforms, making it easier to separate one character transmission from the next. We could have used a local variable in the RD_process instead. But, using a shared variable keeps all of the parameters that you might want to change together in the declaration section of the test bench entity.

- Declare a **shared variable** named “rx_frame” of type std_logic_vector(10 downto 0). This variable will be used to save the value of the data sent to the UART receiver so that it can be compared with the actual data received.
- Write a second VHDL procedure named “*receive_data*” that will accept an 11-bit input data parameter named “*frame*” and output that frame data, one bit at a time, onto the *RXD* input of the UART, beginning with bit 0 of the *frame* parameter. Wait for one *rdata_period* between the start of each bit to set the bit rate of 9600 baud. After the last bit is generated, set *RXD* to a ‘1’. Then, use an assert statement to verify that the correct data was received. This functionality can be accomplished with code that does the following:
  - A ***for loop*** that assigns each consecutive bit of the 11-bit data parameter to the *RXD* signal and wait for a period of *rdata_period* between each bit.
  - Insert the following code after the RXD assignment statement:

```
if I = frame'high then
        report "TEST BENCH SENT FRAME TO UART RX";
end if;
```

- After the ***for loop***, set *RXD* to ‘1’ so that the signal returns to a ‘1’ level when the transmission is finished.After the stimulus code of Part A, add code to do the following:
  - Assign rx_frame with the data pattern “11101010100”
  - Use the receive_data procedure to send the data in rx_frame
- Add a new process placed after the *CLK_process* in the test bench with the label “*RD_process*”. We want this process to generate a *RD* pulse whenever the *RDA* signal goes to a ‘1’. The *RDA* signal indicates a frame was received on *RXD*, the data is available on the *DBIN* bus, and the UART error flags are valid. The *RD* signal acknowledges that the data of the frame was received and will cause the flags to be cleared. For our testing purposes, we want to have a variable time delay between the time the *RDA* signal becomes a ‘1’ and the generation of the *RD* pulse. Therefore, the process should include code that does the following:

  - Wait until *RDA* = ‘1’
  - Check that the data received is correct. Insert the following code:

```
report "RECEIVED DATA IS AVAILABLE ON DBOUT";
if DBOUT = rx_frame(8 downto 1) then
      report "DBOUT MATCHES DATA SENT TO RXD";
else
      report "DBOUT DOES NOT MATCH DATA SENT TO RXD" severity error;
end if;
```

  - Wait for rdata_period**DELAY* Report any flags that are set. Insert the following code:

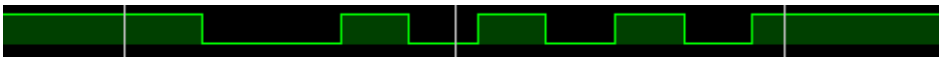
```
if pe = '1' then report "PARITY ERROR FLAG IS SET"; end if;
if fe = '1' then report "FRAME ERROR FLAG IS SET"; end if;
if oe = '1' then report "OVERWRITE ERROR FLAG IS SET"; end if;
```

  - Generate a pulse on signal *RD* for 40 µs

Similar to the CLK_process, the code in this process will loop indefinitely. However, the operation of the RD_process is influenced by the *RDA* signal which comes from the UART and the *DELAY* variable which can be specified in the stimulus process (stim_proc).

Run the Behavioral Simulation

- Run a new behavioral simulation for 5 milliseconds using iSim with your saved waveform configuration file.
- Verify in the waveform view that the signal "*RXD*" is the desired serial data stream for the frame data pattern "11101010100" as shown below (the first bit of the waveform is the most right bit of the pattern, or its LSB), that the signal "*RDA*" went active when the data was received, and that the "10101010" data pattern is presented on *"DBOUT[7:0]"*.



- Add the signals of the "UART_Rx" instance to the waveform viewer and group them with the name "RECEIVER".
- Save the waveform configuration
- Rerun the simulation for 5 milliseconds to capture the additional signal waveforms
- The following should be displayed in the Console window:

```
ISim>
# run 5.00ms
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 55343750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).
at 192843750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).
at 1752843750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).
at 2794513750 ps: Note: TEST BENCH SENT FRAME TO UART RX (/test_uart/).
at 2863234375 ps(4): Note: RECEIVED DATA IS AVAILABLE ON DBOUT (/test_uart/).
at 2863234375 ps(4): Note: DBOUT MATCHES DATA SENT TO RXD (/test_uart/).
ISim>
```

- Save your waveforms, showing all of the signals in the TEST BENCH and RECEIVER groups, to a PDF by printing with the "Microsoft Print to PDF" printer option. Include this PDF with your lab submission on myCourses.

Analysis of the Receiver Section of the UART

- Analyze the operation of the receiver portion of the UART. Answer the following questions and submit your answers with the Lab1 assignment in myCourses as a PDF or Word document:
  1. What is the function of rdSReg[9:0]?
  2. What is the function of rdReg[7:0]?
  3. What is the function of dataCtr[3:0]?
  4. Can the UART transmit and receive data at the same time (i.e. operate in a full-duplex mode)?
- Draw the state diagram for the receiver FSM (rxState). Show all states, outputs, and transitions. Label all state names and transition conditions. Note that this one is tricky

in that it has some Mealy outputs. Recall that Mealy outputs are a function of inputs as well as the current state and are shown on the transition to the next state (i.e. <transition condition> / <output assignment(s)>).

Include this diagram in the file with your answers, or as a separate image or PDF file, or as a hardcopy submitted to the TA.

- Describe the operation of the receiver state machine, where the current state is represented by "rxState". Include a discussion of the functions that the inputs and outputs of the FSM perform and how they relate to the operation of the "ctr" and "dataCtr" counters and the rdSReg, rdReg, and error-flag registers.

Part C

In this part of the lab, you will add more receive data transfers to the test bench. You will explore the operation of the parity error (PE), frame error (FE), and overwrite error (OE) flag outputs of the UART. Then you will explore how variations in the input **data rate** might affect the performance of the UART.

Modify the Simulation Test Bench

- In your test bench, add three more *receive_data* statements with a delay of rdata_period*10 before each of them and assigning *rx_frame* before each using the following data:

  10101110100

  01101010100

  10111110100

- Following the last *receive_data* statement, add a statement to change the value of the DELAY variable to 25. Recall that when assigning a value to a variable, use the symbols ":=".
- Now, add two more *receive_data* statements with a delay of rdata_period*10 before each of them and using the following data assigned to *rx_frame* for each:

  10101111110

  10100011110

Run the Behavioral Simulation

- Run a new behavioral simulation for 15 milliseconds.
- The following messages should appear in the iSim Console window:

  ISim>

  # run 15.00ms

  Simulator is doing circuit initialization process.

  Finished circuit initialization process.

  at 55343750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).

  at 192843750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).

  at 1752843750 ps: Note: TEST BENCH WROTE DATA TO DBIN (/test_uart/).

  at 2794513750 ps: Note: TEST BENCH SENT FRAME TO UART RX (/test_uart/).

  at 2863234375 ps(4): Note: RECEIVED DATA IS AVAILABLE ON DBOUT (/test_uart/).

at 2863234375 ps(4): Note: DBOUT MATCHES DATA SENT TO RXD (/test_uart/).
at 4982020750 ps: Note: TEST BENCH SENT FRAME TO UART RX (/test_uart/).
at 5047234375 ps(4): Note: RECEIVED DATA IS AVAILABLE ON DBOUT (/test_uart/).
at 5047234375 ps(4): Note: DBOUT MATCHES DATA SENT TO RXD (/test_uart/).
at 7169527750 ps: Note: TEST BENCH SENT FRAME TO UART RX (/test_uart/).
at 7237734375 ps(4): Note: RECEIVED DATA IS AVAILABLE ON DBOUT (/test_uart/).
at 7237734375 ps(4): Note: DBOUT MATCHES DATA SENT TO RXD (/test_uart/).
at 7758569375 ps: Note: FRAME ERROR FLAG IS SET (/test_uart/).
at 9357034750 ps: Note: TEST BENCH SENT FRAME TO UART RX (/test_uart/).
at 9421734375 ps(4): Note: RECEIVED DATA IS AVAILABLE ON DBOUT (/test_uart/).
at 9421734375 ps(4): Note: DBOUT MATCHES DATA SENT TO RXD (/test_uart/).
at 9942569375 ps: Note: PARITY ERROR FLAG IS SET (/test_uart/).
at 11648708750 ps: Note: TEST BENCH SENT FRAME TO UART RX (/test_uart/).
at 11716234375 ps(4): Note: RECEIVED DATA IS AVAILABLE ON DBOUT (/test_uart/).
at 11716234375 ps(4): Note: DBOUT MATCHES DATA SENT TO RXD (/test_uart/).
at 13836215750 ps: Note: TEST BENCH SENT FRAME TO UART RX (/test_uart/).
at 14320409375 ps: Note: OVERWRITE ERROR FLAG IS SET (/test_uart/).
ISim>

- Save your waveforms, showing all of the signals in the TEST BENCH group, to a PDF by printing with the “Microsoft Print to PDF” printer option. Include this PDF with your lab submission on myCourses.
- Show your simulation waveforms to a lab proctor and have them check off your group on their sheet.

Analysis of the Flag Outputs of the UART

- Answer the following questions and submit your answers with the Lab1 assignment in myCourses as a PDF or Word document:
    1. Which receive data pattern caused a parity error to occur?
    2. What kind of parity is the UART checking for (odd or even)?
    3. What kind of parity is the UART generating with the transmit data?
    4. Study the UART source code and explain how the receiver checks for parity errors and how the transmitter calculates the parity bit to send.
    5. Which receive data pattern caused a frame error to occur?
    6. Why?
    7. If the bit period of the RXD data has a 4% error, will the UART still capture the data correctly?
    8. Why?
    9. What caused the last received frame to set the overwrite error (OE) flag?
    10. In the test bench, change the value of the rdata_period constant by 4% (choose +4% or -4%). Rerun the simulation.

Suggestion:

Implement this by modifying the constant declaration

```
constant rdata_period : time := 104167 ns;
```

to

```
constant T_ERROR : real := 1.04;
constant rdata_period : time := T_ERROR*104167 ns;
```

Looking at the sample point of each data bit, what do you see happening in the simulation? (Hint: look at the timing of the signal rShift with respect to the input data, RXD).

11. Now, try changing the value of rdata_period by 8% from the original value and explain what you see happening. Again look at the sample point of each data bit.

Lab 1 Assignment Submission

Submit your Lab 1 assignment with the following:

Part A

- PDF of simulation waveforms (5 points)
- Answers to questions (use provided template) (14 points)
- State diagram and description for the transmitter FSM (12 points)

Part B

- PDF of simulation waveforms (5 points)
- Answers to questions (use provided template) (8 points)
- State diagram and description for the receiver FSM (12 points)

Part C

- Proctor Check-off (separate proctor check-off sheet) (10 points)
- PDF of simulation waveforms (5 points)
- Answers to 11 questions (use provided template) (22 points)
- Test Bench VHDL code file (7 points)