EN2111 – Project on UART Implementation

Phase 1 – Find a Verilog RTL for UART transceiver.

Step 1 - Research UART Protocol

Understand the basics of the Universal Asynchronous Receiver/Transmitter (UART) communication protocol, including how it handles data framing, synchronization, and speed settings (baud rate).

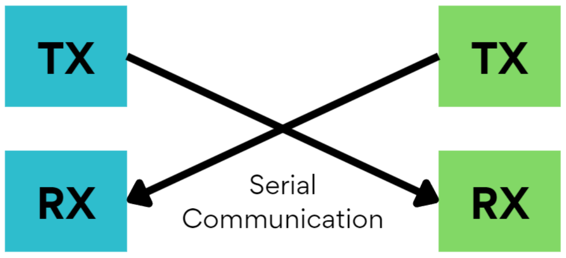

Figure 1 - Typical Communication Between Two UARTs

Step 2 - Search for Existing Implementations

Look for existing Verilog implementations of UART transceivers. Websites like GitHub or forums such as Stack Overflow can be good resources. Make sure to choose a design that is well-documented and fits the FPGA board you will be using.

Step 3 - Review and Select an Implementation

Evaluate the found implementations based on their complexity, required resources, and compatibility with your FPGA. Check if the implementation includes both transmitter (TX) and receiver (RX) functionalities.

Theory and Implementation Strategy:

- A UART transceiver module typically consists of a transmitter that sends data bits serially and a receiver that captures these bits at the destination. Key components include shift registers (for data serialization/deserialization), a baud rate generator, and a control unit to manage the read/write operations and status signals.
- When implementing in Verilog, make sure the design supports configurable baud rate and data format (number of data bits, stop bits, parity bit) to ensure flexibility.

You need to implement a UART with the following specifications.

Table 1 - UART Specifications

Input	**Bits**	**Output**	**Bits**
Data Input	8	Data Output	8
Clock Port	1	Data Valid Port	1
TX Enable Port	1	RX Port 1	
RX Enable Port	1		
TX Port	1		
Reset	1		

The top-level view of the implemented UART module would look like this:

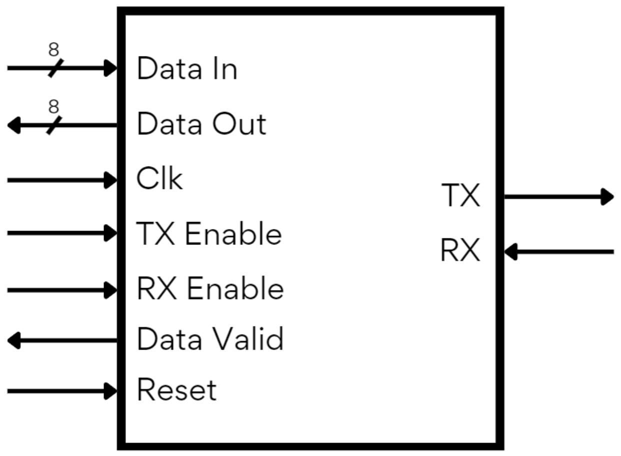

Figure 2 - Top-level View of the UART Module

Your implementation should ultimately perform the following:

A) Scenario 1
   a. Accept 8-bit data. (parallel 8 bits)
   b. Check whether TX Enable is high.
   c. TX transmits the given data using UART module.

B) Scenario 2
   a. Check if RX Enable is high.
   b. Accept serial data from the RX pin.
   c. Convert serial data to an 8-bit value.
   d. Output received data on Data Out port. (parallel 8 bits)
   e. Make Data Valid high to indicate stable data is present.

Phase 2 – Develop a Testbench

Step 4 - Understand the UART Interface

Know the input and output signals of the UART module, such as tx_data, rx_data, tx_enable, rx_enable, clk, reset, etc.

Step 5 - Write a Testbench in Verilog

Create a testbench that simulates the UART environment. This includes generating a clock signal, reset processes, sending test data to the UART transmitter, and checking the output from the UART receiver.

Step 6 - Simulate Edge Cases

Test the UART module with various data patterns, baud rates, and operational edge cases like buffer overflows, framing errors, and parity errors.

Step 7 - Use Simulation Tools

Utilize tools such as ModelSim or Vivado's integrated simulator to run simulations and debug the output.

Theory and Implementation Strategy:

- A testbench is crucial for verifying the functionality of the RTL before hardware implementation. It should comprehensively simulate all possible operational scenarios to ensure robustness.
- Use assertions and checkers to automatically verify that the output matches expected results under various conditions.

Phase 3 – Implement RTL Design in FPGA

Step 8 - Prepare the FPGA Development Environment

Set up the FPGA toolchain, like Xilinx Vivado or Intel Quartus, depending on the FPGA brand being used.

Step 9 - Develop Display Logic

Write additional RTL for driving the 7-segment displays to show transmitted and received data.

Check the following blog posts for further reference:

- https://focuslk.com/design-of-bcd-to-7-segment-display-decoder-using-logic-gates/
- https://focuslk.com/design-of-displaying-0-15-on-7-segment-display-using-logic-gates/
- https://focuslk.com/implementation-of-a-bcd-to-7-segment-display-on-fpga/

Your final implementation should have an architecture as follows. The module in blue should be developed by yourself whereas the module in green is from an existing

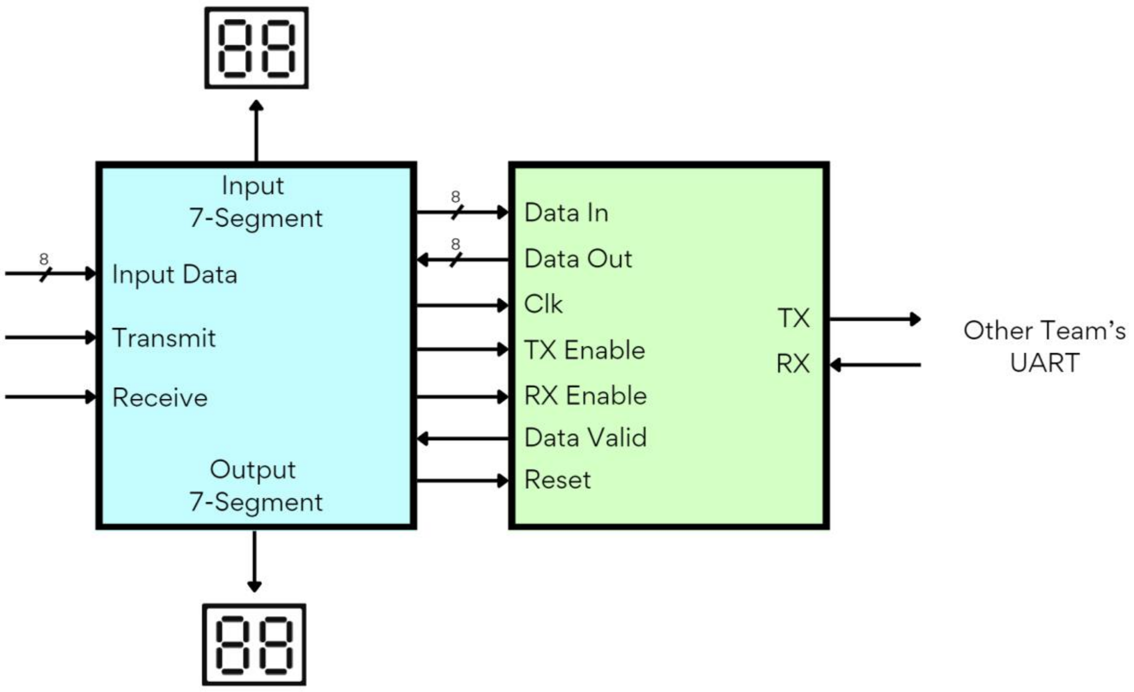

Figure 3 - Architecture of the Final Design

Step 10 – Simulate

Perform a simulation and verify if the UART and 7-segment drive circuit is working properly.

Step 11 - Integrate UART RTL with FPGA Specifics

Watch the provided video on Moodle. Map the general-purpose I/O (GPIO) pins for TX and RX lines and ensure the clock frequency matches your design specifications.

1. FPGA implementation involves considering the physical constraints of the device, such as I/O pin assignments and internal clock usage.
2. Refer to the user manual of the specific development board that you are using and identify the proper pin locations.

Step 12 – Synthesis

Run synthesis and generate bit stream.

Step 13 -Program and Configure the FPGA

Use your FPGA development environment to synthesize the design, perform place and route, and download the bitstream to the FPGA board.

Theory and Implementation Strategy:

- Test inter-group communication to ensure that UART configurations (like baud rate) match and that data is correctly transmitted and received across different FPGA boards.

Phase 4 - View the Transmit Signal Using an Oscilloscope

Step 14 - Setup Oscilloscope

Connect the oscilloscope probes to the UART TX and RX lines. Be cautious about grounding.

Step 15 - Configure Oscilloscope

Set the appropriate time/div and volt/div settings to clearly observe the serial data.

Step 16 - Transmit Data

Send data from the FPGA and observe the waveform corresponding to the UART protocol on the oscilloscope.

Step 17 - Analyze the Signal

Check for correct start bit, data bits, parity bit, and stop bit. Observe any potential noise or signal integrity issues.

Theory and Implementation Strategy:

- Using an oscilloscope allows you to visually verify the physical layer transmission characteristics such as voltage levels, timing, and signal integrity.
- This step is crucial for troubleshooting issues that are not detectable through functional simulation alone, such as ringing, reflections, or crosstalk on the transmission line.