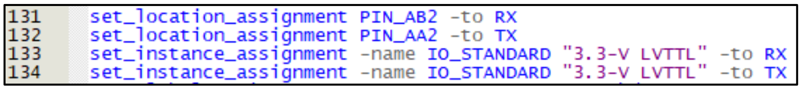
Project 01 – UART

Procedure

The objective of this project was to implement a system capable of UART communication. The system needed to receive an alphabetic character and send in response the same character with its case switched (i.e., lower to upper or upper to lower).

After creating a new project, we had to change the Quartus Settings File (.qsf). We added the necessary lines to use the onboard 40-pin header for TTL communication. The changes made are shown in Figure 1, and the pins we used are shown in Figure 2.



```
set_location_assignment PIN_AB2 -to RX
set_location_assignment PIN_AA2 -to TX
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to RX
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TX
```

Figure 1: QSF Additions

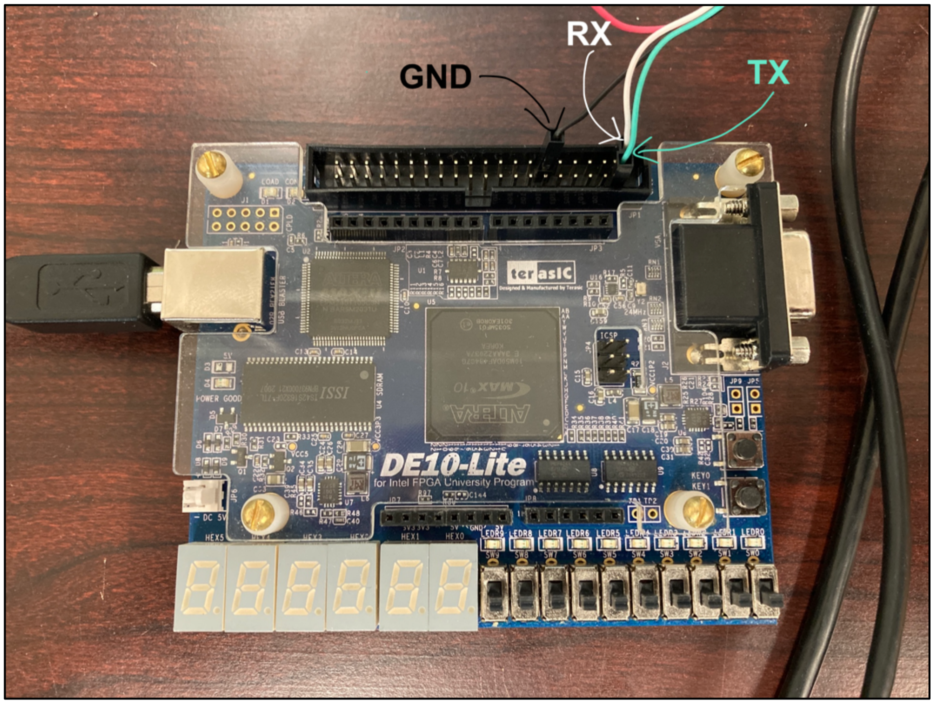
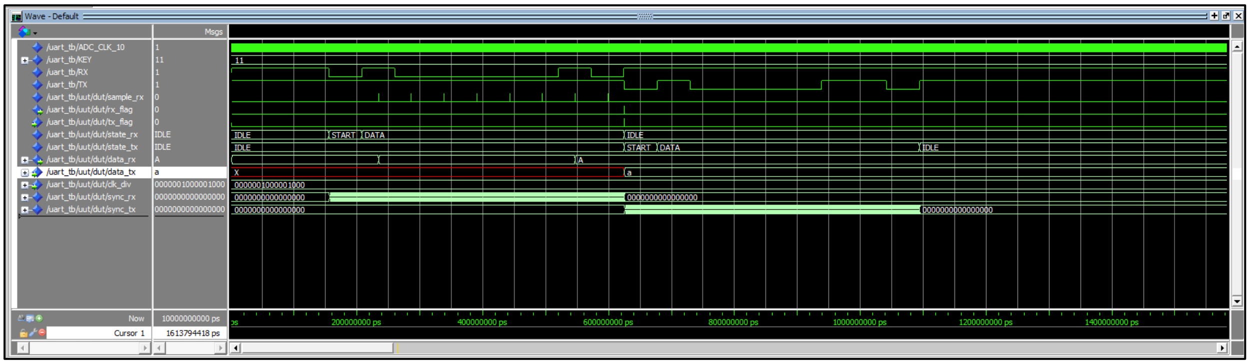


Figure 2: Pin Connections

We then created two modules. The my_UART module connected directly to RX and TX. In addition, it had an 8-bit input data bus (data_tx) and an 8-bit output data bus (data_rx). The module could be told to send the information on the data_tx bus by raising the tx_flag input for one clock cycle. Upon receiving data, the module would place the new data on the data_rx bus and then raise the rx_flag output for one clock cycle.

The UART module was the top module and instantiated the my_UART module. This module oversaw converting the incoming byte and sending it back to the my_UART module.

Once everything was in place, we used ModelSim to simulate our design. The output of our simulation is shown in Figure 3. This shows a capital ‘A’ being received on the RX line, then the rx_flag is asserted. A moment later a lowercase ‘a’ arrives on the date_tx bus and the tx_flag is asserted. The data is then transmitted on the TX line.



	Msgs
/uart_tb/ADC_CLK_10	1
/uart_tb/KEY	11
/uart_tb/RX	1
/uart_tb/TX	1
/uart_tb/uut/dut/sample_rx	0
/uart_tb/uut/dut/rx_flag	0
/uart_tb/uut/dut/tx_flag	0
/uart_tb/uut/dut/state_rx	IDLE
/uart_tb/uut/dut/state_tx	IDLE
/uart_tb/uut/dut/data_rx	A
/uart_tb/uut/dut/data_tx	a
/uart_tb/uut/dut/clk_div	0000001000001000
/uart_tb/uut/dut/sync_rx	0000000000000000
/uart_tb/uut/dut/sync_tx	0000000000000000
Now	10000000000 ps
Cursor 1	1613794418 ps

Figure 3: Simulation

Once the simulation was successful. We tried loading the system onto our FPGA. This failed initially, but after some debugging, we managed to get it working.

Results

Ultimately, we got the project working with the occasional misread/write. This could be due to the fact that we didn’t implement any oversampling or error detection mechanisms, as they only occur occasionally and are inconsistent. Also, if an unknown character was received a capital ‘E’ was returned successfully.

Conclusions

This project helped improve our understanding of the UART protocol. It also gave us much more insight into the challenges of implementing a communication protocol on an FPGA.