PHYS 234 Lab 11: UART Serial Communication

Ethan Yang

UART, short for Universal Asynchronous Receiver/Transmitter, is a hardware protocol for serial communication that takes data and transmits it as a serial stream of bits. Moreover, it's called asynchronous because it doesn't use a shared clock signal.

To use Python to communicate with the FPGA at 19200 baud, we would run the following Python code in Spyder.

```
import serial
a = serial.Serial('COM7',19200)
```

This lab will complete 2 tasks: to design a circuit that receives 1 byte of data from the computer and displays it on the FPGA LEDs and design a circuit that transmits 1 byte of data to the computer. To send a hexadecimal value to the FPGA, use

```
a.write(b'\x41')
```

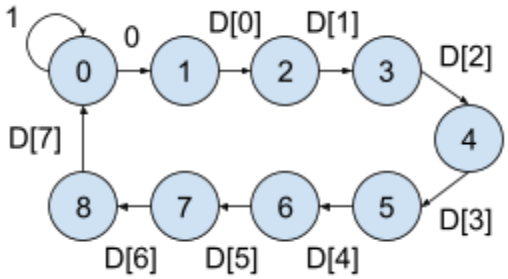
to send 41 in hexadecimal and use

```
a.write(b'a')
```

to send the ascii code for the character 'a'.
To display the transmitted data in base 10, use

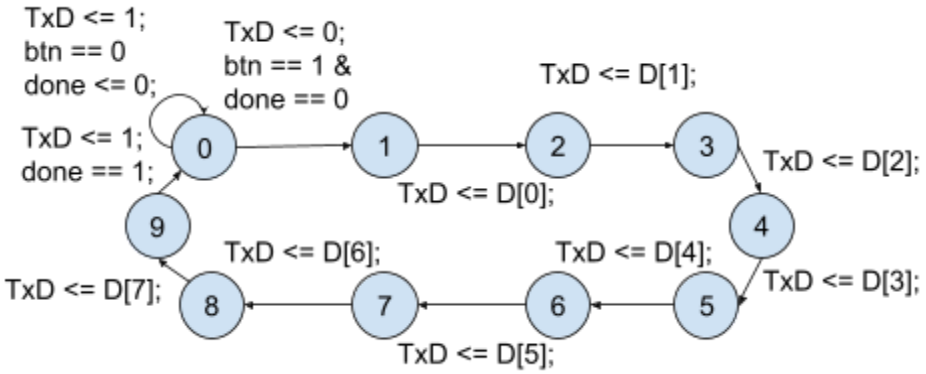
```
a.read()[0]
```

The receiver module operates like a finite state machine that captures incoming serial data bits and reconstructs them into [7:0] D. It starts from the idle state 0 and waits for new transmissions. When data is detected, the receiver transitions through state 1 to 8, where each state represents a data bit. The state diagram is as follows:



On the other hand, the transmitter module is also implemented as a finite state machine. It converts a 8 bit data word into a serial output stream. Similarly, the machine starts with an idle state 0, and it waits for a transmission request signal given by the input btn. After transmission begins, the machine transmits the start bit and the D values subsequently. After all 8 bits are transmitted, the machine enters the stop state 9 where a stop bit is assigned and a done flag is

raised. The “btn == 0” condition in state 0 ensures that one complete transmission is done and that the previous data wouldn’t be repeated. The state diagram is as follows:



The Verilog code that implements the receive and transmit functions are shown below:

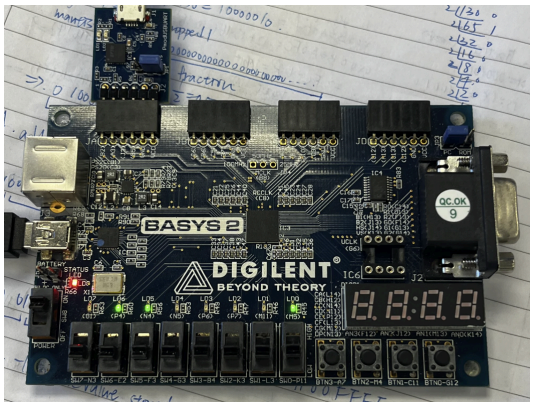
```
`timescale 1ns / 1ps
module receiver(
      input mclk,
      input RxD,
      output reg [7:0] D
      );

reg [3:0] state;
reg [10:0] q;
reg clk19200;


always @ (posedge mclk)
      if (q < 1302)
            q <= q + 1;
      else begin
            clk19200 <= ~clk19200;
            q <= 0;
      end

always @ (posedge clk19200)
      if (state == 0 & RxD ==0)
            state <= 1;
      else if (state == 1) begin
            D[0] <= RxD;
            state <= 2;
```

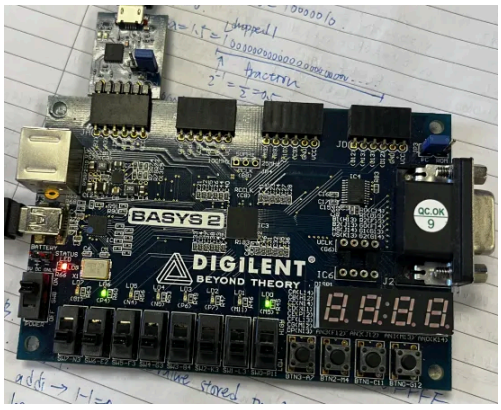
```
      end
      else if (state == 2) begin
            D[1] <= RxD;
            state <= 3;
      end
      else if (state == 3) begin
            D[2] <= RxD;
            state <= 4;
      end
      else if (state == 4) begin
            D[3] <= RxD;
            state <= 5;
      end
      else if (state == 5) begin
            D[4] <= RxD;
            state <= 6;
      end
      else if (state == 6) begin
            D[5] <= RxD;
            state <= 7;
      end
      else if (state == 7) begin
            D[6] <= RxD;
            state <= 8;
      end
      else if (state == 8) begin
            D[7] <= RxD;
            state <= 0;
      end
endmodule

module transmitter(
      input mclk,
      input btn,
      output reg TxD
      );

reg [3:0] state;
reg [10:0] q;
```

```
reg clk19200;
reg done;

reg [7:0] D;
initial D = 65;

always @ (posedge mclk)
      if (q < 1302)
            q <= q + 1;
      else begin
            clk19200 <= ~clk19200;
            q <= 0;
      end

always @(posedge clk19200)
      if (btn == 0) begin
            TxD <= 1;
            state <= 0;
            done <= 0;
      end
      else if (state == 0 & done == 0) begin
            TxD <= 0;
            state <= 1;
      end
      else if (state == 1) begin
            TxD <= D[0];
            state <= 2;
      end
      else if (state == 2) begin
            TxD <= D[1];
            state <= 3;
      end
      else if (state == 3) begin
            TxD <= D[2];
            state <= 4;
      end
      else if (state == 4) begin
            TxD <= D[3];
            state <= 5;
```

```
        end
        else if (state == 5) begin
                TxD <= D[4];
                state <= 6;
        end
        else if (state == 6) begin
                TxD <= D[5];
                state <= 7;
        end
        else if (state == 7) begin
                TxD <= D[6];
                state <= 8;
        end
        else if (state == 8) begin
                TxD <= D[7];
                state <= 9;
        end
        else if (state == 9) begin
                TxD <= 1;
                done <= 1;
                state <= 0;
        end
endmodule
```

As a result, the FPGA board successfully received the hexadecimal value 0x41 and the ascii code of ‘a’ from the computer. The ascii code 97 is 1100001 in binary shown on the left figure and 0x41 is 1000001 in binary shown on the right figure.





The FPGA board also successfully transmitted the decimal value D = 65 to the computer as the python script outputs 65.