Kassandra Marquez
Compe470L
13 October 2021

# COMPE470L LAB 5 UART

**Introduction:**

Within Lab 5, we will be creating a universal asynchronous receiver and transmitter on the Basys3 board with a baud rate of 9600 bits/s. The validity of the design will be tested by using PuTTy as our terminal emulator in which we will input ASCII characters of our last name. In this case, the ASCII characters to be sent are ‘M’, ‘A’, ‘R’, ‘Q’, ‘U’, ‘E’, ‘Z’.

**Source Code:**

For implementing the transmitter and receiver, we will design a finite state machine. Our FSM consists of 5 states: IDLE → START BIT → DATA BITS → STOP BIT → CLEAN UP. Knowing that the Basys3 board has a 100MHz clock, we will set up a ‘CLKS_PER_BIT’ parameter that will serve as a limit for our input clock that will count up to 10416 to account for our 9600 bits/s Baud rate.

For both the transmitter and receiver, we will have input data values that will trigger their own FSM. If enabled, the machine will leave the IDLE state and go on to START BIT state. After START BIT, the machine will then start transmission/reception serially by using a bit index. In this case, our bit index goes from 0 - 7 places. Once it reaches the last bit index, the machine will then go to STOP BIT state and finish in CLEAN UP where our memory registers are resetted.

Then we will have a top module to connect the transmitter, receiver, and Basys3 board. Using the constraint files, ‘RsRx’ will be our input while ‘RxTx’ will be our output. The data coming out of the receiver will then become our input for our transmitter hence, we will connect them with a wire. Lastly, the data values in Rx will be connected to the data values in Tx so that the FSM will act accordingly.

```
//------------uart tx---------
`timescale 1ns / 1ps

module uart_tx
  #(parameter CLKS_PER_BIT = 10416)
  (
   input       i_Clock,
   input       i_Tx_DV,
   input [7:0] i_Tx_Byte,
   output      o_Tx_Active,
   output reg  o_Tx_Serial,
   output      o_Tx_Done
   );

  parameter s_IDLE         = 3'b000;
```

Kassandra Marquez
Compe470L
13 October 2021

```
  parameter s_TX_START_BIT = 3'b001;
  parameter s_TX_DATA_BITS = 3'b010;
  parameter s_TX_STOP_BIT  = 3'b011;
  parameter s_CLEANUP      = 3'b100;

  reg [2:0]    r_SM_Main     = 0;
  reg [7:0]    r_Clock_Count = 0;
  reg [2:0]    r_Bit_Index   = 0;
  reg [7:0]    r_Tx_Data     = 0;
  reg          r_Tx_Done     = 0;
  reg          r_Tx_Active   = 0;

  always @(posedge i_Clock)
    begin

      case (r_SM_Main)
        s_IDLE :
          begin
            o_Tx_Serial   <= 1'b1;         // Drive Line High for Idle
            r_Tx_Done     <= 1'b0;
            r_Clock_Count <= 0;
            r_Bit_Index   <= 0;

            if (i_Tx_DV == 1'b1)
              begin
                r_Tx_Active <= 1'b1;
                r_Tx_Data   <= i_Tx_Byte;
                r_SM_Main   <= s_TX_START_BIT;
              end
            else
              r_SM_Main <= s_IDLE;
          end // case: s_IDLE


        // Send out Start Bit. Start bit = 0
        s_TX_START_BIT :
          begin
            o_Tx_Serial <= 1'b0;

            // Wait CLKS_PER_BIT-1 clock cycles for start bit to finish
```

Kassandra Marquez
Compe470L
13 October 2021

```
          if (r_Clock_Count < CLKS_PER_BIT-1)
            begin
              r_Clock_Count <= r_Clock_Count + 1;
              r_SM_Main     <= s_TX_START_BIT;
            end
          else
            begin
              r_Clock_Count <= 0;
              r_SM_Main     <= s_TX_DATA_BITS;
            end
        end // case: s_TX_START_BIT


      // Wait CLKS_PER_BIT-1 clock cycles for data bits to finish
      s_TX_DATA_BITS :
        begin
          o_Tx_Serial <= r_Tx_Data[r_Bit_Index];

          if (r_Clock_Count < CLKS_PER_BIT-1)
            begin
              r_Clock_Count <= r_Clock_Count + 1;
              r_SM_Main     <= s_TX_DATA_BITS;
            end
          else
            begin
              r_Clock_Count <= 0;

              // Check if we have sent out all bits
              if (r_Bit_Index < 7)
                begin
                  r_Bit_Index <= r_Bit_Index + 1;
                  r_SM_Main   <= s_TX_DATA_BITS;
                end
              else
                begin
                  r_Bit_Index <= 0;
                  r_SM_Main   <= s_TX_STOP_BIT;
                end
            end
        end // case: s_TX_DATA_BITS
```

Kassandra Marquez
Compe470L
13 October 2021

```
      // Send out Stop bit.  Stop bit = 1
      s_TX_STOP_BIT :
        begin
          o_Tx_Serial <= 1'b1;

          // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
          if (r_Clock_Count < CLKS_PER_BIT-1)
            begin
              r_Clock_Count <= r_Clock_Count + 1;
              r_SM_Main     <= s_TX_STOP_BIT;
            end
          else
            begin
              r_Tx_Done     <= 1'b1;
              r_Clock_Count <= 0;
              r_SM_Main     <= s_CLEANUP;
              r_Tx_Active   <= 1'b0;
            end
        end // case: s_Tx_STOP_BIT


      // Stay here 1 clock
      s_CLEANUP :
        begin
          r_Tx_Done <= 1'b1;
          r_SM_Main <= s_IDLE;
        end


      default :
        r_SM_Main <= s_IDLE;

    endcase
  end

assign o_Tx_Active = r_Tx_Active;
assign o_Tx_Done   = r_Tx_Done;
```

Kassandra Marquez
Compe470L
13 October 2021
endmodule

```
//-----uart rx------
`timescale 1ns / 1ps

module uart_rx
  #(parameter CLKS_PER_BIT = 10416)
  (
   input        i_Clock,
   input        i_Rx_Serial,
   output       o_Rx_DV,
   output [7:0] o_Rx_Byte
   );

  parameter s_IDLE         = 3'b000;
  parameter s_RX_START_BIT = 3'b001;
  parameter s_RX_DATA_BITS = 3'b010;
  parameter s_RX_STOP_BIT  = 3'b011;
  parameter s_CLEANUP      = 3'b100;

  reg          r_Rx_Data_R = 1'b1;
  reg          r_Rx_Data   = 1'b1;

  reg [7:0]     r_Clock_Count = 0;
  reg [2:0]     r_Bit_Index   = 0; //8 bits total
  reg [7:0]     r_Rx_Byte     = 0;
  reg          r_Rx_DV       = 0;
  reg [2:0]     r_SM_Main     = 0;

  // Purpose: Double-register the incoming data.
  // This allows it to be used in the UART RX Clock Domain.
  // (It removes problems caused by metastability)
  always @(posedge i_Clock)
    begin
      r_Rx_Data_R <= i_Rx_Serial;
      r_Rx_Data   <= r_Rx_Data_R;
    end


  // Purpose: Control RX state machine
```

Kassandra Marquez
Compe470L
13 October 2021

```
 always @(posedge i_Clock)
   begin

     case (r_SM_Main)
       s_IDLE :
         begin
           r_Rx_DV       <= 1'b0;
           r_Clock_Count <= 0;
           r_Bit_Index   <= 0;

           if (r_Rx_Data == 1'b0)          // Start bit detected
             r_SM_Main <= s_RX_START_BIT;
           else
             r_SM_Main <= s_IDLE;
         end

       // Check middle of start bit to make sure it's still low
       s_RX_START_BIT :
         begin
           if (r_Clock_Count == (CLKS_PER_BIT-1)/2)
             begin
               if (r_Rx_Data == 1'b0)
                 begin
                   r_Clock_Count <= 0;  // reset counter, found the middle
                   r_SM_Main     <= s_RX_DATA_BITS;
                 end
               else
                 r_SM_Main <= s_IDLE;
             end
           else
             begin
               r_Clock_Count <= r_Clock_Count + 1;
               r_SM_Main     <= s_RX_START_BIT;
             end
         end // case: s_RX_START_BIT


       // Wait CLKS_PER_BIT-1 clock cycles to sample serial data
       s_RX_DATA_BITS :
         begin
```

Kassandra Marquez
Compe470L
13 October 2021

```
          if (r_Clock_Count < CLKS_PER_BIT-1)
            begin
              r_Clock_Count <= r_Clock_Count + 1;
              r_SM_Main     <= s_RX_DATA_BITS;
            end
          else
            begin
              r_Clock_Count          <= 0;
              r_Rx_Byte[r_Bit_Index] <= r_Rx_Data;

              // Check if we have received all bits
              if (r_Bit_Index < 7)
                begin
                  r_Bit_Index <= r_Bit_Index + 1;
                  r_SM_Main   <= s_RX_DATA_BITS;
                end
              else
                begin
                  r_Bit_Index <= 0;
                  r_SM_Main   <= s_RX_STOP_BIT;
                end
            end
        end // case: s_RX_DATA_BITS


      // Receive Stop bit.  Stop bit = 1
      s_RX_STOP_BIT :
        begin
          // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
          if (r_Clock_Count < CLKS_PER_BIT-1)
            begin
              r_Clock_Count <= r_Clock_Count + 1;
              r_SM_Main     <= s_RX_STOP_BIT;
            end
          else
            begin
              r_Rx_DV       <= 1'b1;
              r_Clock_Count <= 0;
              r_SM_Main     <= s_CLEANUP;
            end
```

Kassandra Marquez
Compe470L
13 October 2021

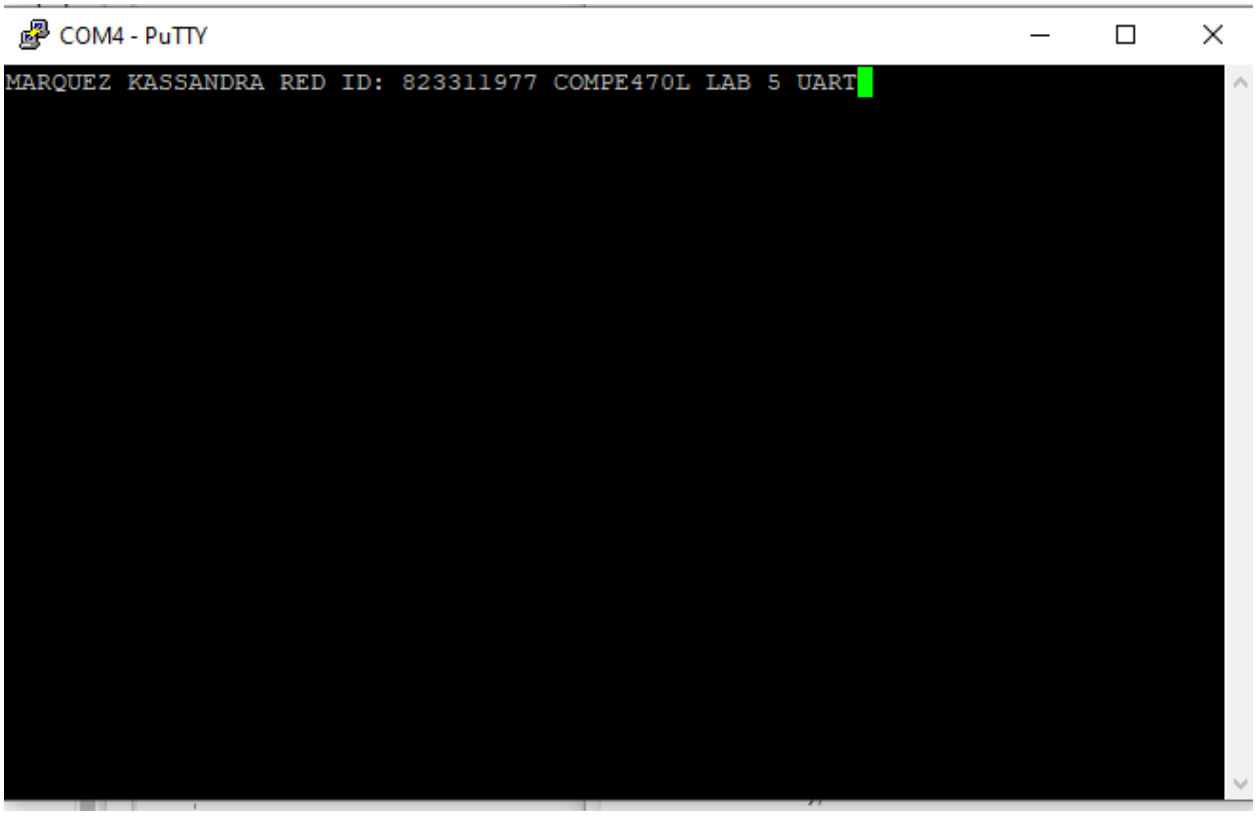
```
        end // case: s_RX_STOP_BIT


       // Stay here 1 clock
       s_CLEANUP :
         begin
           r_SM_Main <= s_IDLE;
           r_Rx_DV   <= 1'b0;
         end


       default :
         r_SM_Main <= s_IDLE;

     endcase
   end

  assign o_Rx_DV   = r_Rx_DV;
  assign o_Rx_Byte = r_Rx_Byte;

endmodule
//-------------top module--------------------
`timescale 1ns / 1ps

module top(
   input clk,
   input RsRx,
   output RsTx
   );

   wire [7:0] data;
   wire dv;

   parameter CLKS_PER_BIT = 10416;

   uart_rx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uut0 (.i_Clock(clk), .i_Rx_Serial(RsRx),
.o_Rx_Byte(data), .o_Rx_DV(dv));

   uart_tx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uut1 (.i_Clock(clk), .i_Tx_Byte(data),
.o_Tx_Serial(RsTx), .i_Tx_DV(dv));
```

Kassandra Marquez
Compe470L
13 October 2021

endmodule
//----------------------------------------

**Demo**
https://youtu.be/uFRN6UUm8q8

**Screenshot**




**Conclusion**

Within this lab, I realized that I must pay attention to the specific frequency of the clock so that I can calculate the baud rate behavior correctly. In this case, since the Basys3 is a 100Mhz clock and we want a 9600 bit/s baud rate, the amount ‘clks_per_bit’ I must count up to was 10,416. If it was significantly less than 10,416, then the baud rate would be too fast and PuTTy would output wrong values. If it was significantly greater than 10,416, then it would be too slow and again PuTTy would output wrong values. So essentially, I learned to be careful and use the Boards components and characteristics, because I mistakenly assumed it was a 10Mhz clock.