Course Name:Embedded Systems Hardware/Software

Course Number and Section: **14:332493:03**

Experiment: Lab #3, Where no clock has gone before

Lab Instructor: Phillip Southard

Date Performed: 03/14/19

Date Submitted: 03/28/19

Submitted by: Raul Mori   183001439

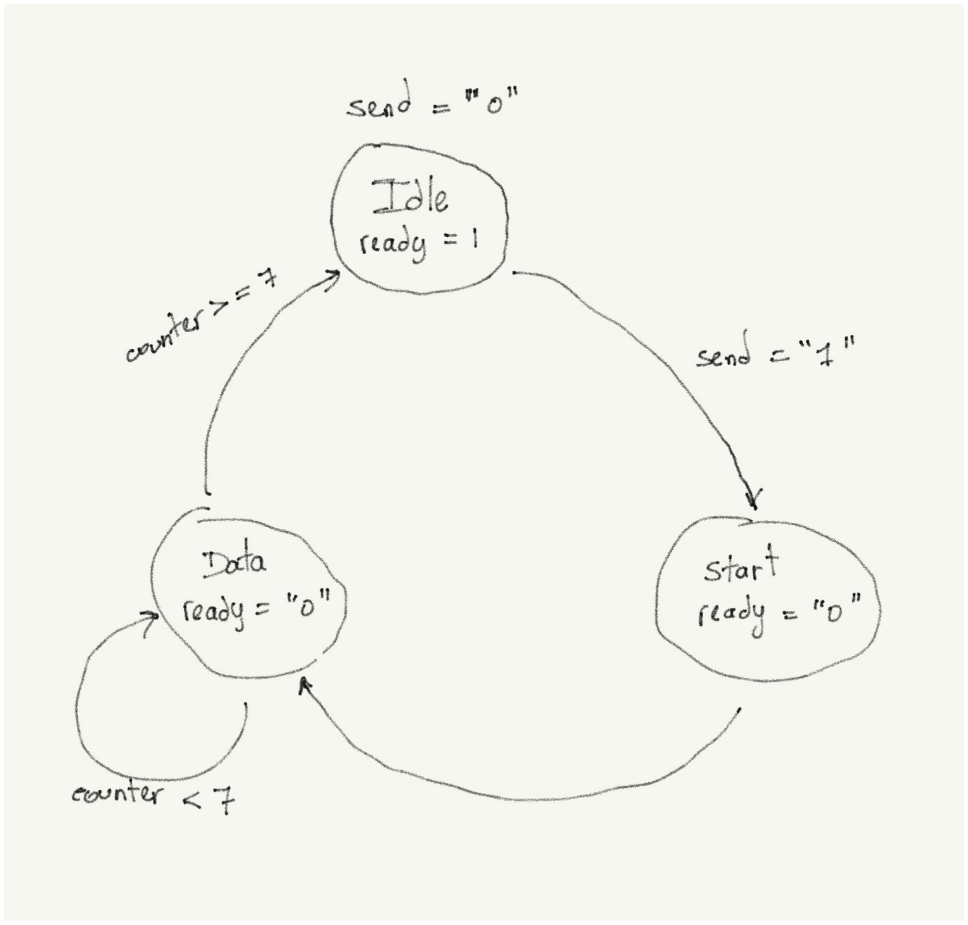
Contents

Purpose (for all parts of the Lab)

The purpose of this lab is to create an Universal Asynchronous Receiver Transmitter (UART) by designing certain parts of the UART in the form of a Finite State Machine (FSM). In the first portion of this lab, we will be designing a transmitter for the UART based on the RS232 protocol. The receiving side of the uart and the top level is provided along with the testbench so that we can test if the transmitter is working properly. The top level design is formed in a way such that the test signals loop back to the transmitter. Running a successful simulation should show the test ASCII characters being looped back.

In the second part of the lab, we will be making another state machine called 'sender' which will send the ASCII version of our NETID, one character at a time. This will be done by connecting a USB<-->UART pmod to send signals to the computer through the Zybo board pins.

Prelab: U-what?



Part 1: We can build him

1.1. Theory of Operation

In this lab we design a transmitter that relies on RX operation. A State Diagram for TX is used to design an FSM using VHDL. We use 8 data bits, 0 parity bits and 1 stop bit and send them using the RS232 protocol. The inputs of TX will have a clock, a clock enable, send, reset and an 8 bit input called char. We will have 1-bit outputs called ready and tx. When reset is asserted (presed) during a rising clock edge, the Finite State Machine (FSM) goes to an idle state and internal registers cleared. When the FSM is in idle state, ready is "1". During rising edge, and send and clock enable are 1,the sequence in char is loaded into a register and the FSM sends data.

1.2. Design

a. VHDL Code

--Code for uart_tx (We don't include the other codes because they were given to us

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity uart_tx is
    Port (
        clk : in STD_LOGIC;
        en : in STD_LOGIC;
        send : in STD_LOGIC;
        rst : in STD_LOGIC;
        char : in STD_LOGIC_VECTOR (7 downto 0);
        ready : out STD_LOGIC;
        tx : out STD_LOGIC);
end uart_tx;


architecture Behavioral of uart_tx is
```

```
        type state is (idle, start, data);
        signal curr : state := idle;

        signal D :std_logic_vector(7 downto 0) := X"00";


        begin              --This starts the part  that has to do with the receiving side Of the Finite State Machine (FSM)
            FSM:process(clk)
            variable count : natural := 0;
                begin
                  if rising_edge(clk) then    --.This is the Main "IF" . This is if "reset" is high
                      if rst = '1' then
                          D <= X"00";
                          curr <= idle;
                      end if;

                      if en = '1' then          --This is the "SUB-IF" because the main-if wasn't closed
                          case curr is                  --Here we start a "CASE"
                              when idle =>
                                  ready <= '1';
                                  tx <= '1';
                                  if send = '1' then
                                    curr <= start;
                                  end if;
                              when start =>
                                  ready <='0';
                                  tx <= '0';
                                  D <= char;
                                  count := 0;
                                  curr <= data;
                              when data =>
                                  if count < 8 then
                                      tx <= D(count);
                                          count := count +1;
                                  else
                                    tx <= '1';
                                    curr <= idle;
                                  end if;
                          end case;
                      end if;             --Here we end the second "SUB-IF"
                  end if;                  --Here we end the "MAIN-IF"
            end process;
    end Behavioral;
```

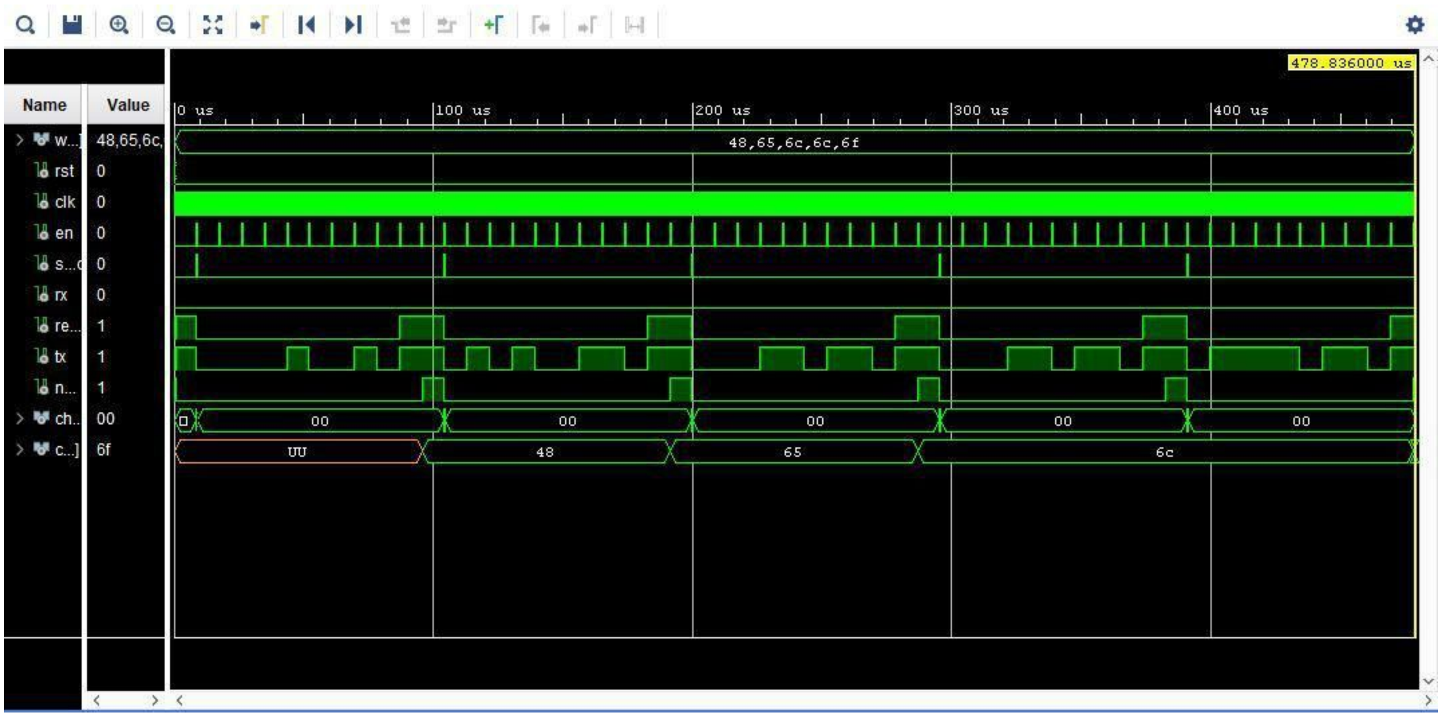
1.3. Test

a. Test Bench VHDL for uart_tb

```
-- written by Raul Mori
-- Spring 2018
-- This is just a TESTBENCH for the "TOP" design
--We bring in 2 designs. One is the "Clock" and the other design is the "Enable"
--We only use those two design because they are the only ones where their inputs cause a difference in the outputs.

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity uart_tb is
end uart_tb;

architecture tb of uart_tb is

    type str is array (0 to 4) of std_logic_vector(7 downto 0);
    signal word : str := (x"48", x"65", x"6C", x"6C", x"6F");

    signal rst : std_logic := '0';
    signal clk, en, send, rx, ready, tx, newChar : std_logic := '0';
    signal charSend, charRec : std_logic_vector(7 downto 0) := (others => '0');


    component uart port (
    clk, en, send, rx, rst  : in std_logic;
    charSend                : in std_logic_vector(7 downto 0);
    ready, tx, newChar      : out std_logic;
    charRec                 : out std_logic_vector(7 downto 0)
    );
    end component;


    begin               --Notice we have a total of 3 processes

        process begin           -- This is for the clock (clk) @125 MHz
            clk <= '0';
            wait for 4 ns;
            clk <= '1';
            wait for 4 ns;
        end process;


        process begin           --This is the ENABLE (en)  @ 125 MHz / 1085 = ~115200 Hz
            en <= '0';
            wait for 8680 ns;
            en <= '1';
            wait for 8 ns;
        end process;


        process begin             -- This is the process for signal stimulation . This is the part of the processes the image.
            rst <= '1';
            wait for 100 ns;
            rst <= '0';
            wait for 100 ns;

            for index in 0 to 4 loop            --Here we create a loop
                wait until ready = '1' and en = '1';
                charSend <= word(index);
                send <= '1';
                wait for 200 ns;
                charSend <= (others => '0');        --This sets "charSend" as "0", no matter how many bits it is
                send <= '0';                        --This sets "send" as "0", no matter how many bits it is
                wait until ready = '1' and en = '1' and newChar = '1';

                if charRec /= word(index) then                                  --This is the "IF" condition inside the loop
                    report "Send/Receive MISMATCH at time: " & time'image(now) &
                    lf & "expected: " &
                    integer'image(to_integer(unsigned(word(index)))) &
                    lf & "received: " & integer'image(to_integer(unsigned(charRec)))
                    severity ERROR;
                end if;

            end loop;           --Here we end a loop

            wait for 1000 ns;
            report "End of testbench" severity FAILURE;

        end process;

end tb;
```

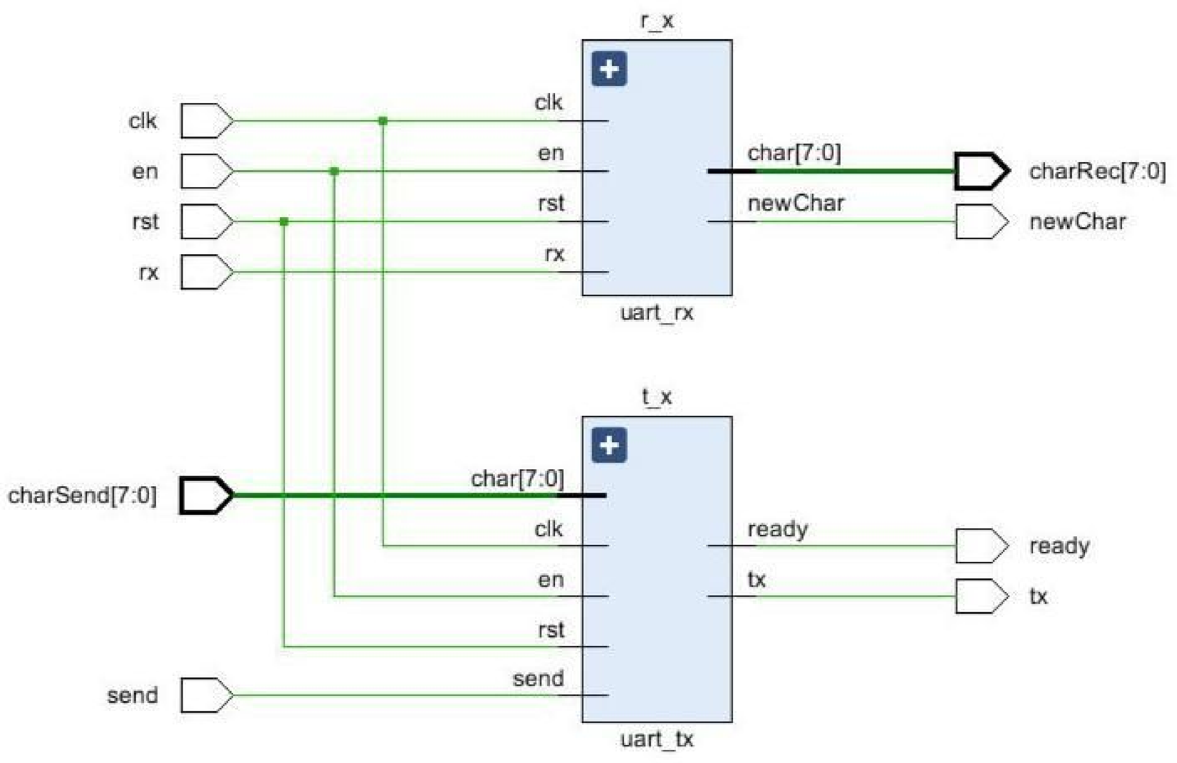
Test bench was provided in the lab.

b. Simulation Results

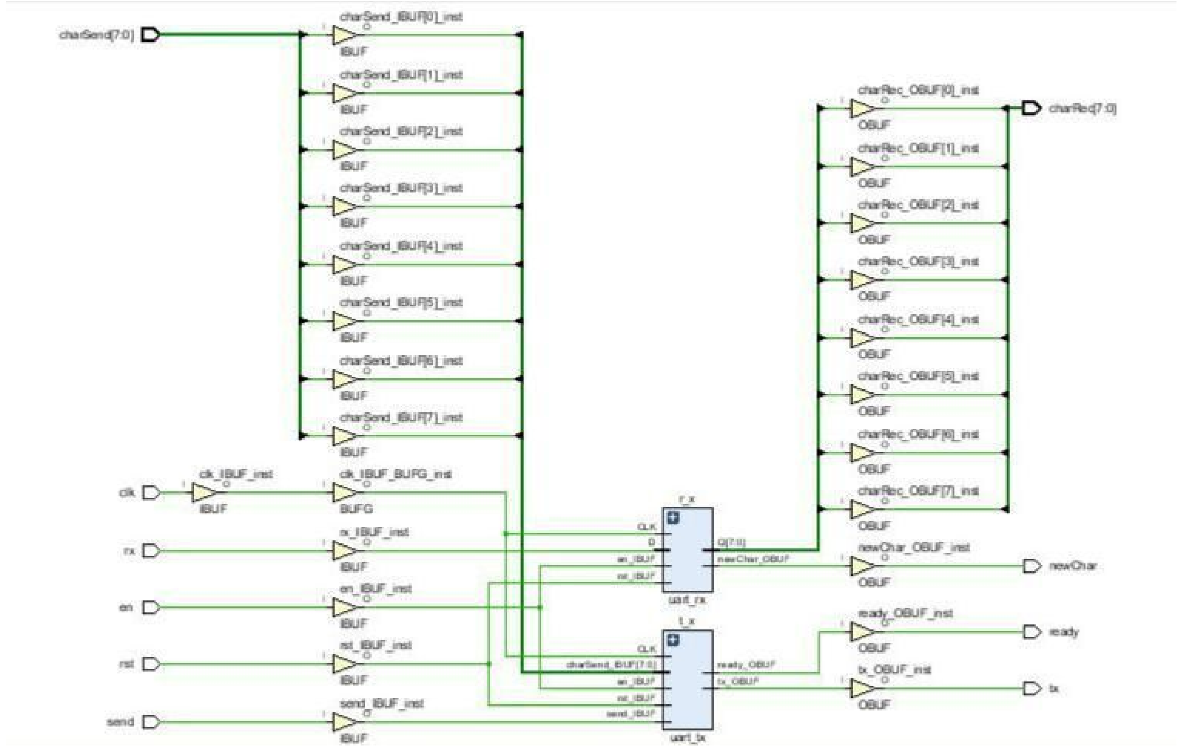


1.4.Implementation

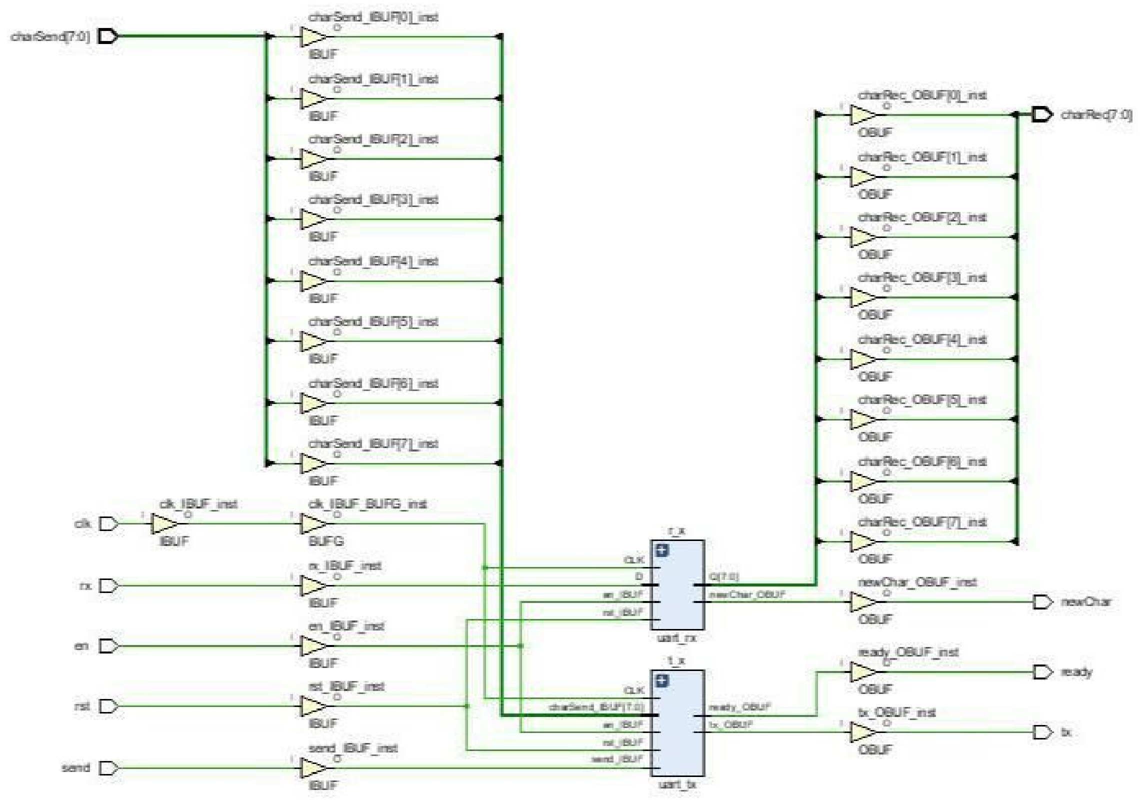
a. Elaboration Schematic



b. Synthesis Schematic

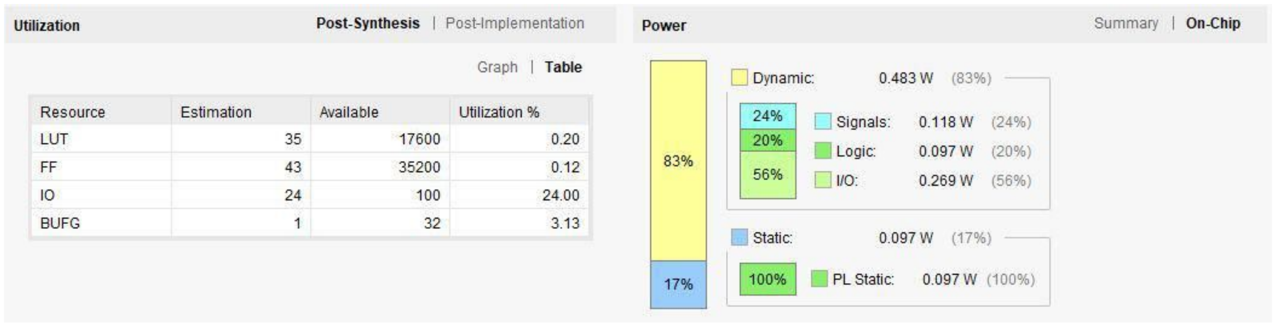


c. Implementation Schematic



d. Project Summary Images

i. Post-Synthesis Utilization table/On-chip Power Graphs



Utilization | Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	35	17600	0.20
FF	43	35200	0.12
IO	24	100	24.00
BUFG	1	32	3.13

Power | Summary | On-Chip

Dynamic:	0.483 W	(83%)
Signals:	0.118 W	(24%)
Logic:	0.097 W	(20%)
I/O:	0.269 W	(56%)
Static:	0.097 W	(17%)
PL Static:	0.097 W	(100%)

Part 2: We have the technology

2.1. Theory of Operation

We make another state machine here named "SENDER". As inputs is has a "clock", "clock-enable", "reset", "button" and a signal called "ready". As output it has a 1-bit send signal and an 8-bit signal called char. It also has an internal counter which counts up until all characters of our NETID are sent in the form of ASCII. The instantiation of this NETID is similar to what we saw in the testbench. This FSM will have 4 states: IDLE, BUSYA, BUSYB and BUSYC. Our NETID will consist of 6 characters so the array will be six characters long. When RESET is asserted while clock is on rising edge, the SENDER clears all outputs and the counter.. When SEND is asserted to 1, "i" is incremented by 1, and the bit of the NETID corresponding to the counter value is loaded onto char. This occurs only when BUTTON and READY are both "1" and counter value is less than n.

When equal to n with ready and button unchanged, it should go to idle state and the counter should go to 0. Otherwise it enters BUSYA state for the former condition. BUSYA should go straight to BUSYB state. In BUSYB, SEND is set to 0 and the state changes to busyC. This last state is maintained until ready changes to 1 and button is 0. When that occurs, busyC goes to an idle state.

2.2. Design

a. VHDL Code

--Sender---
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sender is
    Port (
        rst : in STD_LOGIC;
        clk : in STD_LOGIC;
        en : in STD_LOGIC;
        btn : in STD_LOGIC;
        ready : in STD_LOGIC;
        send : out STD_LOGIC;
        char : out STD_LOGIC_VECTOR (7 downto 0));
end sender;


architecture Behavioral of sender is

    type word is array (0 to 4) of STD_LOGIC_VECTOR(7 downto 0);
    type state is (idle, busyA, busyB, busyC);
    signal curr : state := idle;
    signal NETID : word := (x"6A", x"64", x"65", x"34", x"38");
    signal i : STD_LOGIC_VECTOR(2 downto 0) := "000";


    begin
      FSM:process(clk)
          begin
            if rising_edge(clk) and en = '1' then
                if rst = '1' then
                    send <= '0';
                    char <= x"00";
                    i <= "000";
                    curr <= idle;
                end if;

                case curr is
                    when idle =>
                        if ready = '1' and btn = '1' then
                            if unsigned(i) < 5 then
```

```
                                                send <= '1';
                                                char <= netid(natural(to_integer(unsigned(i))));
                                                i <= STD_logic_VECTOR(unsigned(i) + 1);
                                                curr <= busyA;
                                            else
                                                i <= "000";
                                            end if;
                                        end if;

                                    when busyA =>
                                          curr <= busyB;
                                    when busyB =>
                                          send <= '0';
                                          curr <= busyC;
                                    when busyC =>
                                         if ready = '1' and btn = '0' then
                                             curr <= idle;
                                         end if;
                                end case;
                            end if;
                      end process;
end Behavioral;
```

```
------------------------------------------------------------debounce--------------------------------------------------------
--The point of this counter is that we will only get an output after the counter has counted a "number" of times for a certain
entitiy input called "btn"

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

--declare entity
entity debounce is          --Remember the entitiy values are values that the user manually inputs into the system
    Port (clk: in std_logic;
        btn: in std_logic;
        dbnc: out std_logic );  --Remember this is the Main Output
end debounce;

architecture DB of debounce is
        --Remember that when we assign a "0" to a more than one bit value, we must use the "others" command
       signal counter: std_logic_vector (25 downto 0) := (others => '0');    --use 26 bits to be able to count to 6.25 million
       signal sft_Reg: std_logic_vector(1 downto 0);                        -- we create a 2 bit long shift register
begin
```

```
    process(clk, btn) begin --declare process. Notice we have two temporaries  (a clock period is a loop)

        if rising_edge(clk) then  --This is the loop of one period

            sft_Reg(1) <= sft_Reg(0);       --bit-1 gets value from bit-0

            sft_Reg(0) <= btn;        --Here we put the entity "btn" into register bit-0 (this happens no matter what the value of
entity "btn" is)
                                        --Althought the button is not "held" pressing for 50ms will generate 6.25 million counts. So a "1"
will change
                                        --positions within the 2 shift register 6.25 million times, because button "1" will always be
during 6.25 million counts
                                         --need to count to 6.25 million
                                         --at the sample rate of 125MHZ, using this number of counts will give 50ms debounce time
                                          --The counter starts counting every rising edge, and as long as it detects a button input of "1"
                if(unsigned(counter) < 6250000) then         --In this case (indepedent of the value of "btn") if temporary
"counter" is less than 6.25 million
                                                     --sherif said to use 50ms. This is only possible because we have a 125Mhz clock.
So the numbers of counts we give it
                                                    --will determine the total time the circuit will debounce.
                                                    --For example if we get a "1" signal that was caused by interference within the
50ms, but not long enough that it
                                                    --lasts the entire 50ms, so the counter will not count for the entire 50ms.
                                                    --Remember the counter and the 50ms time are equivalent
                                                   --while less than 6.25 million output 0. This is within the period where the
oscillation button interferance is happening, and the high signal dissapered before the 6.25 million count
                                                  --This gives a debounce output of "0" if the debounce simply thought that the
"high" btn was just interference.
                    dbnc <= '0';          -- A "0" is put into the  output entity port "dbnc"

                        if(sft_Reg(1) = '1' ) then    --Addttionally, if bit-1 register is high
                            counter <= std_logic_vector( unsigned(counter) + 1 ); --We add a "1-integer" to temporary
"counter"
                        else

                            counter <= (others => '0'); -- This just puts a 26-bit "0" in the temporary "counter" (same as
resetting the counter)
                        end if;

                else    -- This is if the counter has reached exaclty 6.25 million counts or more

                    dbnc <= '1';          --A value of high is put on the output entity "dbnc"
                              --once the button is released, reset the output and counter.
                              --Remember that the "btn" (button) is not the output of the debouncer, but it is the actual
value of the button

                              --!!!!!Remember that the button may still be oscilating so for the debounce circuit to
generate a "0" it would have to
                              --detect a "0" at any count for the debounce to give an output of "0"
                              --This is the debounce giving an output of "0" after the debounce detected a corrent and
sufficient period for a "high btn"
                        if(btn = '0') then  --Additionaly, if a "0" is detected for  entity "btn", while the counter reached the
6.25 million
                            dbnc <= '0';           --"0" is put in the output entity "dbnc"
                            counter <= (others => '0');    --We give a "0" to the temporary 26-bit "counter" (resets
counter)
```

```
                                             --Remember that this works differently than a "1" btn output. This is because the "1"
output of the button
                                             --must occur 62.5 million times for the debounce output to finally give a "1". Whereas for
the debounce to give an
                                             --output of "0" it must detect a "0" from the btn at any point in time
                                  end if;

                    end if;  --This end the "if/else" condition. Remember the 2nd Main if condition is part of the first "if" because
there is no "endif"
          end if;  --This ends the first "if" condition

     end process;

end DB;
```

--clk_div (modified)---

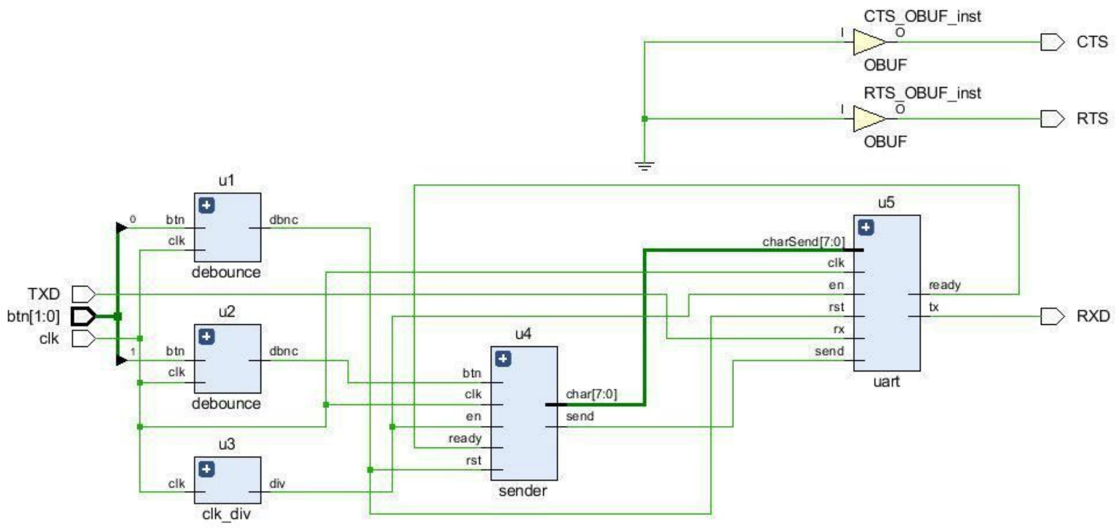
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity clk_div is
      port (
           clk : in std_logic;
           div : out std_logic
      );
end clk_div;

architecture behavioral of clk_div is

       signal count : std_logic_vector (10 downto 0) := (others => '0');     --use 11 bits to be able to count to 1085 (125MHz/115.2hz =
1085.07)

begin
     process(clk) begin        --declare process
          if rising_edge(clk) then        --check on rising edge
                if(unsigned(count) < 1085) then                --update counter while less than 1085
                      div <= '0';                                         --while less than the desired count div = '0'
                      count <= std_logic_vector( unsigned(count) + 1 );                              -- This updates counter
                else                                                     --once you reach 1085 reset counter
                      count <= (others => '0');
                      div <= '1';                                         --output 1 pulse
                end if;                                        --This ends the second "IF"
          end if;                                 --This ends the first "IF"
     end process;

end behavioral;
```

```
------------------------------------------------sender_top------------------------------------------------------- -
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sender_top is
    Port (
          TXD : in STD_LOGIC;
          btn : in STD_LOGIC_VECTOR (1 downto 0);
          clk : in STD_LOGIC;
          RXD : out STD_LOGIC;
          CTS : out STD_LOGIC;
          RTS : out STD_LOGIC);
end sender_top;


architecture Structural of sender_top is

    signal rstbtn, btn1: std_logic;
    signal div, snd, ready : STD_LOGIC;
    signal char: std_logic_vector(7 downto 0);
--------------------------------------------------------------------------

    component uart
        port (
            clk, en, send, rx, rst      : in std_logic;
            charSend                    : in std_logic_vector (7 downto 0);
            ready, tx, newChar          : out std_logic;
            charRec                     : out std_logic_vector (7 downto 0));
    end component;
--------------------------------------------------------------------------
    component debounce
        Port (
             clk : in STD_LOGIC;
             btn : in STD_LOGIC;
             debounced : out STD_LOGIC);
    end component;

--------------------------------------------------------------------------

    component clockdivider
        Port (
            clk_slow : out STD_LOGIC;
            clk : in STD_LOGIC);
    end component;

--------------------------------------------------------------------------

    component sender
```

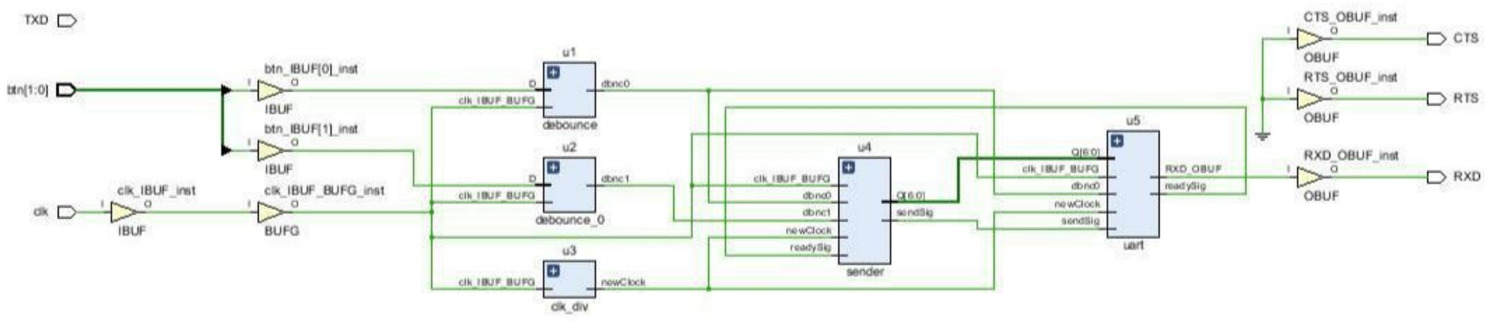
```
            Port (
                  rst : in STD_LOGIC;
                  clk : in STD_LOGIC;
                  en : in STD_LOGIC;
                  btn : in STD_LOGIC;
                  ready : in STD_LOGIC;
                  send : out STD_LOGIC;
                  char : out STD_LOGIC_VECTOR (7 downto 0));
      end component;
------------------------------------------------------------------------------


      begin                   --We start the PORT-MAPS to connect ENTITY-COMPONENT with MAIN-TEMPORARY
signals
            rts <= '0';
            cts <= '0';

            clkdiv: clockdivider port map(
                   clk=> clk,
                   clk_slow =>div);

            rstdbnc: debounce port map(
                        clk => clk,
                        btn => btn(0),
                        debounced => rstbtn);

            btndbnc:debounce port map(
                        clk => clk,
                        btn => btn(1),
                        debounced => btn1);

            snder: sender port map(
                        clk => clk,
                        btn => btn1,
                        en => div,
                        ready => ready,
                        rst => rstbtn,
                        send => snd,
                        char => char);

            u5: uart port map(
                        clk => clk,
                        en => div,
                        send => snd,
                        rx => TXD,
                        rst => rstbtn,
                        charSend => char,
                        ready => ready,
                        tx => RXD);
end Structural;
```

2.3. Implementation

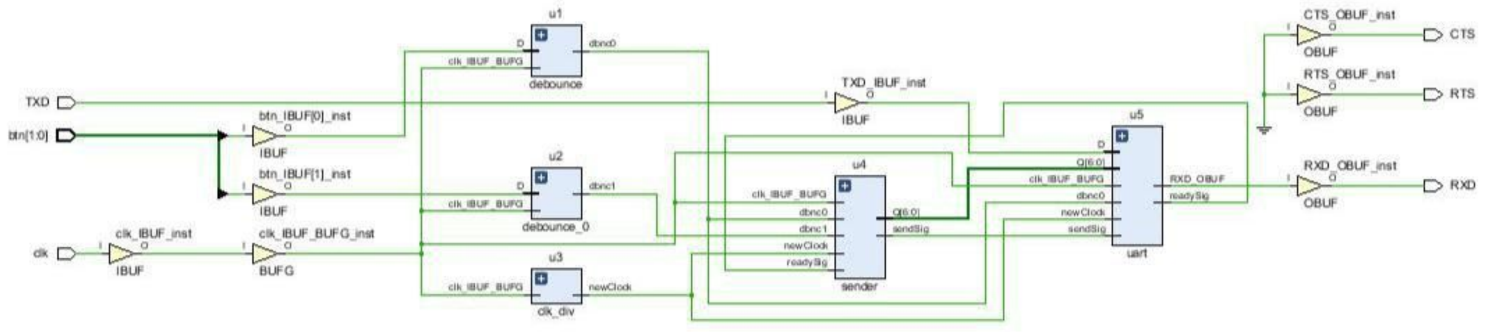
a. Elaboration Schematic



b. Synthesis Schematic

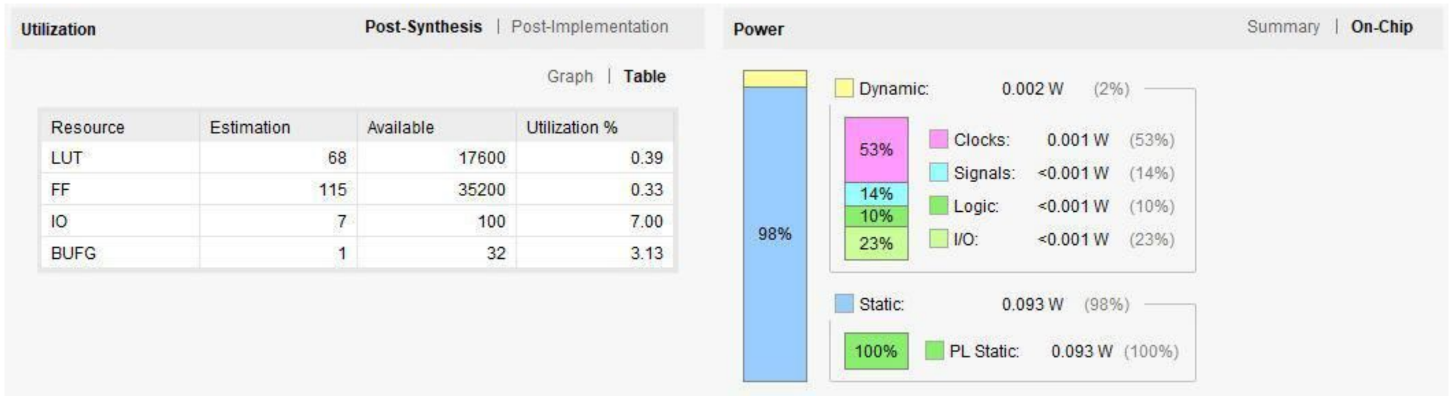


c. Implementation Schematic



d. Project Summary Images

i. Post-Synthesis Utilization table and On-chip Power Graphs



Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	68	17600	0.39
FF	115	35200	0.33
IO	7	100	7.00
BUFG	1	32	3.13

Power — Summary | **On-Chip**

Dynamic:	0.002 W	(2%)
Clocks:	0.001 W	(53%)
Signals:	<0.001 W	(14%)
Logic:	<0.001 W	(10%)
I/O:	<0.001 W	(23%)
Static:	0.093 W	(98%)
PL Static:	0.093 W	(100%)

e. XDC File

Using the provided Zybo_Master.xdc file, we uncommented "clock", two "buttons" (btn[0] and btn[1]) and pins JB1 though JB4 (T20, U20, V20, W20). The lines used are included below:

```
##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L11P_T1_SRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Buttons
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }];
#IO_L20N_T3_34 Sch=BTN0
set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }];
#IO_L24N_T3_34 Sch=BTN1
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }];
#IO_L18P_T2_34 Sch=BTN2
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }];
#IO_L7P_T1_34 Sch=BTN3


##Pmod Header JB
set_property -dict { PACKAGE_PIN T20   IOSTANDARD LVCMOS33 } [get_ports { RTS }];
#IO_L15P_T2_DQS_34 Sch=JB1_p
set_property -dict { PACKAGE_PIN U20   IOSTANDARD LVCMOS33 } [get_ports { RXD }];
#IO_L15N_T2_DQS_34 Sch=JB1_N
set_property -dict { PACKAGE_PIN V20   IOSTANDARD LVCMOS33 } [get_ports { TXD }];
#IO_L16P_T2_34 Sch=JB2_P
set_property -dict { PACKAGE_PIN W20   IOSTANDARD LVCMOS33 } [get_ports { CTS }];
#IO_L16N_T2_34 Sch=JB2_N
```

Discussion (for all parts of the Lab)

<u>Observations/Discoveries:</u>

This lab helped us to more understand the utility potential of VHDL with the Zybo board. We learned to make the Zybo board communicate with the computer using the Termite terminal. We learned to successfully implement an Finite State Machine (FSM) using VHDL. Designing a FSM with VHDL becomes simpler when being guided by a state diagram . Adding on to the learning experience, was using VHDL's capabilities for enumerated types, generic map values and, case statements to specify separate-state conditions.

<u>Questions/Follow up:</u>

I am taking wireless communication this semester and saw that transmitters can be used as MIMO, SIMO and MISO systems. However, they require the use of real and imaginary signals to be transmitted in order to use the degrees of freedom and to avoid complexity. Should wired channels like the UART face such trouble when multiple transmitters and receivers are sent?