Course Name: Embedded Systems Design

Course Number: 16:332:579

Assignment: Lab 3– Where no clock has gone before

Course Instructor: Prof. Phillip Southard

Date Submitted: 3/28/2019

Submitted by: Sanjana Devaraj (NetID: sd1049)

Contents

1 Purpose

The main purpose of this lab was to introduce the concept of a Finite State Machine (FSM) and use it to create a sequential design, a Universal Asynchronous Receiver Transmitter (UART). By using a FSM to control the behavior of a circuit in a certain sequence, an output is produced that was read by the computer and displayed in a terminal emulator.
The UART takes bytes of data and transmits the individual bits in a sequential fashion (serial manner). Serial transmission of digital data (bits) through a single wire or other medium is much more effective than parallel transmission through multiple wires because it minimizes capacitive coupling of wires. The UART transmits data asynchronously. Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. In this case, the sender and receiver must agree on timing parameters (Baud Rate) prior transmission and special bits are added to each word to synchronize the sending and receiving units (which are optional and are used depending on user requirements). Common serial communication uses the protocol shown below:

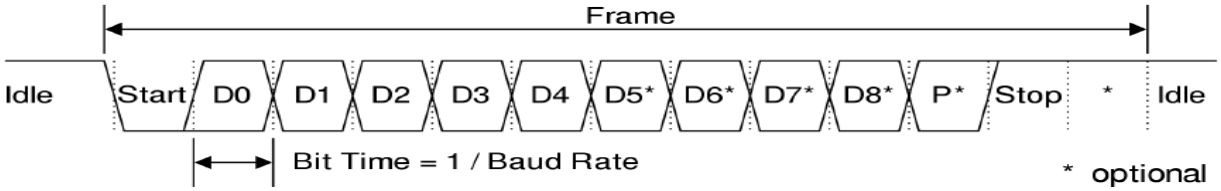
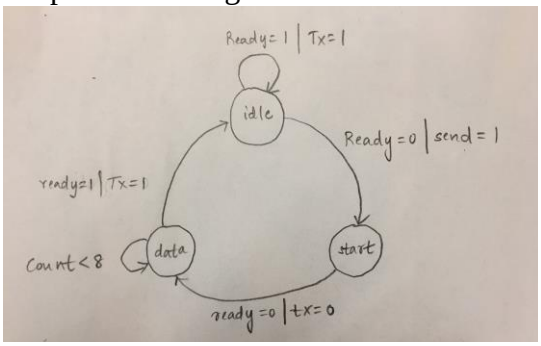


Figure 1: UART Frame Waveform

The lab is comprised of two tasks. The first task consisted of two sub-sections, design of the transmitter part of the UART that sends data using the protocol shown in Figure 1 and testing the top level UART design consisting of UART receiver, UART transmitter sub-modules with the provided UART testbench. The second consisted of two sub-sections, design of a sender module which sends data using a FSM and implementation phase consisted of instantiating the UART, sender, modified clock divider that produces a 115200 Hz output, and a pair of debouncing logic designs in the final sender top level design.

Prelab Task:

The FSM state diagram for a device that takes as input an 8-bit data packet and transmits it using the protocol in Figure 1 is shown below:



2 Lab Assignment 1: Design of UART transmitter

2.1 Theory of Operation

The UART transmitter transmits sends data using the protocol in Figure 1 with specific behavior. The signal ready is ‘1’ when it is in idle state. All internal registers are cleared and the transition happens to idle state when reset is ‘1’. The data is stored into a register and the sequence of sending data begins on the rising clock edge when send is ‘1’ and enable is ‘1’.

2.2 Design

Listing 1: sender_tx.vhd

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

entity uart_tx is
port (clk , en , send , rst : in std_logic ;
char : in std_logic_vector (7 downto 0);
ready , 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) := (others => '0');
signal d_count : std_logic_vector(3 downto 0);

begin
process(clk)
begin
if rising_edge(clk) then
   if rst = '1' then
      d <= (others => '0');
      ready <= '1';
      tx <='1';
      curr <= idle;
      d_count <= (others => '0');

   end if;

   if en = '1' then
      case curr is
      when idle =>
      ready <= '1';
```

```
      tx <= '1';
      if send = '1' then
         d <= char;
         curr <= start;
      else
         ready <= '1';
         tx <= '1';
         curr <= idle;
      end if;

       when start =>
       ready <='0';
       tx <= '0';
       curr <= data;

       when data =>
       if unsigned(d_count) < 8 then
          tx <= d(to_integer(unsigned(d_count)));
          d_count <= std_logic_vector(unsigned(d_count) + 1);
          curr <= data;
       else
           ready <= '1';
           tx <= '1';
           d_count <= (others => '0');
           curr <= idle;
       end if;

       end case;
    end if;
 end if;
end process;
end Behavioral;
```

2.3 Test

The top level UART design consisting of UART receiver, UART transmitter sub-modules is tested with the provided UART testbench.

Simulations results:

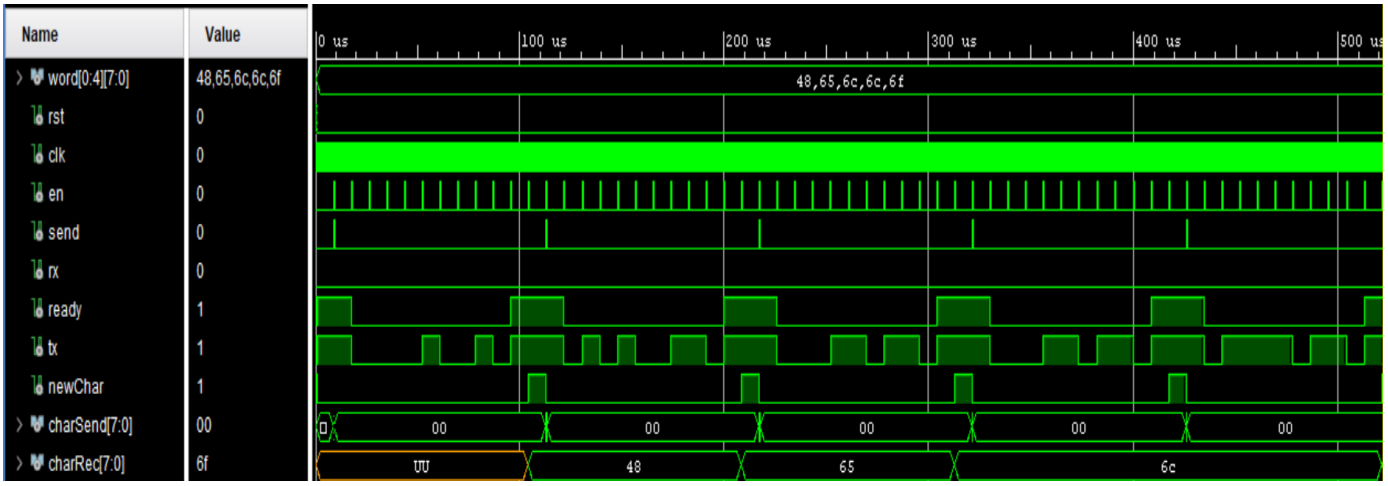


Figure 2: UART Simulation

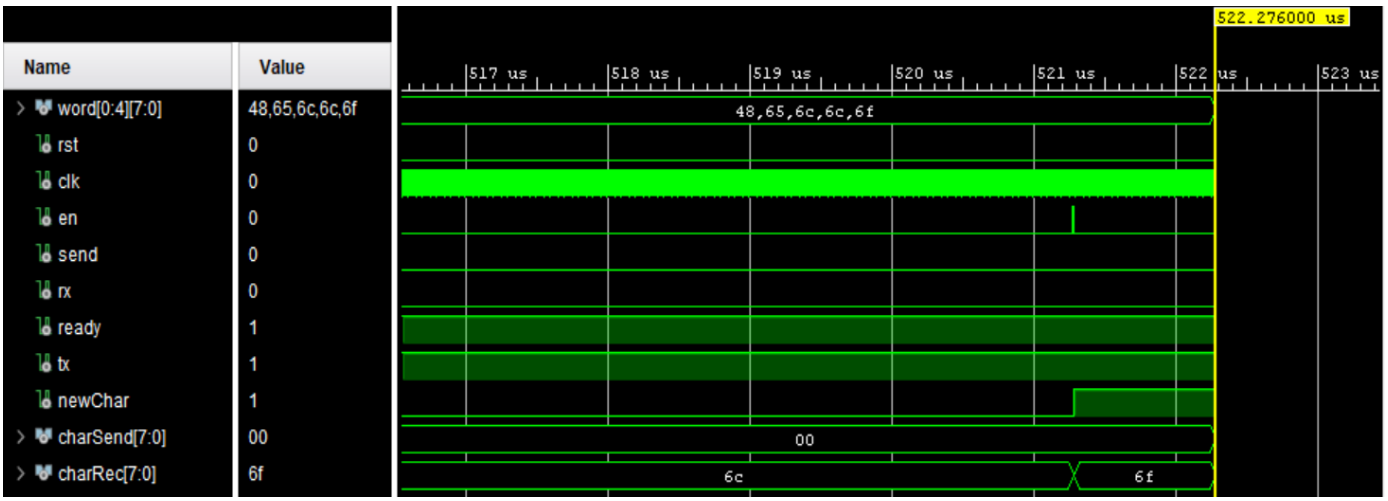


Figure 3: UART Simulation, Zoomed in

2.4 Implementation

2.4.1 UART Transmitter Design

2.4.1.1 Elaboration Schematic:

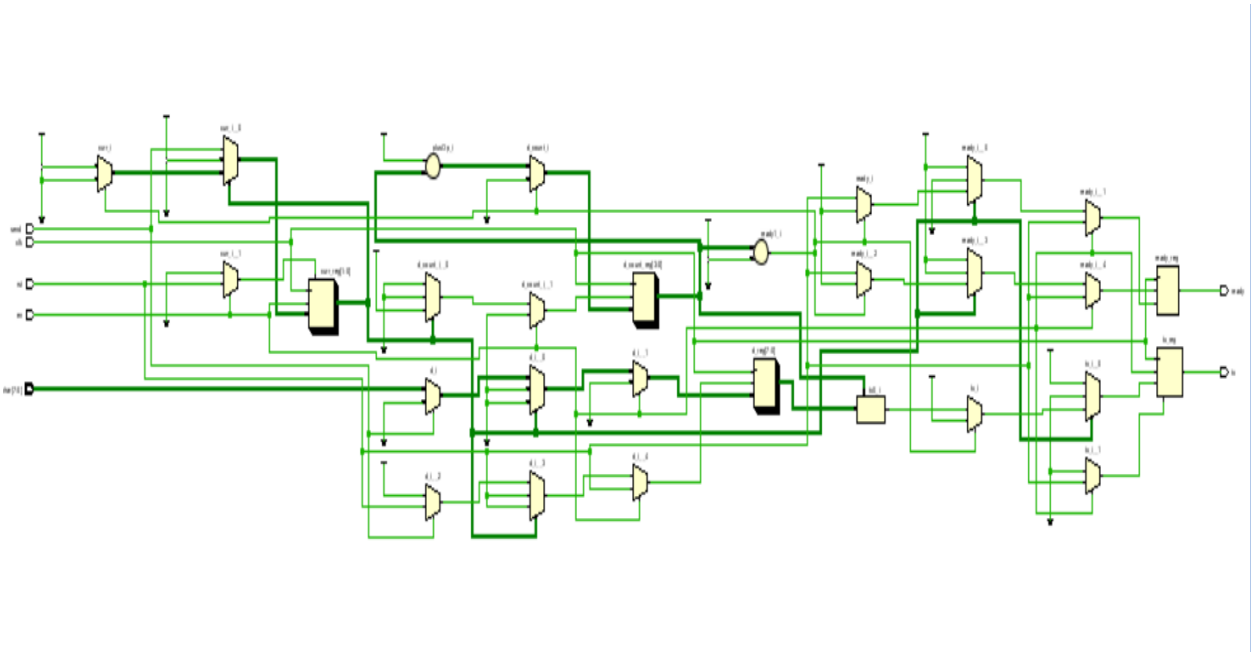


Figure 4: UART_tx RTL Schematic

2.4.1.2 Synthesis Schematic:

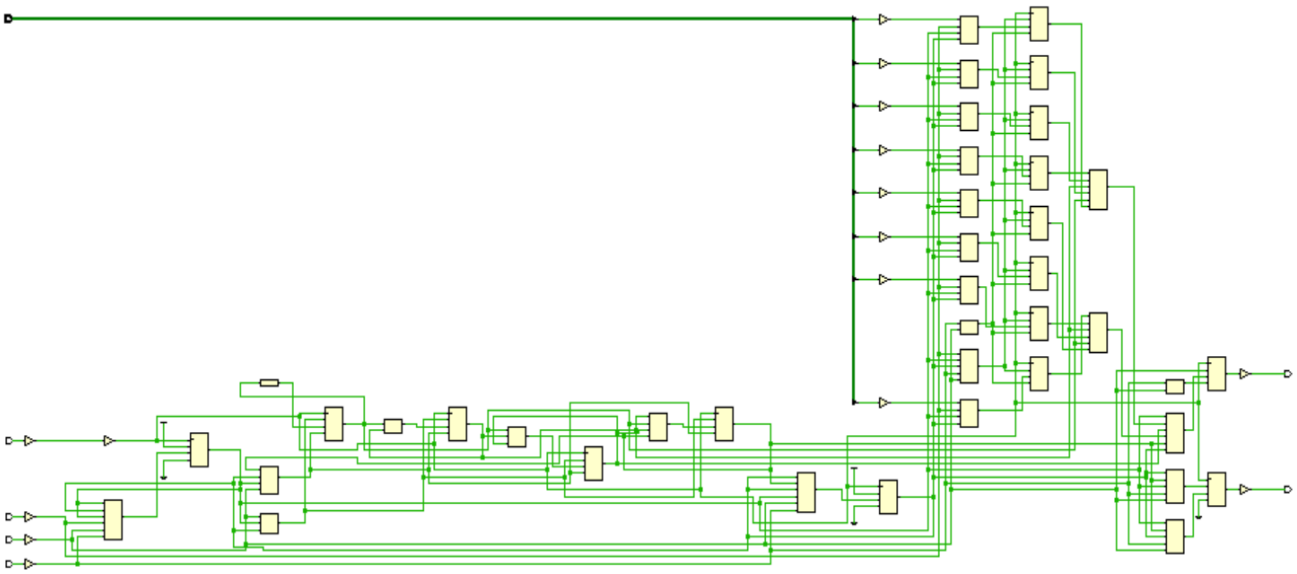


Figure 5: UART_tx Synthesis Schematic

2.4.1.3 Post-Synthesis Utilization Table:

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

Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	22	17600	0.13
FF	16	35200	0.05
IO	14	100	14.00
BUFG	1	32	3.13

Figure 6: UART_tx Post-Synthesis Utilization Table

2.4.1.4 On-chip Power Graph:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.093 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.1°C**
Thermal Margin:	58.9°C (5.0 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

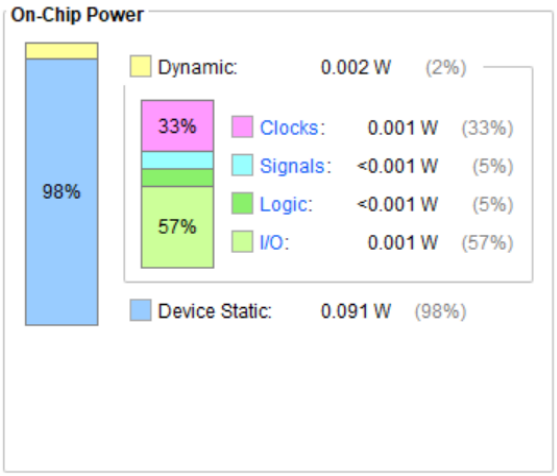


Figure 7: UART_tx On-chip Power Graph

2.4.2 Top level UART Design

2.4.2.1 Elaboration Schematic:

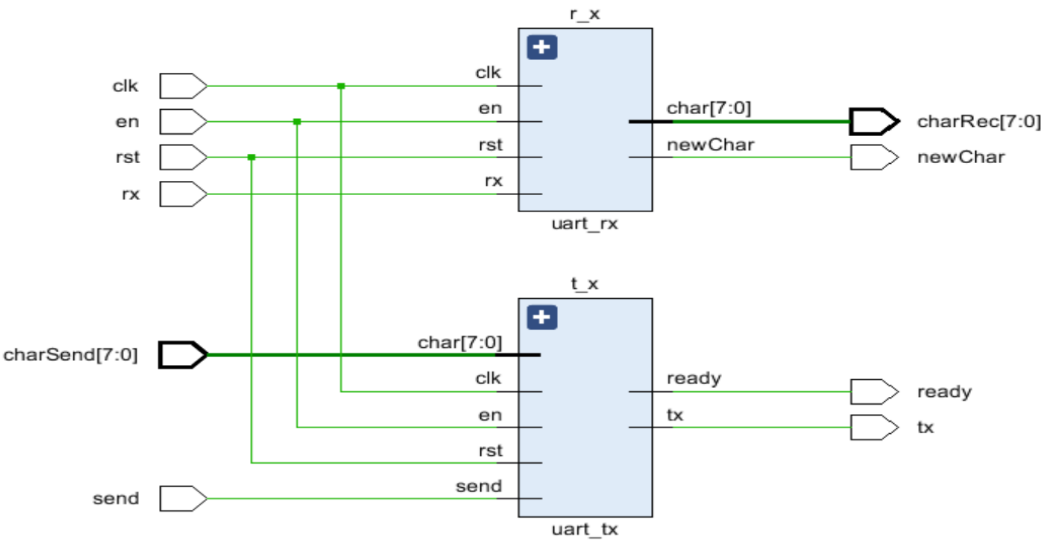


Figure 8: UART RTL Schematic

2.4.2.2 Synthesis Schematic:

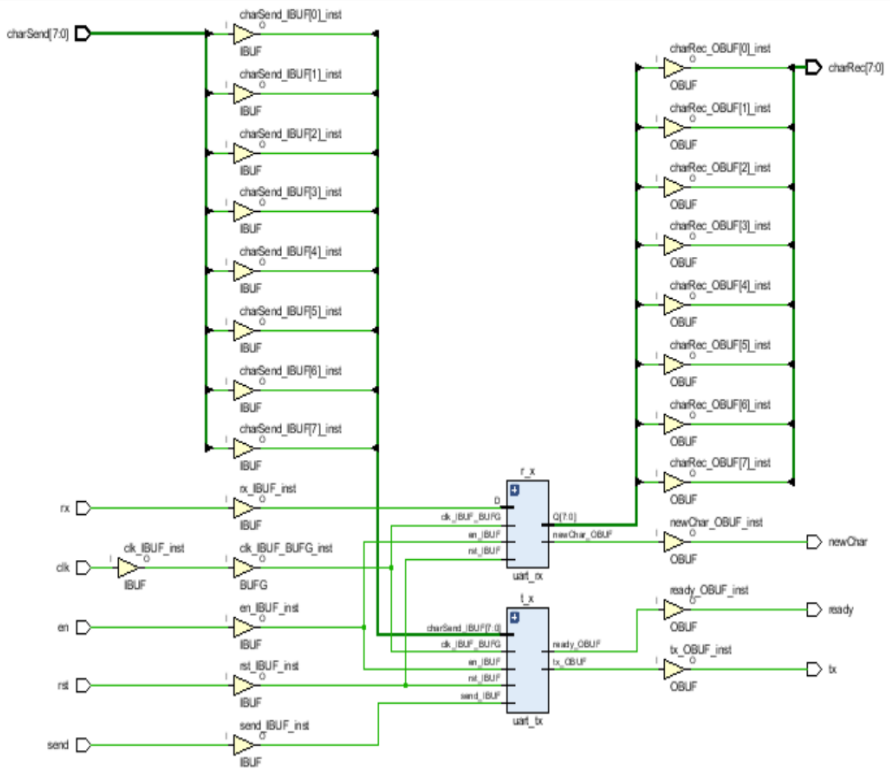


Figure 9: UART Synthesis Schematic

2.4.3.3 Post-Synthesis Utilization Table:

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

Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	27	17600	0.15
FF	43	35200	0.12
IO	24	100	24.00
BUFG	1	32	3.13

Figure 10: UART Post-Synthesis Utilization Table

2.4.3.4 On-chip Power Graph:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.094 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.1°C**
Thermal Margin:	58.9°C (5.0 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

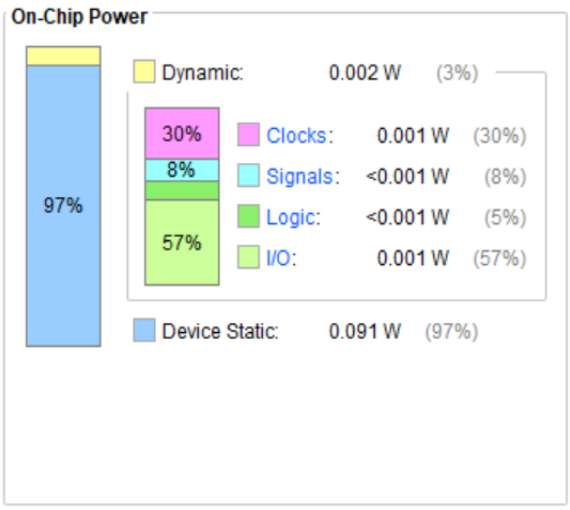


Figure 11: UART On-chip Power Graph

3 Lab Assignment 2: Sender

3.1 Theory of Operation

The sender module transmits my NetID (sd1049) which is a n-long array of 8-bit vectors where n=6 using a FSM having a specific behavior. The sender module takes as input a reset, a clock, a clock enable, a button and a ‘ready’ signal and outputs a ‘send’ signal and an 8-bit ‘char’. The FSM is initialized to an idle state and changes on the rising edge of the clock when enable is ‘1’

according to the following mentioned conditions. When a reset signal is asserted, all outputs and counter *i* are cleared and transition happens to idle state. When ready is '1', button is '1' and *i* <6, 'send' is set to '1' and NETID(i) is stored on 'char', counter *i* is incremented, and transition happens to a 'busyA' state. The counter *i* to reset to '0' and state remains in idle state when 'ready' is '1' and button is '1' but i = 6. After entering 'busyA' state, transition happens to 'busyB' state. After entering 'busyB' state, 'send' value is changed to '0' and transition happens to 'busyC' state. If 'ready' is '1' and button is '0' then transition happens to idle state, otherwise it remains in 'busyC' state.

3.2 Design

Listing 2: sender.vhd

```
21  library IEEE;
22  use IEEE.std_logic_1164.all;
23  use IEEE.numeric_std.all;
24
25  entity sender is
26  port(rst, clk, en, btn, rdy : in std_logic;
27  send : out std_logic;
28  char : out std_logic_vector(7 downto 0));
29  end sender;
30
31  architecture Behavioral of sender is
32
33  type str is array (0 to 5) of std_logic_vector(7 downto 0);
34  signal NETID : str := (X"73", X"64", X"31", X"30", X"34", X"39");
35  type state is (idle, busyA, busyB, busyC);
36  signal curr : state := idle;
37  signal i : std_logic_vector(4 downto 0);
38
39  begin
40  FSM: process(clk, en)
41       begin
42
43          if rising_edge(clk) and en = '1' then
44              if rst = '1' then
45                  char <= "00000000";
46                  i <= "00000";
47                  curr <= idle;
48                  send <= '0';
49              end if;
50          case curr is
51          when idle =>
52          if rdy = '1' and btn = '1' then
53             if unsigned(i) < 6 then
54                send <= '1';
```

```
55                  char <= netid(to_integer(unsigned(i)));
56                  i <= std_logic_vector(unsigned(i) + 1);
57                  curr <= busyA;
58               else
59                  i <= "00000";
60                  curr <= idle;
61               end if;
62            end if;
63
64            when busyA =>
65            curr <= busyB;
66
67            when busyB =>
68            send <= '0';
69            curr <= busyC;
70
71            when busyC =>
72            if rdy = '1' and btn = '0' then
73               curr <= idle;
74            end if;
75         end case;
76      end if;
77   end process;
78   end Behavioral;
```

3.3 Implementation

3.3.1 Elaboration Schematic:

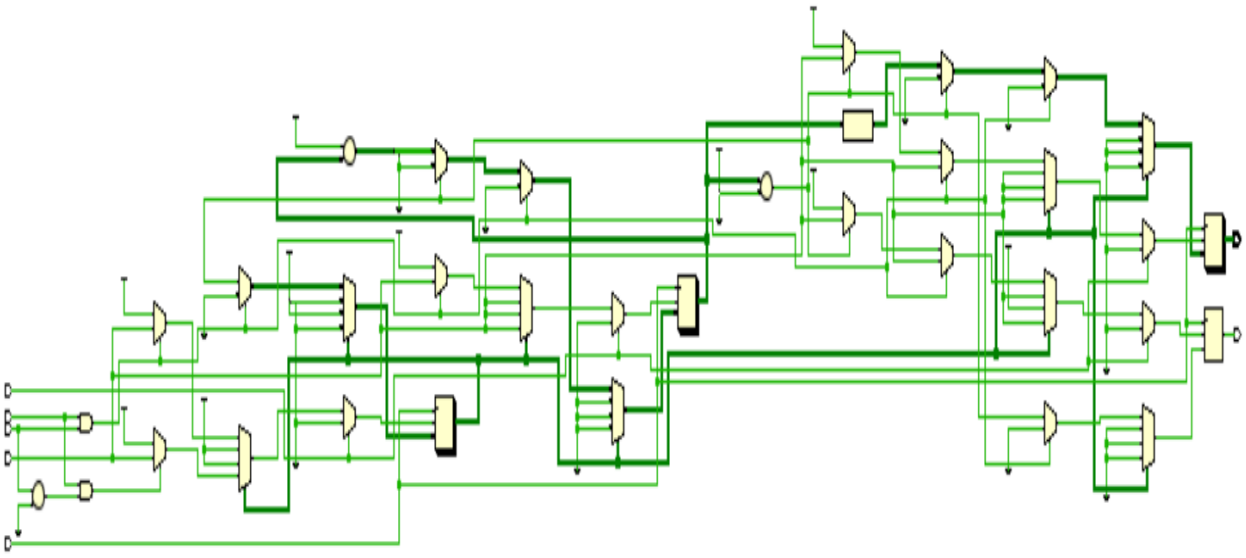


Figure 12: Sender RTL Schematic

3.3.2 Synthesis Schematic:

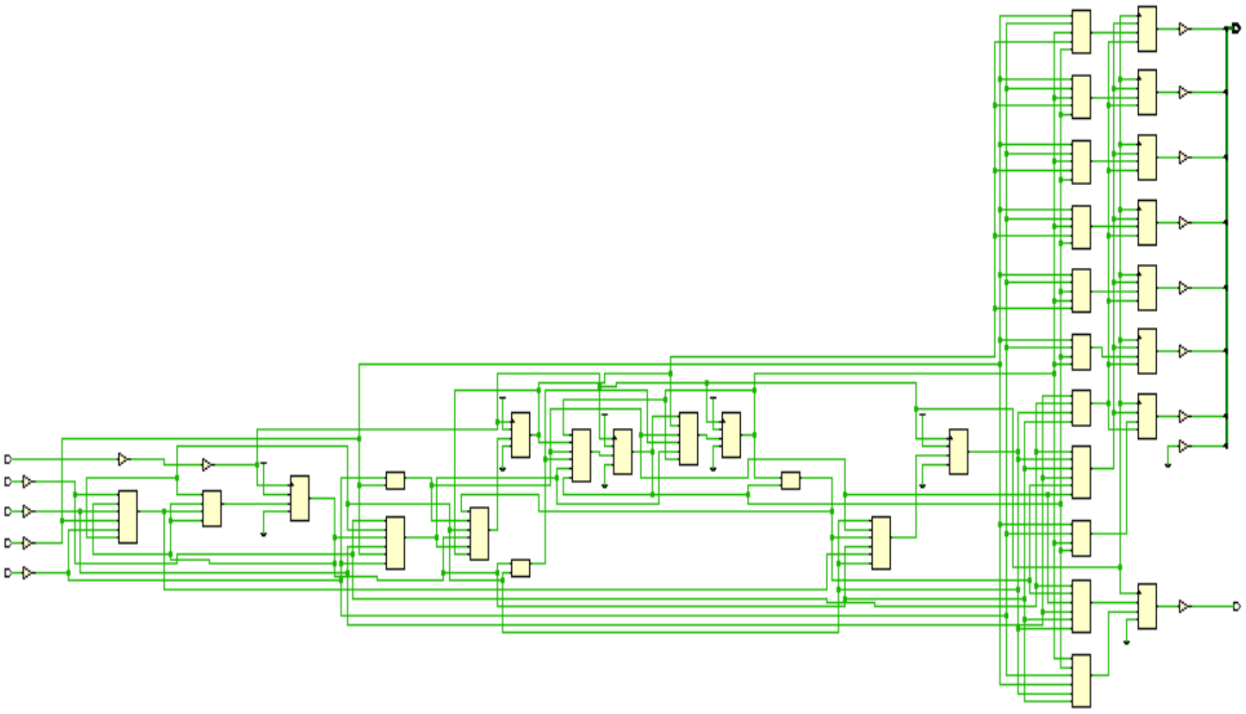


Figure 13: Sender Synthesis Schematic

3.3.3 Post-Synthesis Utilization Table:

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

Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	17	17600	0.10
FF	13	35200	0.04
IO	14	100	14.00
BUFG	1	32	3.13

Figure 14: Sender Post-Synthesis Utilization Table

3.3.4 On-chip Power Graph:

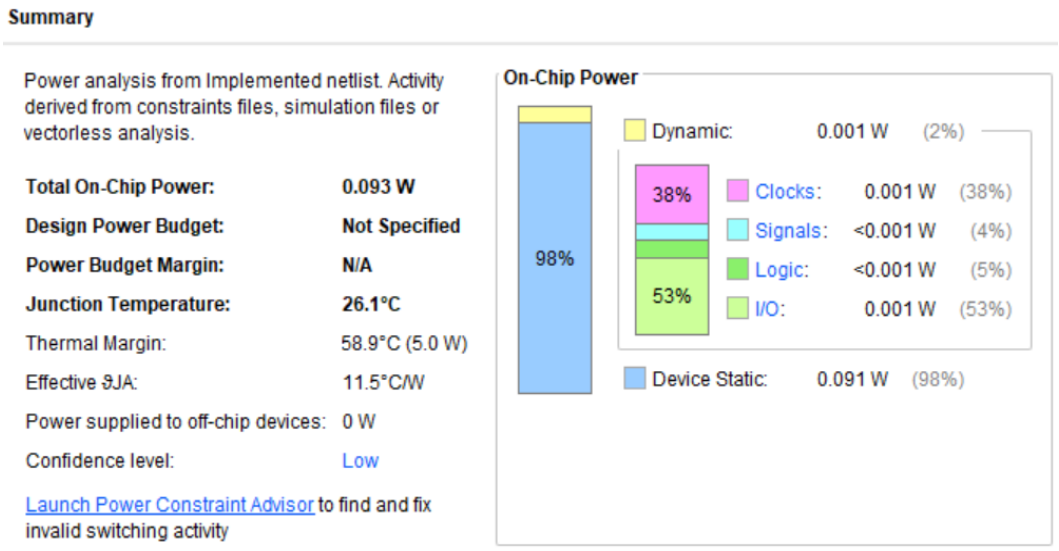


Figure 15: Sender On-chip Power Graph

4 Lab Assignment 3: Sender Top level

4.1 Theory of Operation

The designed UART, sender, modified clock divider that produces a 115200 Hz output, and a pair of debouncing logic designs are instantiated using structural VHDL modeling in this final sender top level design.

4.2 Design

4.2.1 Clock divider

The clock divider circuit produces the clock divided signal as output with a frequency of 115200 Hz which is the specified Baud Rate for this assignment, the rate at which the Transmitter sends data and the Receiver receives data.

The input clock frequency is 125 MHz.

- ⇨ $125 * 10^6 / 1085 = 115207.37$ Hz (approximately equal to the required baud rate 115200 Hz)

Listing 3: clock_div.vhd

```
22  library IEEE;
23  use IEEE.std_logic_1164.all;
24  use IEEE.numeric_std.all;
25
26  entity clk_div is
27  port (clk : in std_logic;
28    div : out std_logic);
29  end clk_div;
30
31  architecture clk_ckt of clk_div is
32  signal counter : std_logic_vector (25 downto 0) := (others => '0');
33
34  begin
35  process(clk)
36  begin
37  if rising_edge(clk) then
38     if(unsigned(counter) < 1085) then
39        div <= '0';
40        counter <= std_logic_vector( unsigned(counter) + 1 );
41     else
42        counter <= (others => '0');
43        div <= '1';
44    end if;
45  end if;
46  end process;
47
48  end clk_ckt;
```

4.2.2 Sender Top Level Design

Listing 4: top_level.vhd

```
22  library IEEE;
23  use IEEE.std_logic_1164.all;
24
25  entity top_level is
26  port(TXD, clk : in std_logic;
27  btn : in std_logic_vector(1 downto 0);
28  CTS, RTS, RXD: out std_logic);
29  end top_level;
30
31  architecture top_level_ckt of top_level is
32
33  signal dbnce: std_logic_vector(1 downto 0);
34  signal div, send, ready, tx : std_logic;
35  signal char: std_logic_vector(7 downto 0);
36
37  component debounce is
38  port( clk: in std_logic;
39  btn: in std_logic;
40  dbnc: out std_logic);
41  end component;
42
43  component clk_div
44  port( clk : in std_logic;
45  div : out std_logic);
46  end component;
47
48  component sender is
49  port( rst, clk, en, btn, rdy : in std_logic;
50  send : out std_logic;
51  char : out std_logic_vector(7 downto 0));
52  end component;
```

```
54  component uart is
55  port( clk, en, send, rx, rst : in std_logic;
56  charSend : in std_logic_vector(7 downto 0);
57  ready, tx, newChar : out std_logic;
58  charRec : out std_logic_vector (7 downto 0));
59  end component;
60
61  begin
62
63  CTS <= '0';
64  RTS <= '0';
65
66  u1: debounce
67  port map(clk => clk,
68  btn => btn(0),
69  dbnc => dbnce(0));
70
71  u2: debounce
72  port map(clk => clk,
73  btn => btn(1),
74  dbnc => dbnce(1));
75
76  u3: clk_div
77  port map(clk => clk,
78  div => div);

79
80  u4: sender
81  port map( btn => dbnce(1),
82  clk => clk,
83  en => div,
84  rdy => ready,
85  rst => dbnce(0),
86  char => char,
87  send => send);
88
89  u5: uart
90  port map( charSend => char,
91  clk => clk,
92  en => div,
93  rst => dbnce(0),
94  rx => TXD,
95  send => send,
96  ready => ready,
97  tx => RXD);
98
99  end top_level_ckt;
```

4.3 Implementation

4.3.1 Elaboration Schematic:

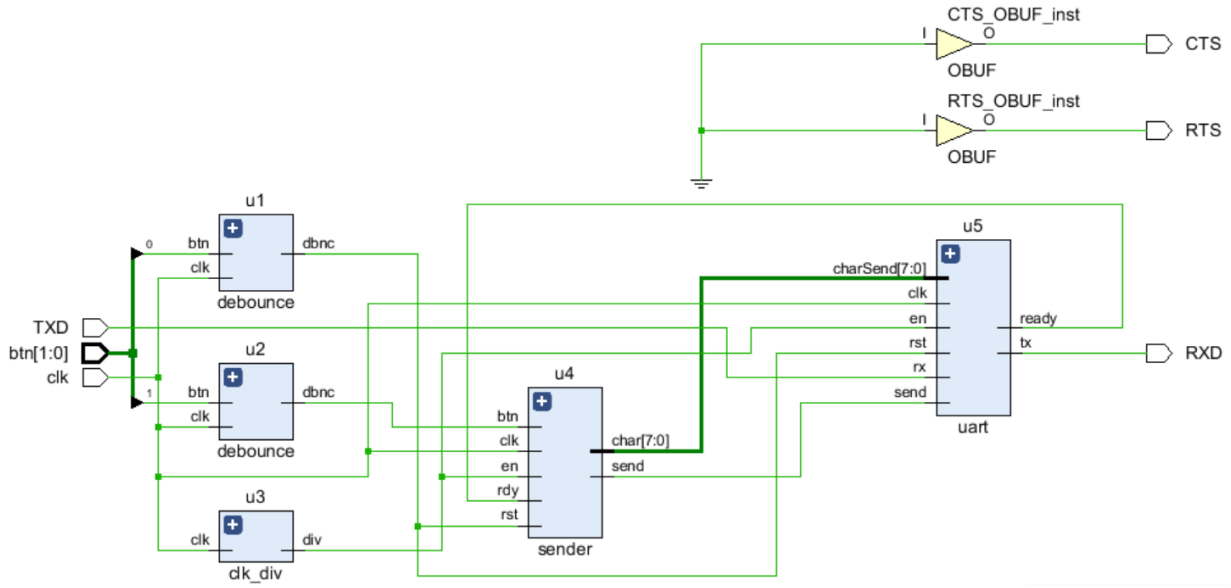


Figure 16: Sender Top Level RTL Schematic

4.3.2 Synthesis Schematic:

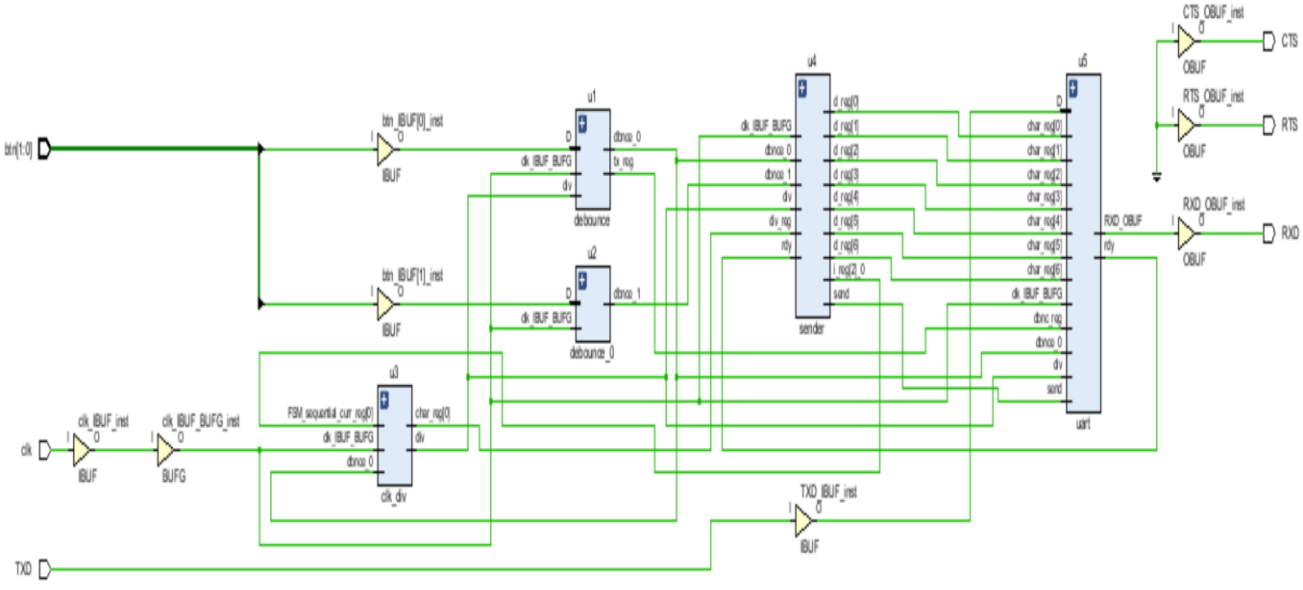
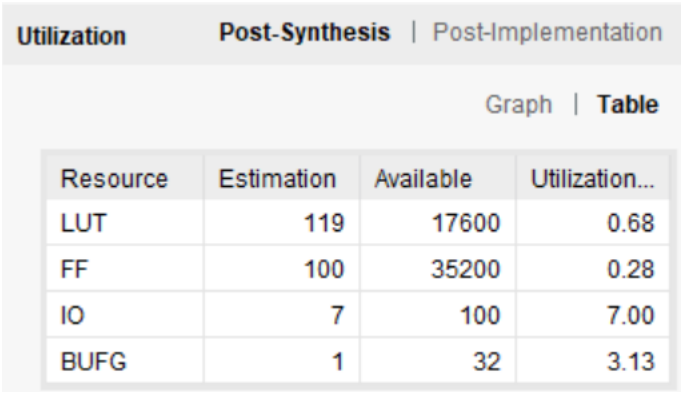


Figure 17: Sender Top Level Synthesis Schematic

4.3.3 Post-Synthesis Utilization Table:



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

Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	119	17600	0.68
FF	100	35200	0.28
IO	7	100	7.00
BUFG	1	32	3.13

Figure 18: Sender Top Level Post-Synthesis Utilization Table

4.3.4 On-chip Power Graph:

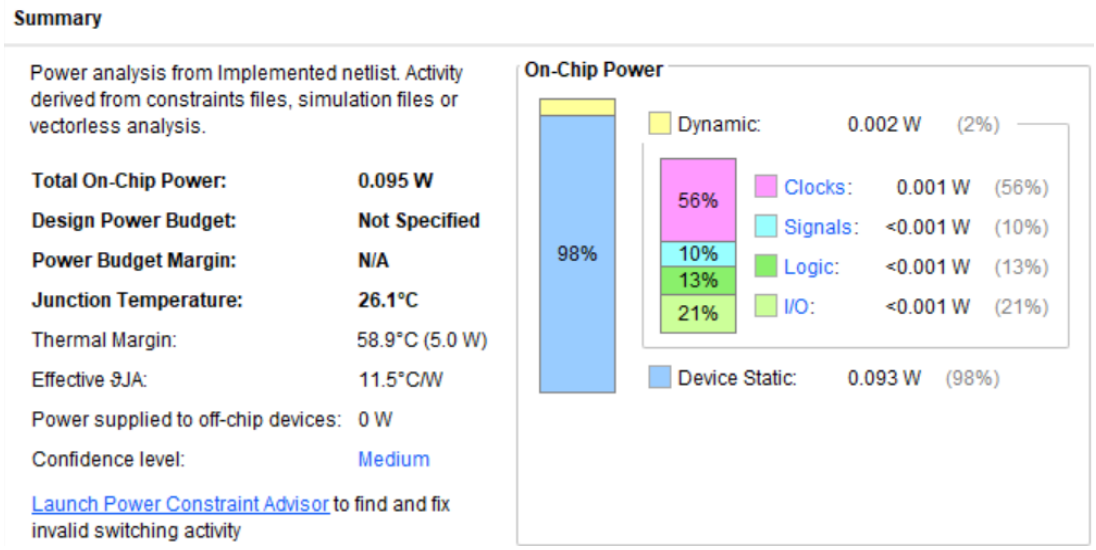


Figure 19: Sender Top Level On-chip Power Graph

4.3.5 Constraints File

The constraints (XDC) file was changed to map the appropriate signals on the board to the final sender top level design. The lines uncommented in the XDC file are shown in the below figures. A operating frequency of clock 125 MHz needs to sent to the Zybo board, 2 push buttons need to be enabled to observe the transmission of dataon the serial terminal. Button 0 is used for reset and button 1 is used for sending data in a sequential manner. The UART Pmod is connected to the Pmod header JB on the board. The lines of code corresponding to Pmod header JB was uncommented so that pins T20, U20, V20, W20 are enabled for the signals RTS, RXD, TXD, CTS respectively.

Listing 5: ZYBO_Master.xdc

```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##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 }];


##Switches
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];  #IO_L24P_T3_34 Sch=SW1
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##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
#set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports { jb_p[2] }]; #IO_L17P_T2_34 Sch=JB3_P
#set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports { jb_n[2] }]; #IO_L17N_T2_34 Sch=JB3_N
#set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports { jb_p[3] }]; #IO_L22P_T3_34 Sch=JB4_P
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports { jb_n[3] }]; #IO_L22N_T3_34 Sch=JB4_N
```

4.3.6 Results:

The serial terminal ‘Termite’ settings are shown below:

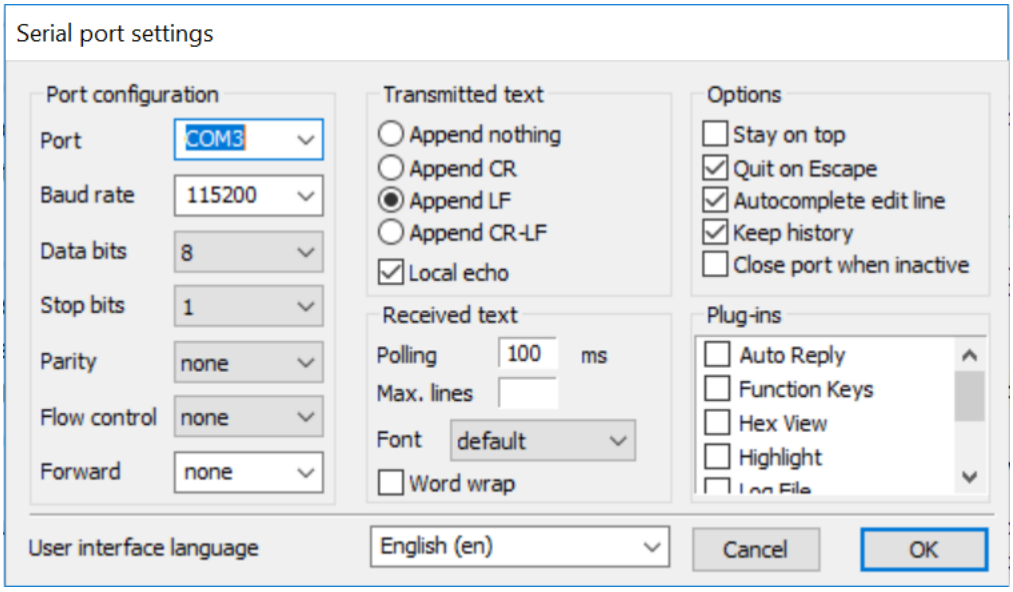


Figure 20: Termite Settings

When button 1 is pressed 6 times consecutively, the output is as shown in the below figure:

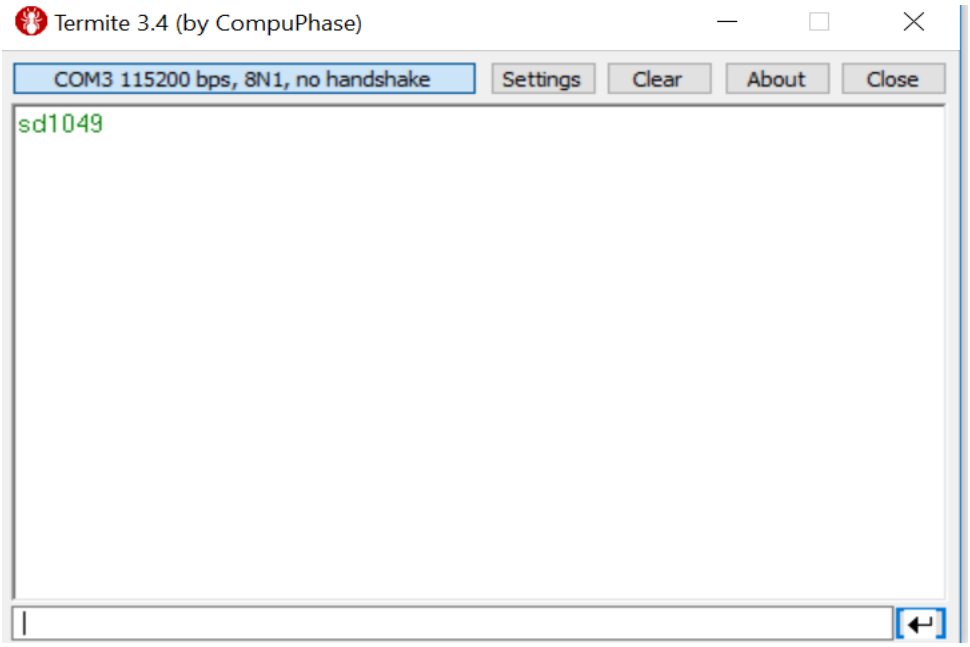


Figure 21: Output

5 Discussion

5.1 Observations

This lab assignment helped to understand the concept of FSM, serial communication, the working of UART in depth. My concepts of FSM are much clear now after using FSM in a practical application such as a hardware device like UART.

5.2 Problem areas

I spent long time in the first task getting the UART transmitter part correct. I am getting better at debugging by running the testbench code to observe the simulation results and tracing back to where I went wrong. My problem was that when I ran the UART module with the testbench the first byte which was received was ‘ff’ instead of ‘48’. I initially though it was a problem with the testbench because the time period for which the ready and tx signal were high was longer than the other times these signals were high as shown in the figure below because of which the received byte was ‘ff’. This problem was corrected by initializing the data counter variable ‘d_count’ to zero when reset is high.

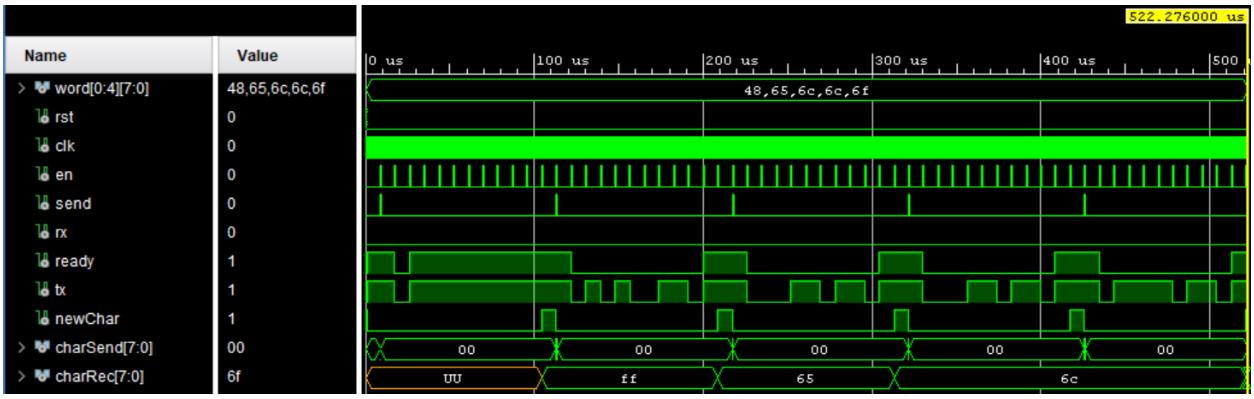


Figure 22: Simulation of UART with incorrect first byte

The sender FSM design and sender top level design were fairly simple.