Course Name: Embedded Systems Design

Course Number: 14 : 332 : 493 : 03

Assignment: Lab 3 - Where No Clock Has Gone Before

Instructor: Southard

Date Submitted: 3/28/19

Submitted By: Fareen Pourmoussavian 165-00-7750

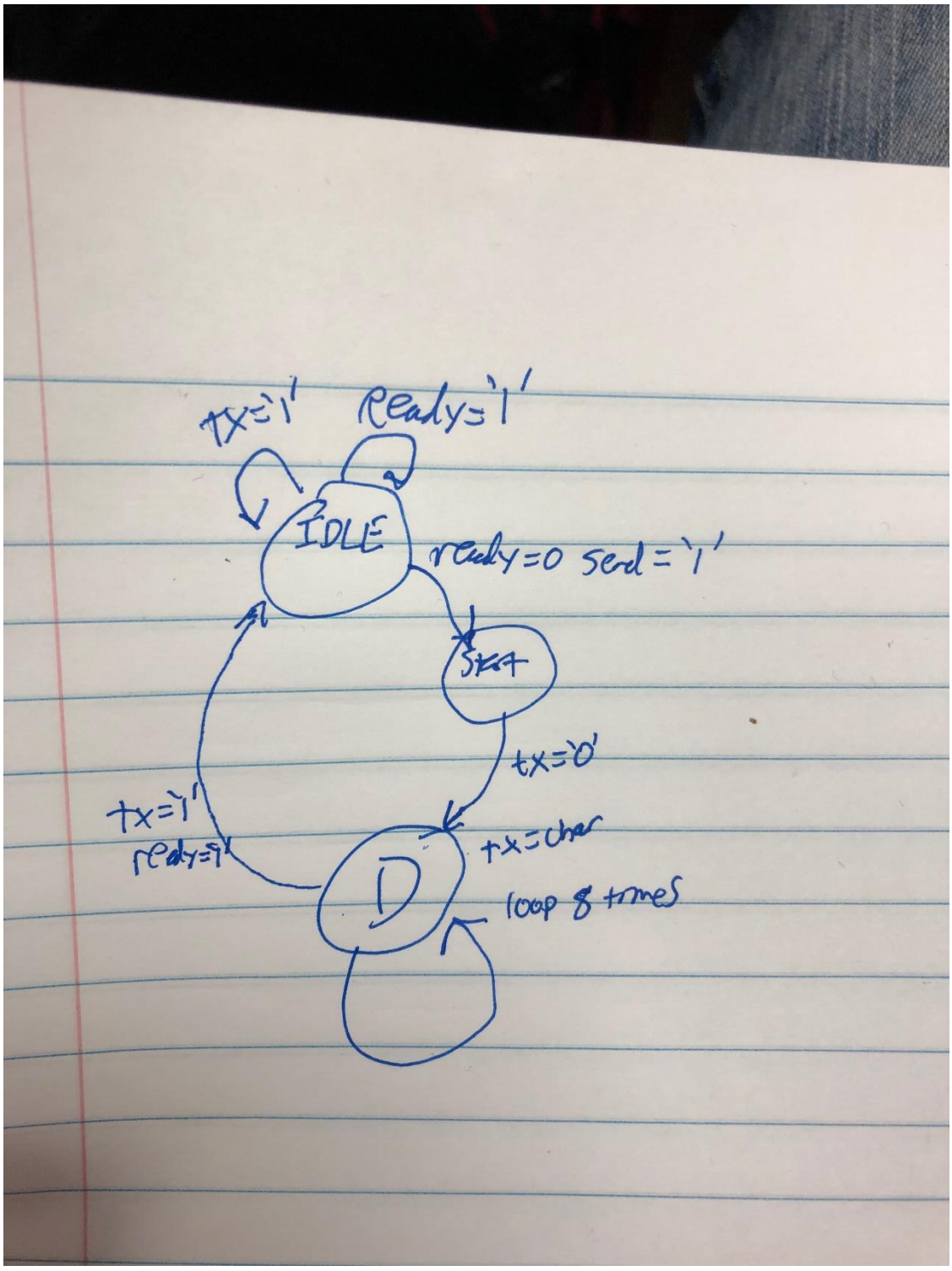
Electrical and Computer Engineering Department

School of Engineering

Rutgers University, Piscataway, NJ 08854

Contents

Prelab


tx='1' ready='1'
IDLE
ready=0 send='1'
START
tx='0'
tx='1'
ready='1'
tx=char
D
loop 8 times

1 Purpose

In this lab, we are introduced to the concept of a Finite State Machine (FSM) and we will use the FSM to create a common sequential design, a Universal Asynchronous Receiver Transmitter (UART). By using an FSM to control the behavior of a circuit in a certain sequence, we will be able to produce output that can be read by a computer and displayed in a terminal emulator.

2 Lab Assignment 1: We Can Rebuild Him

2.1 Theory of Operation

For data transmission, instead of sending multiple bits at once, one bit is sent at a time. This allows us to avoid using thousands of high frequency operating wires which suffer from a large amount of cross-talk and enables us to operate at higher transmission speeds. We do this by utilizing a UART and only have one wire for transmitting and one wire for receiving. To ensure things are sent and received in a consistent manner, the clock frequency must be the same for both wires. Therefore, it is asynchronous because both ends are operating on different clock domains.

For the UART, While the line is held high, no data is being transferred and the line is in an idle state. Once the line is low, that signifies the start of the data transmission. For the next data bits, clock cycles, we sample the line and store the char into a shift register (LSB is transmitted first). After data bits cycles, an optional parity bit is sent (usually XOR of the data bits). Finally, stop bits of logic high are sent to signify the end of the transmission.

2.2 Design

UART_tx

```
1  Library IEEE;
2  use ieee.std_logic_1164.all,ieee.numeric_std.all;
3
4  entity uart_tx is
5  port(clk,en,send,rst :in std_logic;
6       char :in std_logic_vector(7 downto 0);
7       ready,tx :out std_logic);
8  end uart_tx;
9
10 architecture uart_test of uart_tx is
11 type state_type is (IDLE,START,D);
12 signal P : state_type :=IDLE;
13 signal creg : std_logic_vector(7 downto 0);
14 signal count : std_logic_vector(2 downto 0);
15 begin
16     sync: process(clk)
17     begin
18     if(rising_edge(clk)) then
19         if(rst='1') then
20             P<=IDLE;
21             tx<='1';
22             creg<="00000000";
23             count<="000";
24             ready<='1';
25         elsif(en='1') then
26         case P is
27             When IDLE =>
28                if(send='1') then
29                     tx<='0';
30                     ready<='0';
31                     creg<=char;
32                     P<=start;
33                else
34                 ready<='1';
35                 tx<='1';
36               end if;
37             when START =>
38                count<="000";
39                tx<=creg(0);
40                creg<='0' & creg(7 downto 1);
41                P<=D;
42             when D=>
43                if (unsigned(count)<7) then
44                     tx<=creg(0);
45                        count<=std_logic_vector(unsigned(count)+1);
46                     P<=D;
47                     creg<='0' & creg(7 downto 1);
48                else
49                     tx<='1';
50                     P<=IDLE;
51                end if;
52
53      when others=>
54             P<=IDLE;
55     end case;
56     end if;
57     end if;
58     end process sync;
59 end architecture;
60
61
```

2.3 Test

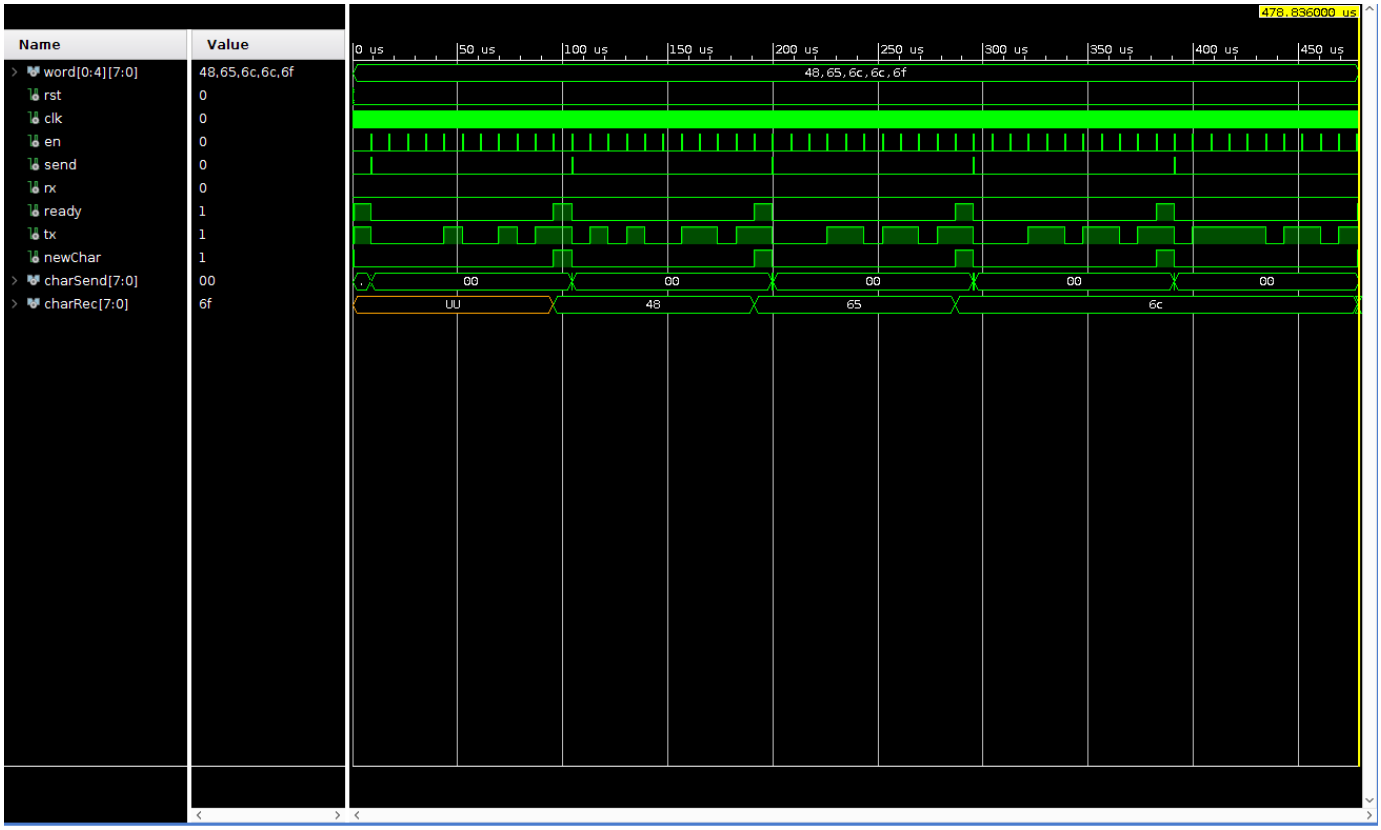
2.3.1 Test Bench Code, UART

```
7   library IEEE;
8   use IEEE.std_logic_1164.all;
9   use ieee.numeric_std.all;
10
11  entity uart_tb is
12  end uart_tb;
13
14  architecture tb of uart_tb is
15
16      component uart port (
17      clk, en, send, rx, rst  : in std_logic;
18      charSend                : in std_logic_vector(7 downto 0);
19      ready, tx, newChar      : out std_logic;
20      charRec                 : out std_logic_vector(7 downto 0)
21      );
22      end component;
23
24      type str is array (0 to 4) of std_logic_vector(7 downto 0);
25      signal word : str := (x"48", x"65", x"6C", x"6C", x"6F");
26
27      signal rst : std_logic := '0';
28      signal clk, en, send, rx, ready, tx, newChar : std_logic := '0';
29      signal charSend, charRec : std_logic_vector(7 downto 0) := (others => '0');
30
31  begin
32
33      -- the sender UART
34      dut: uart port map(
35          clk => clk,
36          en => en,
37          send => send,
38          rx => tx,
39          rst => rst,
40          charSend => charSend,
41          ready => ready,
42          tx => tx,
43          newChar => newChar,
44          charRec => charRec);
45
46
47      -- clock process @125 MHz
48      process begin
49          clk <= '0';
50          wait for 4 ns;
51          clk <= '1';
52          wait for 4 ns;
53      end process;
54
55      -- en process @ 125 MHz / 1085 = ~115200 Hz
56      process begin
57          en <= '0';
58          wait for 8680 ns;
59          en <= '1';
```

```
60          wait for 8 ns;
61      end process;
62
63      -- signal stimulation process
64      process begin
65
66          rst <= '1';
67          wait for 100 ns;
68          rst <= '0';
69          wait for 100 ns;
70
71          for index in 0 to 4 loop
72              wait until ready = '1' and en = '1';
73              charSend <= word(index);
74              send <= '1';
75              wait for 200 ns;
76              charSend <= (others => '0');
77              send <= '0';
78              wait until ready = '1' and en = '1' and newChar = '1';
79
80              if charRec /= word(index) then
81                  report "Send/Receive MISMATCH at time: " & time'image(now) &
82                  lf & "expected: " &
83                  integer'image(to_integer(unsigned(word(index)))) &
84                  lf & "received: " & integer'image(to_integer(unsigned(charRec)))
85                  severity ERROR;
86              end if;
87
88          end loop;
89
90          wait for 1000 ns;
91          report "End of testbench" severity FAILURE;
92
93      end process;
94
95  end tb;
96
```

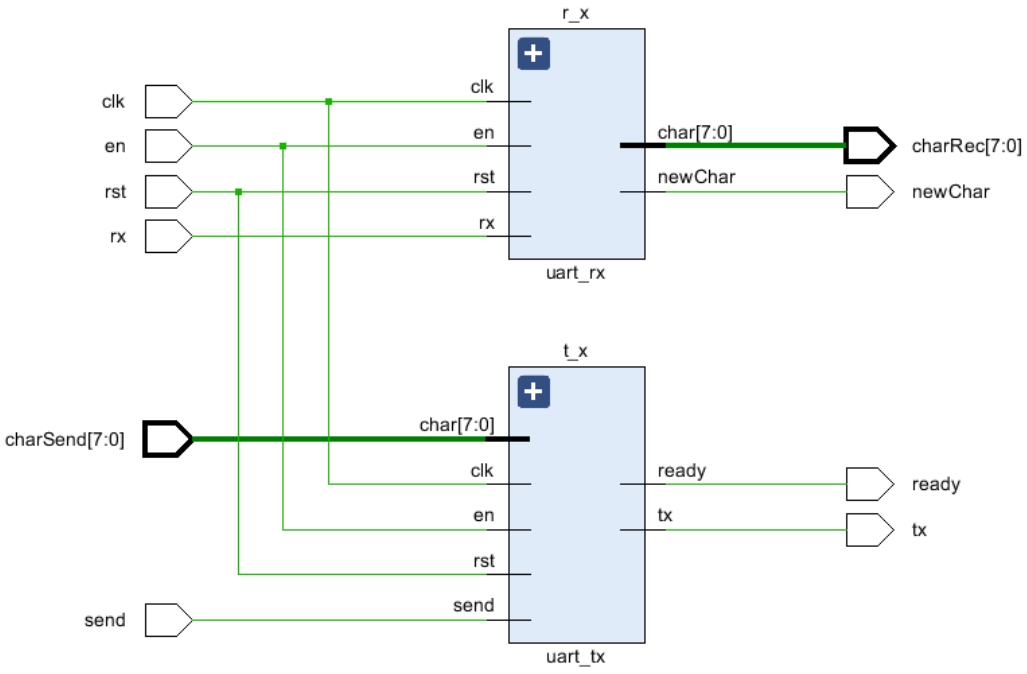
2.3.2 Simulation Results, UART

Ran for 5ms(exits early)

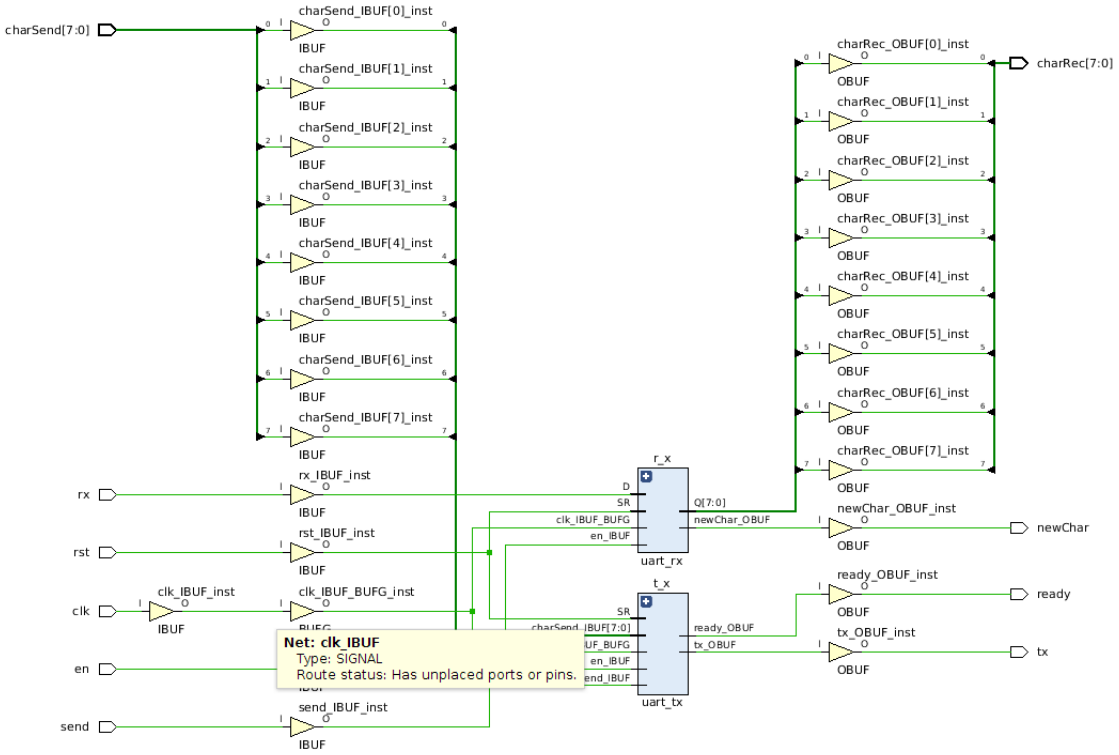


2.4 Implementation

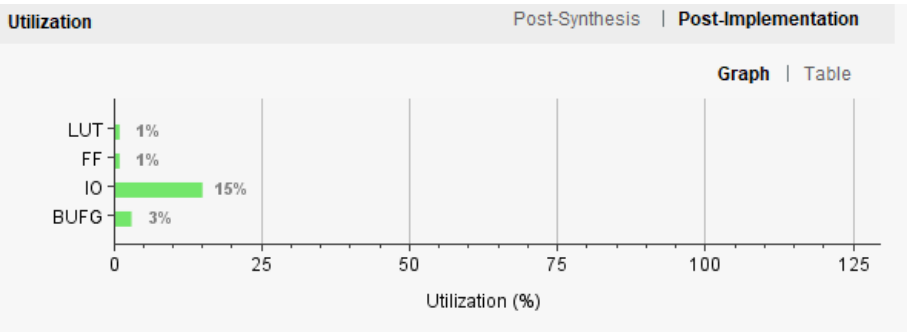
2.4.1 Elaboration Schematic

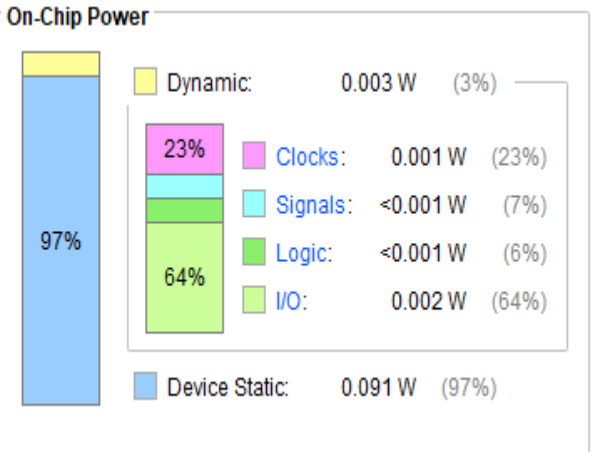


2.4.2 Synthesis Schematic



2.4.3 Project Summary





2.4.4 Constraints

2.4.4 Constraints File

The ‘clk’ line of the document had to be uncommented. This was so that Vivado was able to extract pin information from the file. The rest of the lines were deleted, since they weren’t needed.

```
## 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 }];

```

3 Lab Assignment 2: We have the technology

3.1 Theory of Operation

In order to use the UART and talk to the computer, we need to drive some output through it by sending bytes. In order to do so, an FSM will be created which will drive the ASCII codes for my NetID along with a control signal to the UART. To have our UART work with the zybo board, we will need to use a USB UART pmod.

3.2 Design

3.2.1

Clk_div

```
1  library ieee;
2  use ieee.std_logic_1164.all, ieee.numeric_std.all;
3
4  entity clk_div is
5  port(clk :in std_logic;
6      div :out std_logic);
7  end clk_div;
8
9  architecture clk of clk_div is
10 signal counter : std_logic_vector (25 downto 0) := (others => '0');
11 begin
12     process(clk)
13     begin
14         if(rising_edge(clk)) then
15             if(unsigned(counter)<1085) then
16                 counter <= std_logic_vector(unsigned(counter)+1);
17                 div <= '1';
18             else
19                 counter <= (others => '0');
20                 div <= '0';
21             end if;
22               if (unsigned(counter) = 543) then
23                  div <= '1';
24
25               else
26                  div <= '0';
27
28               end if;
29
30       end if;
31     end process;
32 end clk;
```

Debouncer

```
1  library IEEE;
2  use ieee.std_logic_1164.all, ieee.numeric_std.all;
3
4  entity debounce is
5  port (clk : in std_logic;
6        btn : in std_logic;
7        dbnc : out std_logic);
8  end debounce;
9
10 architecture deb of debounce is
11 signal counter : std_logic_vector(22 downto 0);
12 signal btn_reg : std_logic_vector(1 downto 0);
13 begin
14
15     process(clk)
16     begin
17     if(rising_edge(clk)) then
18            btn_reg(0)<=btn; --bit 0 gets 1
19            btn_reg(1)<=btn_reg(0);  --update bit 1 with bit 0 value
20             if(btn_reg(1)='1') then --if bit 1 is a 1 increment counter
21                if(unsigned(counter)<2500000) then
22                   counter<=std_logic_vector(unsigned(counter)+1);
23                     dbnc<='0';
24                 else
25                     dbnc<='1';
26                 end if;
27             else--btn(1)==0
28                 counter<=(others=>'0');
29                 dbnc<='0';
30             end if;
31          end if; --end if button pressed or not
32     end process;
33 end deb;
```

Sender

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity sender is
port(btn,clk,en,rdy,rst : in std_logic;
     char : out std_logic_vector(7 downto 0);
     send : out std_logic);
end sender;

architecture send_test of sender is
type state_type is (IDLE,busyA,busyB,busyC);
signal N : state_type;
signal P: state_type :=IDLE; --initialized to IDLE
signal i : std_logic_vector(2 downto 0); --counts up to 5 cause FP183 is 5 characters
type id is array(0 to 4) of std_logic_vector(7 downto 0);
signal NETID : id := (X"66", X"70", X"31", X"38", X"33");
begin

    sync: process(clk,en)
    begin
        if(rising_edge(clk) and en='1') then
            if(rst='1') then
                P<=IDLE;
                i<="000";
                char<="00000000";
                send<='0';
             end if;
             case P is
                when IDLE=>
                    if(rdy='1' and btn='1' and unsigned(i)<5) then
                       send<='1';
                       char<=NETID(to_integer(unsigned(i)));
                       i<=std_logic_vector(unsigned(i)+1);
                       P<=busyA;
                    elsif(rdy='1' and btn='1' and unsigned(i)=5) then
                        i<="000";
                        P<=IDLE;
                    end if;
                When busyA=>
                    P<=busyB;
                When busyB=>
                    send<='0';
                    P<=busyC;
                When busyC=>
                    if(rdy='1' and btn='0') then
                        P<=IDLE;
                    end if;
            end case;
         end if;
    end process sync;
    end send_test;
```

Top_level

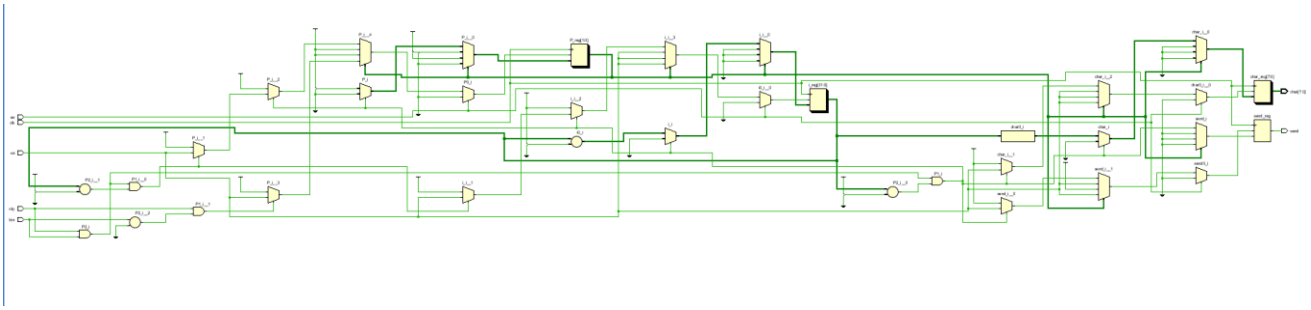
```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity top_lev is
port(TXD,clk: in std_logic;
     btn : in std_logic_vector(1 downto 0);
     RXD,RTS,CTS : out std_logic);
end top_lev;

architecture top_lev_crt of top_lev is
component clk_div
    port(clk :in std_logic;
         div :out std_logic);
end component;

component debounce
    port(clk : in std_logic;
         btn : in std_logic;
         dbnc : out std_logic);
end component;

component sender
    port(btn,clk,en,rdy,rst : in std_logic;
         char : out std_logic_vector(7 downto 0);
         send : out std_logic);
end component;

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;

signal dbout1,dbout2,clk_out,red_out,send_out : std_logic;
signal char_out : std_logic_vector(7 downto 0);

begin

dbnc1 : debounce
    port map(clk=>clk,
             btn=>btn(0),
             dbnc=>dbout1);

dbnc2 : debounce
     port map(clk=>clk,
              btn=>btn(1),
              dbnc=>dbout2);
clk_divider : clk_div
    port map(clk=>clk,
             div=>clk_out);
```

```
52              div=>clk_out);
53
54  send : sender
55  port map(btn=>dbout2,
56           clk=>clk,
57           en=>clk_out,
58           rdy=>red_out,
59           rst=>dbout1,
60           char=>char_out,
61           send=>send_out);
62  uart_blk : UART
63  port map(charSend=>char_out,
64           clk=>clk,
65           en=>clk_out,
66           rst=>dbout1,
67           rx=>TXD,
68           send=>send_out,
69           ready=>red_out,
70           tx=>RXD);
71  cts<='0';rts<='0'; --not sure what to set them to
72  end top_lev_crt;
73
```

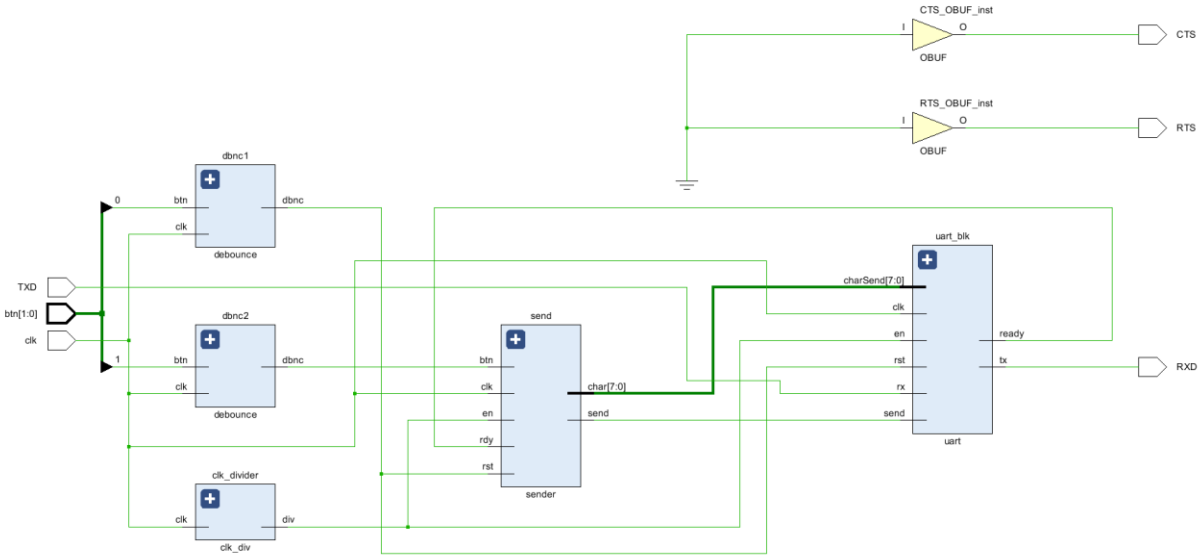
3.3 Implementation

Elaboration Schematic

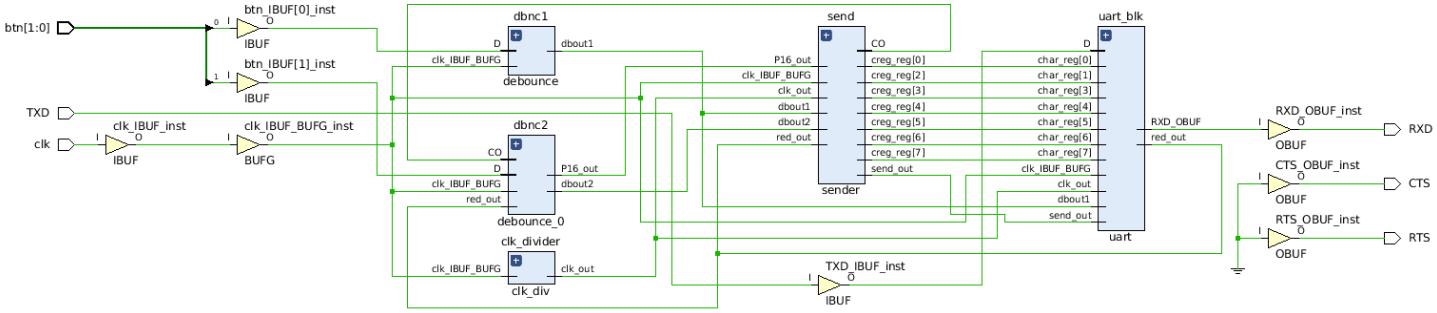
3.3.1 Sender



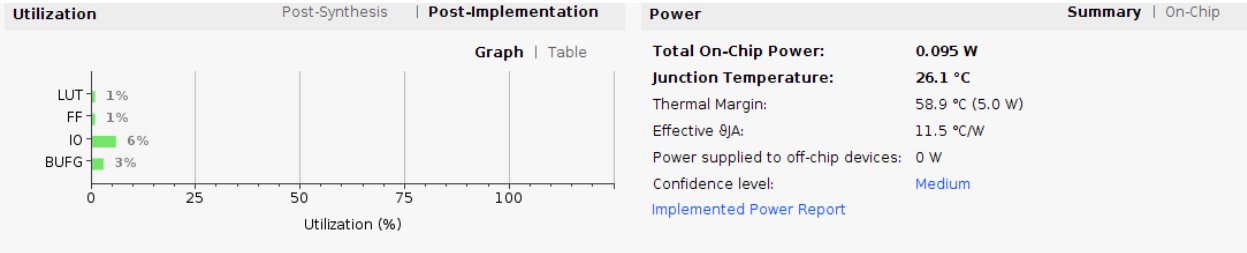
3.3.1 Top Level

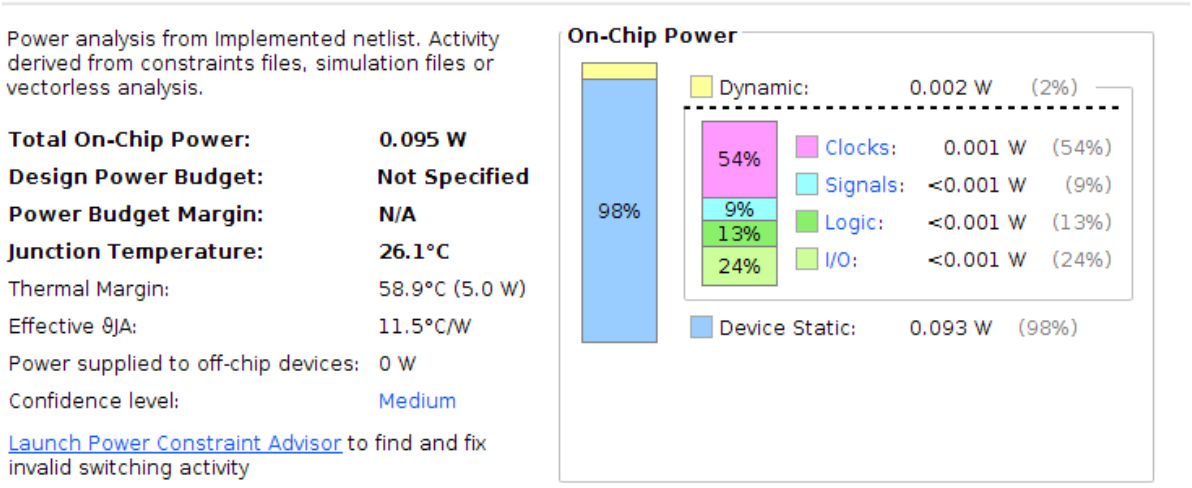


3.3.2 Synthesis Schematic



3.3.3 Project Summary





3.3.4 Constraints File

I had to uncomment the clk, btn[1] and btn[0] as well as the first 4 JB pins and rename those parameters to the ones that correspond to the ones listed in the lab 3 manual