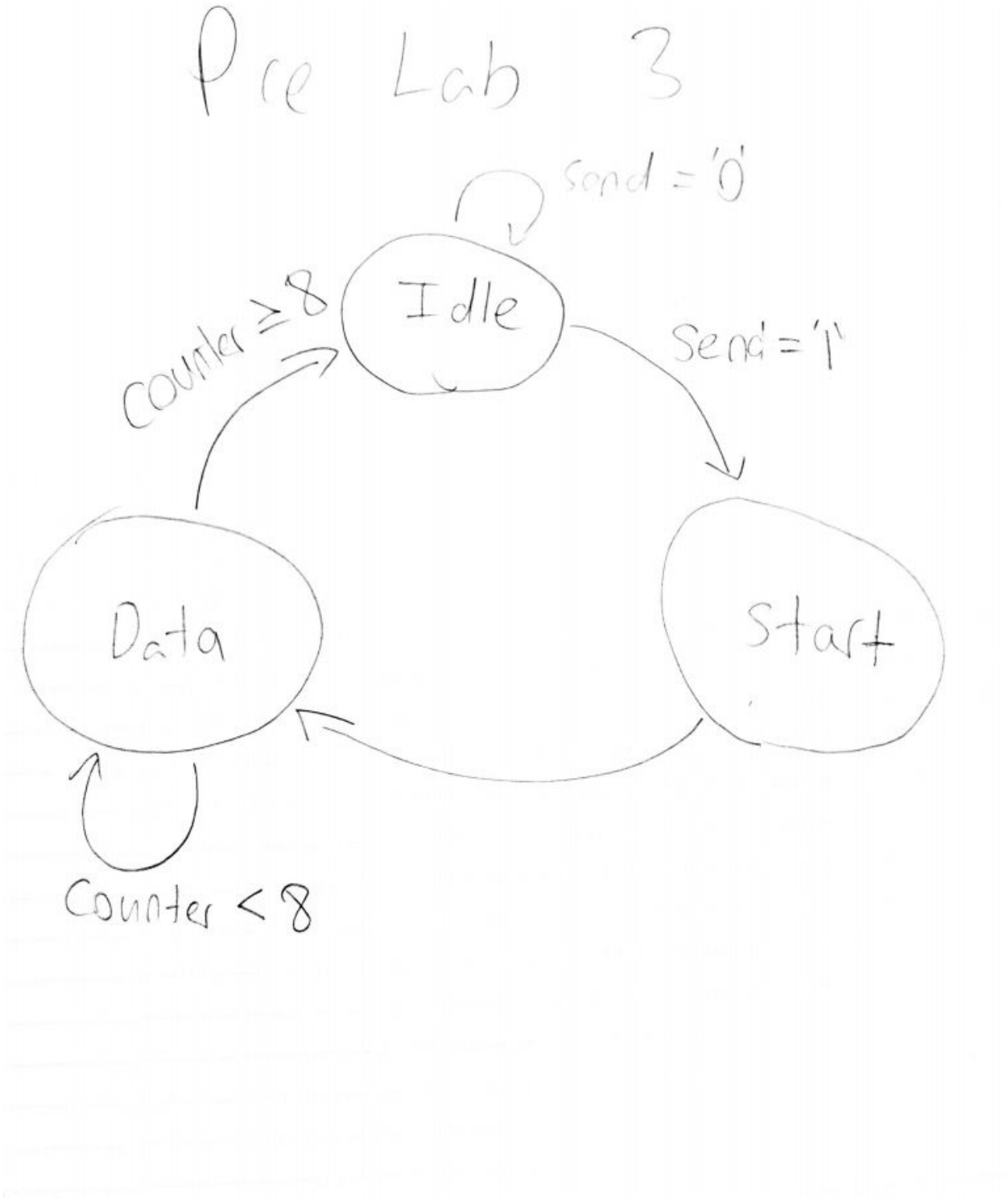
Embedded Systems 1

Lab Report 3

Gavin McKim
3/28/2019

Pre Lab:



Purpose:

The main purpose of the lab was to introduce the idea of Finite State Machines(FSM). FSM are computational models that utilize “states”. The program performs certain actions depending on the state it is currently in. The state changes on clock ticks when certain conditions are met. The concept of FSM was demonstrated by creating a basic Universal Asynchronous Receiver Transmitter(UART). A UART uses FSM to transmit data between two machines.

1. We Can Rebuild Him

Theory:

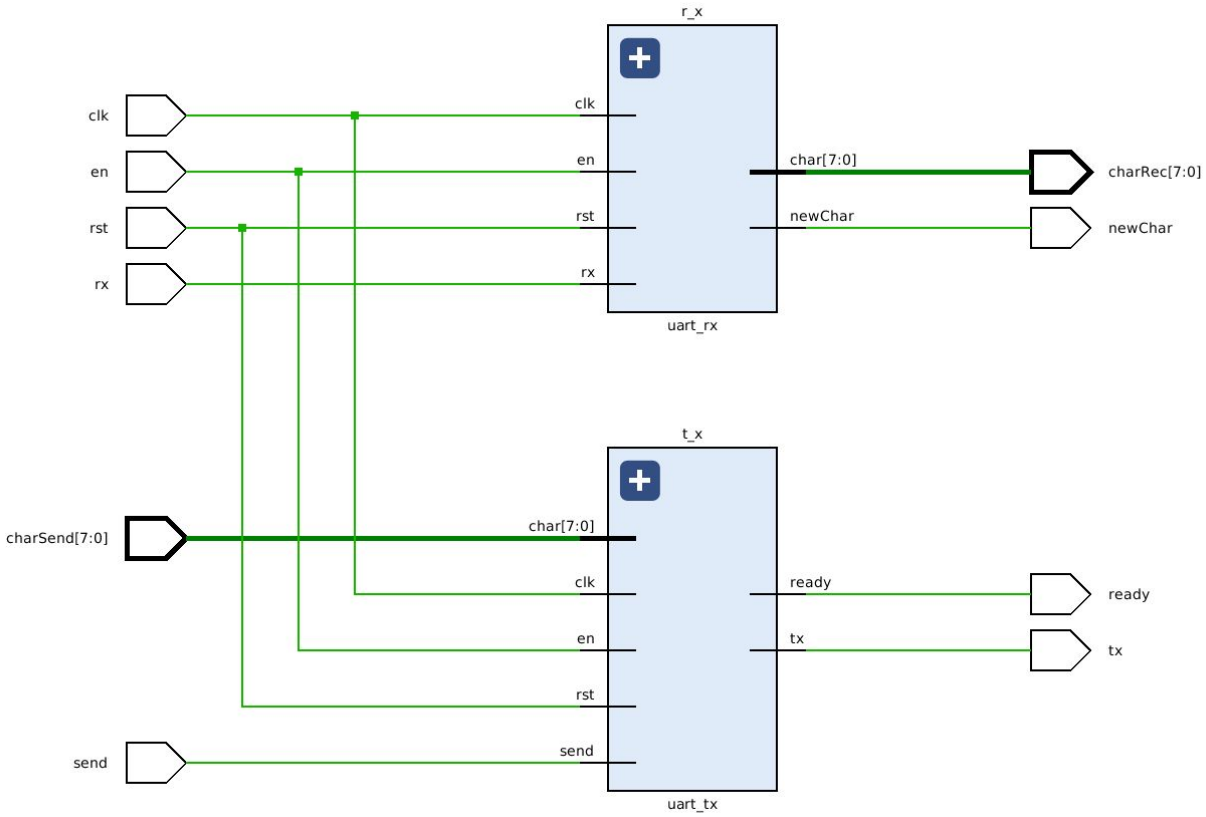
The first part of the lab involved completing the UART. The lab provided the final design of the UART and the receiver aspect of it. The first part had us create the transmitter side of the UART. This transmitter is triggered when the “line” in the UART goes low. When that happens the transmitter sends an 8-bit character one bit at a time. It does this using FSM. When the transmitter is triggered it goes in to a “start” state where it prepares to send the character. Then it transitions into the “data” state where it sends the character one bit at a time. When run with the provided test bench, it should show the correct hexadecimal values in charRec.

VHDL Code:

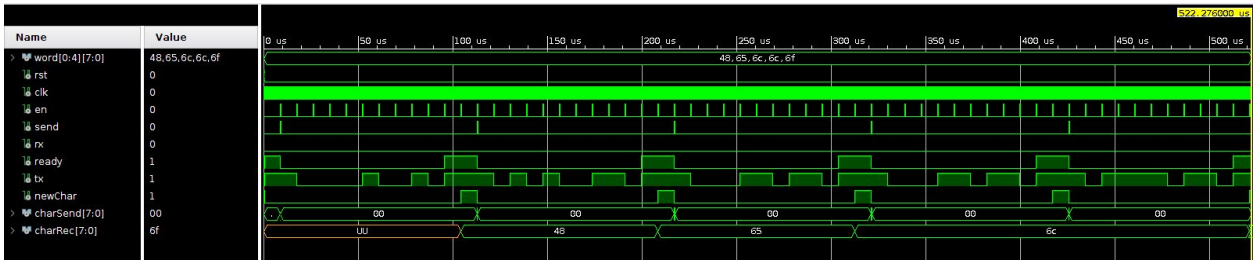
```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.NUMERIC_STD.ALL;
5
6   -- Uncomment the following library declaration if instantiating
7   -- any Xilinx leaf cells in this code.
8   --library UNISIM;
9   --use UNISIM.VComponents.all;
10
11  entity uart_tx is
12  port (
13      clk , en , send , rst : in std_logic;
14      char : in std_logic_vector (7 downto 0);
15      ready , tx : out std_logic  );
16  end uart_tx ;
17
18  architecture Behavioral of uart_tx is
19
20      -- state type enumeration and state variable
21      type state is (idle, start, data);
22      signal curr : state := idle;
23
24      signal temp : std_logic_vector (7 downto 0);
25      signal counter : std_logic_vector (3 downto 0);
26  begin
27      process(clk) begin
28          if rising_edge(clk) then
29              if(rst = '1') then
30                  temp <= (others => '0');
31                  counter <= (others => '0');
32                  curr <= idle;
33                  tx <= '1';
34                  ready <= '1';
35              end if;
36
37
38
39              if(en = '1') then
40                  case curr is
41
42                      when idle =>
43                      if(send = '1') then
44                          temp <= char;
45                          curr <= start;
46                          ready <= '0';
47                      else
48                          tx <= '1';
49                          ready <= '1';
50                      end if;
51
52                      when start =>
53                      tx <= '0';
54                      counter <= "0000";
55                      curr <= data;
```

```
56
57                      when data =>
58                      if(unsigned(counter) < 8) then
59                          tx <= temp(to_integer(unsigned(counter)));
60                          counter <= std_logic_vector(unsigned(counter) + 1);
61
62                      else
63                          tx <= '1';
64                          ready <= '1';
65                          curr <= idle;
66                      end if;
67
68
69                  end case;
70
71              end if;
72          end if;
73
74      end process;
75
76  end Behavioral;
77
```

UART RTL Schematic



Simulation



2. We Can Rebuild Him

Theory

For the next part of the lab we create a program to test sending the computer information from the board. This program holds a four character array that contains my RUID(gqm4). The sender program sends this RUID utilizing a single FSM. It defaults in the idle state and when the button is pressed while the system is ready, it outputs one character of the RUID. Then it cycles through a few “busy” states. When the button is unpressed it goes from the last busy state back to idle. When it reaches the end of the RUID it starts over. This was the implemented in a top level design along with a clock divider, debouncers, and the uart. This top level design would be implemented onto a zybo board along with a PMOD JB. After using the consle to connect the devices, the Zybo’s buttons should control the program. When one button is pressed it should send only one character of my RUID until reaching the end at which point it starts over. If the reset button is pressed it should reset back to the first character in my RUID.

VHDL Code

Sender

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.NUMERIC_STD.ALL;
5
6
7   entity sender is
8     Port ( rst, clk, en, btn, ready : in std_logic;
9            send : out std_logic;
10           char : out std_logic_vector(7 downto 0));
11  end sender;
12
13  architecture Behavioral of sender is
14  type my_array is array (0 to 3) of std_logic_vector(7 downto 0);
15  type state is (idle, busyA, busyB, busyC);
16
17  signal NETID : my_array;
18  signal curr : state := idle;
19  signal i : std_logic_vector(3 downto 0) := (others => '0');
20
21
22
23  begin
24  NETID(3) <= x"67";
25  NETID(2) <= x"71";
26  NETID(1) <= x"6D";
27  NETID(0) <= x"34";
28
29
30  process(clk) begin
31  if rising_edge(clk) and en = '1' then
32
33  if(rst = '1') then
34      send <= '0';
35      char <= (others => '0');
36      i <= (others => '0');
37      curr <= idle;
38
39  else
40  case curr is
41      when idle =>
42      if(ready = '1' and btn = '1') then
43          if (unsigned(i) < 4) then
44              send <= '1';
45              char <= NETID(to_integer(unsigned(i)));
46              i <= std_logic_vector( unsigned(i) + 1 );
47              curr <= busyA;
48          else
49              i <= (others => '0');
50          end if;
51      end if;
52
53      when busyA =>
54      curr <= busyB;
55
56      when busyB =>
57      send <= '0';
58      curr <= busyC;
59
60      when busyC =>
61      if(ready = '1' and btn = '0') then
62          curr <= idle;
```

```
63          else
64              curr <= busyC;
65          end if;
66
67      end case;
68      end if;
69      end if;
70      end process;
71
72
73
74      end Behavioral;
```

Clock Divider

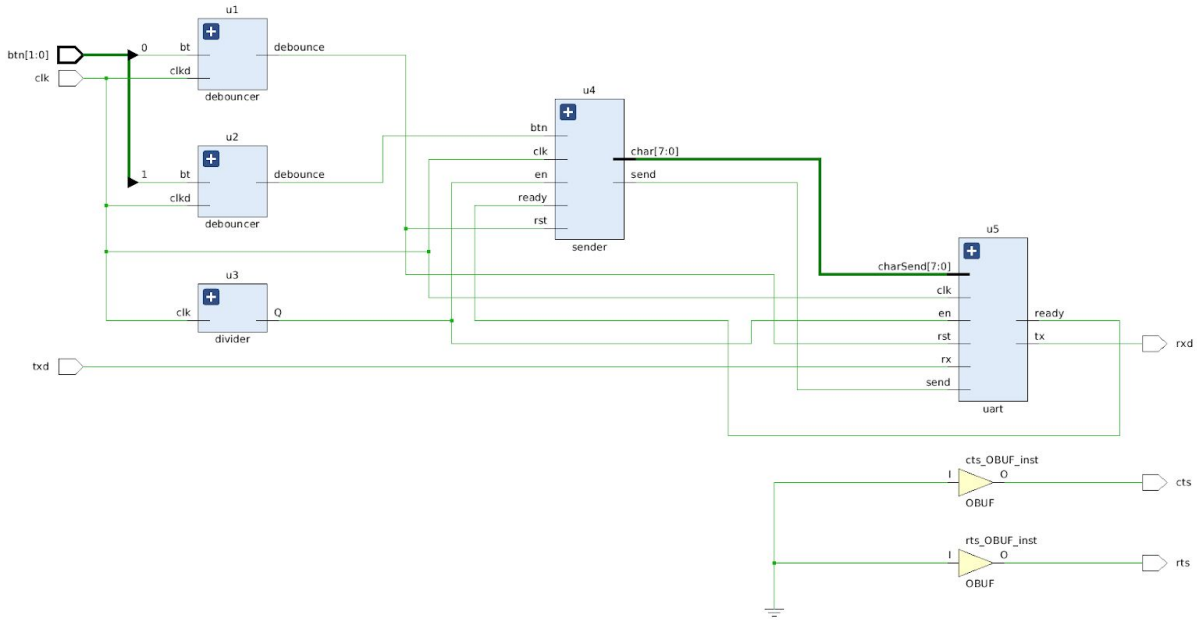
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- 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 divider is
port (
  clk_in : in std_logic;
  div : out std_logic
);
end divider;

architecture cnt of divider is
  signal count : std_logic_vector (25 downto 0) := (others => '0');
begin
process(clk_in) begin
  if rising_edge(clk_in) then
    count <= std_logic_vector( unsigned(count) + 1);

    if ( unsigned(count) = 1085) then
      div <= '1';
      count <= (others => '0');

    else
      div <= '0';
    end if;

  end if;
end process;

end cnt;
```

Top Level Design

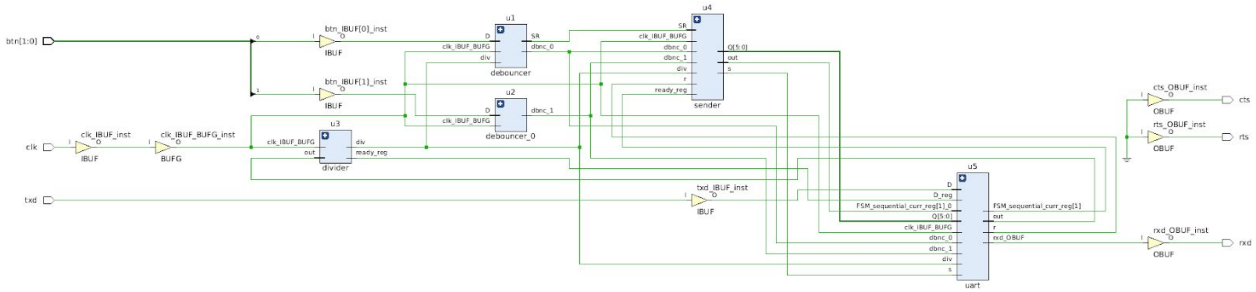
```vhdl

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity top_level is
  Port (txd, clk : in std_logic;
        btn : in std_logic_vector(1 downto 0);
        rxd : out std_logic;
        cts : out std_logic := '0';
        rts : out std_logic := '0' );
end top_level;

architecture Behavioral of top_level is

component debouncer is
 Port ( clkd, bt : in std_logic;
        debounce: out std_logic);
end component;

component divider is
  Port ( clk : in std_logic;
         Q : out std_logic);
end component;

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

component uart is
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 dbnc : std_logic_vector(1 downto 0);
signal c : std_logic_vector(7 downto 0);
signal s, r, div : std_logic;

begin

u1: debouncer
port map ( bt => btn(0),
           clkd => clk,
           debounce => dbnc(0));

u2: debouncer
port map ( bt => btn(1),
           clkd => clk,
           debounce => dbnc(1));

u3: divider
port map ( Q => div,
           clk => clk );

```

```
63  u4: sender
64  port map ( btn => dbnc(1),
65             clk => clk,
66             en => div,
67             ready => r,
68             rst => dbnc(0),
69             send => s,
70             char => c);
71
72  u5: uart
73  port map ( charSend => c,
74             clk => clk,
75             en => div,
76             rst => dbnc(0),
77             rx => txd,
78             send => s,
79             tx => rxd,
80             ready => r );
81  end Behavioral;
82
```

Elaborated Schematic



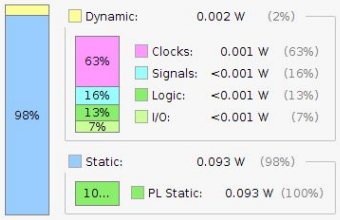
Synthesis Schematic



On-Chip Power and Utilization

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	67	17600	0.38
FF	104	35200	0.30
IO	6	100	6.00
BUFG	1	32	3.13



Constraint File

```
## 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 }];

##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

```

Extra Credit

Theory

In the previous part of the lab, we designed sender to test the transmission aspect of the UART. In this part of the lab we replace “sender” with “echo”. Echo is designed to test the receiver side of the UART. Echo will receive an input from the computer and echo it back. Like sender, echo does this using a single FSM. The FSM contains the states, idle BusyA, BusyB and BusyC. The FSM is initialized to idle. When the newChar and send inputs are both asserted to ‘1’, the program will send the char in charIn to charOut and then transition to BusyA. Then it goes to BusyB followed by BusyC. It stays in BusyC until ready is asserted to ‘1’ in which case it transitions back to idle. As mentioned earlier the device should echo back any input sent from the computer. When characters are typed on the keyboard, the device should echo the characters back and they should appear on the minicom terminal.

Echo Code

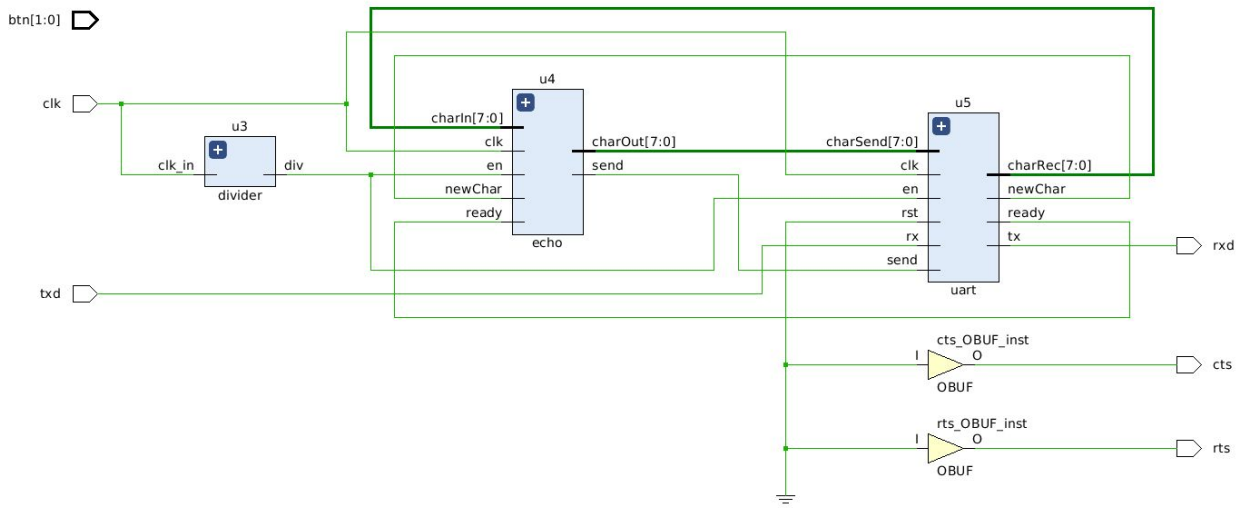
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity echo is
6    Port (clk, en, ready, newChar : in std_logic;
7          charIn : in std_logic_vector(7 downto 0);
8          send : out std_logic;
9          charOut : out std_logic_vector(7 downto 0) );
10 end echo;
11
12 architecture Behavioral of echo is
13 type state is (idle, busyA, busyB, busyC);
14 signal curr : state := idle;
15 begin
16
17 process(clk) begin
18 if rising_edge(clk) then
19     if (en = '1') then
20         case(curr) is
21         when idle =>
22             if(newChar = '1') then
23                 send <= '1';
24                 curr <= busyA;
25             end if;
26         when busyA =>
27             curr <= busyB;
28         when busyB =>
29             send <= '0';
30             curr <= busyC;
31         when busyC =>
32             if(ready = '1') then
33                 curr <= idle;
34             end if;
35         end case;
36     end if;
37 end if;
38 end process;
39
40 end Behavioral;
41
```

New Top Level Code

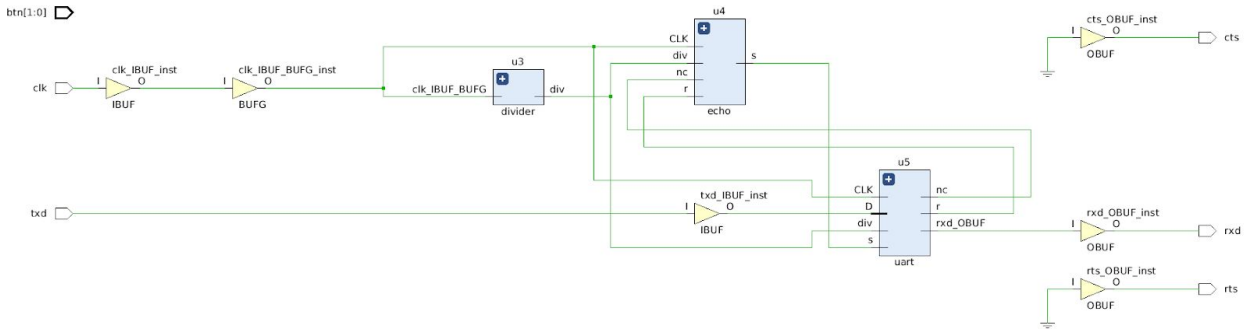
```vhdl


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity top_level is
  Port (txd, clk : in std_logic;
        btn : in std_logic_vector(1 downto 0);
        rxd : out std_logic;
        cts : out std_logic := '0';
        rts : out std_logic := '0' );
end top_level;

architecture Behavioral of top_level is



component divider is
port (
  clk_in : in std_logic;
  div : out std_logic
);
end component;

component echo is
  Port (clk, en, ready, newChar : in std_logic;
        charIn : in std_logic_vector(7 downto 0);
        send : out std_logic;
        charOut : out std_logic_vector(7 downto 0) );
end component;

component uart is
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 dbnc : std_logic_vector(1 downto 0);
signal c1, c2 : std_logic_vector(7 downto 0);
signal s, r, nc, div : std_logic;

begin


u3: divider
port map ( div => div,
           clk_in => clk );

```

```
54  u4: echo
55  port map ( newChar => nc,
56             clk => clk,
57             en => div,
58             ready => r,
59             charIn => c2,
60             send => s,
61             charOut => c1);
62
63  u5: uart
64  port map ( charSend => c1,
65             clk => clk,
66             en => div,
67             rst => dbnc(0),
68             rx => txd,
69             send => s,
70             tx => rxd,
71             ready => r,
72             newChar => nc,
73             charRec => c2 );
74  end Behavioral;
```

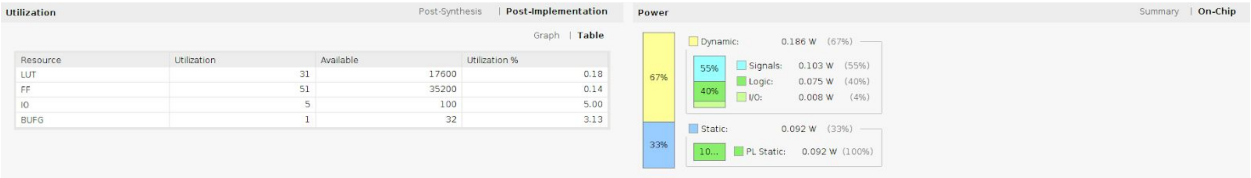
Elaboration Schematic



Synthesis Schematic



On-Chip Power and Utilization



Constraint

```
## 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 }];

##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

The lab was very useful in understanding Finite State Machines. This system was useful for understanding how FSM's can be utilized and how they work. It also gave me an idea of how UART's transfer information bit by bit. It also helped understand how to use a PMOD and write a constraint file for it. I feel as I completely understand FSM and how they work. However, there is still some confusion with UART's. For example I created the extra credit echo program,

but was unsure how it fit into the top level design. Also I am still uncomfortable with the debugging process. I had a bug with my top level implementation and it took me a very long time to figure out what was wrong.