Course Name: Embedded Systems Design

Course Number: 14 : 332 : 493 : 02

Assignment: Lab 3 – Where No Clock Has Gone Before

Instructor: [Phillip Southard]

Date Submitted: [March 28, 2019]

Submitted By: [Timothy Langer] (RUID#: 189005424)

Contents

1 Purpose

2 Pre-Lab

3 Lab Assignment 1: We Can Rebuild Him

3.1 Theory of Operation . . . .

3.2 Design . . . .

3.3 Test . . . .

4 Lab Assignment 2: We have The Technology

4.1 Theory of Operation . . . .

4.2 Design . . . .

4.3 Implementation . . . .

5 Discussion

5.1 Observations . . . .

5.2 Questions/Followup . . . .

1 Purpose

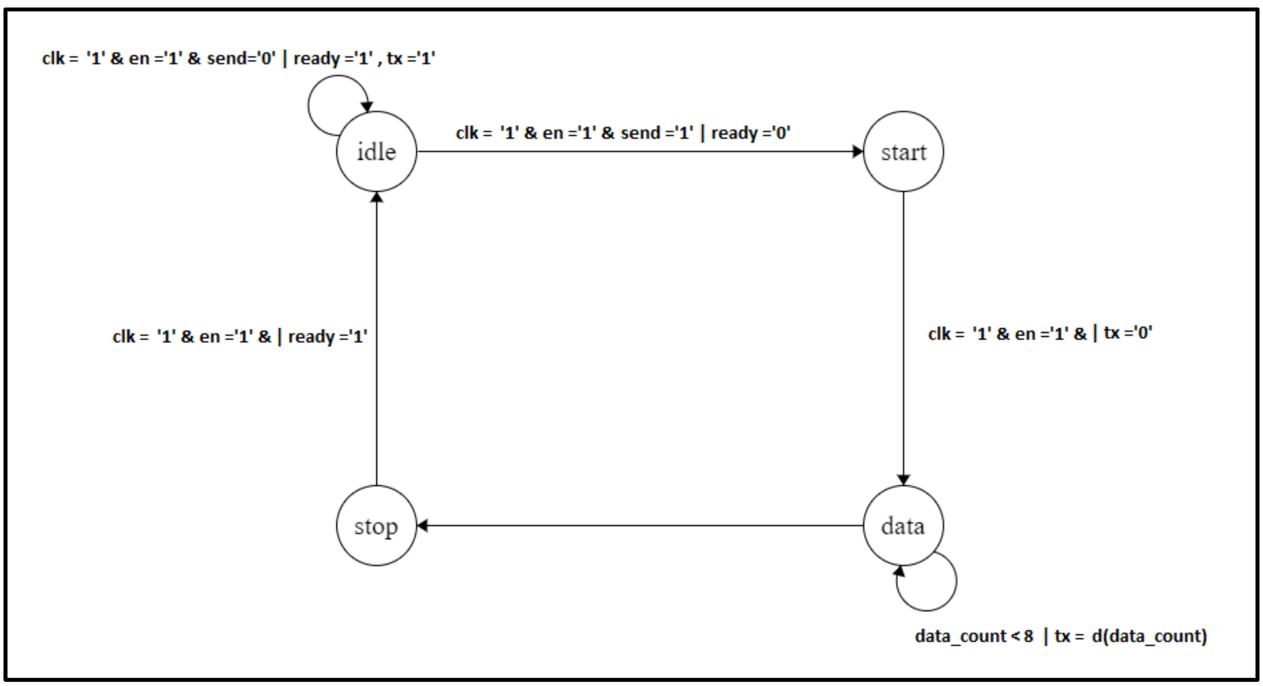
The purpose of this lab is to further our understanding of finite state machines and UARTs and implementing them in VHDL. A finite state machine is a system that is in one of a finite number of states at a given time. The state can change based on inputs and the current state and then resulting in a specific output. In this lab we created a state-machines to implement a UART,(Universal Asynchronous Receiver/Transmitter). UART's are devices used for communications between devices. Using a Finite State Machine to control the behavior a circuit in a specific sequence, can produce an output that can be read with a computer and displayed in a terminal window.

2 Pre-lab

Pre-Lab Task:

Draw the FSM state diagram for a device that takes as input an 8-bit data packet and transmits it using the protocol in figure (3.1) with 8 data bits, 0 parity bits, and 1 stop bit. It should take as input both a clock and a clock enable signal for timing, with input control signal send, and output a signal called tx that is the serial output as well as a signal called ready that indicates the machine is in an idle state.

Diagram:



UART RX STATE DIAGRAM

3 Lab Assignment 1: We Can Rebuild Him

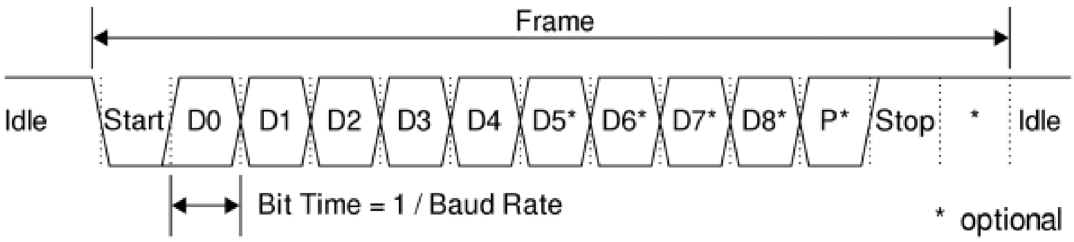


Figure 3.1: UART Frame Waveform [2]

3.1 Theory of Operation

In this portion of the lab we will be creating the entity uart_tx in VHDL. This entity is responsible for sending data using the protocol from figure 3.1. it will send 8 data bits as well as a start and stop bit. A parity bit can be used for error checking, but will not be necessary for this lab. This entity will transmit the data serially, bit by bit, to the receiving device.

The entity uart_tx in this design will have the following behavior:

A) To begin the circuit will only operate on the rising clock edge of 125MHz.
B) If **restart** is asserted with the **rising clock edge,** all internal registers are cleared and it goes into the idle state.
C) When **Rising clock edge** and **en** are asserted:
   a. If in idle state, **ready** = '1'.
   b. In idle, when send is asserted, char will be stored in a register d, and the current state will be changed to the start state(begin transmitting data).
   c. When in the start state **tx** <='0' to signify the begin of data transmission, the data-count is set to 0, and the current state is changed to the data state.
   d. In the data state, if data_count is less than the number of bits, tx = d(data_count)
   e. If in the data state, data_count is not less than the number of bits, **tx**='1' signifying end of data transmission, and the current state will be changed to stop.
   f. When in the stop state, **ready** ='1', to signify its ready to send more data, and the current state is changed to idle.

Overall the circuit will wait in idle, ready to transmit data. Once a send signal is asserted, the 8-bit char will be loaded into a register called "d" and move out of the idle state. Tx will output a start bit to signify the beginning of data transmission Each individual bit will then be transmitted and finish with a stop bit to signify the end of transmission. The circuit will then assert a ready='1' signifying its ready to transmit more data and will change back into the idle state waiting to repeat the cycle. Simulating this design with a testbench, its functionality was confirmed.

3.2 Design

Uart_tx.vhdl

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.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, stop);
 signal curr : state := idle;
 signal count : std_logic_vector(3 downto 0) := (others => '0');
 signal d : std_logic_vector(7 downto 0) := (others => '0');
begin

process(clk)
begin
if rising_edge(clk) then
if rst ='1' then
curr<= idle;
count <=(others => '0');
d<=(others => '0');
ready<='1';
tx<='1';
elsif en ='1' then
case curr is

    when idle =>

    if send ='1' then
    ready<='0';
    d<= char;

    curr <= start;
    else
    ready<='1';
    tx<='1';
    curr <=idle;
    end if;

    when start =>
    tx <= '0';
    count<="0000";
```

```
    curr<=data;

    when data =>
    if (unsigned(count)<8) then

     tx<=d(to_integer(unsigned(count)));
     count <= std_logic_vector(unsigned(count)+1);

     curr <=data;
    else
    tx <='1';
    curr<=stop;
    end if;

    when stop =>
    ready <='1';
    curr<= idle;


    when others =>
    curr <= idle;

    end case;

end if;
end if;
end process;


end Behavioral;
```

Uart_rx.vhdl

```
--
-- written by Gregory Leonberg
-- fall 2017
--
---------------------------------------------------------------------------

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


entity uart_rx is
    port (
    clk, en, rx, rst    : in std_logic;
    newChar             : out std_logic;
    char                : out std_logic_vector (7 downto 0)
);
end uart_rx;

architecture fsm of uart_rx is

    -- state type enumeration and state variable
    type state is (idle, start, data);
    signal curr : state := idle;

    -- shift register to read data in
    signal d : std_logic_vector (7 downto 0) := (others => '0');

    -- counter for data state
    signal count : std_logic_vector(2 downto 0) := (others => '0');

    -- double flop rx plus 2 extra samples to take majority vote of 3
    -- majority vote of samples helps mitigate noise on line
    signal inshift : std_logic_vector(3 downto 0) := (others => '0');
    signal maj : std_logic := '0';

begin

    -- double flop input to fix potential metastability
    -- plus 2 extra samples to take majority vote of 3 inputs (oversampling)
    -- majority vote of samples helps mitigate noise on line
    process(clk) begin
        if rising_edge(clk) then
            inshift <= inshift(2 downto 0) & rx;
        end if;
    end process;

    -- majority vote of 3 samples (oversampling)
    -- majority vote of samples helps mitigate noise on line
    process(inshift)
    begin
        if (inshift(3) = '1' and inshift(2) = '1' and inshift(1) = '1') or
           (inshift(3) = '1' and inshift(2) = '1') or
           (inshift(2) = '1' and inshift(1) = '1') or
```

```vhdl
           (inshift(3) = '1' and inshift(1) = '1') then
                maj <= '1';
        else
            maj <= '0';
        end if;
    end process;

    -- FSM process (single process implementation)
    process(clk) begin
    if rising_edge(clk) then

        -- resets the state machine and its outputs
        if rst = '1' then

            curr <= idle;
            d <= (others => '0');
            count <= (others => '0');
            newChar <= '0';

        -- usual operation
        elsif en = '1' then
            case curr is

                when idle =>
                    newChar <= '0';
                    if maj = '0' then
                        curr <= start;
                    end if;

                when start =>
                    d <= maj & d(7 downto 1);
                    count <= (others => '0');
                    curr <= data;

                when data =>
                    if unsigned(count) < 7 then
                        d <= maj & d(7 downto 1);
                        count <= std_logic_vector(unsigned(count) + 1);
                    elsif maj <= '1' then
                        curr <= idle;
                        newChar <= '1';
                        char <= d;
                    else
                        curr <= idle;
                    end if;

                when others =>
                    curr <= idle;

            end case;
        end if;

    end if;
    end process;

end fsm;
```

Uart.vhdl

```vhdl
--
-- written by Gregory Leonberg
-- fall 2017
--
---------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity 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 uart;

architecture structural of uart is
    component uart_tx port
    (
        clk, en, send, rst  : in std_logic;
        char                : in std_logic_vector (7 downto 0);
        ready, tx           : out std_logic
    );
    end component;

    component uart_rx port
    (
        clk, en, rx, rst    : in std_logic;
        newChar             : out std_logic;
        char                : out std_logic_vector (7 downto 0)
    );
    end component;

begin

    r_x: uart_rx port map(
        clk => clk,
        en => en,
        rx => rx,
        rst => rst,
        newChar => newChar,
        char => charRec);

    t_x: uart_tx port map(
        clk => clk,
        en => en,
        send => send,
        rst => rst,
```

```
        char => charSend,
        ready => ready,
        tx => tx);

end structural;
```

3.3 Test

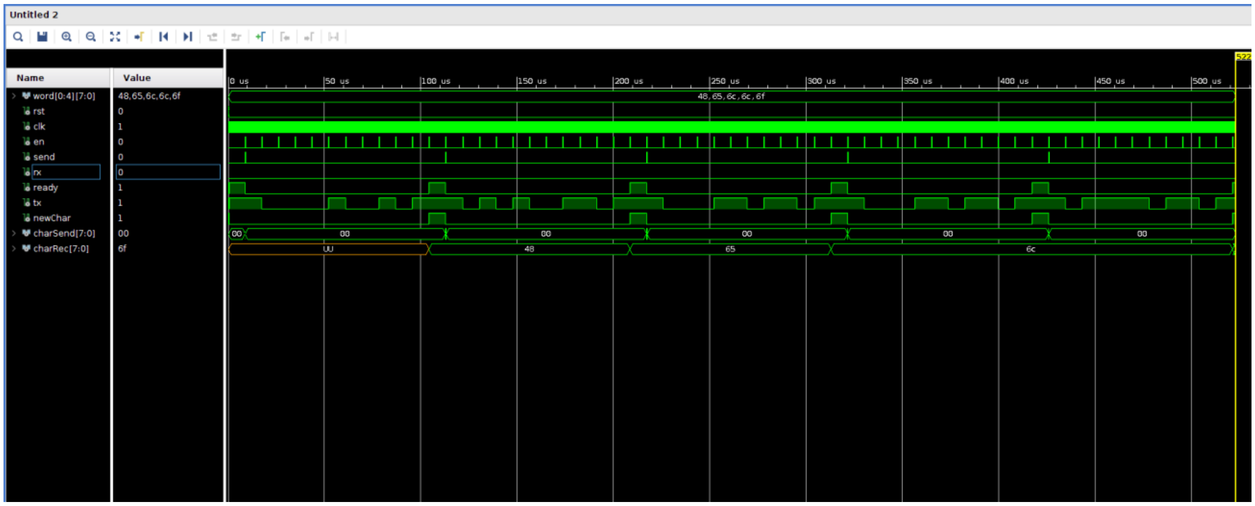
Testbench VHDL:

Uart_tb

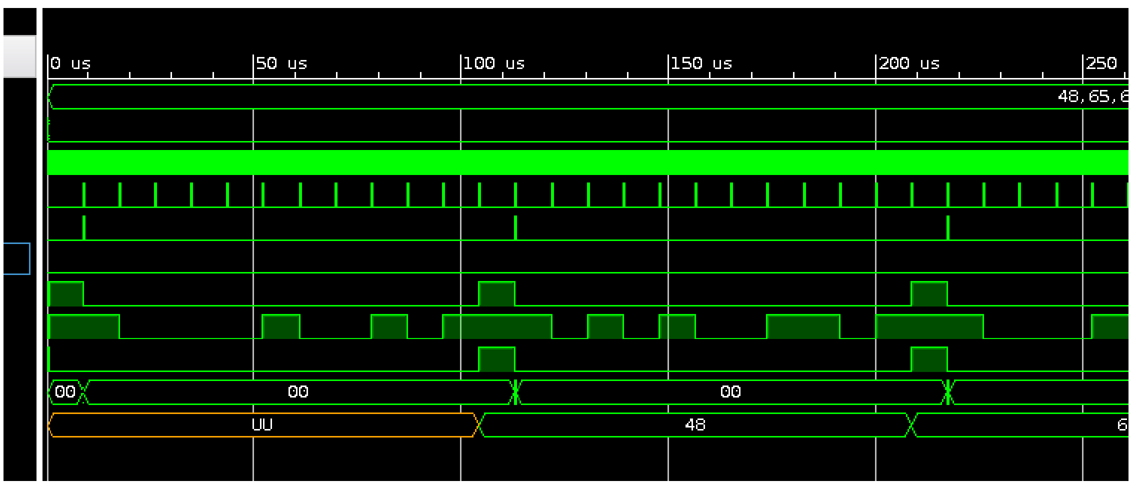
```
--
-- written by Gregory Leonberg
-- fall 2017
--
----------------------------------------------------------------------------

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

    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;

    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');

begin

    -- the sender UART
    dut: uart port map(
        clk => clk,
        en => en,
        send => send,
        rx => tx,
        rst => rst,
        charSend => charSend,
        ready => ready,
        tx => tx,
        newChar => newChar,
        charRec => charRec);


    -- clock process @125 MHz
    process begin
        clk <= '0';
```

```vhdl
        wait for 4 ns;
        clk <= '1';
        wait for 4 ns;
    end process;

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

    -- signal stimulation process
    process begin

        rst <= '1';
        wait for 100 ns;
        rst <= '0';
        wait for 100 ns;

        for index in 0 to 4 loop
            wait until ready = '1' and en = '1';
            charSend <= word(index);
            send <= '1';
            wait for 200 ns;
            charSend <= (others => '0');
            send <= '0';
            wait until ready = '1' and en = '1' and newChar = '1';

            if charRec /= word(index) then
                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;
            elsif charRec = word(index) then
            report "correct";
            end if;

        end loop;

        wait for 1500 ns;
        report "End of testbench" severity FAILURE;

    end process;

end tb;
```

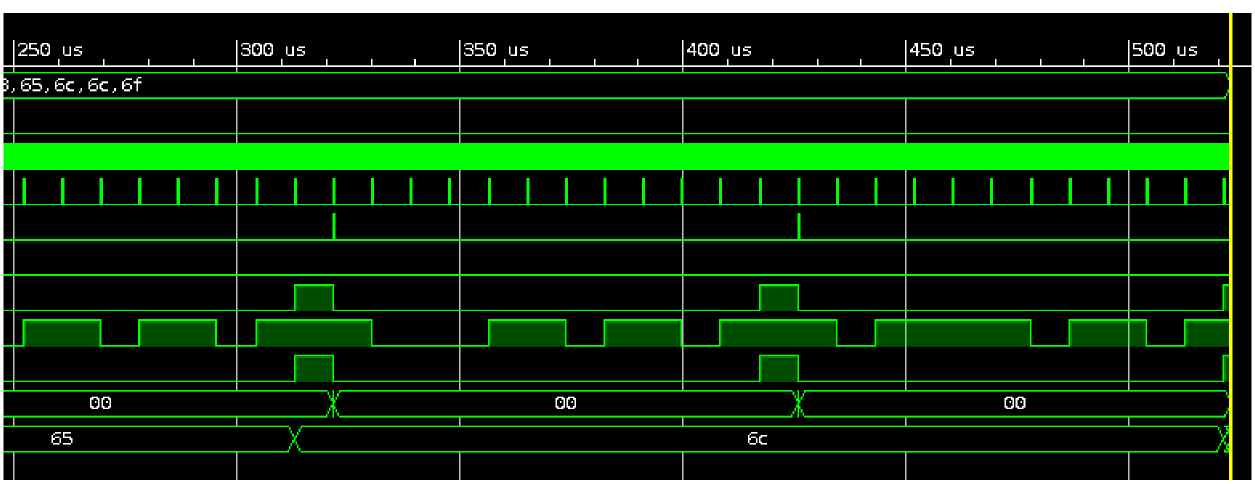
Simulation Results:



Zoom in (LEFT SIDE):



Zoom in (RIGHT SIDE):



4 Lab Assignment 2: We Have The Technology

4.1 Theory of Operation

In order for the UART to actually communicate data we need to drive some output through it by sending bytes. We created a finite state machine that will send the ASCII code for my netID as well as a control signal.

We created a sender that takes as input a **reset**, **clock**, **clock enable**, **button**, and a **ready** signal and as outputs a **send** signal and an 8-bit vector **char**. The sender circuit will have the following behavior.

It stores a n-long array of 8-bit vectors called NETID where n is the number characters in the netid. Also a binary counter i and can count up to n. It has an initial state of idle and will change states only when clock is on the rising edge and enable is asserted. When reset is asserted, it will clear all the outputs as well as the counter I and the current state will change to idle.

When ready ='1' and button ='1' and i<n, send='1' and will place NETID(i) on char. It will also increment i by 1, and transition to busyA state. If ready ='1' and button='1' but i=n, it will reset i to 0 and stay in idle.

After entering busyA it will change to busyB. After entering BusyB it will change send to 0 and go to BusyC state. It should now stay in busyC state until ready changes to 1 and button is 0, then it change back to the idle state.

After creating the sender entity we can now create a top level design to incorporate the UART and the sender entities. To control the baud rate to send each bit individually, we used an entity called clock_div created from a preivous lab. We modified the new output clock to be that of the baud rate of 115200Hz.

We also utilized a debouncer entity created in a previous lab to smooth out the button response due to its mechanical nature.

Incorporating the debouncer, clck_divider, sender, and UART into a top level design, we can then test it on a zybo board.

Overall the circuit will send character by character, bit by bit, of the netid. It is instantiated in the sender circuit and will transmit to the computer and present on a terminal.

4.2 Design

sender.vhdl

```vhdl
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 ( clk, en, reset, button, ready : in STD_LOGIC;
           send : out STD_LOGIC;
           char : out STD_LOGIC_VECTOR (7 downto 0));
end sender;

architecture Behavioral of sender is

type state is (idle, busyA, busyB, busyC );
signal curr : state := idle;
signal i : std_logic_vector(2 downto 0) := (others=>'0');

type str is array (0 to 5) of std_logic_vector(7 downto 0);
signal word : str := (x"74", x"6d", x"6c", x"31", x"34", x"33");

begin

process(clk)
begin
if rising_edge(clk) then
if reset ='1' then
send <='0';
char <=(others=>'0');
i<=(others=>'0');
curr<=idle;

elsif en='1' then
    case curr is

     when idle =>
     if (ready ='1' and button ='1' and unsigned(i)<6)then
     send <='1';
     char <= word(to_integer(unsigned(i)));
     i<= std_logic_vector(unsigned(i) + 1);
     curr<=busyA;
     elsif  (ready ='1' and button ='1' and unsigned(i)=6) then
     i<=(others=>'0');
     curr<=idle;
     end if;
```

```vhdl
     when busyA =>
     curr<=busyB;

     when busyB =>
     send <='0';
     curr<= busyC;

     when busyC =>
     if ready ='1' and button ='0' then
     curr<= idle;
     end if;

    when others =>
    curr <= idle;

    end case;

end if;
end if;

end process;

end Behavioral;
```

Uart.vhdl

```vhdl
--
-- written by Gregory Leonberg
-- fall 2017
--
---------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity 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 uart;

architecture structural of uart is
    component uart_tx port
    (
        clk, en, send, rst  : in std_logic;
        char                : in std_logic_vector (7 downto 0);
        ready, tx           : out std_logic
    );
    end component;

    component uart_rx port
    (
        clk, en, rx, rst    : in std_logic;
        newChar             : out std_logic;
        char                : out std_logic_vector (7 downto 0)
    );
    end component;

begin

    r_x: uart_rx port map(
        clk => clk,
        en => en,
        rx => rx,
        rst => rst,
        newChar => newChar,
        char => charRec);

    t_x: uart_tx port map(
        clk => clk,
        en => en,
        send => send,
        rst => rst,
```

```
        char => charSend,
        ready => ready,
        tx => tx);

end structural;
```

Top_level_design.vhdl

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



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

architecture Behavioral of top_level_design is

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 sender port
        (
           clk, en, reset, button, ready : in STD_LOGIC;
           send : out STD_LOGIC;
           char : out STD_LOGIC_VECTOR (7 downto 0)
            );
        end component;

component clock_div port
(
  clk  : in std_logic;
  new_clock : out std_logic
);
end component;

component debounce
port(
    BTN : in STD_LOGIC;
    CLK : in STD_LOGIC;
    DBNC : out STD_LOGIC
);end component;

signal s_u1, s_u2, s_new_clock, s_ready, s_send : std_logic;
signal s_char : std_logic_vector(7 downto 0);
begin
```

```
u1: debounce
port map(
BTN => btn(0),
CLK =>  clk,
DBNC =>s_u1
);

u2: debounce
port map(
BTN => btn(1),
CLK =>  clk,
DBNC =>s_u2
);

u3: clock_div
port map(
clk => clk,
new_clock => s_new_clock
);

u4: sender
port map(
clk => clk,
en => s_new_clock,
button => s_u2,
reset => s_u1,
ready => s_ready,
char => s_char,
send => s_send
);

u5: uart
port map(
charSend =>s_char,
clk => clk,
en => s_new_clock,
rst => s_u1,
rx => TXD,
send => s_send,
ready => s_ready,
tx => RXD
);
CTS <='0';
RTS <='0';
end Behavioral;
```

Debounce.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;


entity debounce is
    Port ( BTN : in STD_LOGIC;
           CLK : in STD_LOGIC;
           DBNC : out STD_LOGIC);
end debounce;

architecture Behavioral of debounce is
signal shiftreg : std_logic_vector (1 downto 0) := (others => '0');
signal counter : std_logic_vector(21 downto 0) := (others =>'0');
begin

process (clk)
begin

if (rising_edge(clk)) then
    shiftreg(1) <= shiftreg(0);
    shiftreg(0) <= BTN;

if (unsigned(counter)<= 2499999) then
    DBNC <= '0';
    if shiftreg(1) = '1' then
    counter <= std_logic_vector(unsigned(counter)+1);
    elsif (shiftreg(1)='0') then
    counter <= (others => '0');

    end if;

elsif (unsigned(counter)= 2500000) then
    if (shiftreg(1)='1') then
    DBNC <='1';
    else
    DBNC <='0';
    counter <= (others =>'0');
    end if;

end if;
end if;



end process;

end Behavioral;
```

clock_div.vhdl

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

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

entity clock_div is
port(
      clk  : in std_logic;
      new_clock : out std_logic
    );

end clock_div;

architecture behavior of clock_div is

signal s_new_clock : std_logic:='0';
signal counter : std_logic_vector(26 downto 0) := (others => '0');

begin

 process(clk)
 begin
 new_clock <= s_new_clock;
 if rising_edge(clk) then
         --1085.06944444...
         if (unsigned(counter) <= 1085) then
                s_new_clock <= '0';
                counter <= std_logic_vector(unsigned(counter) + 1);
         else
                new_clock <= (not s_new_clock);
                counter <= (others => '0');

         end if;
                 end if;

    end process;

end behavior;
```

CONSTRAINT FILE

```
##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 }];

##LEDs and buttons
set_property -dict { PACKAGE_PIN U20   IOSTANDARD LVCMOS33 } [get_ports { RXD}];
set_property -dict { PACKAGE_PIN V20   IOSTANDARD LVCMOS33 } [get_ports { TXD }];
set_property -dict { PACKAGE_PIN W20   IOSTANDARD LVCMOS33 } [get_ports { CTS }];
set_property -dict { PACKAGE_PIN T20   IOSTANDARD LVCMOS33 } [get_ports { RTS }];


set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }];
set_property -dict { PACKAGE_PIN p16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }];
```

Explanation of constrains:

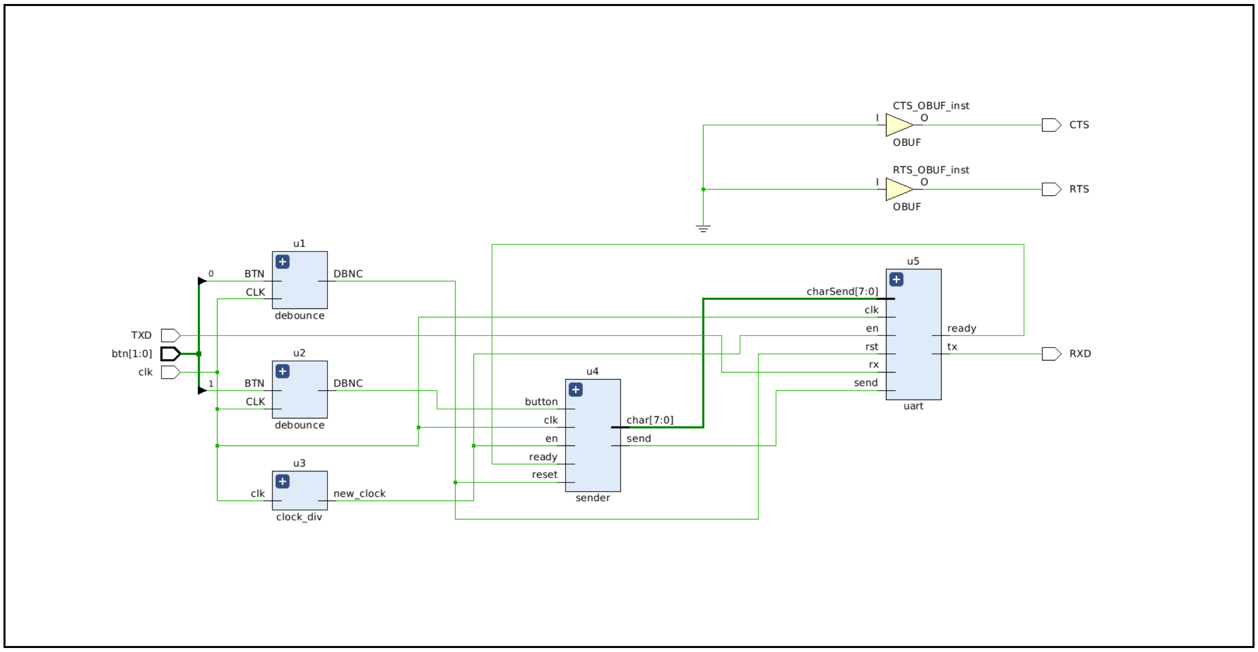
The changes to the constraint files for the connection of pmod and the buttons:

The pmod connections are T20, U20, V20, and W20 to output RTS, RXD, TXD, CTS respectively.

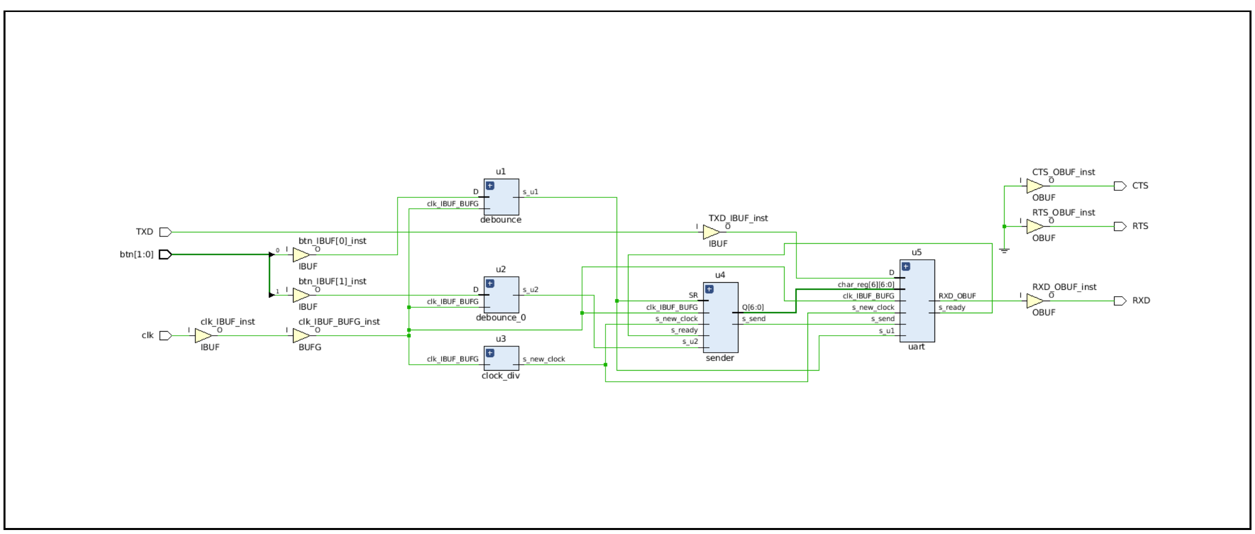
Button 1 and 0 are connected to pins R18 and P16.

4.3 Implementation

Top_level_design.vhdl Elaborate Schematic



Top_level_design.vhdl Synthesis Schematic

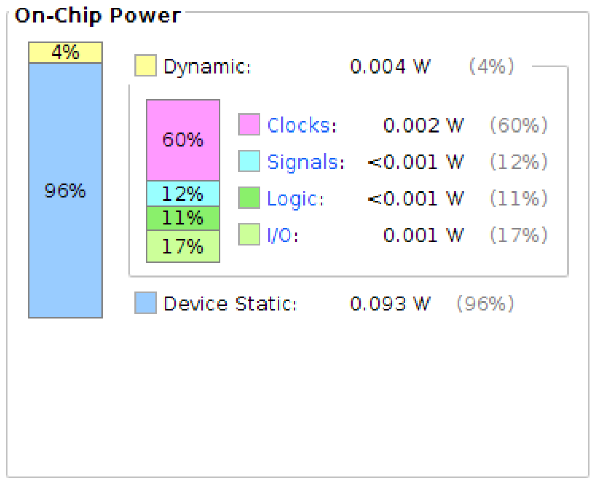


POWER

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

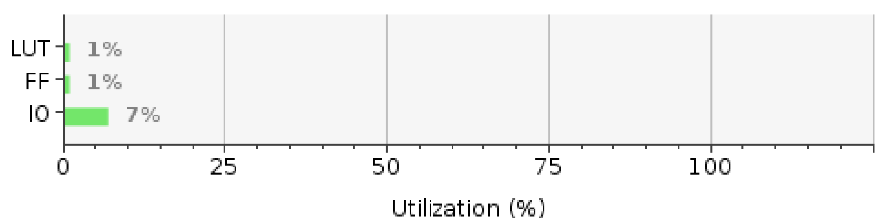
Total On-Chip Power:	**0.097 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.1°C**
Thermal Margin:	58.9°C (5.0 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



UTILIZATIZATION

Resource	Utilization	Available	Utilization %
LUT	117	17600	0.66
FF	100	35200	0.28
IO	7	100	7.00



Name	Slice LUTs (17600)	Slice Registers (35200)	Bonded IOB (100)	BUFGCTRL (32)
top_level_design	117	100	7	1
u1 (debounce)	36	25	0	0
u2 (debounce_0)	36	25	0	0
u3 (clock_div)	11	12	0	0
u4 (sender)	13	13	0	0
u5 (uart)	21	25	0	0

5.1 OBSERVATIONS

What did you learn?

It was very interesting to learn about UARTs. It seems like a very important topic for embedded systems with devices to communicate. I learned how parallel data transfer can not be sent at high frequencies due to capacitance, and how sending the data serially, bit by bit, can handle the higher frequencies. Because of this lab I now understand how UARTs are created and how they transmit and receive data.

5.2 Questions/follow ups

I am a little confused with how UARTs are connected to devices and the exact data flow of information from device to device.

I had trouble understanding how the UARTs physical inputs and outputs are connected to external devices and how they exactly communicate. I hope to get further understanding to implement in capstone and project design.