Digital Circuit Design2 Lab

Dr. Ashraf Armoush

Wednesday 2:00 pm – 5:00 pm

First Semester

Experiment Information	
Experiment Name: IP cores and Serial Communication	Experiment Number: #6
Performed: 2 Dec 2020	Submitted: 7 Dec 2020

Partner Students		
1- Taher Anaya	2- Hadi Jml	3 – Ameed Omar

Introduction:

In fact, our experiement consists of two parts, the first is IP Cores and the second is Serial Communication. in the first part, we are going to use IP cores. In the second part we are going to use the serial port RS-232 to exchange data between the computer and the FPGA, and to so, an IP core, called MURAT, was used.

Objectives:

- Using IP cores and integrate them into other VHDL Projects.
- Implementing a serial communication between the FPGA and the computer.

Procedure:

IP Cores

We use HDL to describe complex logic functions. But coding everything from scratch would be like reinventing the wheel and a waste of our time. Because of this, we may use an IP core.

An IP core is a block of HDL code that other engineers have already written to perform a specific function and to reduce design time. Open cores contain many open-source projects where the designer has full access to codes and design documents that are built to specific standards. In this lab, we will be given a chance to incorporate a Xilinx IP core into a simple project.

In this part, we will use the Xilinx's Core Generator System to incorporate a multiplier IP core into a VHDL project. Xilinx’s system comes with an IP core, called Multiplier, that accepts two inputs and gives an output of multiplying them. To design the given idea, we followed the instructions given in the lab’s manual to make use of the multiplier.

Procedure: *(cont.)*

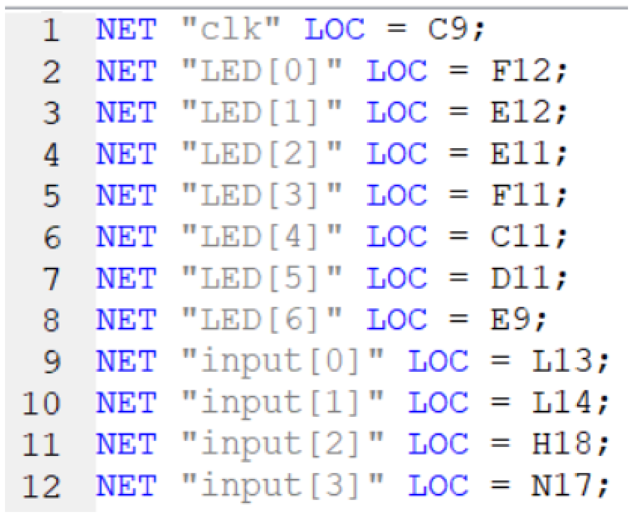
And our VHDL code for this part will seem like this:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
Library XilinxCoreLib;
library UNISIM;
use UNISIM.VComponents.all;



entity main is
   Port ( clk : in STD_LOGIC;
   LED : out STD_LOGIC_VECTOR (6 downto 0);
   input : in STD_LOGIC_VECTOR (3 downto 0));
end main;

architecture Behavioral of main is
signal A: std_logic_vector(2 downto 0);

COMPONENT multi
  PORT (
    clk : IN STD_LOGIC;
    a : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    b : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    p : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
   END COMPONENT;
begin

A <= "101";

u1 : multi PORT MAP (clk, A, input, LED );

end Behavioral;
```

Note that the first input always equals 5 because we have only four switches and we have used it for the second input. As for the multiplier code, it was used as it is without change.

Procedure: *(cont.)*

Switches were used as inputs, and LEDs were the indicators of the outputs. As a final step, we need to type our constraint file below.



```
NET "clk" LOC = C9;
NET "LED[0]" LOC = F12;
NET "LED[1]" LOC = E12;
NET "LED[2]" LOC = E11;
NET "LED[3]" LOC = F11;
NET "LED[4]" LOC = C11;
NET "LED[5]" LOC = D11;
NET "LED[6]" LOC = E9;
NET "input[0]" LOC = L13;
NET "input[1]" LOC = L14;
NET "input[2]" LOC = H18;
NET "input[3]" LOC = N17;
```

Up to here, we will have finished the first part of the experiment, and now let's move on to the second part, which is Serial Communication.

Serial Communication

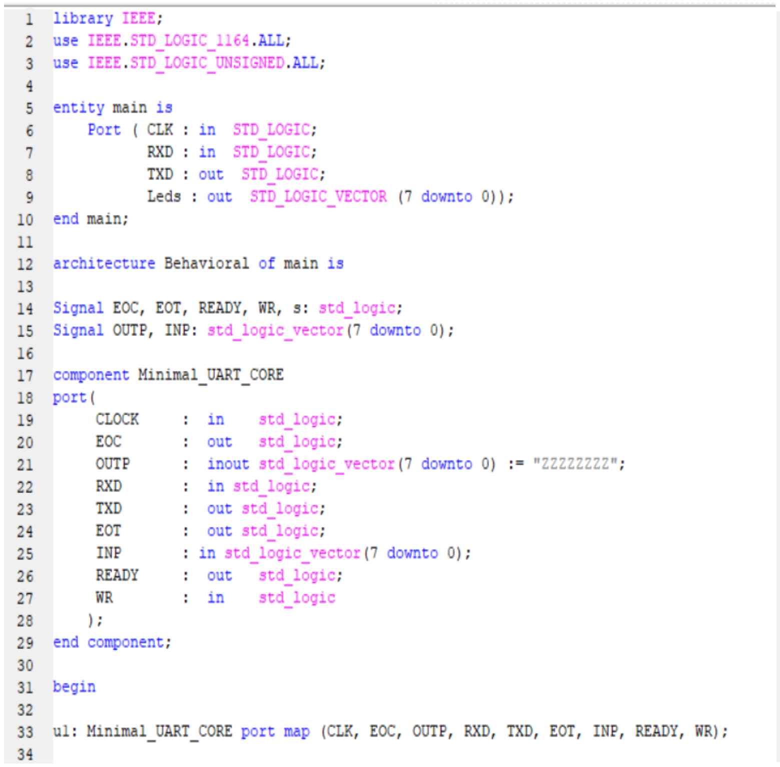
Serial communication is a method for transferring data between two systems. Where the data is sent one bit at a time, but since computers usually need buses of data, it has to be serialized before sent.

In this part, we have to use the serial port RS-232 to exchange data between the computer and the FPGA, and to so, an IP core, called MURAT, was used. Considering the IP core's documentation provided in the lab, we write a VHDL code that is responsible for recognizing the letter a computer sends to send back the letter next to it immediately.

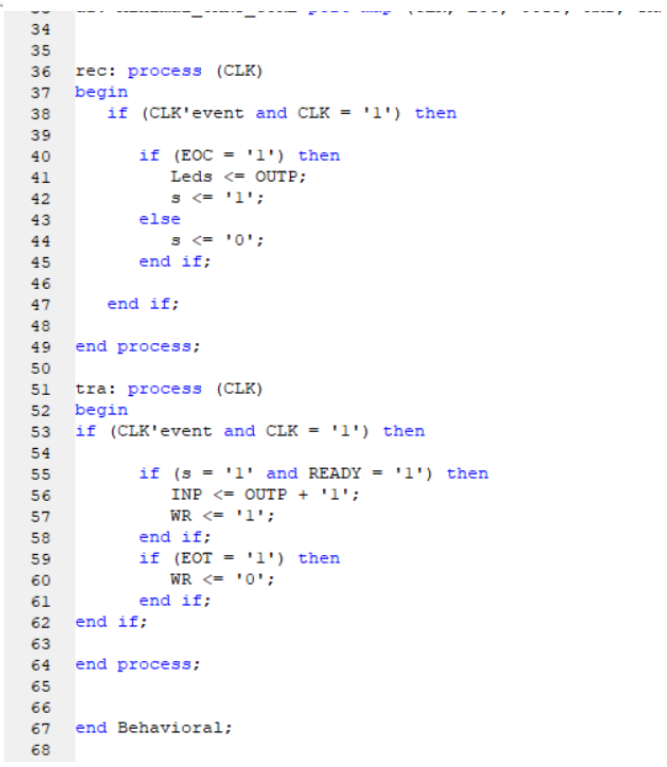
The Hyperterminal software was used to display data, LEDs on the kit were included in the process to indicate receiving messages, and a cable was only needed to connect the two systems.

Procedure: *(cont.)*

And our VHDL code for this part will seem like this:

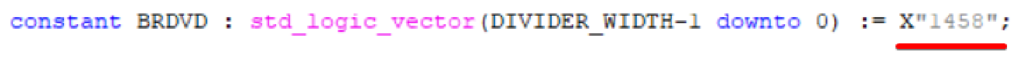


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity main is
    Port ( CLK : in  STD_LOGIC;
           RXD : in  STD_LOGIC;
           TXD : out  STD_LOGIC;
           Leds : out  STD_LOGIC_VECTOR (7 downto 0));
end main;

architecture Behavioral of main is

Signal EOC, EOT, READY, WR, s: std_logic;
Signal OUTP, INP: std_logic_vector(7 downto 0);

component Minimal_UART_CORE
port(
     CLOCK     :  in    std_logic;
     EOC       :  out   std_logic;
     OUTP      :  inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
     RXD       :  in std_logic;
     TXD       :  out std_logic;
     EOT       :  out std_logic;
     INP       : in std_logic_vector(7 downto 0);
     READY     :  out   std_logic;
     WR        :  in    std_logic
    );
end component;

begin

u1: Minimal_UART_CORE port map (CLK, EOC, OUTP, RXD, TXD, EOT, INP, READY, WR);
```



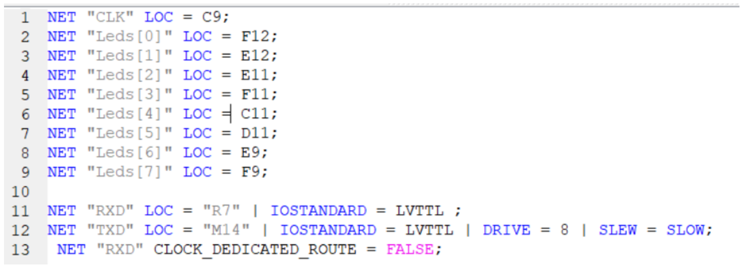
```


rec: process (CLK)
begin
   if (CLK'event and CLK = '1') then

      if (EOC = '1') then
         Leds <= OUTP;
         s <= '1';
      else
         s <= '0';
      end if;

   end if;

end process;

tra: process (CLK)
begin
if (CLK'event and CLK = '1') then

      if (s = '1' and READY = '1') then
         INP <= OUTP + '1';
         WR <= '1';
      end if;
      if (EOT = '1') then
         WR <= '0';
      end if;
end if;

end process;


end Behavioral;
```

The files related to the MURAT core were imported and not changed but used as they are, except for the baud rate in the BRG.vhd file that was set to the value of 1458H, to fit the clock cycle of the kit we have, which is 50MH.



```
constant BRDVD : std_logic_vector(DIVIDER_WIDTH-1 downto 0) := X"1458";
```

As a final step, we need to type our constraint file as below:

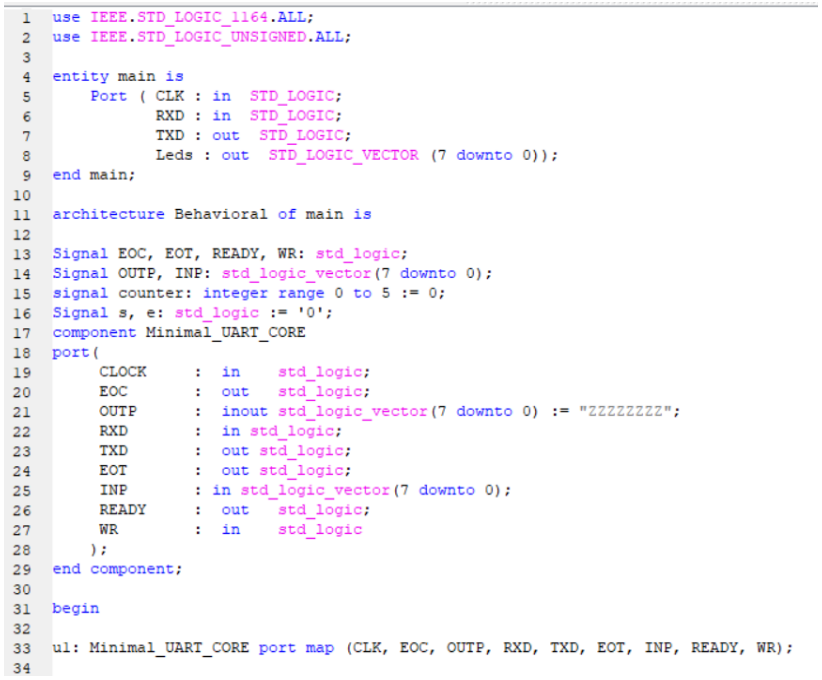


```
NET "CLK" LOC = C9;
NET "Leds[0]" LOC = F12;
NET "Leds[1]" LOC = E12;
NET "Leds[2]" LOC = E11;
NET "Leds[3]" LOC = F11;
NET "Leds[4]" LOC = C11;
NET "Leds[5]" LOC = D11;
NET "Leds[6]" LOC = E9;
NET "Leds[7]" LOC = F9;

NET "RXD" LOC = "R7" | IOSTANDARD = LVTTL ;
NET "TXD" LOC = "M14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = SLOW;
 NET "RXD" CLOCK_DEDICATED_ROUTE = FALSE;
```

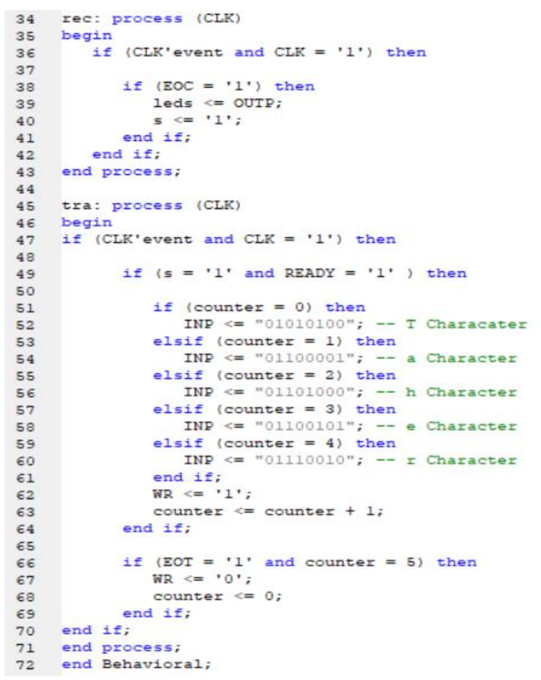
Procedure: *(cont.)*

After finishing this part successfully, we had to improve our code to make it applicable to transmitting a word instead of one letter per each letter received. We chose the name “Taher”.

In the last meeting, many attempts were done to achieve this goal, but none of them was successful enough to give the desired result. We were able to send the word infinite times, but we can not send the word once per letter received. In the bellow the code we wrote to send the word separately, hoping that our efforts do not go to waste.



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

entity main is
    Port ( CLK : in  STD_LOGIC;
           RXD : in  STD_LOGIC;
           TXD : out  STD_LOGIC;
           Leds : out  STD_LOGIC_VECTOR (7 downto 0));
end main;

architecture Behavioral of main is

Signal EOC, EOT, READY, WR: std_logic;
Signal OUTP, INP: std_logic_vector(7 downto 0);
signal counter: integer range 0 to 5 := 0;
Signal s, e: std_logic := '0';
component Minimal_UART_CORE
port(
     CLOCK     :  in    std_logic;
     EOC       :  out   std_logic;
     OUTP      :  inout std_logic_vector(7 downto 0)  := "ZZZZZZZZ";
     RXD       :  in std_logic;
     TXD       :  out std_logic;
     EOT       :  out std_logic;
     INP       : in std_logic_vector(7 downto 0);
     READY     :  out   std_logic;
     WR        :  in    std_logic
    );
end component;

begin

u1: Minimal_UART_CORE port map (CLK, EOC, OUTP, RXD, TXD, EOT, INP, READY, WR);

```



```vhdl
rec: process (CLK)
begin
   if (CLK'event and CLK = '1') then

      if (EOC = '1') then
         leds <= OUTP;
         s <= '1';
      end if;
   end if;
end process;

tra: process (CLK)
begin
if (CLK'event and CLK = '1') then

      if (s = '1' and READY = '1' ) then

         if (counter = 0) then
            INP <= "01010100"; -- T Characater
         elsif (counter = 1) then
            INP <= "01100001"; -- a Character
         elsif (counter = 2) then
            INP <= "01101000"; -- h Character
         elsif (counter = 3) then
            INP <= "01100101"; -- e Character
         elsif (counter = 4) then
            INP <= "01110010"; -- r Character
         end if;
         WR <= '1';
         counter <= counter + 1;
      end if;

      if (EOT = '1' and counter = 5) then
         WR <= '0';
         counter <= 0;
      end if;
end if;
end process;
end Behavioral;
```

Note that Constraint file for this part is the same that we used in the previous part.

Up to here, we finish our experiment, that’s All we have to do.

Conclusion:

In this experiment, we basically learned how to deal with IP cores and how to use them. It saved us much time and effort by offering us a block of VHDL instead of designing them from scratch. In addition to that, we learned about the principles of serial communication and how to transfer data between two systems.