CCE 3202 – Advanced Digital System Design

Lab Exercise #3

This lab exercise will show you how to create, synthesize, and test a 3-bit ripple counter. A ripple counter is simply a circuit that outputs the value "000" then "001" then "010", etc., until it hits "111" and resets to zero. You will be implementing this counter from within VHDL, and utilizing the BASYS onboard clock and you can watch one of the 7-segment display count from 0 to 7. Once the bitstream for this counter has been created, you simply need to download the file and the counter will begin automatically.

Deliverables for Lab Exercise #3

When completed, you will hand in the following deliverables for Lab Exercise #3, in this order:

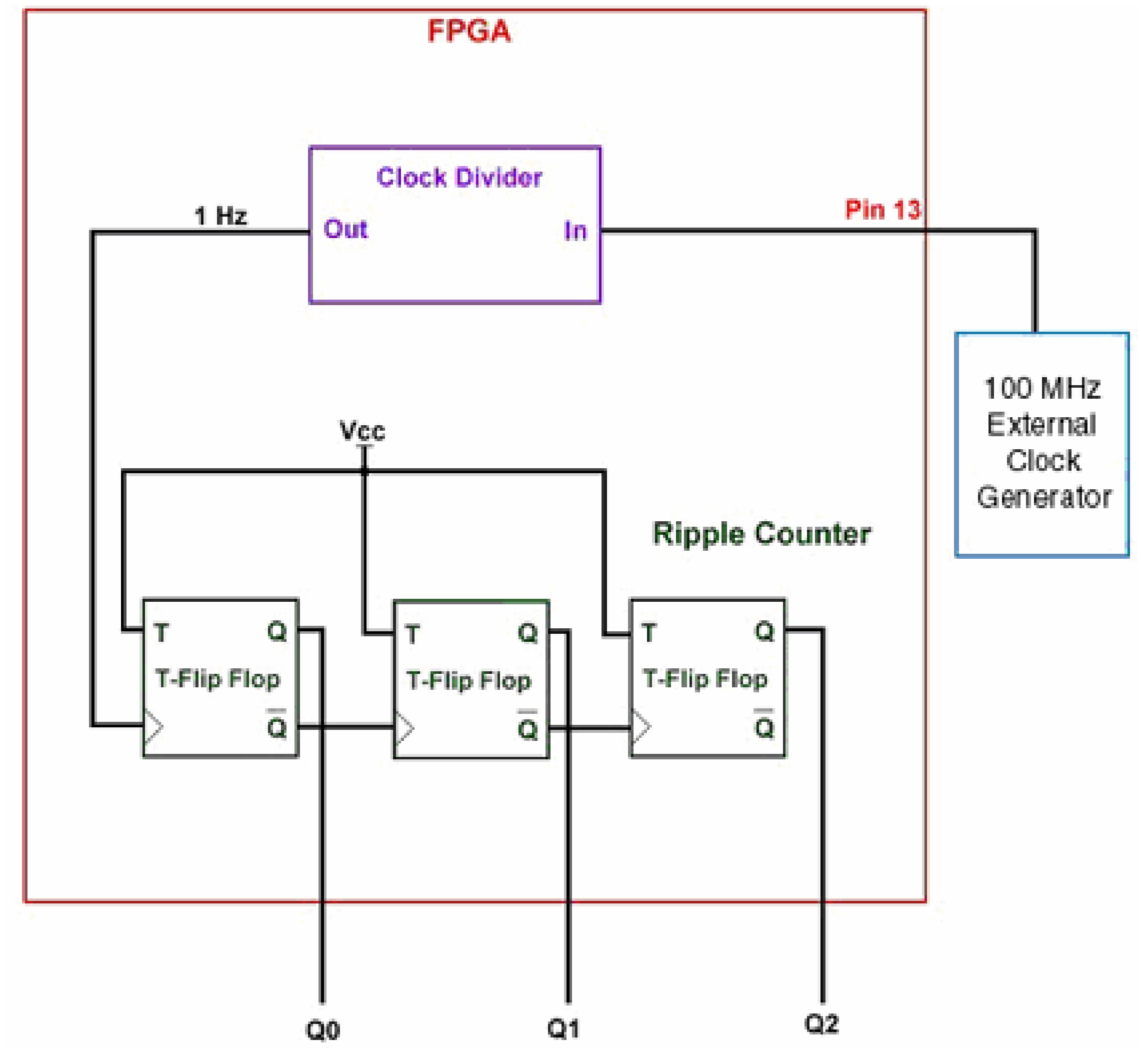
1. Title Page
2. Circuit Diagram
3. Your VHDL Code. Make your code *readable* and *neatly organized*.
4. Simulation Waveforms as proof that your code works.
5. Any comments

Introduction

You will use Xilinx Webpack v9.1 to allow the synthesis and creation of VHDL-based designs. This lab will outline the steps necessary to synthesize, download, and test the 3-bit ripple counter using VHDL.

All synthesis tools use files called "netlists." Netlists are simply text descriptions of how various circuit components are connected. The HDL design flow is similar to Schematic Capture, except that it deals with VHDL code instead of schematic drawings. Xilinx ISE can take VHDL and create a netlist based on your code.

Before starting the VHDL code that implements the 3-bit ripple counter, first take a look at its schematic diagram:



The BASYS boards have a programmable external clock of 100 MHz that can be access through pin 54. However, this frequency is far too fast for us to see the counter's output, so you need to use a clock divider to reduce the frequency to about 1 Hz, so the counter will count once per second.

The easiest way to implement a ripple counter is to use T-Flip Flops. When the "T" input is a logic 1, the output Q will toggle on every clock transition. When the "T" input is logic 0, the output Q will not change on clock transitions.

To implement the 3-bit ripple counter within VHDL, you will be using three VHDL blocks. The first block is the clock divider, the second block is a T-Flip Flop, and the third block is a decoder to map the output of the ripple counter to the 7-segment display. Then a top-level block that instantiates and interconnects the clock divider, flip flops and decoder is required. This top level block can be synthesized and downloaded into the FPGA, and the counter will immediately begin to count.

The Clock Divider

The VHDL code used to implement the clock divider is:

```
library IEEE;
use IEEE.std_logic_1164.all;
        entity Clock_Divider is
            port (
                CIN: in STD_LOGIC;
                COUT: out STD_LOGIC  );
        end Clock_Divider;
        architecture Clock_Divider of Clock_Divider is
             constant TIMECONST : integer := 84;
             signal count0, count1, count2, count3: integer range 0 to 1000;
             signal D: STD_LOGIC := '0';
        begin
             process (CIN)
             begin
              if CIN'event and CIN = '1' then
                  count0 <= count0 + 1;
                  if count0 = TIMECONST then
                  count0 <= 0;
                  count1 <= count1 + 1;
                  elsif count1 = TIMECONST then
                  count1 <= 0;
                  count2 <= count2 + 1;
                  elsif count2 = TIMECONST then
                  count2 <= 0;
                  count3 <= count3 + 1;
                  elsif count3 = TIMECONST then
                  count3 <= 0;
                  D <= not D;
                  end if;
                end if;
                COUT <= D;
             end process;
        end Clock_Divider;
```

You do not necessarily have to understand how this code works. The component is in its "entity" declaration, this component has only two ports, CIN and COUT. We will feed the 100 MHz external clock into CIN, and then a 1 Hz signal will come out of COUT. This frequency is adjustable, however, according to the following formula:

Output Frequency = 100000000 / (2 * (TIMECONST ^ 4))

Inside your VHDL code you can set the "TIMECONST" parameter to a different value according to your desired frequency.

Note that we are using "STD_LOGIC" declarations. STD_LOGIC allows you to create code that will behave more like a real-world circuit would, so in general most VHDL is written using STD_LOGIC signal declarations. The use of STD_LOGIC requires us to add the std_logic.1164 library, which is the first two lines of our code.

The T-Flip Flop

Here is the code for our T-Flip Flop:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity TFF is
   port (
      T: in STD_LOGIC;
      Q: out STD_LOGIC;
      TFF_CLOCK: in STD_LOGIC
   );
end TFF;
architecture TFF of TFF is
    signal D: STD_LOGIC := '0';

  begin

    process (TFF_CLOCK)  -- Execute process only when the clock changes
    begin

       if T = '0' then null;  -- no toggle, so don't do anything

       elsif TFF_CLOCK = '1' and TFF_CLOCK'event then
           D <= not D;  -- rising edge of clock and T = 1, so
                     -- toggle the output
       end if;

       Q <= not D;

    end process;
end TFF;
```

The entity port map shows three signals - the "T" input, the output "Q", and the clock input "TFF_CLOCK." You may be wondering why there is no inverted output "QN." As you will see later, it is unnecessary to include this output in our T Flip Flops when we implement our final design.
The signal "D" is a signal used to implement the flip flop operation. It is initially set to a value of '0'. When the clock transitions from low to high, and our "T" input is a logic '1', we then invert D and store it back into itself. However, if the clock transitions and our "T" input is a logic '0', then the process stops until the next transition, and no change on D occurs.
At the end of the process the inverse of D is assigned into Q, effectively implementing the T Flip Flop operation.

The 7-Segment Display

Develop the seven segment display code. Use the code in Lab 2 as an example.

The Ripple Counter

So, now we have the three main components needed for our ripple counter. The following code puts it all together:

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

        -- A 3-bit Ripple Counter
        entity ripple is
           port (
              CLOCK_IN: in STD_LOGIC;
              Q0: out STD_LOGIC;
              Q1: out STD_LOGIC;
              Q2: out STD_LOGIC;
              Q3: out STD_LOGIC;
              Q4: out STD_LOGIC;
              Q5: out STD_LOGIC;
              Q6: out STD_LOGIC
           );
        end ripple;
        architecture ripple of ripple is
            component TFF   -- component declaration for the T-Flip Flop
             port (
                T: in STD_LOGIC;
                Q: out STD_LOGIC;
                TFF_CLOCK: in STD_LOGIC
                );
            end component;

            component Clock_Divider  -- component declaration for the clock divider
             port (
                CIN: in STD_LOGIC;
                COUT: out STD_LOGIC
                );
            end component;

            component DEC_7seg  -- component declaration for 7 segment decoder
            port (
                  Din: in STD_LOGIC_VECTOR (2 downto 0);
                  Segments: out STD_LOGIC_VECTOR (6 downto 0)
                );
           end component;
    signal S0, S1, S2, S3, Mainclock: STD_LOGIC;  -- various intermediate signals
    signal F: STD_LOGIC_VECTOR (6 downto 0);
    signal G: STD_LOGIC_VECTOR (2 downto 0);

     begin

       S0 <= '1';  -- We assign signal S0 with '1', so that we can feed this logic
```

```
        -- value into all of the "T" inputs.

    -- Now instantiate the clock divider component, so that we can
    -- feed our ripple counter with a 1 Hz clock
        CLOCK: Clock_Divider port map (CLOCK_IN, Mainclock);

        -- Now instantiate the 3 T-Flip Flops needed to implement
        -- the 3-bit ripple counter
        TFF1: TFF port map (S0,S1,Mainclock);
        TFF2: TFF port map (S0,S2,S1);
        TFF3: TFF port map (S0,S3,S2);

          G(0) <= S1;
          G(1) <= S2;
          G(2) <= S3;

       --Now initiate the 7-segment display
        DISPLAY: DEC_7seg port map (G, F);

        Q0 <= F(0);
        Q1 <= F(1);
        Q2 <= F(2);
        Q3 <= F(3);
        Q4 <= F(4);
        Q5 <= F(5);
        Q6 <= F(6);

     end ripple;
```

As you can see, in order to connect the three flip flops together we use intermediate signals S0, S1, S2, and S3, and Mainclock.

Synthesizing the VHDL with Xilinx ISE

You are now ready to check the syntax of the code and to synthesize it as well. Open the Synthesize XST tree within the process dialog box and run "Check Syntax". Notice the output of this operation in the window at the bottom of the HDL editor.

If for some reason your file has an error in it, a red x would appear next to file in place of the green check. If this happens, right click on the filename and click on "Edit" to fix the problem.

Now we have to synthesise the circuit. First we need to define the pins that will be used. In the sources dialog box select Synthesis/Implementation. In the process dialog box open the tree of the Implement Design and then Translate. Select the Assign Package Pins Post-Translate, right click and “Run”. This will open the PACE utility.
Enter the locations of the pins by using the datasheet.

You can now run the Synthesis and Implementation. Select Synthesize in the process dialog box, click the right mouse button and select “*Run*”. Then select Implement Design in the process dialog box, click the right mouse button and select “*Run*”. This will translate the design, map it on to the FPGA, select the slices to be used and do the routing. Finally select the Generate Programming File in the process dialog box, click the right mouse button and select “*Run*”. This will generate the .bit file required by the FPGA.

Downloading and Testing

Now all that's left is to find the ripple.bit file that was generated, and download it to the BASYS board (using the same procedure as in last labs).