Objective:

The objective of this lab is to use the PROCESS construct in VHDL to implement sequential logic, implementing counters in different ways. Also, for demonstration on the DE10-lite board, a clock divider (again using PROCESS) is created to use a push button to produce a clock signal to drive a counter; but is slow enough for human to see.

Required tools and parts:

Intel Quartus Prime, ModelSim-Altera, Terasic DE10-Lite board

Important: For this and the subsequent labs, put your name and section number as a comment at the beginning all VHDL files and other materials that you submit.

Pre-lab Requirements:

Part 1: 4-bit Up/Down Counter

Provided for you is the template for counter.vhd with the following entity specification:

```
entity counter is
  port (
    clk   : in  std_logic;
    rst   : in  std_logic;          -- active high, asynch.
    up_n  : in  std_logic;          -- active low
    load_n: in  std_logic;          -- active low
    input : in  std_logic_vector(3 downto 0);
    output: out std_logic_vector(3 downto 0));
end counter;
```

Create and verify (functional simulation) a 4-bit up/down counter which functions as follows:

- If rst = '1', load_n = 'X', and up_n = 'X', the counter is asynchronously reset.
  I.e., output (3 downto 0) = "0000" ('X' represents "don't care").
- If rst = '0', load_n ='0', and up_n = 'X', output (3 downto 0) is synchronously loaded with input (3 downto 0).
- If rst = '0', load_n = '1', up_n = '0', the counter will synchronously count upward, incrementing the output by 1 at each clock transition. If the output is currently "1111", it will become "0000" at the next clock transition.
- If rst = '0', load_n = '1', up_n = '1', the counter will synchronously count downward, decrementing the output by 1 at each clock transition. If the output is currently "0000", it will become "1111" at the next clock transition.

You are free to implement the counter however you want, as long as you conform to the provided entity specification. Be aware that this counter can be implemented with very little code, so if your architecture description is getting long, consider a different way of implementing it.

Part 1 Pre-lab submission (on Canvas)

Include the following content in **`your-UFID/P1`** directory (see end of document for more information about submission formatting):

- VHDL code for Part 1: counter.vhd and counter_tb.vhd.
- A pdf file of the functional simulation, with added annotations to illustrate all operations (reset, load, up, down) are functioning correctly.

Part 2: 4-bit Gray code counter

Design and verify (functional simulation) a synchronous Gray code counter using a finite state machine, and using the provided template for gray2.vhd with the following entity specification:

```
entity gray2 is
  port (
    clk    : in  std_logic;
    rst    : in  std_logic;    -- active high, synchronous!
    output: out std_logic_vector(3 downto 0));
end gray;
```

Gray code is a numerical system where two successive values differ by only a single bit. Therefore, the binary sequence for the output (3 downto 0) of a 4-bit Gray code counter is defined as follows: (note that the corresponding hex value is given in the parenthesis)

0000 (0)
0001 (1)
0011 (3)
0010 (2)
0110 (6)
0111 (7)
0101 (5)
0100 (4)
1100 (C)
1101 (D)
1111 (F)
1110 (E)
1010 (A)
1011 (B)
1001 (9)
1000 (8)

If the current output of the counter is “1000”, the counter should output “0000” at the next clock.

- Use a single PROCESS with nothing except clock and reset in the sensitivity list to produce the state machine.
- The combinational logic to produce the output (3 downto 0) should be outside of the PROCESS.

Part 2 Pre-lab submission (on Canvas)

Include the following content in **`your-UFID/P2`** directory:

- VHDL code for Part 2: gray.vhd the modified gray_tb.vhd.
- A pdf file of the functional simulation.

Part 3: Clock Generator

To be able to see the output of the counters in real time using the DE10 board, you will need to control the counters using a button press (on the board) to generate a clock that is slow enough to see. To do this, you will create two entities: **clk_div** and **clk_gen**, with the following entity declarations.

Clock divider: converts an input clock to an output clock with a lower frequency.

```
entity clk_div is
 generic(clk_in_freq  : natural;  -- incoming clock frequency
         clk_out_freq : natural); -- desired output clock frequency
 port(
      clk_in  : in  std_logic;
      clk_out : out std_logic;
      rst     : in  std_logic);
 end clk_div;
```

- clk_div.vhd uses generics to specify the input and output clock frequencies. In other words, the code should be able to convert any input clock frequency to any (lower) output clock frequency.
- For this lab, you will use clk_div to converts the incoming 50 MHz board clock to a 1000 Hz output clock, with any duty cycle you want.
- Note: 1000 Hz clock results in a clock period is 1ms (and 1000 * 1ms = 1 second).

Clock generator: uses clk_div.vhd (PORT MAP) to generate a clk_out to control and view the counter outputs.

```
entity clk_gen is
 generic (
     ms_period : positive); -- amount of ms for button to be
                            -- pressed before creating clock pulse
 port(
      clk50MHz : in  std_logic; -- board clock
      rst      : in  std_logic; -- active high
      button_n : in  std_logic; -- active low
      clk_out  : out std_logic);
 end clk_gen;
```

Specification for clk_gen. vhd

- The clk_gen entity also uses generics. Clk_gen works by specifying how many milliseconds the button must be pressed to generate a clock pulse. Note for this lab, this value is 1000 milliseconds (i.e., 1 second).
- The input clock to clk_gen (named clk50MHz) is the 50 MHz board clock.
- Inside clk_gen, the clk_div component is used (i.e., PORT MAP) to reduce the clock frequency to 1000 Hz.
- Using a PROCESS statement, count 1000 Hz clock pulses (each is 1 millisecond) while the button is pressed down until 1000ms (i.e., 1 second) have elapsed,
  - At that point, clk_out will be high for one 1000 Hz clock period.
- If the button is continually held down, it should generate a pulse every second until the button is released.

Additional notes for clk_gen.vhd

- Note that it is not possible to guarantee that a generated clock pulse occurs after exactly 1s because the button may be pressed in the middle of the 1ms clock. Therefore, the actual time for the generated clock after the first button press will be between [1000ms, 1001ms). In other words, the first generated clock will occur anytime being 1000ms and 1001ms after the button was pressed.
- However, for repeated generations of the clock, it is possible to produce a clock after exactly 1000ms because you know that the button was already pressed at the beginning of the cycle.
- The provided testbench tests these times, so be aware that being off by a single cycle will cause assertion errors.

Finally, a basic testbench has been provided to help with clk_div. However, this testbench does not check for correctness. It just gives you a starting point for your own tests.

A testbench for clk_gen is also provided, which does check for correctness of both values and timing. As similar testbench will be used for grading, so make sure there are no errors messages. Also, make sure to test different ms times for the button press.

Part 3 Pre-lab submission (on Canvas)

Include the following content in **`your-UFID/P3`** directory:

- VHDL code for Part 3: clk_div.vhd, clk_gen.vhd, clk_div_tb.vhd, clk_gen_tb.vhd
- A pdf file of simulation screenshot of clk_div with a ratio of 2; simulation screenshot of clk_div with a ratio of 4

Part 4: Top Level

Read the code for the provided top_level entity (top_level.vhd) to determine what it does.

Part 4 Pre-lab submission (on Canvas)

Include the following content in **`your-UFID/P4`** directory:

- Summary of the top level entity - be specific. Determine how the counters get mapped onto the board by analyzing the provided top_level entity.

In-lab procedure (do as much as possible ahead of time):

1. Using Quartus, assign pins to each of the top_level.vhd inputs/outputs such that the signals are connected to the appropriate locations on the board. Note that the exact connections are purposely omitted so that you have to understand the top_level.vhd file. Make sure to add the 7-segment decoder code to your project.
2. Download your design to the board for Part A (up/down counter) and Part B (gray code counter), and test them for different inputs and outputs. Demonstrate the correct functionality for the TA.
3. For the quiz, be prepared to answer simple questions or to make simple extensions that your TA may request. There is no need to memorize the different packages. If you have done the pre-lab exercises, these questions should not be difficult.

Grade Breakdown

Note: In general, check this rubric against the associated rubric on the Canvas assignment page to ensure this is up to date.

Criteria	Points
----- Pre-lab -----	
Part 1 up/down counter	10 pts
Part 2 gray code counter	12.5 pts
Part3 clk_div	12.5 pts
Part3 clk_gen	12.5 pts
Top-Level description	2.5 pts
----- In-Lab -----	
Demo up/down counter	20 pts
Demo gray code counter	20 pts
Quiz	10 pts