COEN 313 Project Report

COEN 313 – DIGITAL DESIGN

Concordia University

November 2023

Table of Contents

1. Introduction

2. Main Folder

2.1 Conceptual Diagram

3. VHDL File & Simulation

3.1 VHD file

3.2 TestBench for VHD file

3.3 DO file for VHD file

3.4 Simulation with Testbench

3.5 Simulation with DO file

4. Xilinx Vivado

4.1 Elaborated Design

4.2 Implemented Design

4.3 Implementation Logs

5. Limits and shortcomings of the circuit

Introduction

The objective of this project is to use the designing patterns and techniques taught in COEN 313 to build a circuit with the following characteristics:

1) The system must have 4 inputs:

    A) A tracker for individuals entering the room (X).
    B) A tracker for individuals leaving the room (Y).
    C) A clock (for the register).
    D) A reset option to bring the register value back to 0.

2) The system must accurately keep track of the people entering and leaving the room and must therefore increment the counter by 1 when someone is entering and decrement the counter when someone is leaving the room.
3) The maximum occupancy of the room is 63 and an output (Z) must light up when the maximum occupancy is reached.

Main Folder – Conceptual Diagram

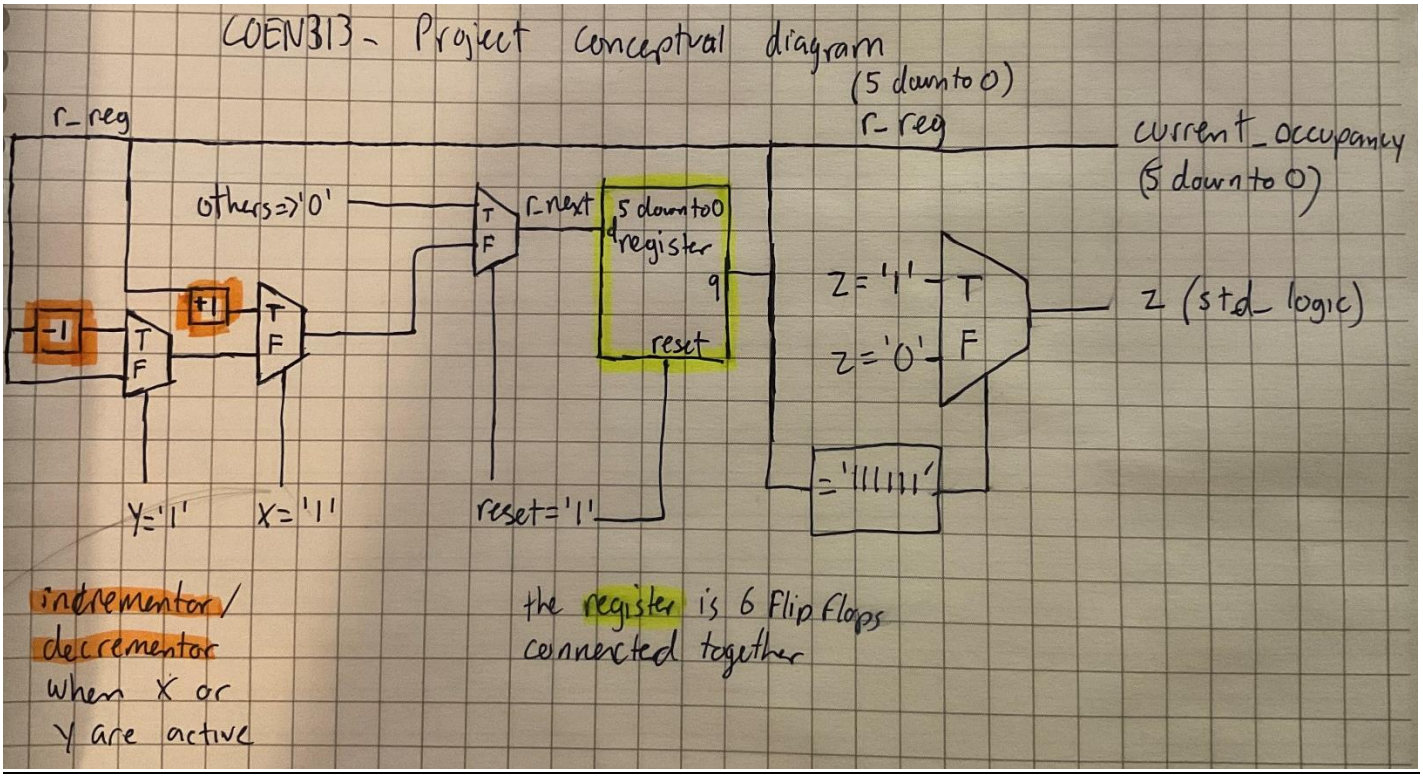
I opted for a single register to keep track of the occupancy as a 6 bit counter can count from 0 to 63 (unsigned 111 111 is 63).

The priority if statements decides the value of r_next as follows:

If reset is 1, then r_next and r_reg are set to others => '0' to start the counter from scratch.

Once r_reg is 0, we can increment the register by one on the rising edge of the clock when x = '1' (meaning someone walked into the room). By the same logic, we can decrement the counter by '1' when someone leaves the room (and y goes to 1). If x and y are both 0, then the occupancy of the room didn't change and the value of r_reg remains the same.

I decided to have an output that displays the current occupancy (again as a 5 downto 0 display) so the occupancy is known at all time. Finally, when the inner register reaches its max value 111111 at 63, 1 is assigned to z and the circuit displays that the max occupancy as been reached.



ModelSim – VHD code

I first declared my entites (inputs and outputs of the system). The logic behind each inputs and outputs are explained in the previous section.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
--declaring the inputs and outputs of the circuit
entity COEN313Project is
    port(
        x: in  std_logic;
        y: in std_logic;
        reset: in std_logic;
        clk : in std_logic;
        z : out std_logic;
        current_occupancy: out std_logic_vector (5 downto 0)
    );
end entity COEN313Project;
```

I declare my architecture and declare the signals needed to establish a 6 bit register.

The current state logic block initializes my circuit to 0 when reset is activated, and assigned r_next to r_reg on the rising edge of the clock. The next state logic either increment r_next by one if x is active, decrement r_next by one if y is active or stays as is if both x and y are 0.

```
--architecture
architecture arch of COEN313Project is
    -- declaring pre and post register signals (to store the 0 to 63 current occupancy)
    signal r_reg: unsigned(5 downto 0); -- max at 0111111 = 63 in unsigned, will trigger z at 0111111 (63)
    signal r_next: unsigned(5 downto 0);

begin
    --current state logic
    process(clk,reset) -- updating the occupancy in the register
    begin
        if (reset = '1') then r_reg <= (others=>'0');
        elsif (clk'event and clk = '1') then r_reg <= r_next;
        end if;
    end process;
--next state logic
    r_next <= (others =>'0') when reset='1' else
        r_reg + 1      when x='1' else --increment when someone enters the room
        r_reg - 1      when y='1' else --decrement when someone leaves the room
        r_reg;                         --no changes to the counter when no one enters/leaves

```

Finally, the last step is to assign the output logic. Z is assigned to 1 when the inner register is 111111, meaning the max occupancy has been achieved. The current occupancy is also kept track of through an output statement.

```
--output logic

      z <= '1' when (r_reg="111111") else '0'; --displays when the room reaches full occupancy
      current_occupancy <= std_logic_vector(r_reg); --display current occupancy

   end architecture arch;
```

ModelSim – TestBench code

The first part of the TestBench is to declare the components of the testbench, the list is the same as the entity statement in the vhd code.

```
ARCHITECTURE tbarch of COEN313Project_tb IS

    component COEN313Project is
        Port (
            x: in  std_logic;
            y: in std_logic;
            reset: in std_logic;
            clk : in std_logic;
            z : out std_logic;
            current_occupancy: out std_logic_vector (5 downto 0)
        );
    end component;
```

Then, I declare the period of the clock as a constant and the runtime of the circuit (allowing enough time for the occupancy to reach 63 to see if the output logic works as intended). The signals are subsequently initialized to 0.

```
CONSTANT max_simulation_time : TIME := 720 ps;
CONSTANT clk_period          : TIME := 10 ps;

signal x: std_logic :='0';
signal y: std_logic :='0';
signal reset: std_logic :='0';
signal clk: std_logic :='0';
signal z: std_logic :='0';
signal current_occupancy: std_logic_vector (5 downto 0);
```

The next step is to link the components to the signals through a port map and to define the clock process.

```
33
34 begin
35     PROJECT : COEN313Project port map (
36         x => x,
37         y => y,
38         reset => reset,
39         clk => clk,
40         z => z,
41         current_occupancy => current_occupancy);
42
43     clk_gen: PROCESS
44     VARIABLE sim_time : TIME;
45     BEGIN
46         sim_time := 0 ps;
47         WHILE sim_time <=  max_simulation_time LOOP
48             clk <= '0';
49             WAIT FOR 5 ps;
50             clk <= '1';
51             WAIT FOR 5 ps;
52             sim_time := sim_time + clk_period;
53         END LOOP;
54         WAIT;
55     END PROCESS;
56
```

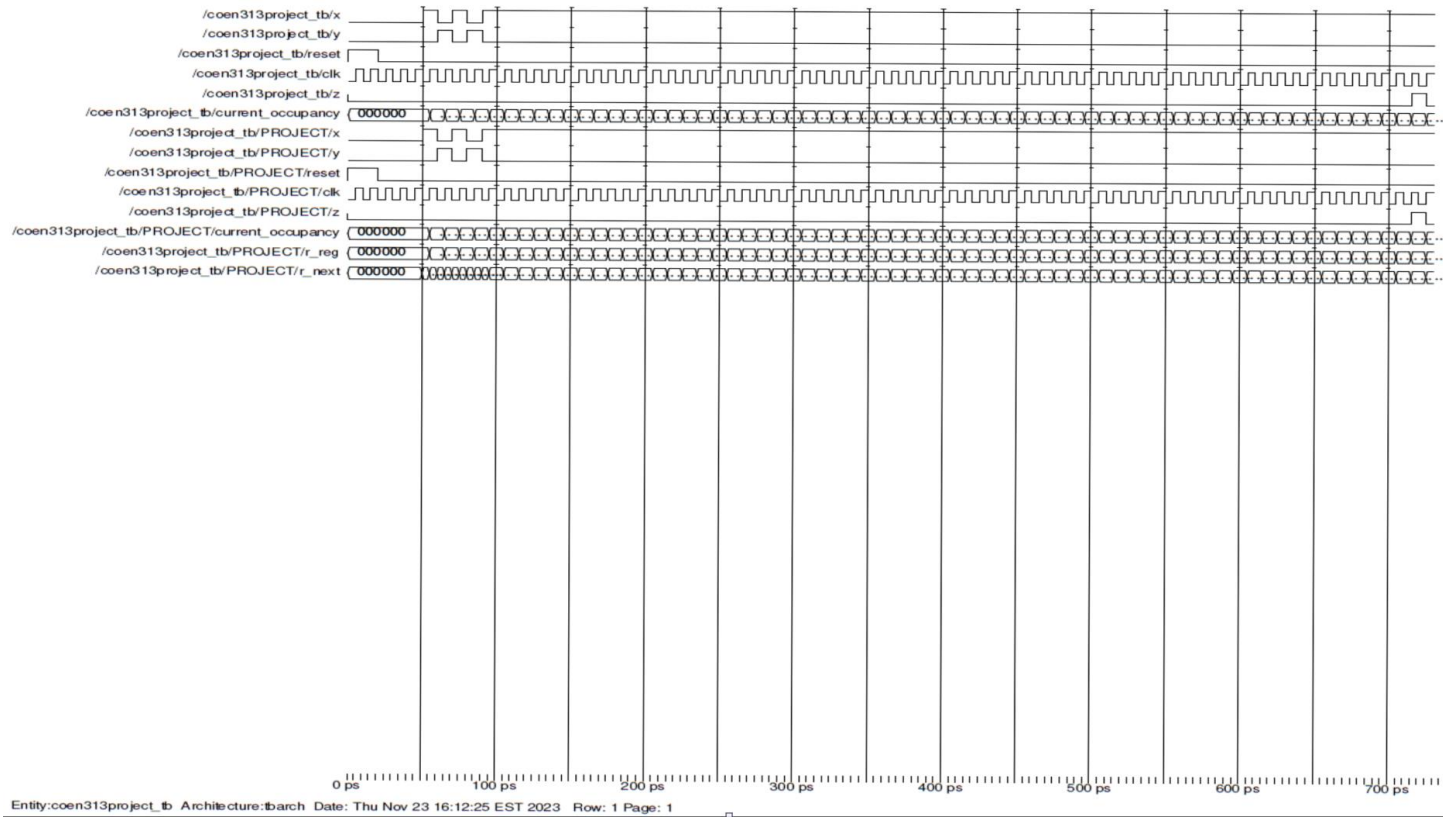
The last step is to declare the input process (we want to make sure that the counter increment as intended when x is 1 and decrement as intended with y is 1). We also want to insure that the output Z correctly displays 1 when the maximum occupancy 111111 is reached. (Hence the wait statement after forcing x = '1').

```
57
58      input_process: process
59      begin
60          reset<= '1'; wait for 20 ps;
61          reset<= '0'; wait for 20 ps;
62          x <= '0'; y <= '0'; wait for 10 ps;
63          x <= '1'; y <= '0'; wait for 10 ps;
64          x <= '0'; y <= '1'; wait for 10 ps;
65          x <= '1'; y <= '0'; wait for 10 ps;
66          x <= '0'; y <= '1'; wait for 10 ps;
67          x <= '1'; y <= '0'; wait for 10 ps;
68          x <= '1'; y <= '0'; wait for 10 ps;
69          wait;
70      end process;
71  END tbarch;
```

The DO file provides the same statements to test for each edge cases (increment on x =1, decrement on y =1 and z =1 when r_reg = 111111) and is included for completeness but is not necessary since it is redundant with the testbench.

ModelSim – TestBench wave

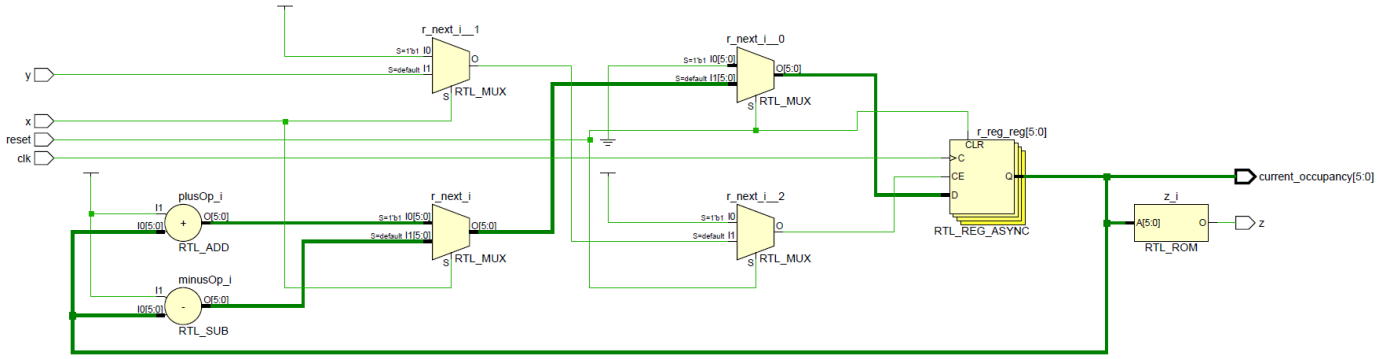
The pdf file is very wide (because the system had to increment the counter 65 times, considering we decremented the circuit twice to ensure that y works as intended) and I wasn't sure how the 63 6-bits values could fit on a single page. I apologize for the lack of clarity but counting the incrementations correctly assigns z to 1 when r_reg reaches 111111. The clock continuously ticks as intended and x is kept at 1 so that the counter increment more rapidly to 63. One drawback of this design is that the clock could increment more than once for a single person if a single person stayed in front of the detector for two rising edges clock cycles.



The wave description obtained through modelsim is also available with the do file and is added as complement to the testbench wave.

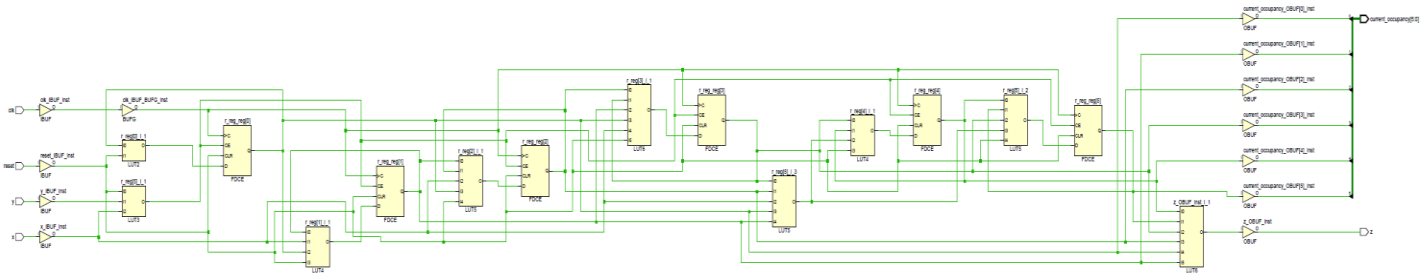
Xiinx Vivado – Elaborated diagram

The Elaborated diagram provided by Xilinx after Synthesis looks similar to my conceptual diagram where the x and y signal MUXs decides the r_next logic which will be stored in r_reg on the clock rising edge.



Xiinx Vivado – Implemented diagram

The implemented diagram shows every flip-flops “individually” linked to the inputs X Y clk and reset and the output Z is set to 1 when all the flip flops are 1. Individually, the state of the flip-flops determines the current_occupancy output.



Xiinx Vivado – Log file

The log file provided by Vivado doesn’t show any critical error and properly compiles. The log file divides the implementation into the following phases:

Starting Placer task:

1) Placer Initialization
2) Global Placement
3) Detail Placement
4) Post-Placement optimization

Starting Routing task:

1) Build RT design
2) Router Initialization
3) Initial Routing
4) Rip-up and reroute
5) Delay and Skew optimization
6) Post hold fix
7) Route finalize
8) Verifying Routed nets
9) Depositing Routes

Limits and shortcomings of the circuit

In conclusion, the circuit works as intended (the incrementation and decrementation by one when x and y are active work), the current occupancy properly displays and Z is triggered when the occupancy reaches 63 as intended. One of the shortcoming and challenge of this circuit is timing the clock in such manner that a single input X only triggers the counter to increment once (the counter could potentially go up twice if a single input X remains on for two rising edges of the clock).