COEN 313 Project Report

COEN 313 – DIGITAL DESIGN

Concordia University

© Mathias Desrochers

(40241734)

June 2024

Table of Contents

Introduction

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

- Inputs:

- Tracker for individuals entering the room (X): Detects and counts people entering the room.
- Tracker for individuals leaving the room (Y): Detects and counts people leaving the room.
- Maximum Occupancy : set the maximum occupancy
- Reset option: Resets the register value to 0.

- Outputs:

- Z Led : lit up if the number of people in the room is equal to the maximum occupancy

- Functionality:

- The system must accurately track the number of people entering and leaving the room.
- Increment the counter by 1 when someone enters.
- Decrement the counter by 1 when someone leaves.

- Maximum Occupancy:

- The maximum occupancy is a user-defined input with a limit of 63.
- An output (Z) must light up when the maximum occupancy is reached.

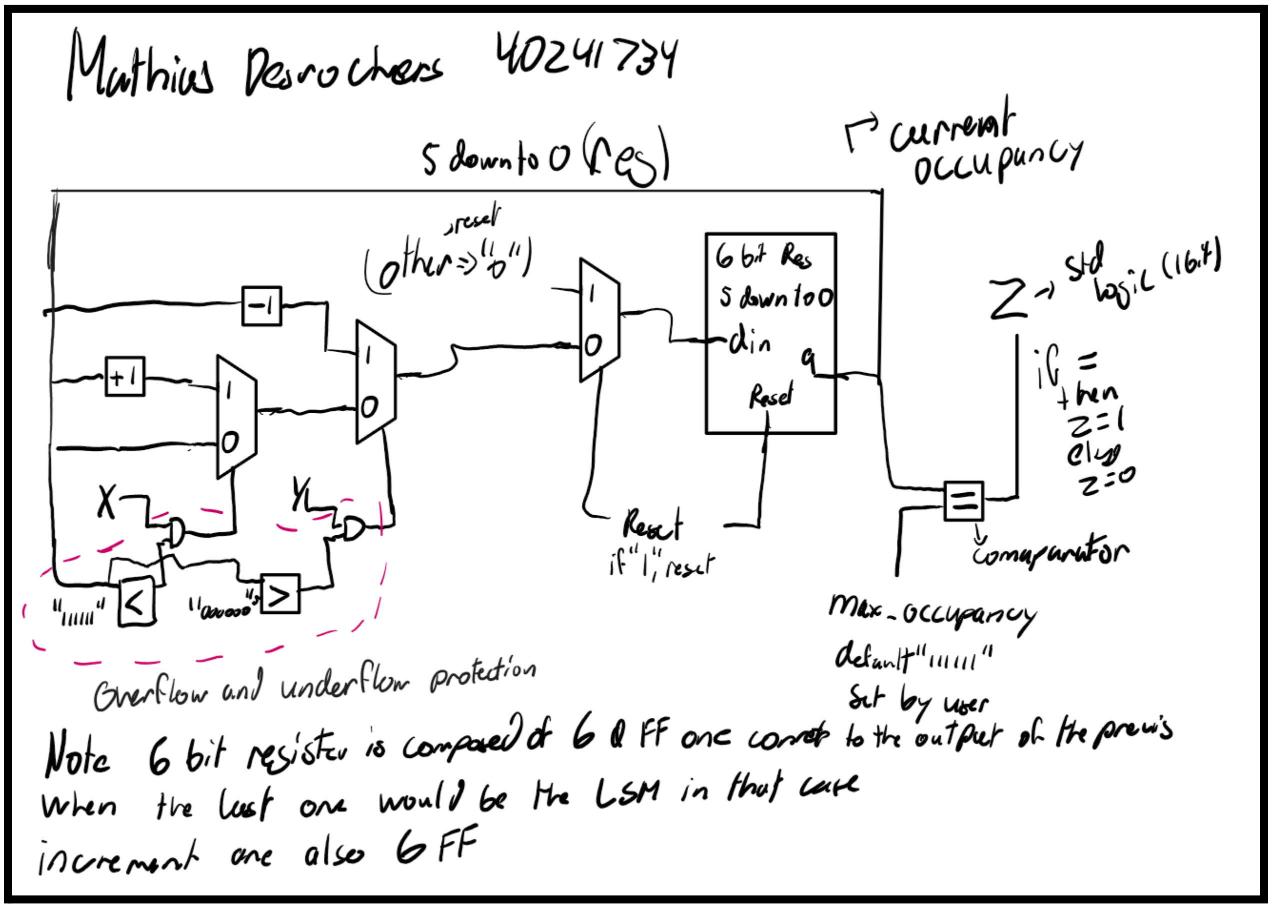
- Overflow and Underflow Protection:

- The system should include mechanisms to protect against overflow and underflow conditions.

Main Folder – Conceptual Diagram

A single register is used to keep track of the maximum occupancy. This is because a 6-bit counter can count from 0 to 63 (since the unsigned value of 111111 is 63), therefore we can define it as the register as a 5 down to 0 logic vector.

Figure 1, Conceptual Diagram


Mathias Desrochers 40241734
5 downto 0 (res)
current occupancy
(others=>"0")
reset
6 bit Res
5 downto 0
din
q
Reset
-1
+1
1
0
X
Y
<
>
"111111"
"000000"
Reset
if "1", reset
Z -> std logic (1bit)
if = then Z=1 else Z=0
=
comparator
max_occupancy default "111111" set by user
Overflow and underflow protection
Note 6 bit resistor is composed of 6 D FF one connect to the output of the previous when the last one would be the LSM in that case
increment one also 6 FF

The following is a translation of the Conceptual diagram into text:

We have a 6-bit counter (count) initialized to 0 and a 6-bit variable (max_occ) representing the default maximum occupancy, initially set to 63 “111111”. If we wanter to show it on a lower level, we could use 6 D Flip Flop to implement it.

We have a mux that responds to changes in the reset.. When the reset is activated (set to 1), the counter is reset to “000000”.

When an individual enters the room (x is set to 1) and the counter is less than 63, the counter increments by 1 by using a Mux that lets the counter value pass through the increment or not. Making sure that the counter is less than 63 “111111” is a choice to react to overflow, it is one way to do it, as the count will just stop to increment once the maximum is reached. We do that by using an AND gate between the Input X and the comparison of the counter value and the maximum value of the counter.

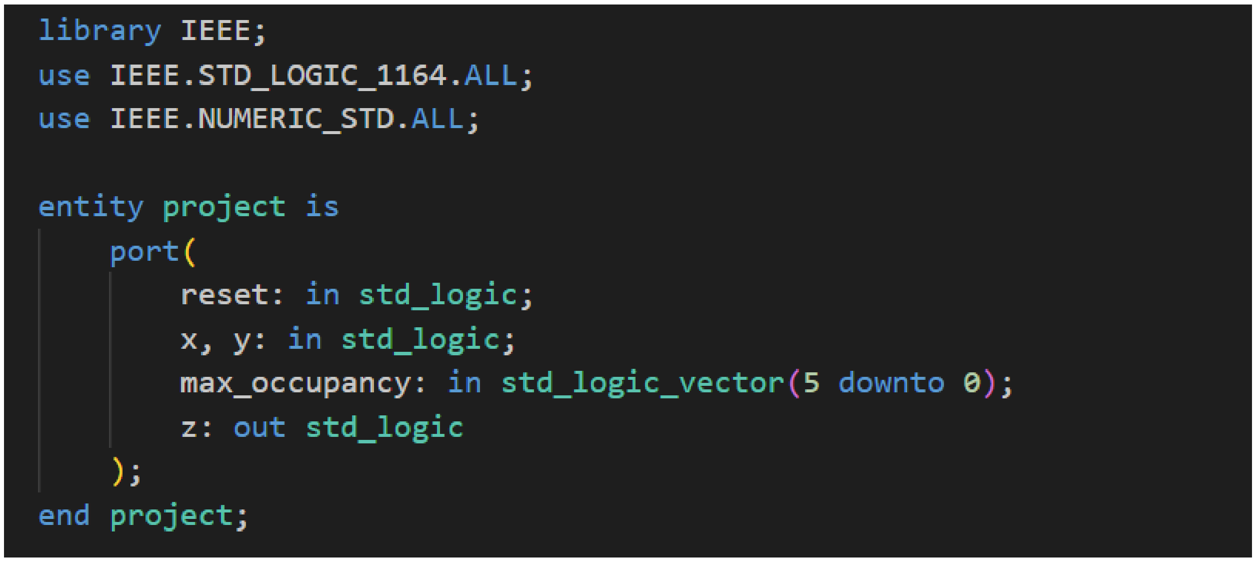
When an individual leaves the room (y is set to 1) and the counter is greater than “000000”, the counter decrements by 1.Again the comparison is for underflow.

Finally, if the counter value is equal to the user defined max occupancy, the Z should be one, else it is 0.

ModelSim – VHD code

I first declared my entities (inputs and outputs of the system). The logic behind each input and output are explained in the intro.

Figure 2, Entity declaration



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

entity project is
    port(
        reset: in std_logic;
        x, y: in std_logic;
        max_occupancy: in std_logic_vector(5 downto 0);
        z: out std_logic
    );
end project;
```

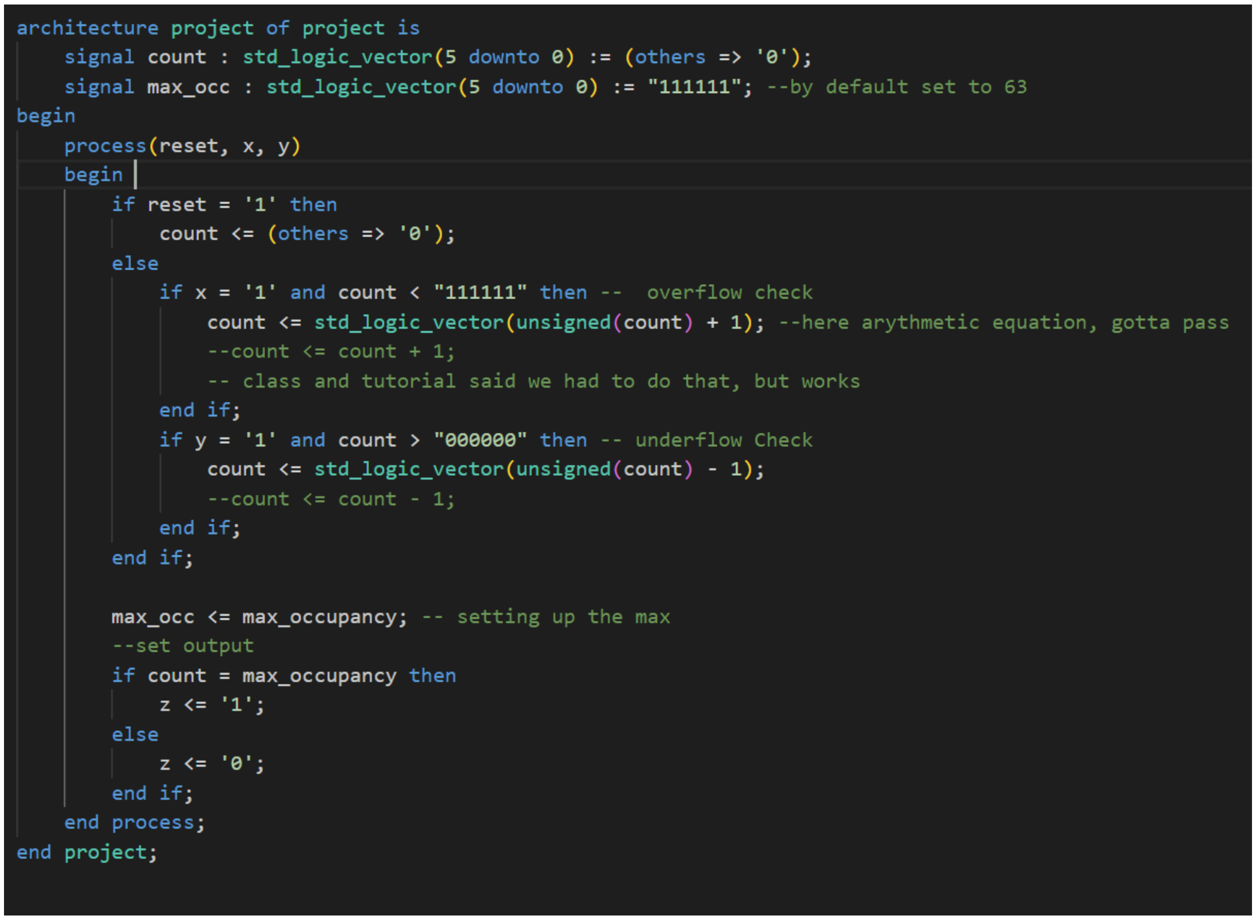
I declare my architecture and declare the signals needed to establish a 6 bit register.

The current state logic block initializes my circuit to 0 when reset is activated, else it checks X and the overflow, if there are both true, we can increment or not, Same for the Y input with the underflow. Also to use the Arithmetic equation, we need to change it to unsigned and then back to the logic vector.

Then I assign the user defined Maximum occupancy, or set it to 63 by default.

Finally, the last step is to assign the output logic. Z is assigned to 1 when the inner register is equal to the maximum occupancy, meaning the max occupancy has been achieved.

Figure 3, architecture

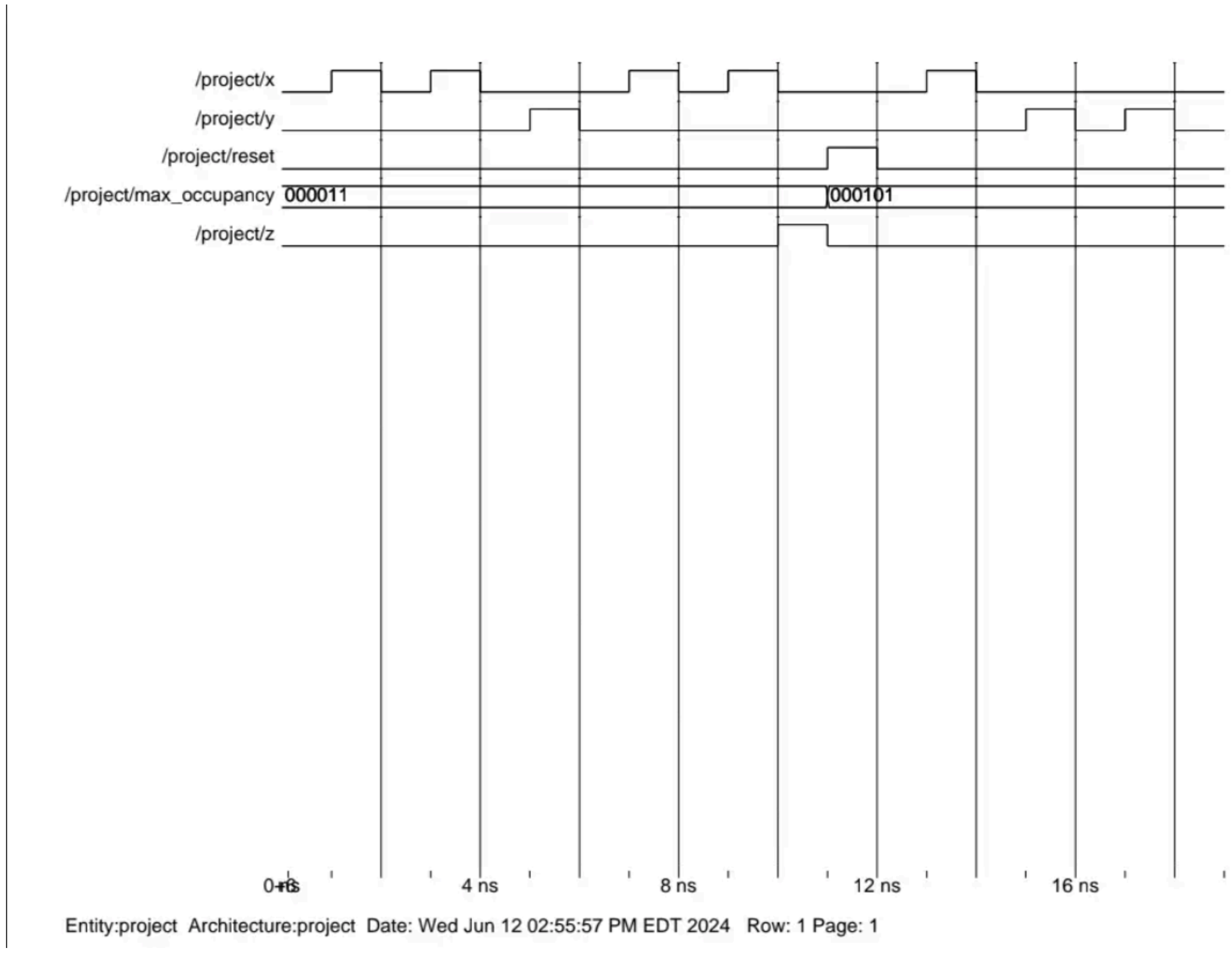


```vhdl
architecture project of project is
    signal count : std_logic_vector(5 downto 0) := (others => '0');
    signal max_occ : std_logic_vector(5 downto 0) := "111111"; --by default set to 63
begin
    process(reset, x, y)
    begin
        if reset = '1' then
            count <= (others => '0');
        else
            if x = '1' and count < "111111" then --  overflow check
                count <= std_logic_vector(unsigned(count) + 1); --here arythmetic equation, gotta pass
                --count <= count + 1;
                -- class and tutorial said we had to do that, but works
            end if;
            if y = '1' and count > "000000" then -- underflow Check
                count <= std_logic_vector(unsigned(count) - 1);
                --count <= count - 1;
            end if;
        end if;

        max_occ <= max_occupancy; -- setting up the max
        --set output
        if count = max_occupancy then
            z <= '1';
        else
            z <= '0';
        end if;
    end process;
end project;
```

ModelSim – TestBench code

Here for simulation To ensure a good simulation, I first wrote a do file, added the waves, to test all case scenario: underflow, increment, decrement and when the counter is equal to the Maximum occupancy :

Figure 4, Wave simulation

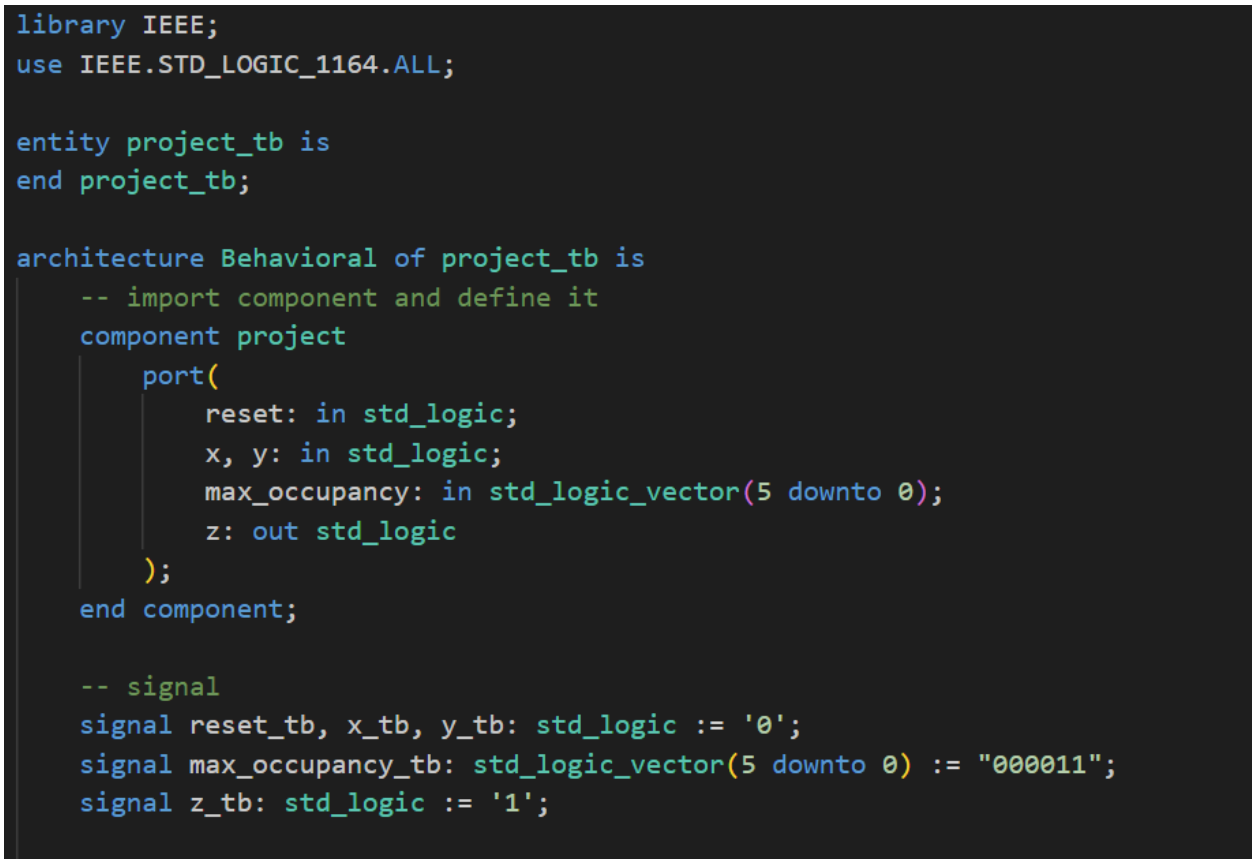


ModelSim – TestBench wave

For a better test Bench using ModelSim full capabilities, I create a project with the main Vhdl file and a test bench one “projecttb.vhd”

This code imported the main entity as a component and we gave input to try :

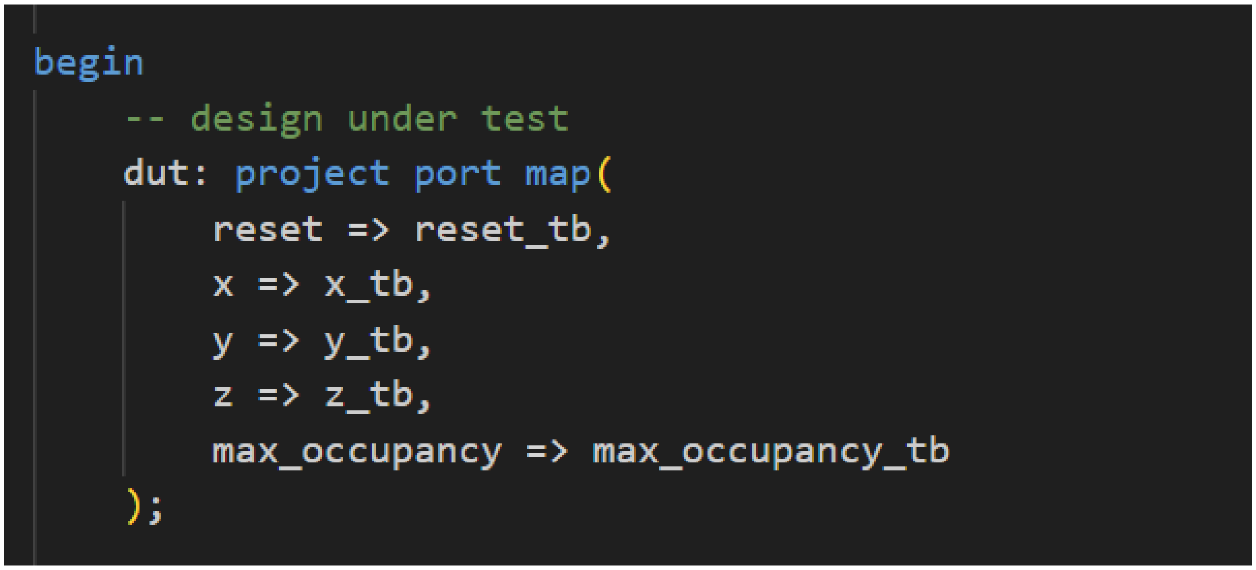
Figure 5, Test bench entity declaration



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

entity project_tb is
end project_tb;

architecture Behavioral of project_tb is
    -- import component and define it
    component project
        port(
            reset: in std_logic;
            x, y: in std_logic;
            max_occupancy: in std_logic_vector(5 downto 0);
            z: out std_logic
        );
    end component;

    -- signal
    signal reset_tb, x_tb, y_tb: std_logic := '0';
    signal max_occupancy_tb: std_logic_vector(5 downto 0) := "000011";
    signal z_tb: std_logic := '1';
```

Then we define the design under tree to link all the input, like port map:

Figure 6, DUT

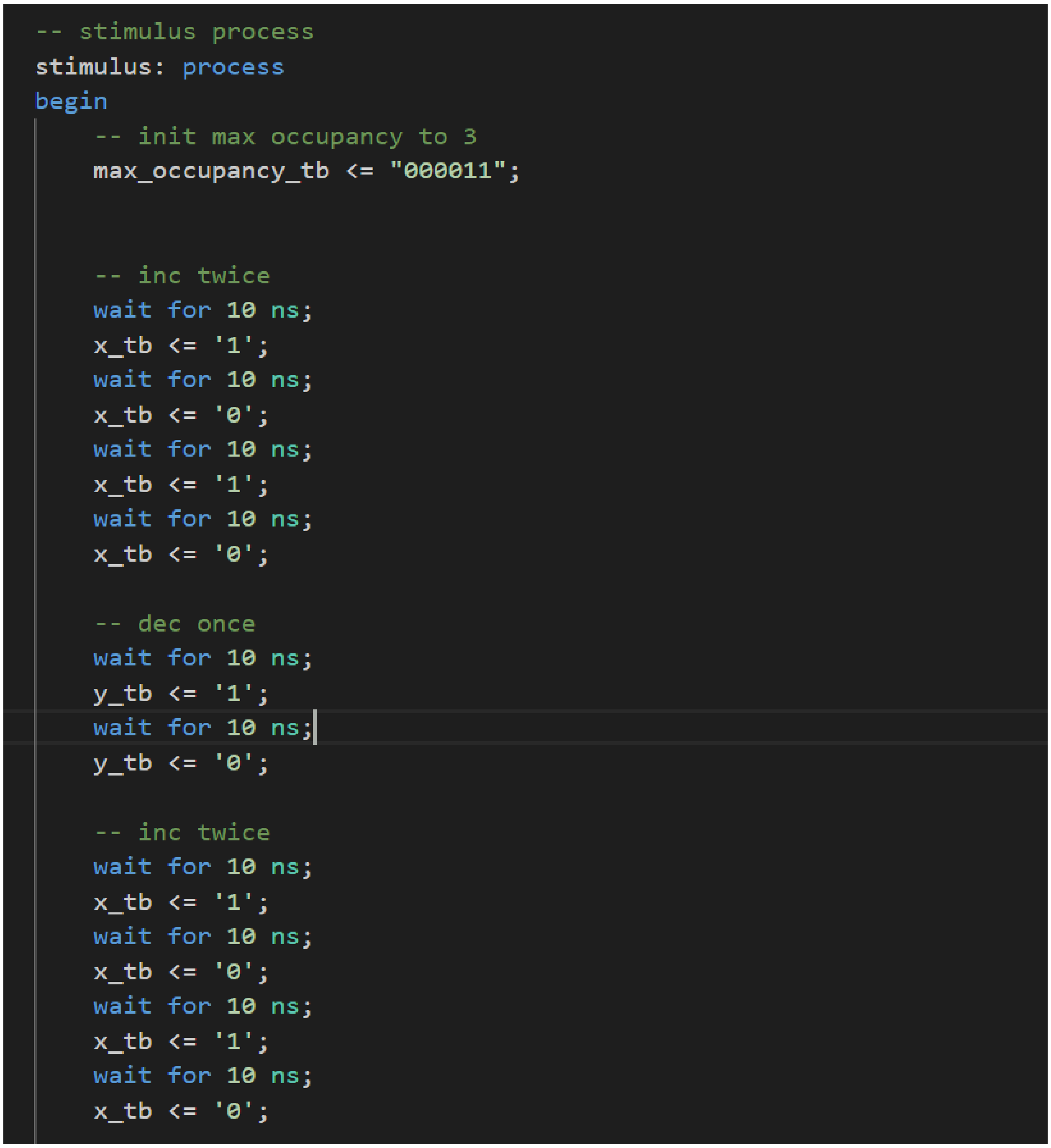


```
begin
    -- design under test
    dut: project port map(
        reset => reset_tb,
        x => x_tb,
        y => y_tb,
        z => z_tb,
        max_occupancy => max_occupancy_tb
    );
```

Then we start the stimulus process to test different inputs :

First round of input test the increment and the decrement by doing +2, -1 +2 and it should equal the chosen max occupancy of 3, thus putting the Z to 1

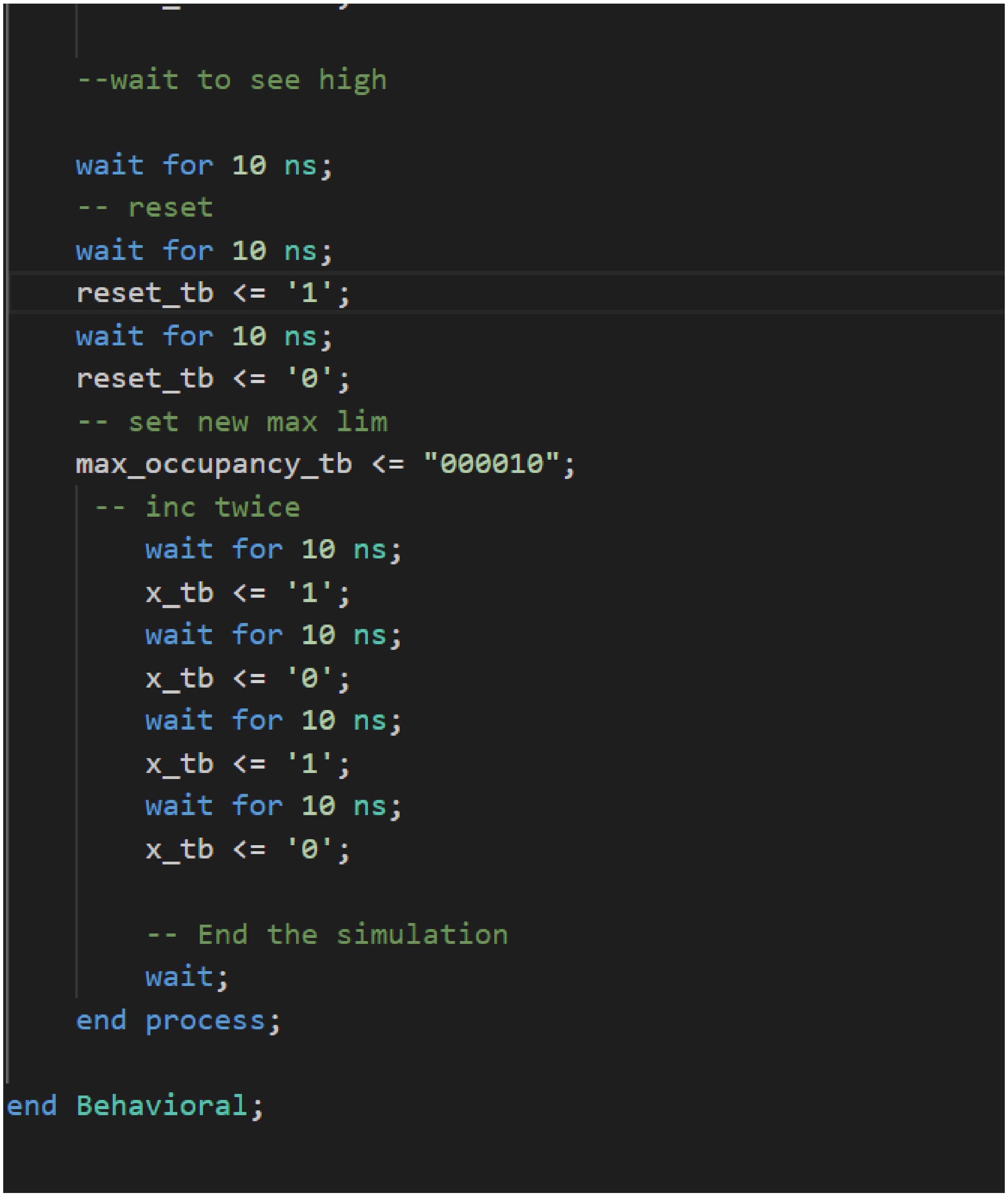
Figure 7, Test bench stimulus



```vhdl
-- stimulus process
stimulus: process
begin
    -- init max occupancy to 3
    max_occupancy_tb <= "000011";


    -- inc twice
    wait for 10 ns;
    x_tb <= '1';
    wait for 10 ns;
    x_tb <= '0';
    wait for 10 ns;
    x_tb <= '1';
    wait for 10 ns;
    x_tb <= '0';

    -- dec once
    wait for 10 ns;
    y_tb <= '1';
    wait for 10 ns;
    y_tb <= '0';

    -- inc twice
    wait for 10 ns;
    x_tb <= '1';
    wait for 10 ns;
    x_tb <= '0';
    wait for 10 ns;
    x_tb <= '1';
    wait for 10 ns;
    x_tb <= '0';
```

Finally we test the reset and setting a new Maximum occupancy :

Figure 8, Stimulus second part

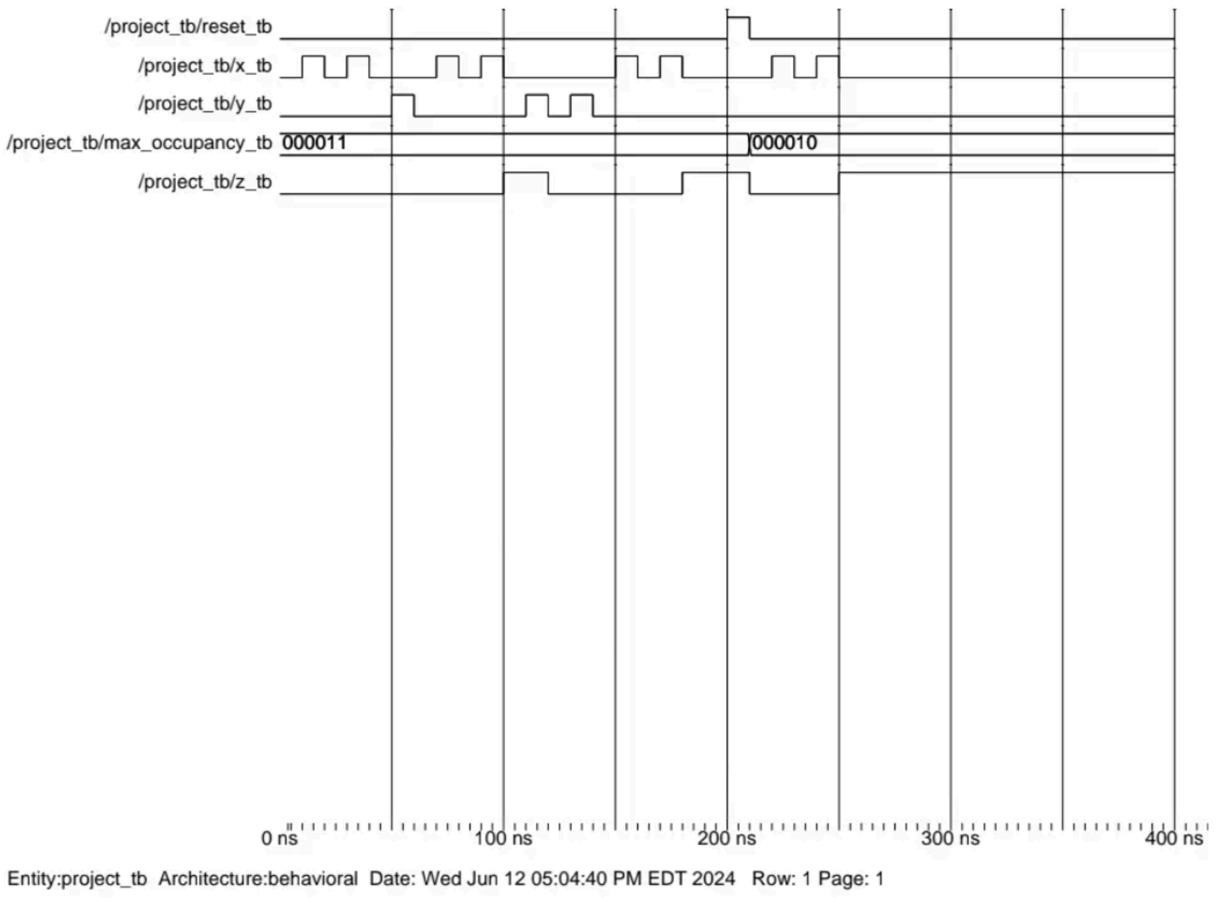


```
    --wait to see high

    wait for 10 ns;
    -- reset
    wait for 10 ns;
    reset_tb <= '1';
    wait for 10 ns;
    reset_tb <= '0';
    -- set new max lim
    max_occupancy_tb <= "000010";
     -- inc twice
        wait for 10 ns;
        x_tb <= '1';
        wait for 10 ns;
        x_tb <= '0';
        wait for 10 ns;
        x_tb <= '1';
        wait for 10 ns;
        x_tb <= '0';

        -- End the simulation
        wait;
    end process;

end Behavioral;
```

There wasn't a need to test Underflow and Overflow as we did it using a do file and it works as expected.

The output wave was the following :

Figure 9, Test bench, waves



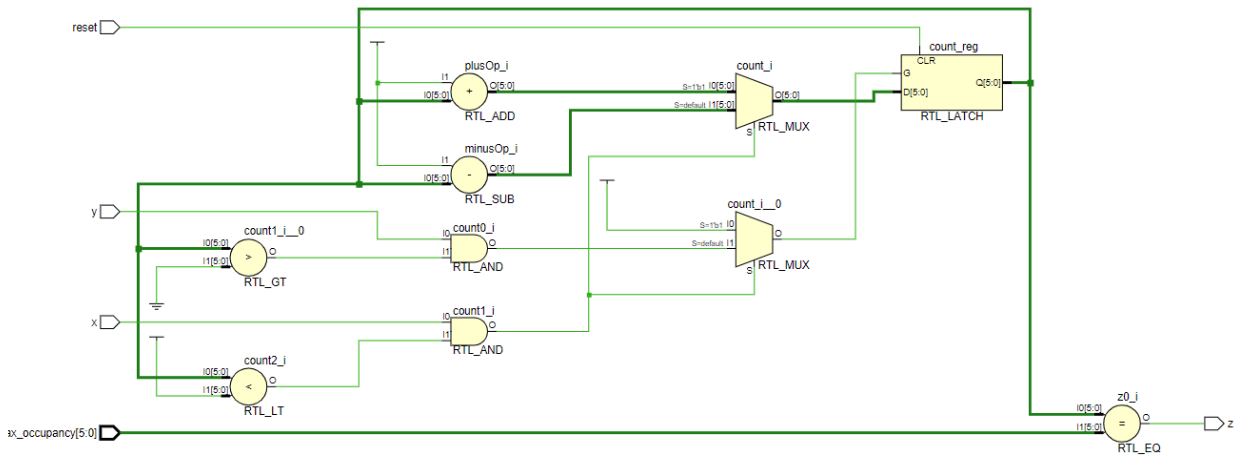
Which reacts exactly as expected as we can see from the output being high when the count is equal to the set maximum occupancy.

Here I could have tested the counter up to the theoretical limite of 63, but i believe the logic is very strong and predictable and therefore there isn't a need to set the maximum occupancy to 63 as it will react the same if we set it to 3.

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 counter logic which will be stored in counter when an input is registered

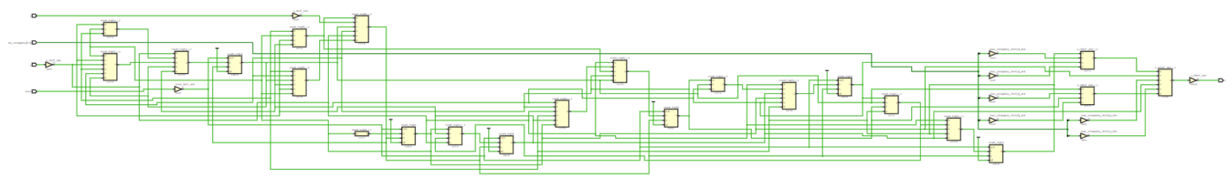
Figure 10, Elaborated design



Xiinx Vivado – Implemented diagram

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

Figure 10, Implemented design



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, given the project, I believe all the essential points were covered. One of the main issues is overflow or underflow of the 6 bits counter and I decided to just stop the incrementation when it reaches the maximum value but there are other ways to address the problem, such as rounding around or making the counter bigger, but they weren't in the scope of the project. Therefore I believe the project was a success since the vhdl code was perfectly compiled and tested on a workbench and with a .do file and in every case the system reacted as expected.