Design & Simulation
of an occupancy digital system counter

Kevin Mandiouba
ID: 40243497

COEN 313
Digital Systems Design II
Section AA

Due date: June 18th, 2024

I certify that this submission is my original work and meets the Faculty's Expectations of Originality

Kevin Mandiouba

1. Abstract

In this project, we will design a digital occupancy counter using VHDL in Xilinx Vivado. A digital occupancy counter is a digital system designed to count the number of people entering and leaving a room. This counter increments when an entry signal is received and decrements with an exit signal. The maximum occupancy of the room used for this project is limited to 63 people.

The aim of this project is to let the students apply their knowledge in digital system design and VHDL programming to real-world scenarios. Through this project, students will gain experience in designing, simulating, and verifying digital systems, enhancing their practical skills and understanding of theoretical concepts learned in class and touched during labs.

2. Procedure

For this project students had to provide a conceptual diagram of their occupancy counter. The conceptual diagram drawn for our design is represented in Figure 1. Our counter is comprised of four inputs, two outputs, an incrementor, a decrementor, three comparators, two registers (D Flip-flops), two 2-to-1 MUXs and one 4-to-1 MUX.

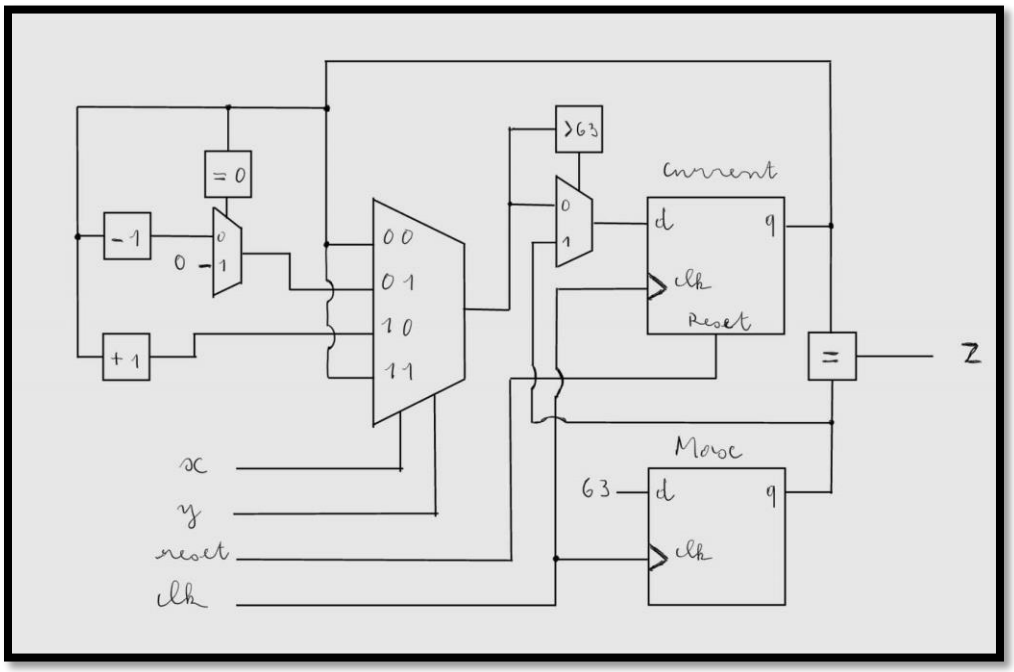

Figure 1: Conceptual Diagram of an occupancy system counter

2.1. Inputs

- **X**: entry signal activated when a person enters the room.
- **Y**: exit signal, activated when a person leaves the room.
- **Reset**: resets the current occupancy count to zero.
- **Clk**: clock signal, helps us synchronizing the system.

2.2. Outputs

- **Current occupancy**: The number of people present in the room.
- **Z**: The maximum occupancy of the room.

2.3. Incrementor and decrementor

- **Incrementor (+1)**: increment the current occupancy count by 1 when X is activated, and Y is not activated.
- **Decrementor (-1):** decrement the current occupancy count by 1 when X is not activated, and Y is activated.

2.4. Comparators

- **Comparator 1 (current = 0)**: prevents the current occupancy to decrement if the room is empty.
- **Comparator 2 (current > 63)**: prevent the current occupancy to increment past the room's max occupancy.
- **Comparator 3 (current = max)**: activates the output Z, which represent a red light showing that the room is currently full.

2.5. Registers

- **Current**: holds the current occupancy count.
- **Max**: holds maximum occupancy.

In this project, students could follow instructions from documents named *Digital logic simulation and synthesis using Modelsim, precision RTL* [1]*, and Xilinx ISE* and *Quickstart Guide to the Nexys-A7100T FPGA board and Xilinx Vivado* [2] to help them compile, debug, synthesize and simulate their design. The VHDL design and testbench codes written for this project are represent in Figure 3 and Figure 4 in the Appendix. We will discuss more about them in the Results and Discussion.

3. Results and Discussion

The code written for this project respects the design of our conceptual diagram. All restrictions our project needs to adhere to were incorporated. The testbench used for testing our design tests it in many different ways. The simulation for our project using Xilinx Vivado is represented in Figure 2. The elaborated and synthesis schematics for this design is represented in Figure 5 and Figure 6 respectively.

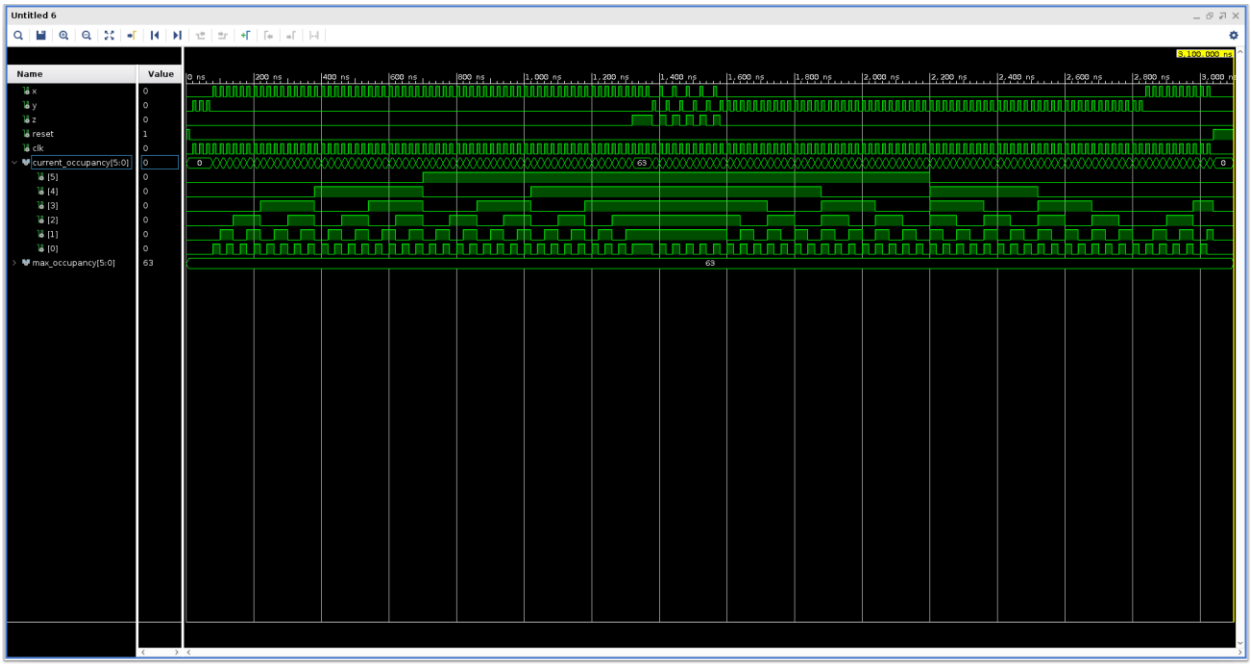

Figure 2: Project simulation

First, we test the functionality of the reset signal by ensuring that it correctly initializes the current_occupancy to zero regardless of its previous state.

Next, we test the increment functionality. We simulate multiple entry signals (x) to verify that the counter correctly increments the current_occupancy value with each entry. This includes testing the boundary condition where the counter should not exceed the maximum occupancy of 63. The testbench runs a series of 65 entry signals to confirm that the counter stops incrementing at 63.

The decrement functionality is also thoroughly examined. We simulate multiple exit signals (y) to ensure the counter correctly decrements the current_occupancy value. This includes testing the boundary condition where the counter should not drop below zero. The testbench runs a series of exit signals when the counter is at zero to confirm that the value does not go negative.

Finally, the overall performance and reliability of the occupancy counter are evaluated. We check that all signals are properly synchronized with the clock (clk) and that the output signal (z) correctly indicates when the counter reaches the maximum occupancy.

Other files supporting the documentation of this project are attached in the ZIP file for this report, such as log files and others test results using Modelsim.

4. Conclusions

The results obtained for this project demonstrate that our design works has intended. The current occupancy increments when the entry signal X is activated and decrements when the entry signal Y is activated. The room's current occupancy cannot go past 63 people and cannot go below 0.

5. References

[1] Department of Electrical and Computer Engineering, *DIGITAL LOGIC SIMULATION AND SYNTHESIS USING MODELSIM, PRECISION RTL , AND XILINX ISE,* Concordia University, 2018.

[2] *Quickstart Guide to the Nexys-A7100T FPGA board and Xilinx Vivado.*

6. Appendix

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

-- Entity
entity project is
port(
    x:                in std_logic;
    y:                in std_logic;
    z:                out std_logic;
    reset:              in std_logic;
    clk:              in std_logic;
    current_occupancy:        out std_logic_vector(5 downto 0)
);
end entity;

-- Architecture
architecture behavior of project is

signal red_light: std_logic;
signal max_occupancy: std_logic_vector(5 downto 0) := "111111";
signal current: std_logic_vector(5 downto 0);

begin

        p0: process(clk, reset)
        begin

                if (reset = '1') then
                    current <= (others => '0');

                elsif (clk' event and clk = '1') then

                    if (x = '1' and current < max_occupancy) then
                          current <= current + 1;
                    end if;

                    if (y = '1' and current > "000000") then
                          current <= current - 1;
                    end if;
```

```
            end if;

        end process;

        p1: process(current, max_occupancy)
        begin

            if (current = max_occupancy) then
               red_light <= '1';

            else
               red_light <= '0';

            end if;

        end process;

        z <= red_light;
        current_occupancy <= current;

end;
```

Figure 3: project.vhd

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

entity project_tb is
end project_tb;

architecture behavior of project_tb is

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

```
    signal x : std_logic := '0';
    signal y : std_logic := '0';
    signal z : std_logic;
    signal reset : std_logic := '1';
    signal clk : std_logic := '0';
    signal current_occupancy : std_logic_vector(5 downto 0);
    signal max_occupancy : std_logic_vector(5 downto 0) := "111111";

begin

    uut: project port map (
        x => x,
        y => y,
        z => z,
        reset => reset,
        clk => clk,
          current_occupancy => current_occupancy
    );

    p0: process
    begin

        reset <= '1';
          clk <= '0';
        x <= '0';
        y <= '0';
        wait for 10 ns;

        reset <= '0';
        wait for 10 ns;

        -- Check decrementor by decrementing three times (current_occupancy should
remain 0)
        for i in 1 to 3 loop
            clk <= '1';
            x <= '0';
            y <= '1';
            wait for 10 ns;

            clk <= '0';
            x <= '0';
            y <= '0';
            wait for 10 ns;
```

```
        end loop;

        -- Check incrementor by incrementing 65 times (current_occupancy should go from
0 to 63 and not exceed 63)
        for i in 1 to 65 loop
            clk <= '1';
            x <= '1';
            y <= '0';
            wait for 10 ns;

            clk <= '0';
            x <= '0';
            y <= '0';
            wait for 10 ns;
        end loop;

        -- Decrementing to 62 then incrementing to 63 five times
        for i in 1 to 5 loop
            clk <= '1';
            x <= '0';
            y <= '1';
            wait for 10 ns;

            clk <= '0';
            x <= '0';
            y <= '0';
            wait for 10 ns;

            clk <= '1';
            x <= '1';
            y <= '0';
            wait for 10 ns;

            clk <= '0';
            x <= '0';
            y <= '0';
            wait for 10 ns;
        end loop;

        -- Decrement back to 0 (current_occupancy should decrement from 63 to 0)
        for i in 1 to 63 loop
            clk <= '1';
            x <= '0';
```

```
        y <= '1';
        wait for 10 ns;

        clk <= '0';
        x <= '0';
        y <= '0';
        wait for 10 ns;
    end loop;

    -- Increment by 10 then reset
    for i in 1 to 10 loop
        clk <= '1';
        x <= '1';
        y <= '0';
        wait for 10 ns;

        clk <= '0';
        x <= '0';
        y <= '0';
        wait for 10 ns;
    end loop;

    reset <= '1';
    wait for 10 ns;

    wait;
  end process;

end architecture;
```

Figure 4: testbench.vhd

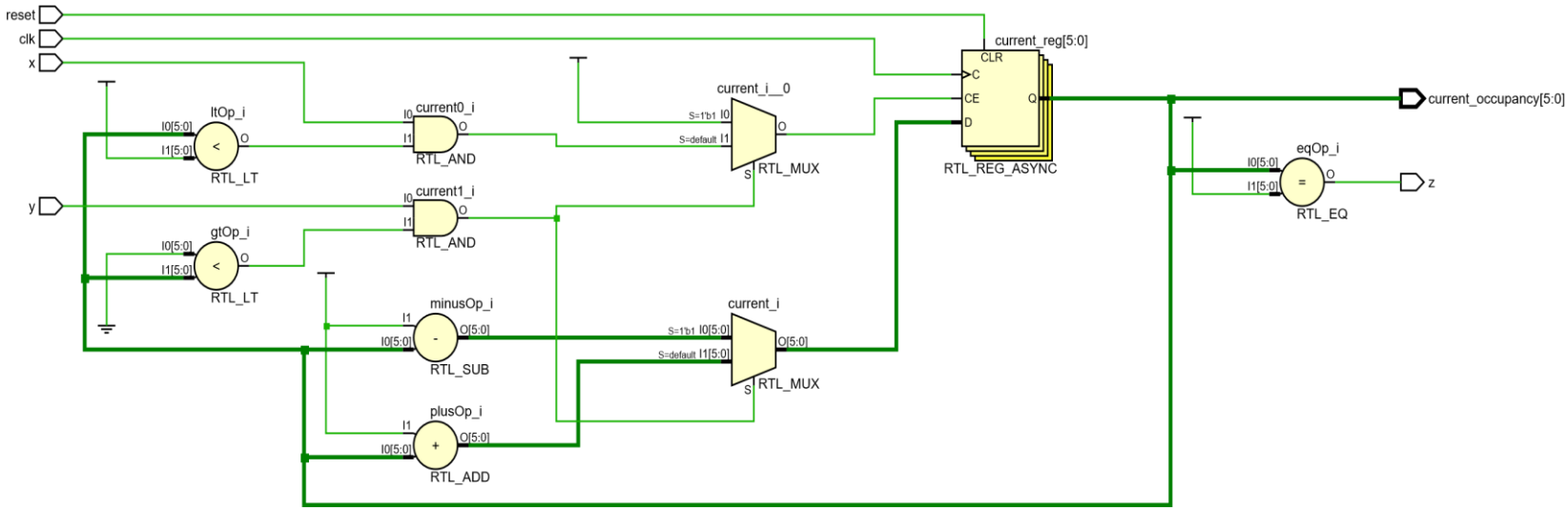

Figure 5: Elaborated schematic

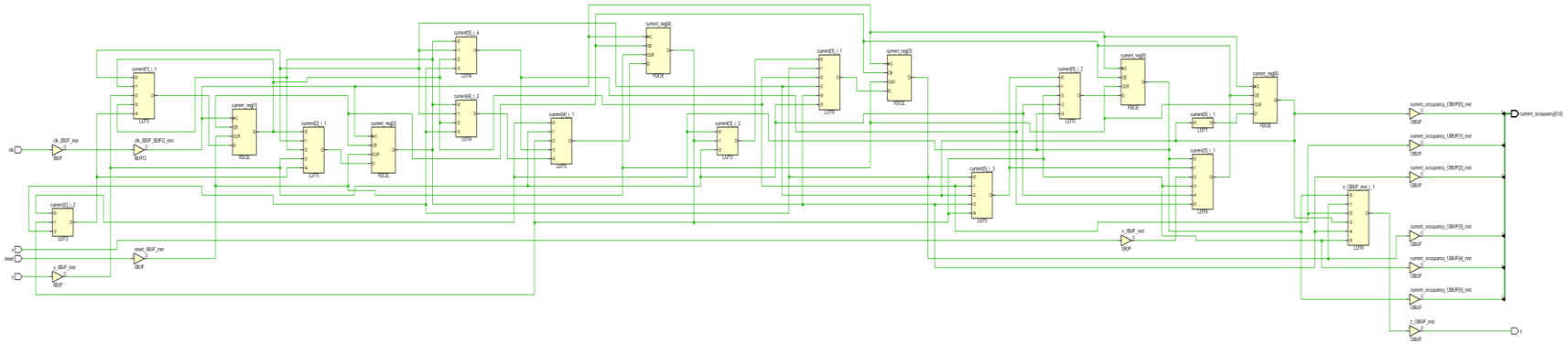

Figure 6: Synthesis Schematic