EEE 102 Term Project: Parking Lot Barrier System

Orkun İbrahim Kök

22003656

EEE102 – 01

Youtube video link:

https://youtu.be/tkgmV18fxQw

Purpose

The aim of this project is implementing the barrier system in parking lots to basys 3 FPGA by using some other external components

Design Specifications

The design has 3 inputs and 1 output. 2 of the 3 inputs are “clock” and “clear” inputs which are present in all designs. The other input is “duty_cycle”, which is related to signals from infrared sensor. “Pwm” (pulse-width modulation) is the output of the circuit which drives the servo motor reagarding the signals coming from the infrared sensor.

Methodology

My design takes signals from the infrared sensor and transfers these signals to basys 3 and via the code that I have written, servo motor starts working. I used a 5V adapter in order to give power to my servo motor because basys 3 can only provides 3,3V which is not enough for servo motor to work. I used a 12V 2A adapter cable for my adapter. I used a logic level converter to connect the servo motor to basys 3 FPGA. I connected my infrared sensor directly to basys 3 FPGA. I used JA and JC pins on my basys 3 FPGA to program the infrared sensor and servo motor via VHDL.

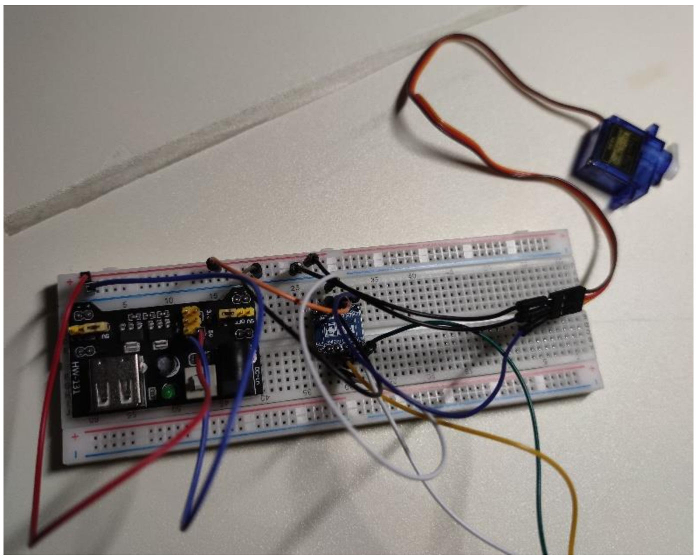
Figure 1.a: Breadboard design

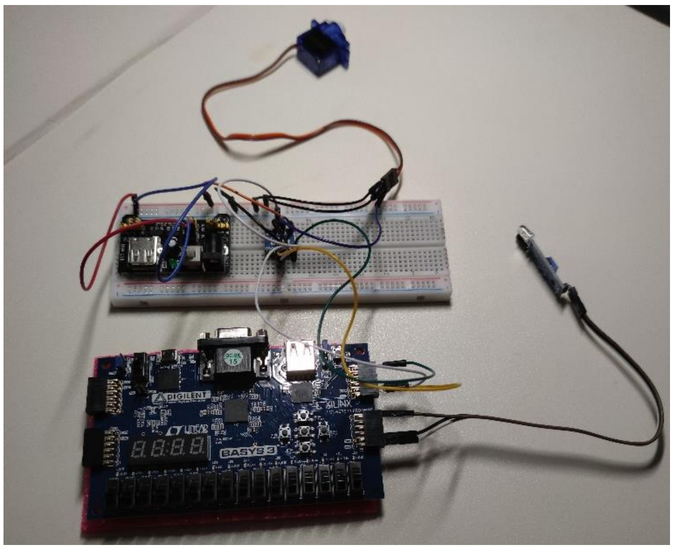
Figure 1.b: Complete design

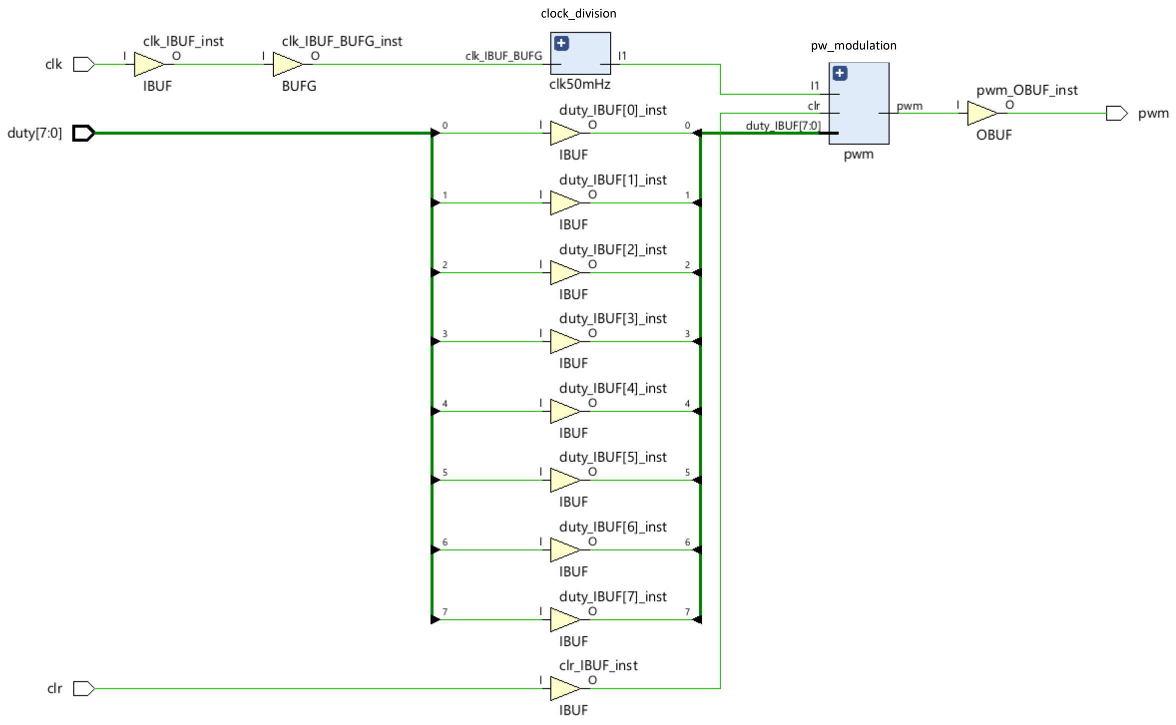

Figure 2: Schematic

VHDL Code

main_module.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_unsigned.all;

entity main_module is

   port(

      clear : in std_logic;

      clock : in std_logic;

      duty_cycle : in std_logic_vector (7 downto 0);

      pw_modulation : out std_logic

   );

end main_module;

architecture main of main_module is
```

```
signal n_clk : std_logic;

begin

clock_division: entity work.clk_freq

        port map(

            clock => clock, reset => '0', clock_output => n_clk);

Pulse_width: entity work.pw_modulation

        port map(

            clear => clear, clock => n_clk, duty_cycle => duty_cycle, period => "11001000",
pw_modulation => pw_modulation);


end main;
```

clk_freq.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity clk_freq is

    Port (

        clock    : in  STD_LOGIC;

        reset  : in  STD_LOGIC;

        clock_output: out STD_LOGIC

    );

end clk_freq;


architecture freq of clk_freq is

    signal temp: STD_LOGIC;

    signal s_count : integer range 0 to 5000 := 0;

begin

    clock_division: process (reset, clock) begin

        if (reset = '1') then
```

```
            temp <= '0';
            s_count  <= 0;
        elsif rising_edge(clk) then
            if (s_count = 5000) then
                temp <= NOT(temp);
                s_count  <= 0;
            else
                s_count <= s_count + 1;
            end if;
        end if;
    end process;

    clock_output <= temp;
end freq;
```

pulse_width.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
entity pulse_width is
    port(
        clear : in std_logic;
        clock : in std_logic;
        duty_cycle : in std_logic_vector (7 downto 0);
        period : in std_logic_vector (7 downto 0);
        pw_modulation : out std_logic
    );
end pw_modulation;
architecture behavioral of pw_modulation is
signal s_count : std_logic_vector(7 downto 0);
```

```
begin
   proc_count: process(clock, clear)
   begin
      if clear = '1' then
         s_count <= "00000000";
      elsif clk event and clk = '1' then
         if s_count = period -1 then
            s_count <= "00000000";
         else
            s_count <= s_count +1;
         end if;
      end if;
   end process proc_count;
   pwm_output: process(s_count, duty_cycle)
   begin
      if s_count < duty_cycle then
         pw_modulation <= '1';
      else
         pw_modulation <= '0';
      end if;
   end process pwm_output;
end behavioral;
```