Lab 5: Pulse Width Modulated LED Array Driver

-Overview-

In this lab, I will design a 16-output Pulse Width Modulation (PWM) device using VHDL to control the brightness of LEDs on the Nexys A7 board. The system will take a 64-bit input from the CPU, with each four bits determining the duty cycle for one of the LEDs. I will implement two 32-bit registers to hold this input and create a clock divider to reduce the 50 MHz system clock to about 5 KHz. Additionally, a modulo-15 counter will enable the PWM outputs, and comparators will determine when to turn the LEDs on based on the duty cycle values. I'll also develop a testbench to simulate and verify the circuit's functionality before programming the FPGA and testing the PWM output with an oscilloscope.

-Design-

The first part of the design starts with the array of registers down the middle that store the values of how bright was want each light to be. In the 2 register design, each register has 32 bits, with every register containing 8 LED brightness values. In the 8 register design, each register has 8 bits, with every register containing 2 LED brightness values. The 8 register design was implemented on the Nexys A7 board, while the 2 register design will be used for the next lab. To select the registers, a decoder with enable was used. The enable of the decoder is used for the write enable. To decrease the blinking rate of the LEDs, a clock divider was made out of a 10 bit counter. The top bit of the counter's output was used and the clock signal for the modulo 14 counter. The modulo counter will be used to compare its output with the values of the stored values, giving a high signal when the values is bigger then the modulo counter, and a low signal when the values is equal to or less than the modulo counter. The modulo counter was implemented using a generic 4 bit counter with combinational logic that reset the counter when the count was "1110". This makes it count to 14 then reset to 0 without counting 15.

Logic Diagram

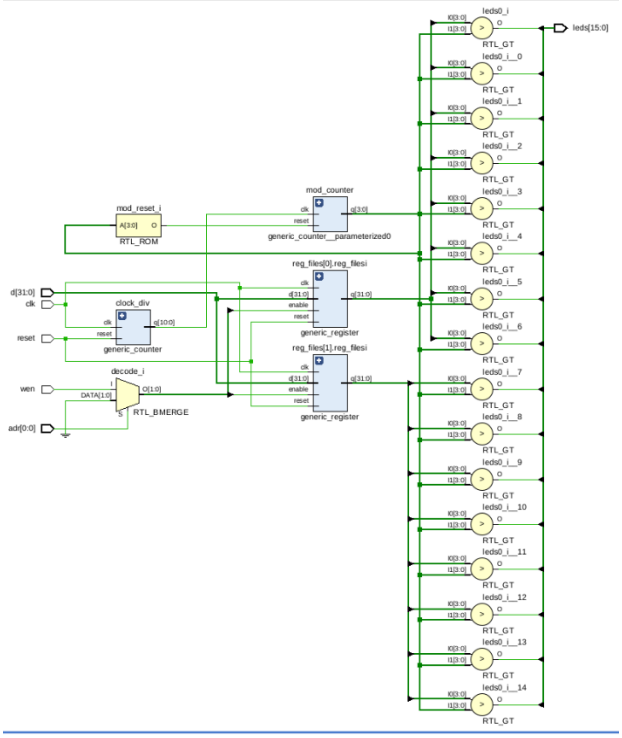

Figure 1: 2 32 bit registers design

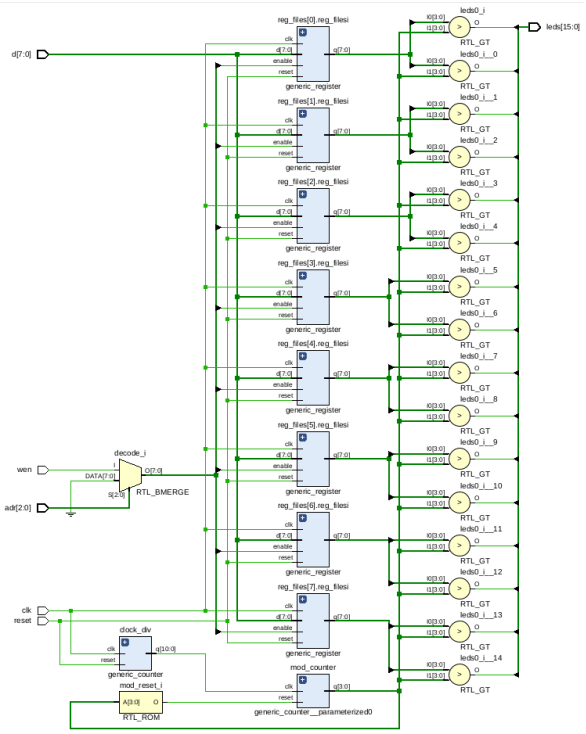

Figure 2: 8 8 bits registers design

-Simulation -

(The top clk is the global clk, and the bottom clk is out of the clock divider)

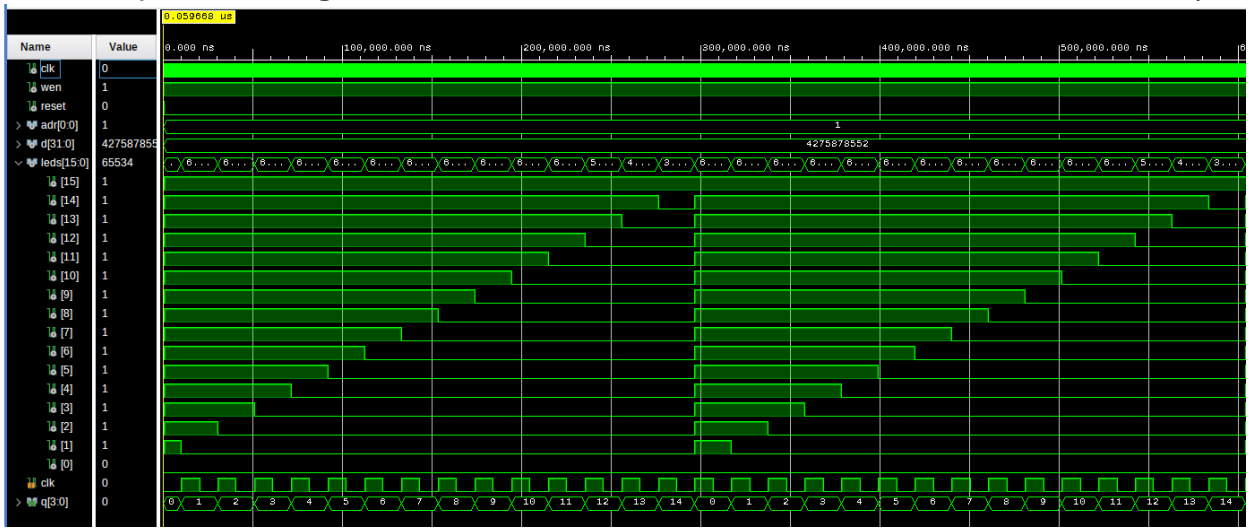

Figure 3: 2 32 bit registers design after loading each register with the valus 0 to 15

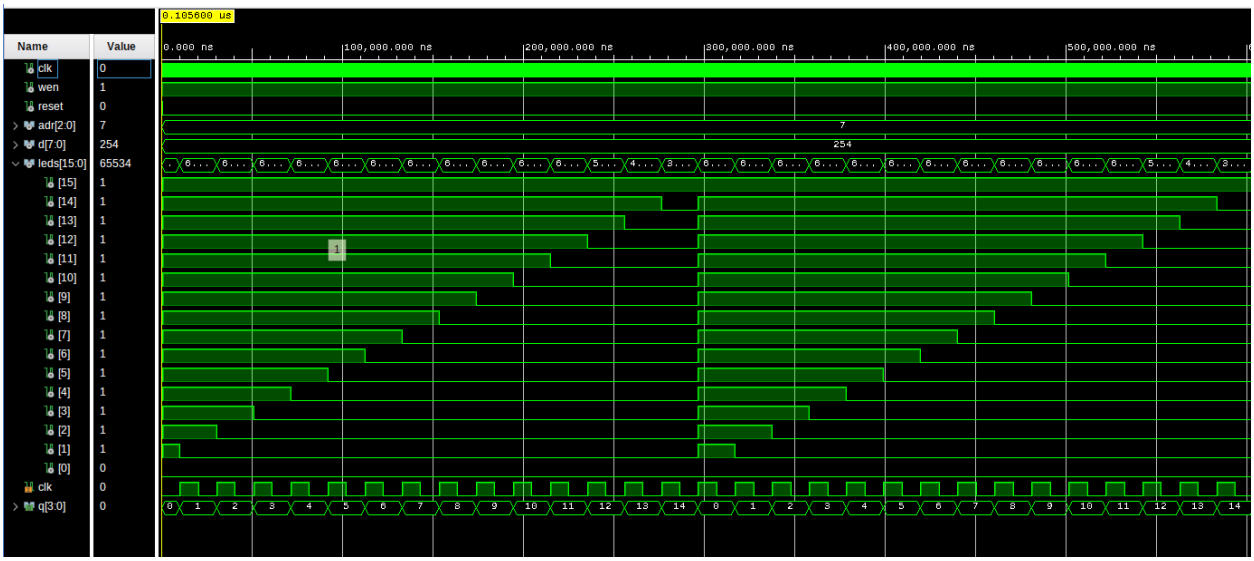

Figure 4: 8 8 bit registers design after loading each register with the valus 0 to 15

Despite the 2 register and 8 register designs being different internally, the output is the same when the registers are loaded with the same data.

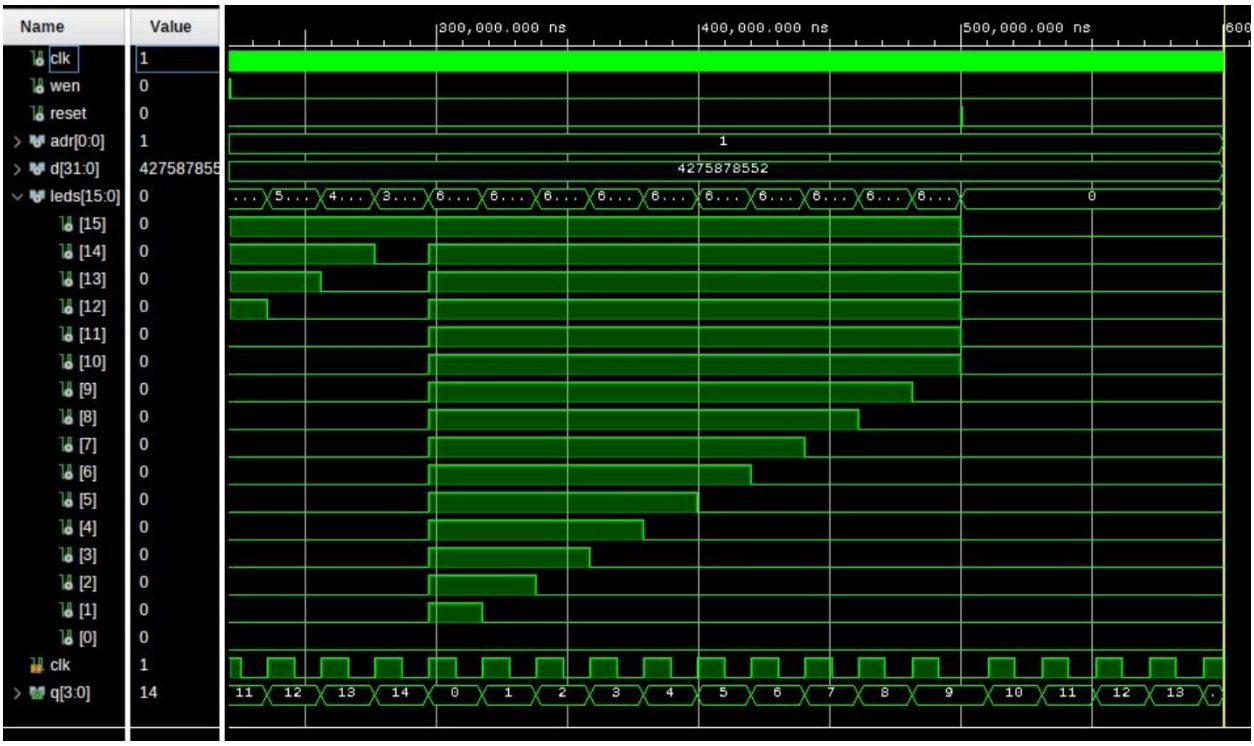

Figure 5: Reset was hit at 500,000 ns. All outputs go to 0

-Implemented Design Verification-

(Switches 15-13 used to select address, 7-0 were value to store)

(Buttons were used for write enable and reset)

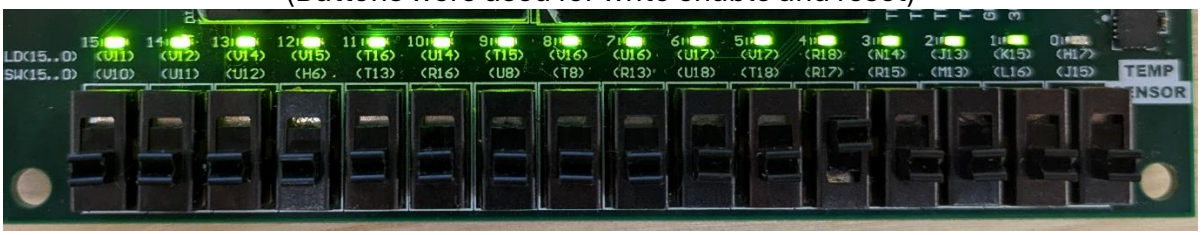

Figure 6: Each LED has be programmed with values 15 through 0.

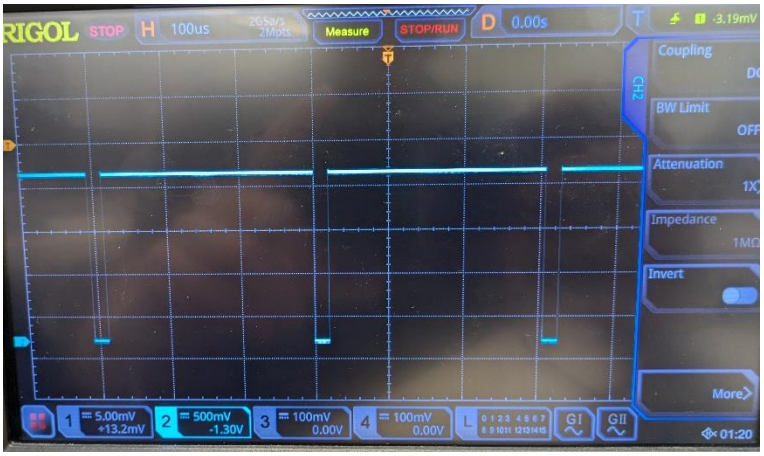

Figure 7: Oscilloscope output of LED set to 14

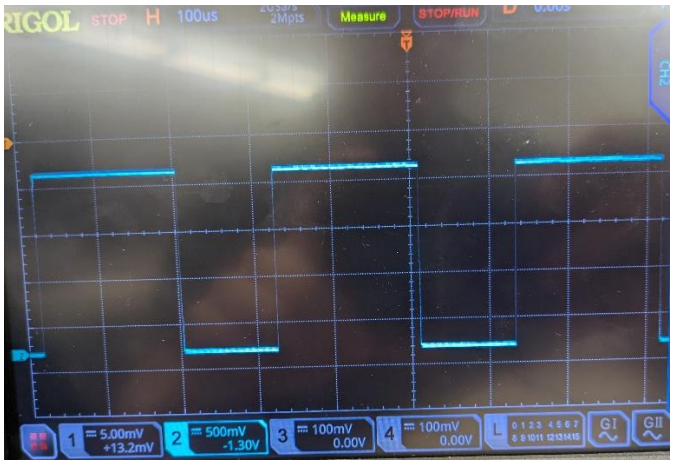

Figure 8: Oscilloscope output of LED set to 9

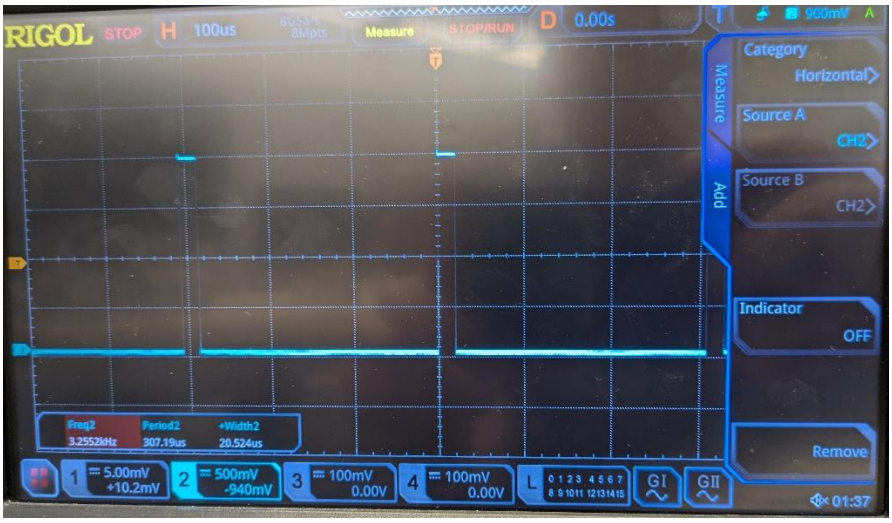

Figure 9: Oscilloscope output of LED set to 1 (with period)

Frequency = 3255 Hz
Period = 307.19 us
Positive Width = 20.524 us

$$\frac{Positive\ Width}{Period} = \frac{20.524}{307.19} = \frac{1}{14.97}$$

This shows that the positive length is 1/15 that of period as we expected.

-Conclusion-

Overall, the lab went pretty smoothly for me. I was able to use past resources like the decoder and register, and implement them into a larger design. I was able to create a generic counter that was used both for the clock divider, and the modulo counter. The modulo counter took some working as I was resetting it in the wrong spot initially. Also, parsing out the data from the registers using a generate statement proved to be a challenge, but I got it down with some mod operators. Finally, getting a clean oscilloscope output took some trail and error, but eventually I got some good readings.

-Appendix-

Main VHDL

```
1
2       library IEEE;
3       use IEEE.STD_LOGIC_1164.ALL;
4       use IEEE.numeric_std.all;
5
6
7       entity generic_counter is
8           generic ( bits : integer := 4 );
9           Port ( clk : in STD_LOGIC;
10                 reset : in STD_LOGIC;
11                 q : out STD_LOGIC_VECTOR ( bits-1 downto 0 ));
12      end generic_counter;
13
14      architecture Behavioral of generic_counter is
15          signal current_count : std_logic_vector ( bits-1 downto 0 ) := ( others => '0' );
16          signal next_count : std_logic_vector ( bits-1 downto 0 ) := ( others => '0' );
17      begin
18          --declare memory
19          current_count <= next_count when rising_edge( clk );
20          --next state logic
21          next_count <= (others => '0') when reset = '1' else
22              std_logic_vector ( unsigned ( current_count ) + 1 );
23          --output logic
24          q <= current_count;
25
26      end Behavioral;
```

Figure 10: Generic Counter Entity

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use work.mi_package.all;

entity PWM_Array_Driver is
    Port (
    clk, wen, reset : in std_logic;
    adr : in std_logic_vector ( 2 downto 0 );
    d : in std_logic_vector ( 7 downto 0 );
    leds : out std_logic_vector (15 downto 0 )
    );
end PWM_Array_Driver;

architecture Behavioral of PWM_Array_Driver is
    type signal_array is array ( 7 downto 0 ) of std_logic_vector ( 7 downto 0 );
    signal array_reg: signal_array;
    signal decoded : std_logic_vector ( 7 downto 0 );
    signal div_clk_vector : std_logic_vector ( 10 downto 0 );
    signal div_clk, mod_reset : std_logic;
    signal compare : std_logic_vector (3 downto 0 );
begin
    --clock divider
    clock_div : entity work.generic_counter ( Behavioral )
        generic map ( bits => 11 )
        port map ( clk => clk, reset => reset, q => div_clk_vector );
     div_clk <= div_clk_vector(10);

    --modulo counter
    mod_counter : entity work.generic_counter ( Behavioral )
        generic map ( bits => 4 )
        port map ( clk => div_clk, reset => mod_reset, q => compare );
    mod_reset <= '1' when compare = "1110" else '0';

    --registers
    decoded <= decode ( adr, wen );

    reg_files: for i in 0 to 7 generate
        reg_filesi: entity work.generic_register ( Behavioral )
        generic map ( bits => 8 )
        port map ( clk => clk, reset => reset, enable => decoded(i),
                   d => d, q => array_reg(i) );
    end generate reg_files;

    --comparing
    leds_loop : for i in 0 to 15 generate
        leds(i) <= '1' when array_reg(i/2)(3 + 4 * (i mod 2) downto 4 * (i mod 2) ) > compare
            else '0';
    end generate leds_loop;

end Behavioral;
```

Figure 11: 8 8 bit regsiters design entity/architecture

```vhdl
architecture two_by_32 of PWM_Array_Driver is
    type signal_array is array ( 1 downto 0 ) of std_logic_vector ( 31 downto 0 );
    signal array_reg: signal_array;
    signal decoded : std_logic_vector ( 1 downto 0 );
    signal div_clk_vector : std_logic_vector ( 10 downto 0 );
    signal div_clk, mod_reset : std_logic;
    signal compare : std_logic_vector (3 downto 0 );
begin
    --clock divider
    clock_div : entity work.generic_counter ( Behavioral )
        generic map ( bits => 11 )
        port map ( clk => clk, reset => reset, q => div_clk_vector );
     div_clk <= div_clk_vector(10);

    --modulo counter
    mod_counter : entity work.generic_counter ( Behavioral )
        generic map ( bits => 4 )
        port map ( clk => div_clk, reset => mod_reset, q => compare );
    mod_reset <= '1' when compare = "1110" else '0';

    --registers
    decoded <= decode ( adr, wen );

    reg_files: for i in 0 to 1 generate
        reg_filesi: entity work.generic_register ( Behavioral )
        generic map ( bits => 32 )
        port map ( clk => clk, reset => reset, enable => decoded(i),
                   d => d, q => array_reg(i) );
    end generate reg_files;

    --comparing
    leds_loop : for i in 0 to 15 generate
        leds(i) <= '1' when array_reg(i/8)(3 + 4 * (i mod 8) downto 4 * (i mod 8) ) > compare
            else '0';
    end generate leds_loop;

end two_by_32;
```

Figure 12: 2 32 bit regsiters design architecture

Testbench VHDL

```vhdl
entity PWM_Array_Driver_tb is
end PWM_Array_Driver_tb;

architecture Behavioral of PWM_Array_Driver_tb is
    signal clk, wen, reset : std_logic := '1';
    signal adr : std_logic_vector ( 2 downto 0 );
    signal d : std_logic_vector ( 7 downto 0 );
    signal leds : std_logic_vector (15 downto 0 );
begin

uut: entity work.PWM_Array_Driver ( Behavioral )
    port map (
        clk => clk,
        wen => wen,
        reset => reset,
        adr => adr,
        d => d,
        leds => leds);

clk <= not clk after 5 ns;
reset <= '0' after 20 ns;

process
begin
    wait for 20 ns;
    d <= "00010000";
    adr <= "000";
    wait for 10 ns;
    d <= "00110010";
    adr <= "001";
    wait for 10 ns;
    d <= "01010100";
    adr <= "010";
    wait for 10 ns;
    d <= "01110110";
    adr <= "011";
    wait for 10 ns;
    d <= "10011000";
    adr <= "100";
    wait for 10 ns;
    d <= "10111010";
    adr <= "101";
    wait for 10 ns;
    d <= "11011100";
    adr <= "110";
    wait for 10 ns;
    d <= "11111110";
    adr <= "111";
    wait;
end process;

end Behavioral;
```

Figure 14: 8 8 bit register design testbench

```vhdl
entity PWM_Array_Driver_tb is
end PWM_Array_Driver_tb;

architecture Behavioral of PWM_Array_Driver_tb is
    signal clk, wen, reset : std_logic := '1';
    signal adr : std_logic_vector ( 0 downto 0 );
    signal d : std_logic_vector ( 31 downto 0 );
    signal leds : std_logic_vector (15 downto 0 );
begin

uut: entity work.PWM_Array_Driver ( two_by_32 )
    port map (
        clk => clk,
        wen => wen,
        reset => reset,
        adr => adr,
        d => d,
        leds => leds);

clk <= not clk after 5 ns;
reset <= '0' after 20 ns;

process
begin
    wait for 20 ns;
    d <= "01110110010101000011001000010000";
    adr <= "0";
    wait for 10 ns;
    d <= "11111110110111001011010010011000";
    adr <= "1";
    wait;
end process;

end Behavioral;
```

Figure 13: 2 32 bit regsiter design testbench

Constraint File

```
## Clock signal
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports {clk}]
#create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports {clk}33]


##Switches
set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports {d[0]}]
set_property -dict {PACKAGE_PIN L16 IOSTANDARD LVCMOS33} [get_ports {d[1]}]
set_property -dict {PACKAGE_PIN M13 IOSTANDARD LVCMOS33} [get_ports {d[2]}]
set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports {d[3]}]
set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports {d[4]}]
set_property -dict {PACKAGE_PIN T18 IOSTANDARD LVCMOS33} [get_ports {d[5]}]
set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports {d[6]}]
set_property -dict {PACKAGE_PIN R13 IOSTANDARD LVCMOS33} [get_ports {d[7]}]
#set_property -dict {PACKAGE_PIN T8 IOSTANDARD LVCMOS18} [get_ports {}]
#set_property -dict {PACKAGE_PIN U8 IOSTANDARD LVCMOS18} [get_ports {}]
#set_property -dict {PACKAGE_PIN R16 IOSTANDARD LVCMOS33} [get_ports {d_addr[0]}]
#set_property -dict {PACKAGE_PIN T13 IOSTANDARD LVCMOS33} [get_ports {d_addr[1]}]
#set_property -dict {PACKAGE_PIN H6 IOSTANDARD LVCMOS33} [get_ports {d_addr[2]}]
set_property -dict {PACKAGE_PIN U12 IOSTANDARD LVCMOS33} [get_ports {adr[0]}]
set_property -dict {PACKAGE_PIN U11 IOSTANDARD LVCMOS33} [get_ports {adr[1]}]
set_property -dict {PACKAGE_PIN V10 IOSTANDARD LVCMOS33} [get_ports {adr[2]}]

## LEDs
set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports {leds[0]}]
set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports {leds[1]}]
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports {leds[2]}]
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports {leds[3]}]
set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports {leds[4]}]
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {leds[5]}]
set_property -dict {PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports {leds[6]}]
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {leds[7]}]
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {leds[8]}]
set_property -dict {PACKAGE_PIN T15 IOSTANDARD LVCMOS33} [get_ports {leds[9]}]
set_property -dict {PACKAGE_PIN U14 IOSTANDARD LVCMOS33} [get_ports {leds[10]}]
set_property -dict {PACKAGE_PIN T16 IOSTANDARD LVCMOS33} [get_ports {leds[11]}]
set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {leds[12]}]
set_property -dict {PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports {leds[13]}]
set_property -dict {PACKAGE_PIN V12 IOSTANDARD LVCMOS33} [get_ports {leds[14]}]
set_property -dict {PACKAGE_PIN V11 IOSTANDARD LVCMOS33} [get_ports {leds[15]}]

set_property -dict {PACKAGE_PIN N17 IOSTANDARD LVCMOS33} [get_ports {wen}]
set_property -dict {PACKAGE_PIN M18 IOSTANDARD LVCMOS33} [get_ports {reset}]
#set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports {buttons[2
```

Figure 15: Constraints File