EEE102-Term Project

Password(Passcode) System With Sensors

Name: Sait Sarper Özaslan

ID: 22002861

Section: 102-2

Youtube Video: [Sait Sarper Özaslan ||Bilkent EEE|| Password System with Sensors|| EE-102 Term Project]

Objective: Aim of the project is to be able to set and try a password using Basys3 FPGA and infrared sensors.

Methodology: The first step was to design the lock system and figure out a way to open a set password. Initially, to achieve this I created a state machine whose password could be set through switches. In this part I used LEDs to understand whether the password I entered through the buttons were correct or not. Here is the state machine of this project.

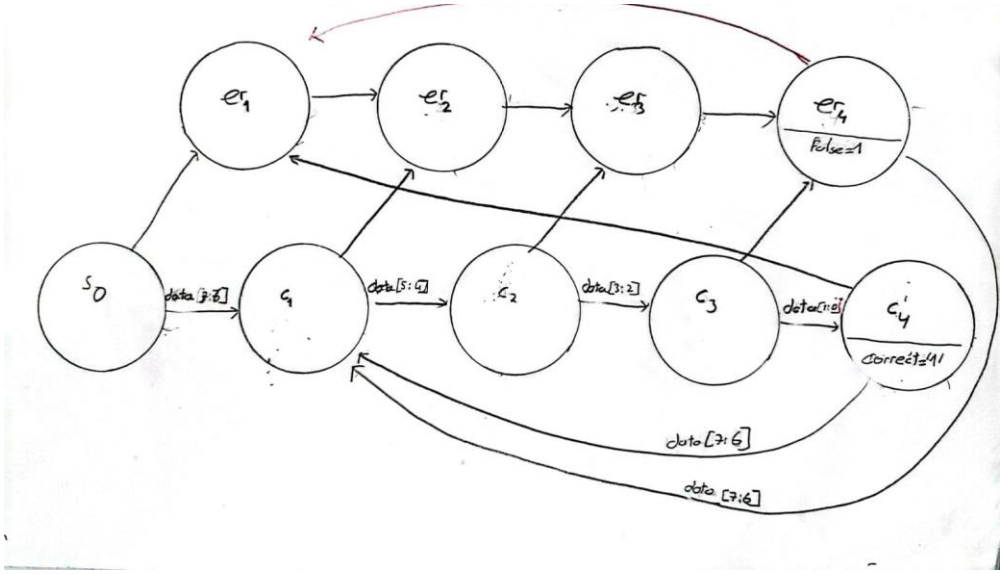

Figure 1: State Machine of the Password

If the password entered is correct, one can go from state s0 to c4. However, if there is a mistake being made the machine will jump into error states and until reaching the end of the error states the system will remain in these states. From the end of the error state er4 and correct

state c4 if one starts with the right input the person will go to correct state s1 and if one enters the wrong input the state machine will go to error state er1. This allows continuous trial of the password without the need of a reset button after each trial. In state er4, the ”fals” input will be ‘1’ and in correct state c4 the “correct” input will be ‘1’.

Lastly, I also added the option through a button to clear the password trial and go back to state s0. This same implementation is also can be used while setting the password as well through a different button.

In the final version of this project, I executed 3 steps in order:

1) A shift-register to set the password, whose inputs are taken from the buttons. The input is displayed in the seven-segment display.
2) A shift-register and the password state-machine, in here inputs are taken from the infrared sensors and displayed in the seven-segment display.
3) If the password is correct the display changes to “CCCC”, if it is false it changes to “FFFF”.

While the buttons only work to enter a password, I still added a multiplexer which works as an “enable” signal to avoid further unwanted results.

Essentially an 8-bit password will be set through buttons and through switching the enable signals, one will try the password with the infrared sensors. The seven-segment display will show whether it’s correct or false.

Design Specification: The top module I have written consists of:

Mclk(in): The main clock of BASYS3

Btn(in): The inputs allowing me to set up a password. It is a vector (2 downto 0).

Sens(in): The inputs allowing me to try the password. It is a vector (2 downto 0).

Enabls(in): The enable signal of the shift register. When active, one can set up a password.

Enable1(in): The enable signal of the password state. When active one can try to enter the password.

Select_digit(out): The anodes of the seven-segment display. It is a vector (3 down to 0).

Select_LED(out): The cathodes of the seven-segment display. It is a vector (6 down to 0).

LD (out): It is used to display whether the password entered is correct or not through LEDs of the BASYS3. It is a vector (2 downto 0). Not necessary for the design, is added to ensure the design is correct.

The design I made has 1 main module “mtop.vhdl” and the top modules under it which are “reg_top.vhdl” and “pw_top.vhdl”. The modules also have sub modules under them which can be seen in the figure below:

Figure 2: Hierarchy

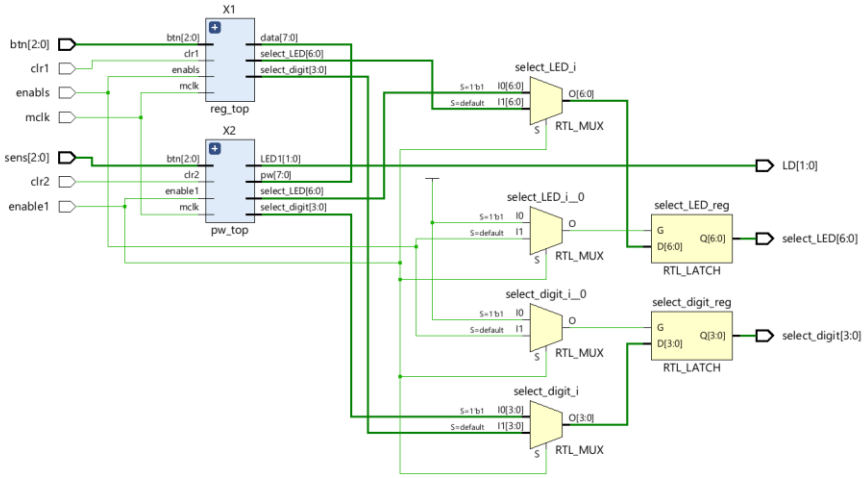

Figure 3: RTL Schematic of mtop.vhdl

The “clkdiv.vhdl” module consists of “clk, clr, clk19 and clk178”. While the first two are inputs corresponding to clock and clear functions, the last 2 are clock divisions: clk190 for buttons and sensors, dclk178 for the seven- segment display.

From the “clkdiv.vhdl” the connection extends to “clock_pulse.vhdl”. The function of this module is to stop buttons and sensors from sending more than one signal at a time which prevents unwanted results.

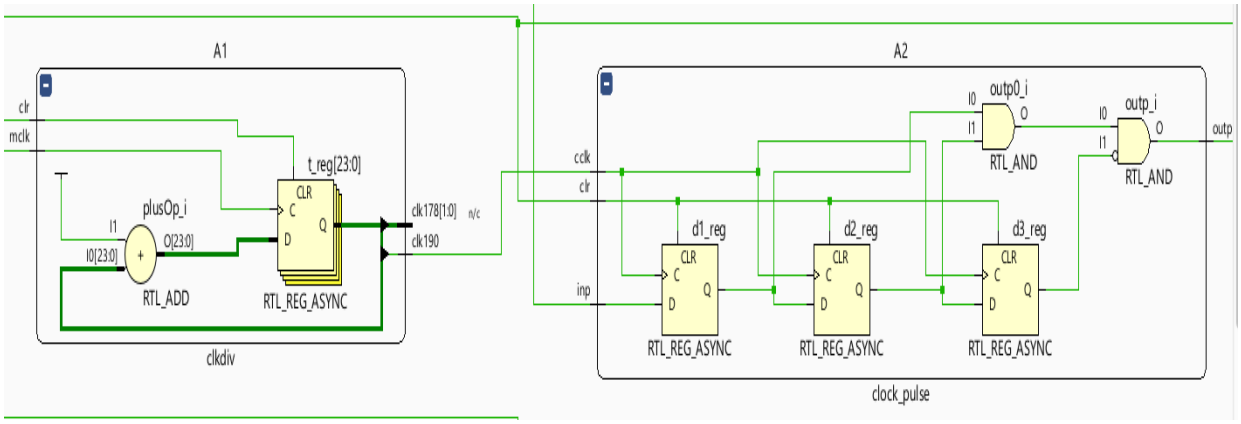

Figure 4: RTL Schematic of clkdiv and clock_pulse

From “clock_pulse.vhdl” the connection extends to “shift_register_top.vhdl”. The function of this module is to register the password entered or the password set.. Consists of “mclk,clr, enable,din, dout”. Data input (Din) is a 2-bit input while data output (dout) is an 8-bit output.

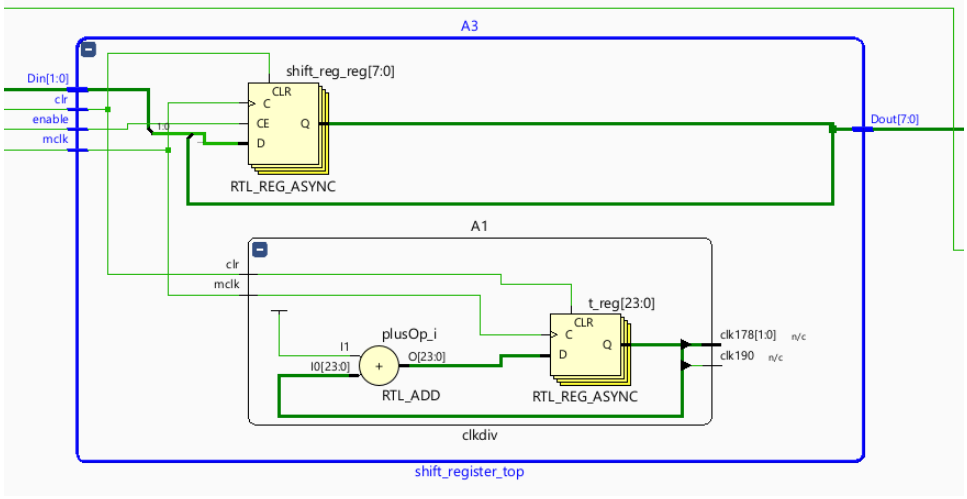

Figure 5: RTL Schematic of Shift_register

The outputs of these datas will be displayed in “segment.vhdl” which is the seven-segment display as a submodule. The display will be seen in “select_digit and select_LED” outputs which corresponds to anode and cathode parts of the display. This module can also display “C” correct and “F” for false.

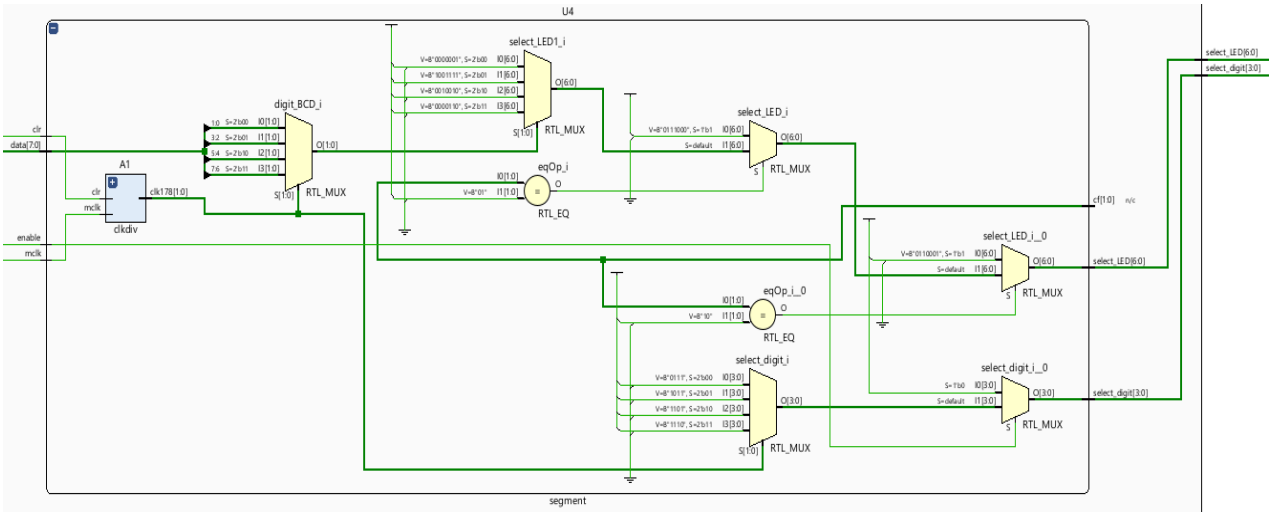

Figure 6: RTL Schematic of segment

Finally, there is “pw_s.vhdl” where the state machine I have mentioned before is written in. It consists of “clk, clr, data, sens, enable, correct and fals”. While data is the input coming from the “reg_top.vhdl”, sens is a two-bit input that allows me to check, the password is. In state c4, the value of “correct” input is ‘1’ which allows seven-segment to display “C”, on the other hand in state er4 “fals” input will be “1” and it will be displayed as “F”.

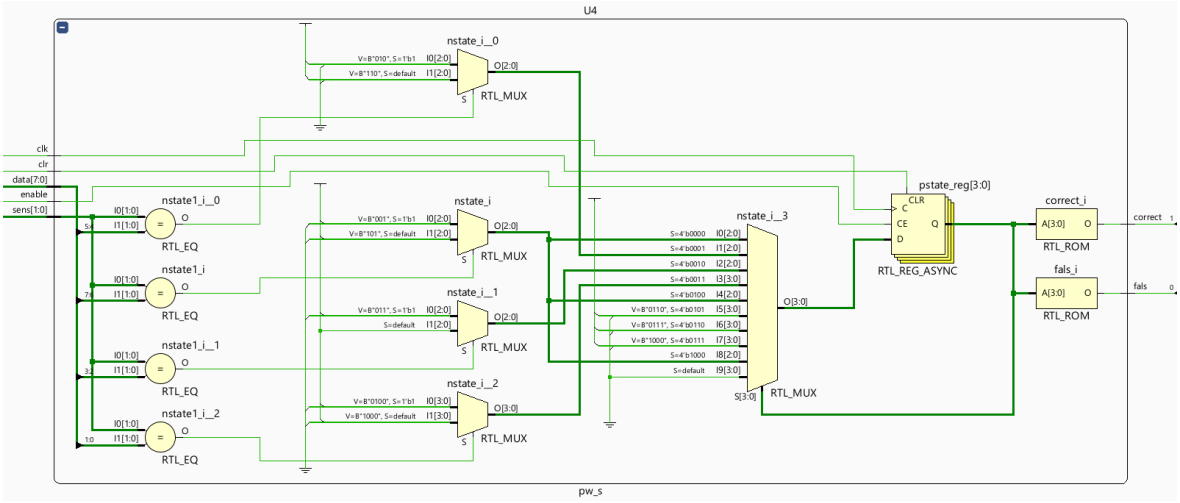

Figure 7: RTL Schematic of pw_s

Results:

- Sensors from left to right represent 0, 1,2 in order.
- Buttons from left to right represent 2,1,0 in order.
- The top button resets the password trial process, the bottom button resets the set password.
- The left most switch enables set password process, one right of the left most switch enables try password process.
- The Rightmost LED is on when the password entered is false, one right of this LED turns on when the password entered is correct.

Considering this info now I will give an example:

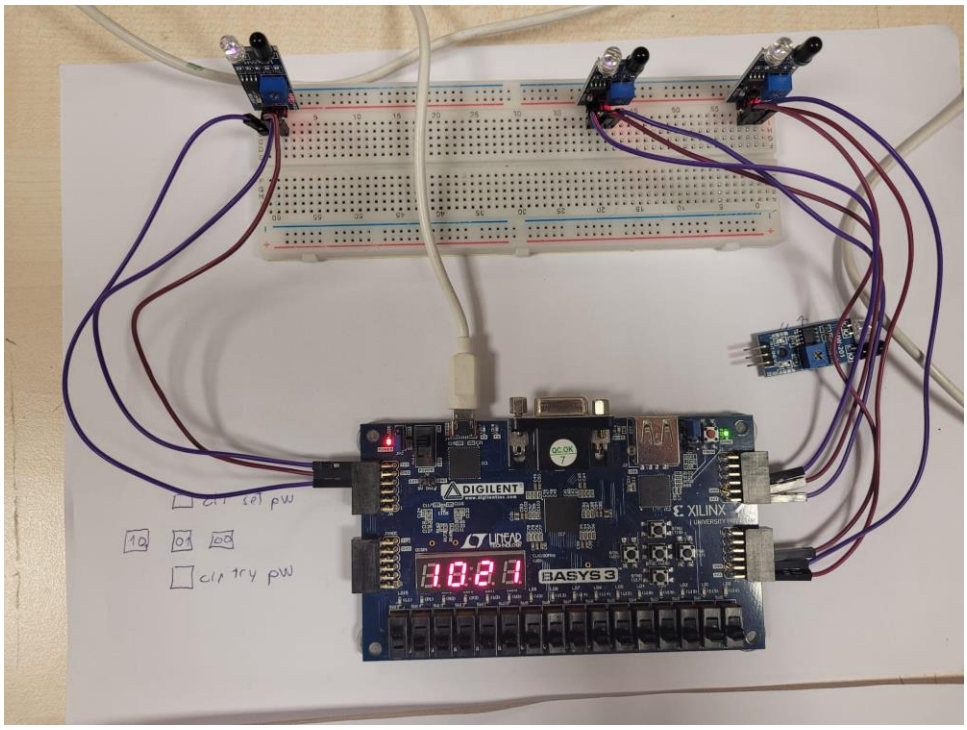

Figure 8: Set Password 1021

In this figure left switch(enabls) is on so it can be said that the password 1021 is the registered password through the buttons.

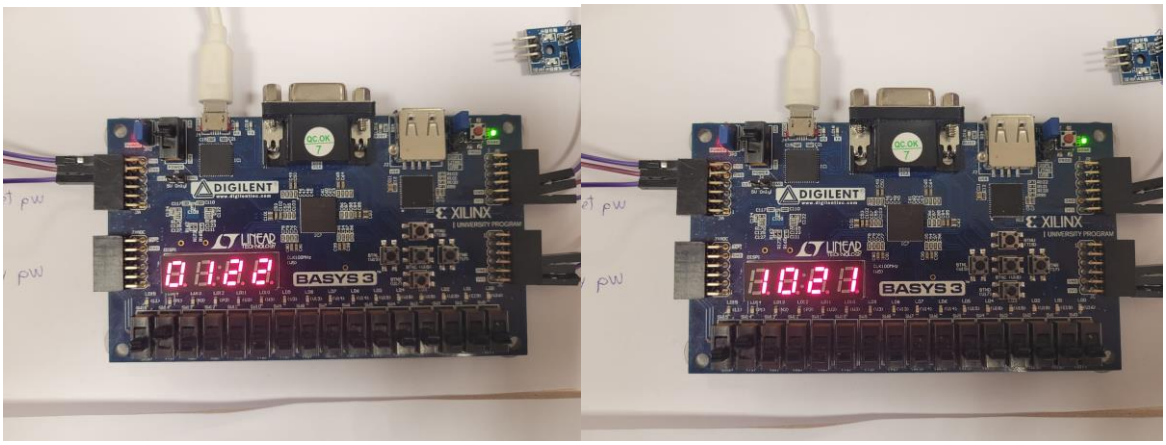

Figure 9 and 10: Try Password 0122 and 1021

In these figures the second switch(enable1) is active, and the leftmost switch is inactive meaning that we are trying to enter the password. The inputs taken from sensors are “0122” and “1021” meaning that the first password entered is false and the second password entered is correct

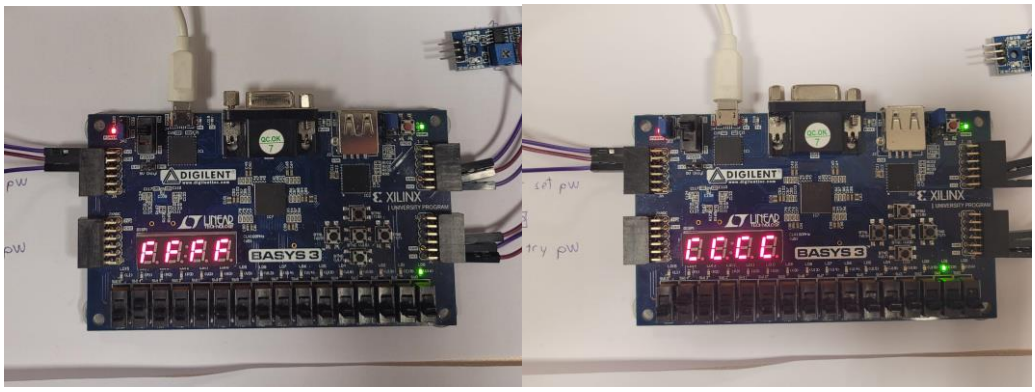

Figure 11 and 12: Display of False for 0122 and display of Correct for 1021

As it can be seen from the figures that if the password entered is wrong it will display “F” if the entered password is correct it will be display “C”

Conclusion:

This project utilizes several parts of the BASYS3 fpga which means I had to create several submodules. The submodules are connected in order so that there are no issues of having multiple or wrong signals being sent.

As there were many submodules I had to create, it was a challenge to learn how to connect each of the submodules in an adequate manner. This forced me to go through several trial-and-error processes to understand how the connection between modules works. To figure out this issue I separated the project into 2 main parts which is registering the password and trying the password. I first collected submodules under these two parts, and to combine these two parts I wrote several multiplexers to ensure the code works without errors.

One problem I hadn’t thought about was the sensitivity of the infrared sensors I used in the project. There were cases of sending multiple inputs if my hands were barely in the reach of the sensors. This situation caused the sensor to register several inputs as my hands are not fully stable and sometimes go outside and inside of the sensors.

In the end, I managed to complete the project I intended to do initially, and this project can be considered a success in such a case.

References:

Digilent – Basys3 Manual. https://digilent.com/reference/_media/basys3:basys3_rm.pdf.

"VHDL Code for Seven-Segment Display on Basys 3 FPGA." *FPGA Projects, Verilog Projects, VHDL Projects - FPGA4student.Com*, https://www.fpga4student.com/2017/09/vhdl-code-for-seven-segment-display.html.

Appendix:

Mtop.vhdl:

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



entity mtop is
Port (mclk: in std_logic;

  btn,sens: in std_logic_vector (2 downto 0);
  enable1,enabls,clr1,clr2: in std_logic ;
  select_digit: out std_logic_vector (3 downto 0);--which digit lights up
select_LED: buffer std_logic_vector (6 downto 0);
  LD:  out std_logic_vector (1 downto 0) );
end mtop;

architecture Behavioral of mtop is

component pw_top is
Port (mclk: in std_logic;
  pw: buffer std_logic_vector (7 downto 0);
  btn: in std_logic_vector (2 downto 0);
  enable1,clr2: in std_logic ;
  select_digit: out std_logic_vector (3 downto 0);--which digit lights up
select_LED: buffer std_logic_vector (6 downto 0);
  LED1:  out std_logic_vector (1 downto 0) );
end component;

component reg_top is
Port ( mclk:in std_logic ;
enabls,clr1: std_logic ;
btn: in std_logic_vector(2 downto 0);
data:buffer std_logic_vector (7 downto 0);
```

```
select_digit: out std_logic_vector (3 downto 0);
select_LED: buffer std_logic_vector (6 downto 0));
end component;

signal clr: std_logic ;
signal select_digit1,select_digit2:std_logic_vector (3 downto 0);
signal select_LED1, select_LED2: std_logic_vector (6 downto 0);
signal dn: std_logic_vector (7 downto 0);
begin

X1: reg_top
port map(mclk=>mclk,
clr1=>clr1,
enabls=> enabls,
btn=>btn,
data=>dn,
select_digit =>select_digit2 ,
select_LED  => select_LED2);

X2: pw_top
port map(mclk=>mclk,
btn=>sens,
enable1=>enable1,
clr2=>clr2,
pw=> dn,
select_digit =>select_digit1 ,
select_LED => select_LED1,
LED1=>LD);

process (enable1,enabls)
begin
if enable1 = '1' then
select_digit<= select_digit1 ;
select_LED<= select_LED1 ;
elsif enabls = '1' then
select_digit <= select_digit2 ;
select_LED <= select_LED2;
end if;
end process;
end Behavioral;
```

Reg_top.vhdl

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

entity reg_top is
Port ( mclk:in std_logic ;
enabls,clr1: std_logic ;
btn: in std_logic_vector(2 downto 0);
data:buffer std_logic_vector (7 downto 0);
select_digit: out std_logic_vector (3 downto 0);
select_LED: buffer std_logic_vector (6 downto 0));
end reg_top;

architecture Behavioral of reg_top is

component clock_pulse is
   port(inp,cclk,clr: in std_logic;
   outp: out std_logic  );
   end component ;

component clkdiv is
   port( mclk: in std_logic ;
      clr: in std_logic ;

      clk190: out std_logic;
      clk178: out std_logic_vector(1 downto 0) );
      end component ;

component shift_register_top is
   port( mclk : in  STD_LOGIC;
         clr: in std_logic ;
         enable: in std_logic ;
        Din  : in  STD_LOGIC_vector(1 downto 0);
        Dout : out STD_LOGIC_VECTOR(7 downto 0));
   end component;
```

```
component segment is
Port ( mclk: in std_logic;
clr: in std_logic;
enable:in std_logic ;
data: in std_logic_vector(7 downto 0);
select_digit: out std_logic_vector (3 downto 0);--which digit lights up
select_LED: buffer std_logic_vector (6 downto 0);
cf:buffer std_logic_vector (1 downto 0));
end component ;

   signal clr,clk190,clkp,btn012: std_logic ;
   signal bn,clk178,ld: std_logic_vector (1 downto 0);

begin

btn012 <= btn(0) or (btn(1)) or (btn(2));

bn<= (btn(2))& (btn(1));



A1:clkdiv
   port map(mclk => mclk,
   clr=> clr1,

   clk190 => clk190,
   clk178=>clk178)
   ;
A2: clock_pulse
port map(inp=> btn012,
   cclk => clk190,
   clr => clr1,
   outp => clkp);
A3: shift_register_top
port map(mclk=>clkp,
clr=>clr1,
enable=>enabls,
Din=>bn,
Dout=>Data);
A4: segment
```

```
port map(mclk =>mclk,
clr => clr1,
enable=>enabls,
data => data,
select_digit => select_digit,
select_LED => select_LED,
cf=> ld
);
end Behavioral;
```

Pw_top.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity pw_top is
  Port (mclk: in std_logic;
  pw: buffer std_logic_vector (7 downto 0);
  btn: in std_logic_vector (2 downto 0);
  enable1,clr2: in std_logic ;
  select_digit: out std_logic_vector (3 downto 0);--which digit lights up
select_LED: buffer std_logic_vector (6 downto 0);
  LED1:  out std_logic_vector (1 downto 0) );
end pw_top;

architecture Behavioral of pw_top is
component clkdiv is
    port( mclk: in std_logic ;
        clr: in std_logic ;
        clk190: out std_logic;
        clk178: out std_logic_vector(1 downto 0) );
  end component;

component pw_s is
    port(clk,clr:in std_logic;
     data: in std_logic_vector (7 downto 0);
     sens: in std_logic_vector (1 downto 0);
     enable: in std_logic ;
     correct: out std_logic ;
```

```
    fals: out std_logic
     );
   end component;

component clock_pulse is
   port(inp,cclk,clr: in std_logic;
   outp: out std_logic  );
   end component ;

component segment is
Port ( mclk: in std_logic;
clr: in std_logic;
data: in std_logic_vector(7 downto 0);
enable: in std_logic ;
select_digit: out std_logic_vector (3 downto 0);--which digit lights up
select_LED: buffer std_logic_vector (6 downto 0);
cf: buffer std_logic_vector (1 downto 0));
end component ;

component shift_register_top is
Port ( mclk : in  STD_LOGIC;
          clr: in std_logic ;
          enable: in std_logic ;
         Din   : in  STD_LOGIC_vector(1 downto 0);
         Dout : out STD_LOGIC_VECTOR(7 downto 0));
end component;

signal clr,clk190,clkp,btn0123: std_logic ;
signal bn,bt3,btn33,clk178 : std_logic_vector (1 downto 0);
signal Dout: std_logic_vector (7 downto 0);
begin

btn0123 <= (not(btn(0)) or not(btn(1))) or not(btn (2));


bn <= not(btn(2))& not(btn(1));


U1: clkdiv
   port map(mclk => mclk,
   clr=> clr2,
```

```
   clk190 => clk190,
   clk178 => clk178)
   ;
U2: clock_pulse
   port map(inp=> btn0123,
   cclk => clk190,
   clr => clr2,
   outp => clkp);
U3: shift_register_top
port map (mclk=> clkp,
clr=>clr2,
enable=> enable1,
Din=>bn,
Dout=> Dout);
U4: pw_s
   port map(clk => clkp,
   clr=> clr2,
   data=> pw,
   sens => bn,
   enable=> enable1,
   correct=> led1(1),
   fals => led1(0));
U5: segment
port map(mclk =>mclk,
clr => clr2,
data => Dout,
enable=>enable1,
select_digit => select_digit,
select_LED => select_LED,
cf=>led1 );

end Behavioral;
```

Clkdiv.vhdl

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

```
entity clkdiv is
Port ( mclk: in std_logic ;
       clr: in std_logic ;
       clk190: out std_logic;
       clk178: out std_logic_vector(1 downto 0 ));
end clkdiv;

architecture clkdiv of clkdiv is
signal t : std_logic_vector (23 downto 0);

begin

process(mclk,clr)
begin

if clr = '1' then
   t <= x"000000";
elsif mclk'event and mclk = '1' then
   t <= t+1;
end if;
end process;
clk190 <= t(17);
clk178 <= t(18 downto 17);
end clkdiv;
```

Clock_pulse.vhdl

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


entity clock_pulse is
Port (inp, cclk,clr: in std_logic;
outp: out std_logic);
end clock_pulse;

architecture behavioral of clock_pulse is
signal d1, d2, d3: std_logic;
begin
   process (cclk,clr)
```

```
      begin
      if clr ='1' then
      d1 <= '0';
      d2 <= '0';
      d3 <= '0';
      elsif cclk'event and cclk='1' then
      d1 <= inp;
      d2<= d1;
      d3 <= d2;
      end if;
   end process;
outp <= d1 and d2 and not d3;

end behavioral;
```

Shift_register_top.vhdl

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

entity shift_register_top is
   Port ( mclk : in  STD_LOGIC;
          clr: in std_logic ;
          enable: in std_logic ;
         Din   : in  STD_LOGIC_vector(1 downto 0);
         Dout : out STD_LOGIC_VECTOR(7 downto 0));
end shift_register_top;

architecture Behavioral of shift_register_top is
component clkdiv is
   port( mclk: in std_logic ;
      clr: in std_logic ;
      clk190: out std_logic;
      clk178: out std_logic_vector(1 downto 0) );
      end component ;

   signal shift_reg : STD_LOGIC_VECTOR(7 downto 0) := X"00";

   signal clk190: std_logic ;
begin
```

```
A1:clkdiv
port map(mclk => mclk,
clr=>clr,
clk190 =>clk190 );



   process (mclk,clr)
   begin

      if clr = '1' then
      shift_reg <= x"00";

      elsif enable = '1' and(mclk'event and mclk = '1') then
         shift_reg(1 downto 0) <= Din;
         shift_reg(3 downto 2) <= shift_reg(1 downto 0);
         shift_reg(5 downto 4) <= shift_reg(3 downto 2);
         shift_reg(7 downto 6) <= shift_reg(5 downto 4);

      end if;
   end process;

      Dout <= shift_reg;

end Behavioral;
```

Pw_s.vhdl

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



   entity pw_s is
    Port (clk,clr:in std_logic;
    data: in std_logic_vector (7 downto 0);
    sens: in std_logic_vector (1 downto 0);
    enable: in std_logic ;
    correct: out std_logic ;
    fals: out std_logic
     );
```

```
end pw_s;

architecture Behavior of pw_s is
type state_type is (s0,c1,c2,c3,c4,er1,er2,er3,er4);
signal pstate, nstate: state_type ;
begin
sreg: process(clk,clr)
begin
   if clr= '1' then
      pstate <= s0;
   elsif enable = '1' and clk'event and clk = '1' then
      pstate <= nstate;
   end if;
end process;

X1: process(pstate,sens,data)
begin
   case pstate is
      when s0=>
      if sens = data(7 downto 6) then
      nstate<= c1;

      else
      nstate <= er1;
      end if;
      when c1=>
      if sens = data(5 downto 4) then
      nstate <= c2;

      else
      nstate <= er2;
      end if;

      when c2 =>
      if sens= data(3 downto 2) then
      nstate <= c3;

      else
      nstate <= er3;
      end if;
```

```
      when c3 =>
      if sens= data( 1 downto 0) then
      nstate <= c4;

      else

      nstate <= er4;
      end if;

      when c4 =>
      if sens = data(7 downto 6) then
      nstate <= c1;
      else
      nstate <= er1;
      end if;
      when er1=>
      nstate<= er2;
      when er2 =>
      nstate <= er3;
      when er3 =>
      nstate <= er4;
      when er4 =>
      if sens = data(7 downto 6) then
      nstate <= c1;
      else
      nstate <= er1;
      end if;
      when others =>
      nstate <= s0;
  end case;
end process;

X2: process (pstate)
begin
   if  pstate = c4 then
         correct <= '1';
    else
         correct <= '0';
    end if;
   if pstate = er4 then
         fals <= '1';
```

```
        else
            fals <= '0';
        end if;

    end process;
   end Behavior;
```

Segment.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity segment is
Port ( mclk: in std_logic;
clr: in std_logic;
data: in std_logic_vector(7 downto 0);
enable: in std_logic ;
cf:buffer std_logic_vector (1 downto 0);
select_digit: out std_logic_vector (3 downto 0)
select_LED: buffer std_logic_vector (6 downto 0))end segment;

architecture Behavioral of segment is

component clkdiv is
    port( mclk: in std_logic ;
        clr: in std_logic ;
        clk190: out std_logic;
        clk178: out std_logic_vector(1 downto 0 ));
  end component;

signal clk178: std_logic_vector(1 downto 0) ;
signal digit_BCD: std_logic_vector (1 downto 0);
signal digit_counter: std_logic_vector (1 downto 0);
signal refresh_counter: std_logic_vector(27 downto 0);
signal select_LED1,select_LED2,select_LED3: std_logic_vector (6 downto 0);
begin
A1: clkdiv
    port map(mclk => mclk,
    clr=> clr,
    clk178 => clk178);
```

```
process(clk178,enable)
begin
case clk178 is
when "00" => select_digit <= "0111";
digit_BCD <= data(1 downto 0);

when "01" =>
select_digit <= "1011";
digit_BCD <= data(3 downto 2);

when "10" =>
select_digit <= "1101";
digit_BCD <= data(5 downto 4);

when "11" =>
select_digit <= "1110";
digit_BCD <= data(7 downto 6);
end case;
if enable = '0'then
select_digit <="1111";
end if;
end process;
digit_BCD <= digit_BCD;
process(digit_BCD)
begin

case digit_BCD is
when "00" => select_LED1 <= "0000001"; -- "0"
when "01" => select_LED1 <= "1001111"; -- "1"
when "10" => select_LED1 <= "0010010"; -- "2"
when "11" => select_LED1 <= "0000110"; -- "3"

end case;


case digit_BCD is
when "00" => select_LED2 <= "0110001"; -- "C"
when "01" => select_LED2 <= "0110001"; -- "C"
when "10" => select_LED2 <= "0110001"; -- "C"
```

```
when "11" => select_LED2 <= "0110001"; -- "C"
end case;


  case digit_BCD is
when "00" => select_LED3 <= "0111000"; -- "F"
when "01" => select_LED3 <= "0111000"; -- "F"
when "10" => select_LED3 <= "0111000"; -- "F"
when "11" => select_LED3 <= "0111000"; -- "F"
end case;

end process;
process(cf) is
begin
if cf= "10" then
select_LED <= select_LED2;
elsif cf= "01" then
select_LED <= select_LED3;
else
select_LED <= select_LED1;
end if;
end process;
end Behavioral;
```