Abstract:

- Code a multiplayer Maze Game using VHDL for use on FGPA.
- Components used:
    1. Basys3
    2. VGA Monitor
    3. Keyboard
- If a player wins the game, a YOU WIN is displayed on that player's VGA screen and the win of the player is updated on the Seven Segment Display.
- If time runs out, TIME'S UP notification appears on the VGA monitor.
- A count down timer is observed on the Seven Segment Display.
- After the game has been won, it can be reset using a switch and the objects appear on their initial position.

Design Specification Plan:

- The maze is hard-coded because it allowed me to keep a track of the wall pixels so I can restrict the movement of my objects - Tim and Elif – whenever they came across these wall pixels.
- Both objects are represented by an x and y pixel which can then be used to draw Tim and Elif on the VGA monitor.
- I use a keyboard to control the movement of my objects because of their user-friendliness.
- To make the game more competitive, I introduce a countdown timer as well as a win counter.
- To make the game more appealing to the eye, the background of my maze is black while the walls and the objects are white.
- In order to communicate with VGA Monitor, I make use of the inbuilt clock of the FGPA board.

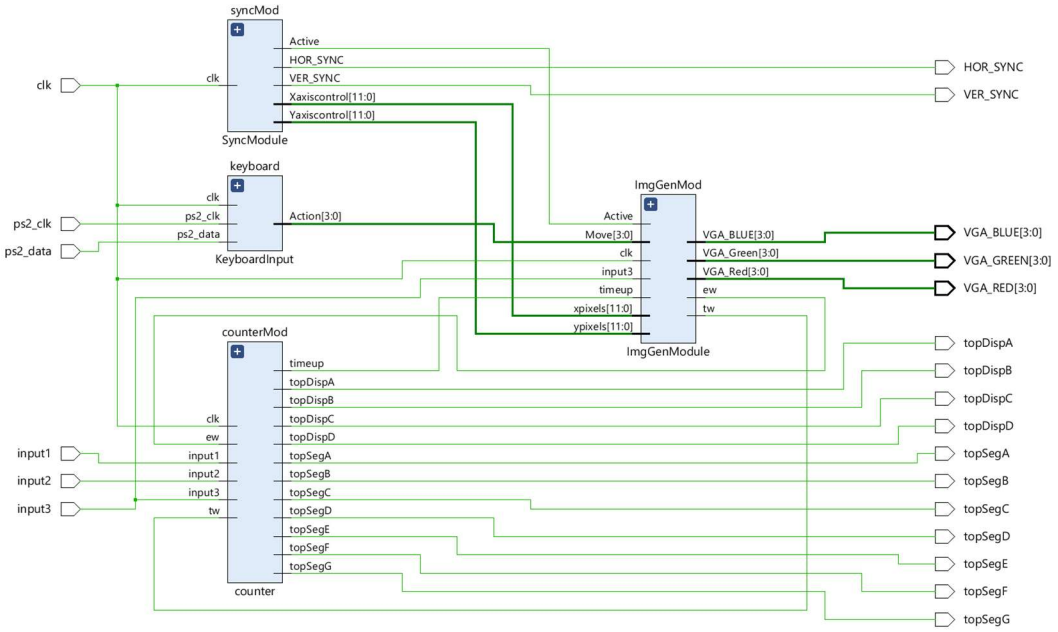
Design Methodology:

1. **Maze Generation:**
    - I hand drew a 6x8 block maze. I kept this maze same for both players to keep the game fair. Then, I assigned pixel values to the walls of the maze. I hard-coded these pixels value in my image generation module. The horizontal walls are coded such that their height is 7. The vertical walls have a width of 7. Whenever a pixel is found to exist in these wall pixel value ranges then that pixel is turned on and the image generation module paints a white pixel on the VGA monitor.
2. **Movement and Restriction:**
    - The step size of my objects is 1. Because I assigned pixel values to the maze myself, I kept a track of the outer pixel values of these walls. Due to movement, if my object came across any of these wall pixels, then my object position wasn't updated. I used a count that counted 12500 rising edges of my clock to serve my purpose of clock divider and read the input value from the keyboard after reasonable intervals of time. I only updated the movement of my objects between the end and start of two consecutive frames. All this is realized in the image generation module as well.
3. **Keyboard:**
    - I read the ps2 data signals from the keyboard and then convert them to their respective Ascii values representing keys on the keyboard. If the Ascii of any of the movement keys is observed e.g.: W, A, S and D, then an action signal is updated which is fed as input to the image generation module to be used in the movement and restriction code.

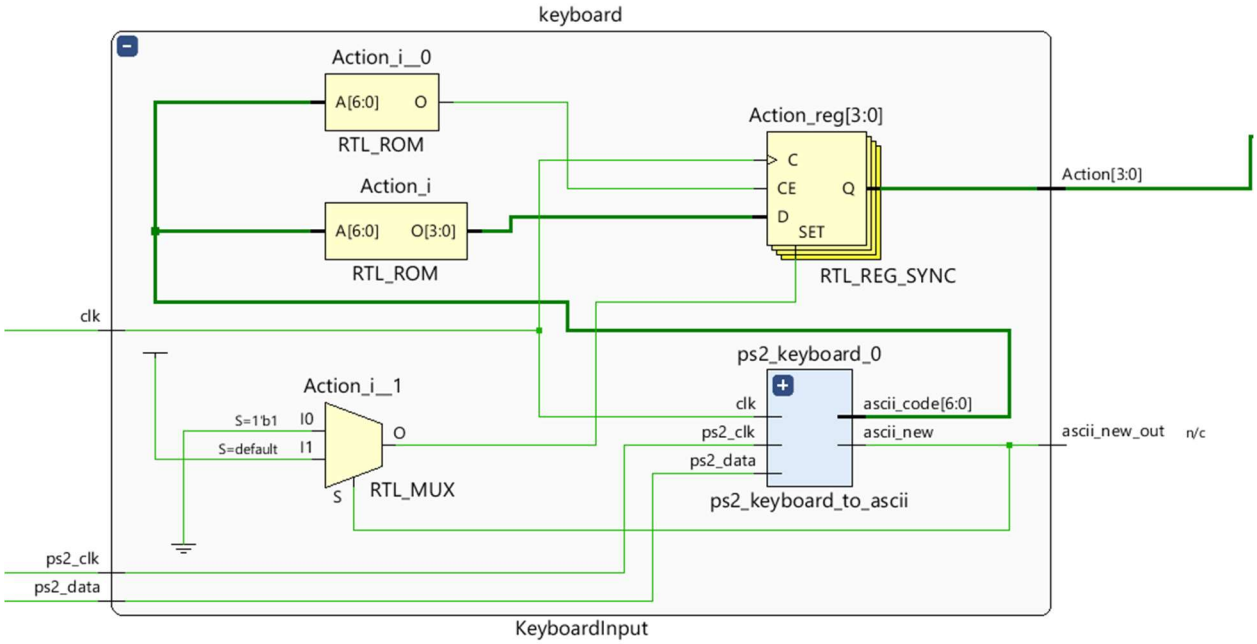
4. **Counter:**
    - Here, I create a count up timer that is displayed on the Seven Segment Display. The refresh rate of my anodes is 10.5ms. I use a segment decoder to decode the current time value into cathode values which is then used to display on the Seven Segment Display.

Modules:

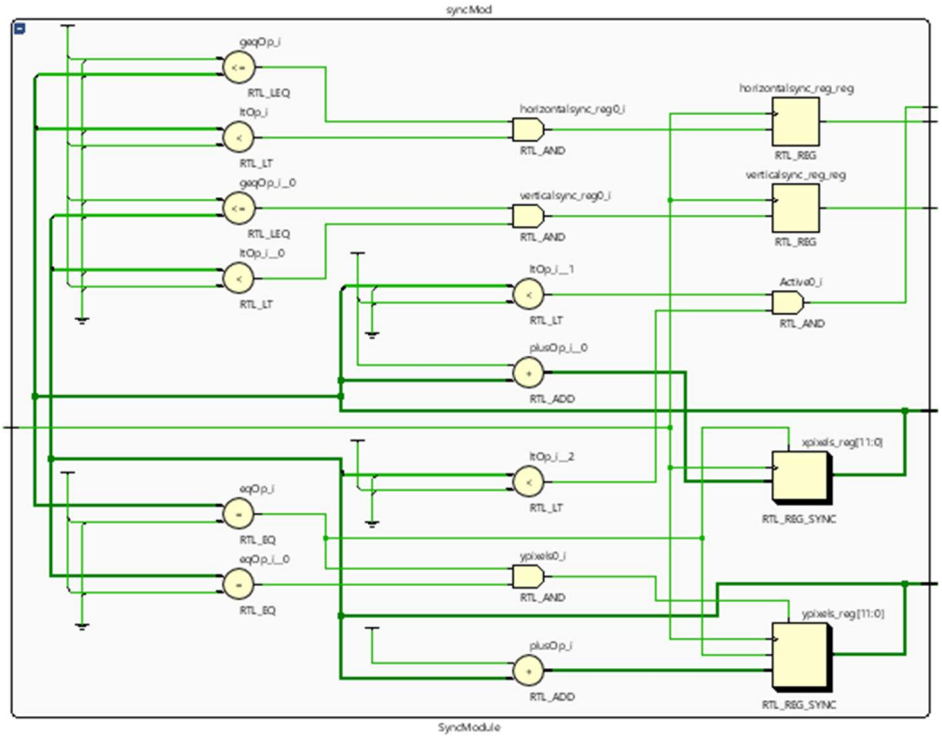
I make use of four modules to realize my game. In the following section, I describe each of these modules briefly and attach their schematic diagrams generated by VIVADO for better understanding as well.



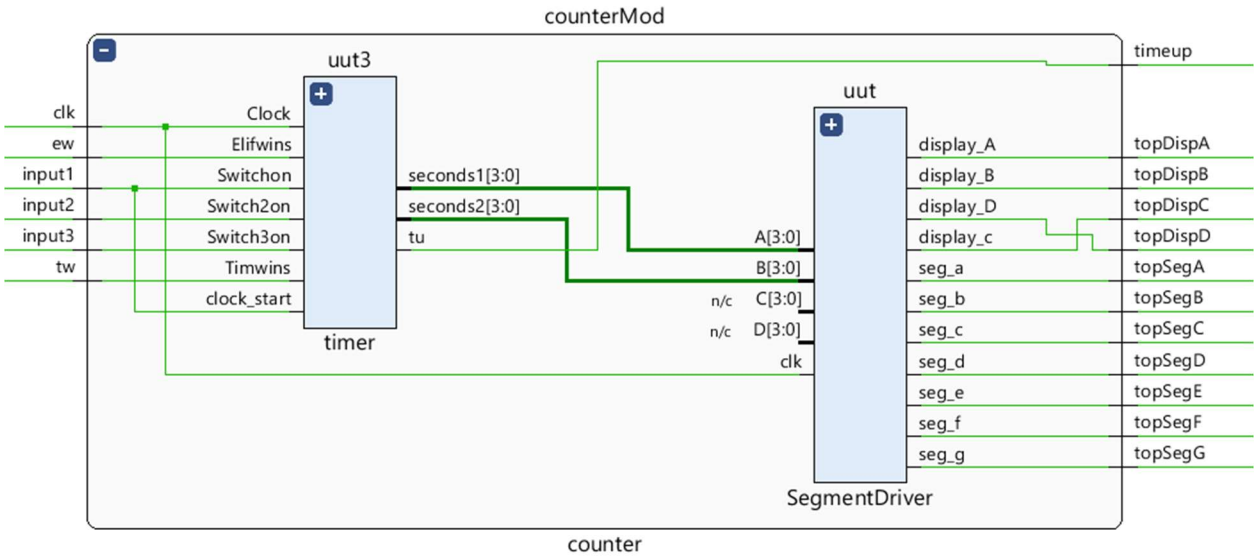
- **KEYBOARD:**
  I use this module to take input values from the user and then use them as input for the movement of my objects. Whenever a key is pressed, a ps2 data signal is generated and then a correspondent Ascii value is created. In my module, I look for the Ascii values of w, a, s , p, l, ; and ‘ because these control the movement of my objects. I create a signal in this module called Action which is updated whenever a key is pressed which is then fed into the Image Generation Module to be used by the Movement and Restriction part of the code. Whenever a key is not pressed, I assign the Action signal a value such that no movement takes place. I have taken the code from my friend Ali Khaqan – a 3rd year Electrical and Electronic Student at Bilkent University. I understand and edit the code to meet my purposes.



- **VGA SYNC:**
  This module generates Horizontal and Vertical Syncs with respect to pixels on the screen. The pixels are drawn horizontally one line at a time. After which, the next line below is drawn. Once the bottom line is reached, it goes back up to the top and starts again. This entire process of starting from the top and going to the bottom is considered a frame. The Horizontal and Vertical Syncs are fed into the top module to display the Maze game on the VGA Monitor.



- **Counter:**
  I took the basic code from my friend, Hatim Zahid, and then added the countup timer logic to achieve my goals. I input the timwin and elifwin signals to stop the counter when either of them wins so that the players can see the time of completion and compete for the best time. A time up signal is an output of this module and is fed into the image generation module to show the Time up notification when 60 seconds have passed.

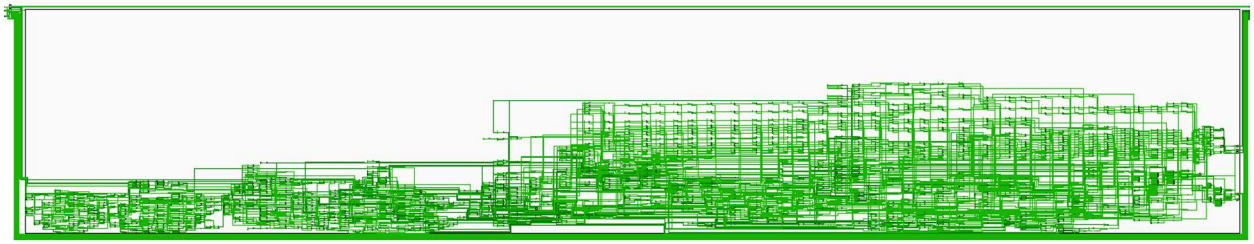


- **IMAGE GENERATION:**
  This is the main module of my Maze Game where I use my other modules to produce my Maze Game. This module is responsible of conveying information to the VGA monitor to display what and where on the screen. The onboard clock of FGPA is used to aid in the process. Every time a rising edge of the clock is observed or the clock signal turns from 0 to 1, a pixel is drawn. I achieve the following in this module:
  - Pixel values of the walls are jotted down. Whenever a pixel is found to be in these walls, it is turned on and a white pixel is drawn on the monitor.
  - I create 4 signals to keep a track of the x and y pixels of my objects. 2 for each. By adding and subtracting x and y pixels from these signals, I draw two 20x20 objects on the VGA Screen.
  - Whenever an Action is updated, my movement and restriction process is executed and it compares the new pixel value after the step size of 1 with my object. If a wall pixel is found, then the object's position is not updated. To move the box at reasonable speed, I make use of a count variable that serves the purpose a clock divider and only after 12500 rising clock edges have been recorded does it allow the object to move. Also, the object's position is only updated between the end and start of two consecutive frames to avoid choppy frames.
  - If the object reaches the bottom right corner of upper or lower maze, the game ends and a you win/you lose notification is displayed on the screen and the maze disappears. This is

checked by introducing two Tim_win and Elif_win signals in our code that our updated to '1' when the pixels of the bottom right corner of upper/lower maze come across Tim or Elif.

- If the countdown timer reaches 60, representing 60 seconds, a time up notification is observed on the monitor and the maze disappears. This time up signal is input from the counter module.
- The game can be thought of having 2 states:
  1) Game is being played
  2) Notification state

  In order to reset from the notification state and be able to play the game **again**, the user resets the game by using a switch that resets the game. This switch input is fed from the Top module.



Problems faced:

- Due to poor VHDL background, I struggled to understand the syntax of VHDL. I tried to use Arrays to store my wall pixels. For some reason – still unknown to me – my code did not work. I spent 2 days trying to figure this out, resorting to online resources and books but all in vain. Because my friends were unaware of the syntax and the teacher assistants could not help, I decided to make my code logically as simple as possible to ensure that it would run which is why I decided to hardcode my maze. I observed similar issues in my movement code.
- Having realized that I just had to be logically correct and avoid basic syntax errors, I coded my entire game over a span of 4 days and then decided to test it. To my dismay, VIVADO showed that my code was error free, yet my code would not synthesize. A couple of hours went by when I realized that something was wrong, and I decided to debug the code. However, again I made a wrong decision because my code was spanned over more than 700 line. I spent an entire day trying to figure my mistake out, but I couldn't. In the end, I decided to create separate codes and tested each component out individually where finally I was able to pinpoint my mistake – I had written pixelon <= p; when I should have written p <= pixelon; in my image generation module. After this my code worked. Having learned from this, I decided to test each code individually and then build on it. I started writing different codes and then adding them together one by one ensuring they worked together in harmony.
- The most troubling phase of my entire project was trying to control the movement of my objects. Even though I was logically correct, my object pixels would either spread or they would move

unreasonable distances. I thought that there was something wrong with my logic or syntax as before so I spent two days just trying to figure this out before realizing that I need to step back and understand the big picture which was that I can not update the objects' positions while a frame is being drawn because then the VGA would not know where to draw the object. As soon as I resolved this issue, I saw that my pixels were no longer spreading; the frames were not getting chopped. This made it easier for me to observe that my movement object logic was correct, however, the image generation module was reading the action key in very short intervals of time which is why it appeared that the object was moving to fast, making it seem that it was moving unreasonable distances. To tackle the issue, I introduced a count in my code which served the purpose of a clock divider and only read data from the keys after 12500 rising edges of the clock went by. I have learned from this experience that one should always doubt his or her logic and syntax but at the same time consider the possibility that he or she has a knowledge gap about the component being used.

- I also came across some syntax mistakes every now and then while I coded. For example, the "if (risingedge_clk) then" statement cannot be written under an if (risingedge_clk) statement and that it has to be the most outer if statement. I also observed that a variable is updated after the entire process has executed which is why in my movement module my timwin <= '1' when the end of game condition is reached, and it does not depend on any variable updated in this same process because they are only updated after the process has executed. I only realized the previous problem when my countdown timer was not stopping when either Tim or Elif had won so I kept coding different logics in my counter module until the point where I realized that I could not be wrong. Only then by trial and error, I became aware of this issue.
- I also faced a lot of issues incorporating modules. The first time I coded the entire game I created a lot of modules and, in the end, I dreaded the process because there was too much to keep a track of which is why the next time, I recoded the game I only made 5 modules in total. This saved me a lot of time. In order to avoid confusion, I made extra use of statements in my code that eased the process of identifying what a certain section of the code is doing.
- The keyboard code that I used stored the Action performed by a certain Ascii values; that is to say, after a key was pressed and no new key was pressed the action to be performed by the pressed key kept on happening. This created problems for me because my object should only move when the key was pressed. Since I had not written the entire code by myself, I thought the problem lied somewhere in conversion of ps2 data signals to ascii values or maybe even at the Debounce section of the code! I was wrong. After understanding the code, I realized that I just had to introduce a new Action which would not perform any movement if a new Ascii value was not detected.

Result:

Successful implementation of Multiplayer Maze Game is observed. As expected, the user is able to move both the objects via Keyboard. Whenever Tim or Elif – my maze game objects – clear the maze a "You Win/You Lose" notification is displayed on the VGA Monitor. If the countup timer is set up and players are unable to finish the game in 60 seconds then a Time Up notification is shown on the VGA Screen. The Seven Segment Display, as demanded, show the count up timer. In order to revert back to the playable state, the user is also able to make use of switch W16 and reset the game. Overall, the game performs the functions it was intended to perform. I attach a couple of pictures below showcasing different components and states of my game.

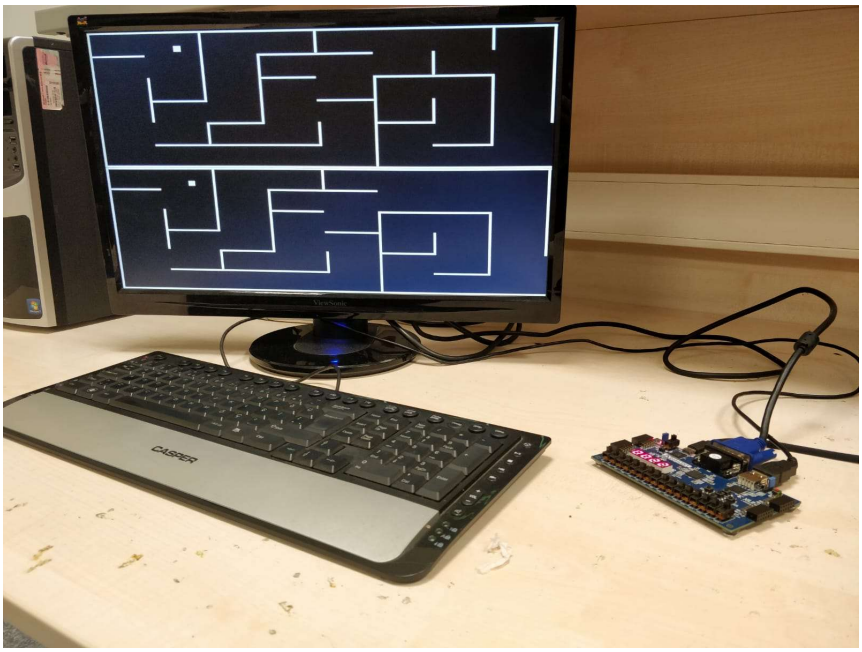
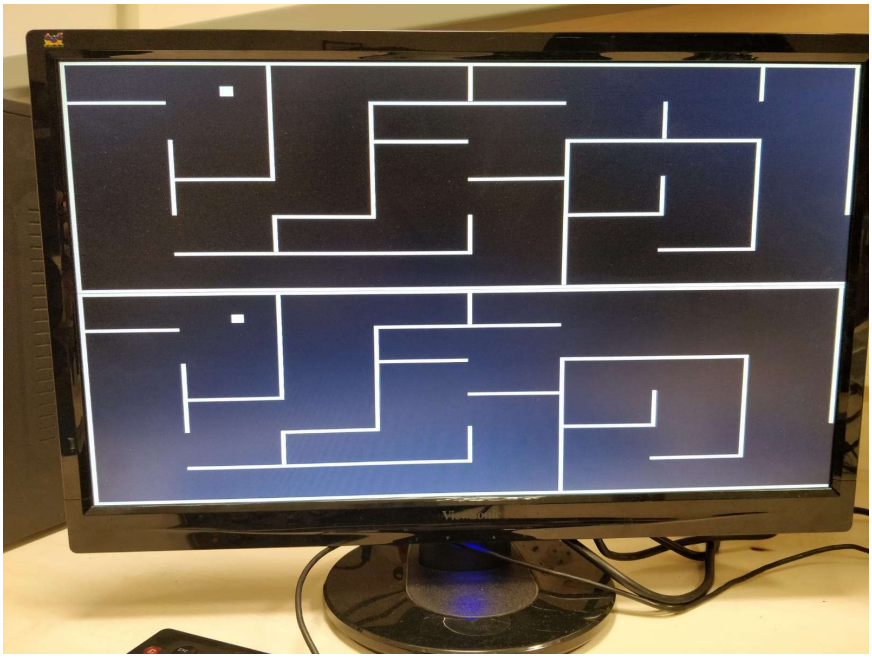
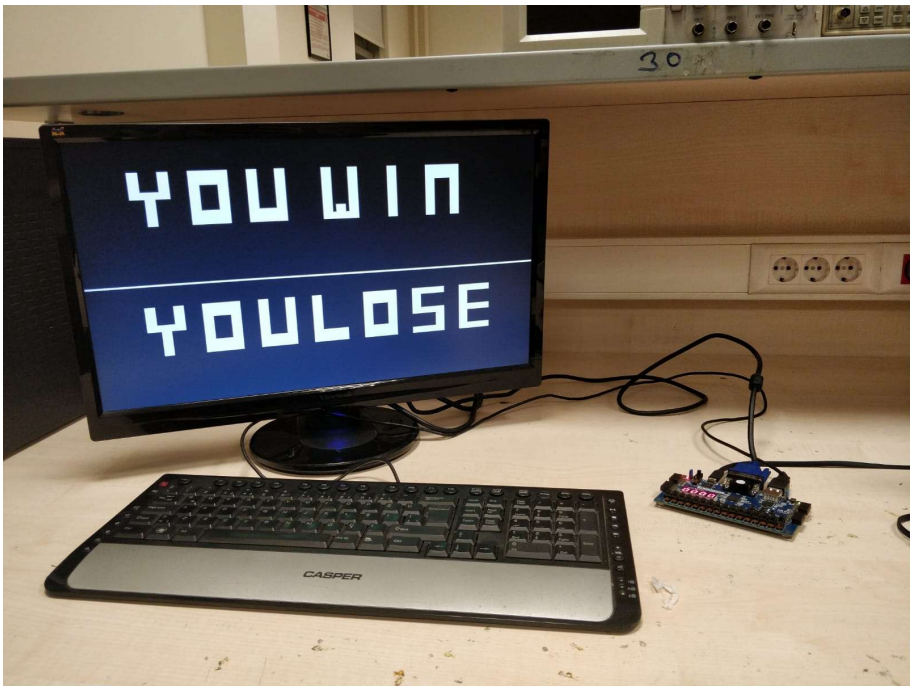


Image showcasing all the components utilized by my MULTIPLAYER MAZE GAME

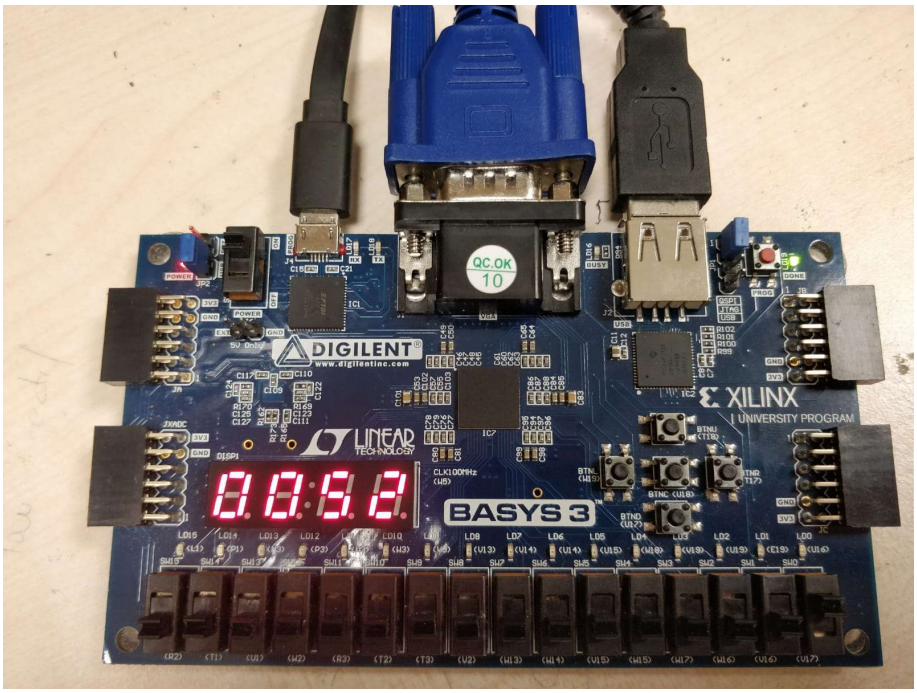


Maze and objects displayed on the VGA Monitor.

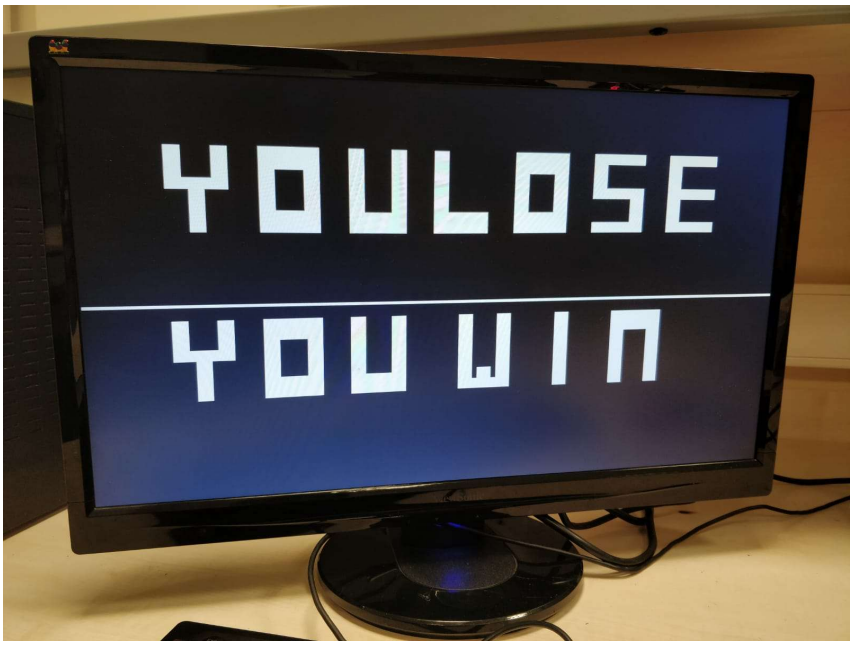
- Black background
- Walls are white
- Objects are white



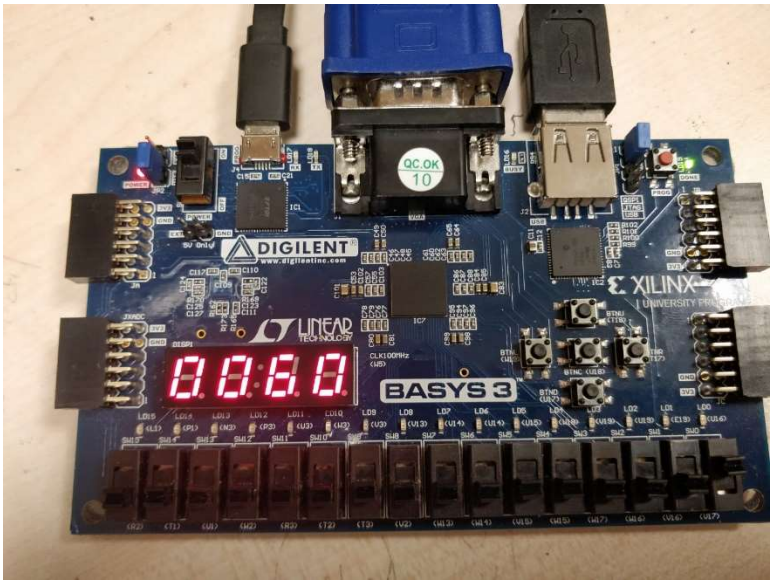
Tim wins the game



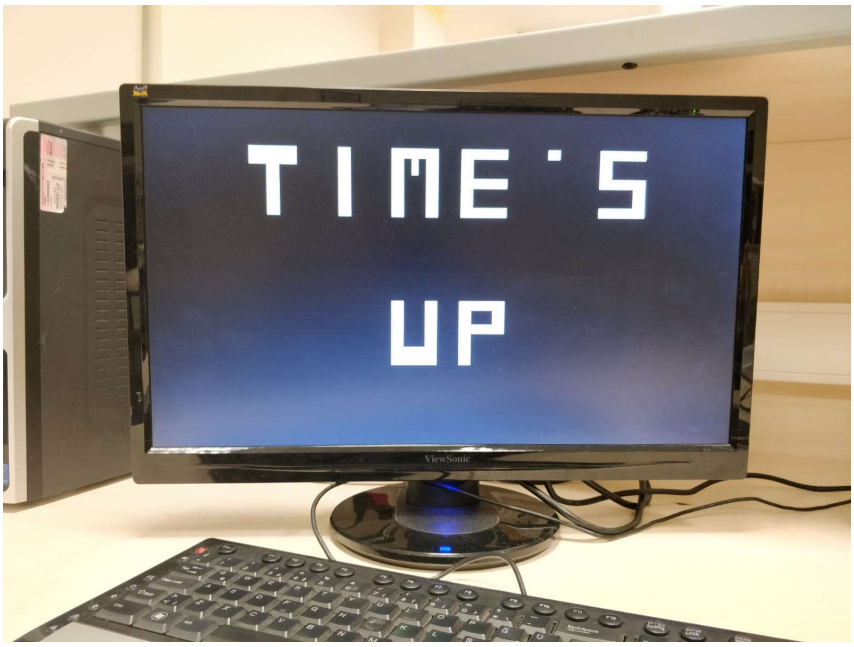
Count up timer showing 52 seconds have passed and seven segment display continues to increment



Elif wins the game



Count up timer reaches 60 seconds and stops incrementing. A time up signal is sent to the image generation module at this point. As a result, the image generation module generates a Time up Notification on the VGA monitor



When 60 seconds reached, The TIME UP notification is displayed

Conclusion:

This Digital Term Project has greatly enriched my understanding of the VHDL, Basys3, VGA Monitors, and Keyboard. I have started to understand the complexities and minute features of each. Because of little help available, I pondered over the issues I described earlier for days and days on before coming across solutions to them. As a result, I believe that I now possess the basic set of skills and thinking required to debug and implement codes from scratch. I have developed a profound interest in coding and hope to take on courses and projects that challenge me more in this similar field. I strongly believe that the set of skills I have developed due to the Multiplayer Maze Game - my digital term project – are not limited to VHDL and will greatly aid me in my future projects.

VIDEO PRESENTATION:

https://drive.google.com/file/d/1A46ZD7v3ndeSbak7oxTeCS5qz1u-bbCz/view?usp =drivesdk

REFERENCES:

- Counter Module taken from Hatim Zahid – 3rd year student Bilkent University
- Keyboard Module taken from Ali Khaqan – 3rd year student Bilkent University

VHDL CODES:

TOP Module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.std_logic_unsigned.all;
--use work.Arrays.all;


entity TopModule is
GENERIC(
clk_freq : INTEGER := 100_000_000; --system clock frequency in Hz
ps2_debounce_counter_size : INTEGER := 9); --set such that 2^size/clk_freq = 5us (size = 8 for 50MHz)

PORT(

   clk : in STD_LOGIC; --system clock input
   ps2_clk : in STD_LOGIC; --clock signal from PS2 keyboard
   ps2_data : in STD_LOGIC; --data signal from PS2 keyboard
--declaring in/out for VGA MONITOR
   HOR_SYNC: out std_logic;
   VER_SYNC: out std_logic;
   VGA_RED: out std_logic_vector (3 downto 0);
   VGA_BLUE: out std_logic_vector (3 downto 0);
   VGA_GREEN: out std_logic_vector (3 downto 0);

--ssd inputs and outputs
        input1 : in std_logic;
        input2:in std_logic;
        input3:in std_logic;
        topDispA : out std_logic ;
        topDispB : out std_logic ;
        topDispC : out std_logic ;
```

```
        topDispD : out std_logic ;
        topSegA : out std_logic ;
        topSegB : out std_logic ;
        topSegC : out std_logic ;
        topSegD : out std_logic ;
        topSegE : out std_logic ;
        topSegF : out std_logic ;
        topSegG : out std_logic
  );

end TopModule;



architecture Behavioral of TopModule is

component SyncModule is
    Port (
    clk: in std_logic;
    xaxiscontrol: out std_logic_vector (11 downto 0);
    yaxiscontrol: out std_logic_vector (11 downto 0);
    HOR_SYNC: out std_logic;
    VER_SYNC: out std_logic;
    Active : out std_logic
    );
end component;



component ImgGenModule is
Port(
  clk: in std_logic;
  xpixels: in std_logic_vector (11 downto 0); -- pixels on x
  ypixels: in std_logic_vector (11 downto 0); -- pixels on y
```

```
  VGA_BLUE: out std_logic_vector (3 downto 0);
  input3: in std_logic;
  VGA_Red: out std_logic_vector (3 downto 0);
  VGA_Green: out std_logic_vector (3 downto 0);
  tw: out std_logic;
  ew: out std_logic;
  Active : in std_logic;
  timeup: in std_logic;
  Move: in std_logic_vector (3 downto 0));

end component;

COMPONENT KeyboardInput is
GENERIC(
     clk_freq                  : INTEGER := 100_000_000; --system clock frequency in Hz
     ps2_debounce_counter_size : INTEGER := 9);         --set such that 2^size/clk_freq = 5us (size = 8 for
50MHz)
   Port (
         clk : in STD_LOGIC;  -- System Clock
         ps2_clk : in STD_LOGIC;  --Clock Signal from ps2 keyboard
         ps2_data : in STD_LOGIC;  --Data signal from ps2 keyboard
         ascii_new_out : out STD_LOGIC;
         Action : out STD_LOGIC_VECTOR (3 downto 0)); --Action to be taken when a certain key is
pressed.
end COMPONENT;

Component counter is
   Port (
         input1 : in std_logic;
         input2:in std_logic;
         input3: in std_logic;
         tw: in std_logic;
         ew: in std_logic;
```

```
        clk : in STD_LOGIC ;
        topDispA : out std_logic ;
        topDispB : out std_logic ;
        topDispC : out std_logic ;
        topDispD : out std_logic ;
        topSegA : out std_logic ;
        topSegB : out std_logic ;
        topSegC : out std_logic ;
        topSegD : out std_logic ;
        topSegE : out std_logic ;
        topSegF : out std_logic ;
        timeup : out std_logic;
        topSegG : out std_logic  );

end component;



signal pixelsonx: std_logic_vector (11 downto 0);
signal pixelsony: std_logic_vector (11 downto 0);
signal clock : std_logic;
signal s_ascii_out : std_logic ;
signal s_action  : std_logic_vector(3 downto 0 );
signal pixelactive: std_logic;
signal Twins: std_logic := '0';
signal Ewins: std_logic := '0';
signal stopsignal: std_logic := '0';



begin
clock <= clk;
syncMod: SyncModule port map(
   clk => clk ,
```

```
    yaxiscontrol => pixelsony ,
    xaxiscontrol => pixelsonx ,
    HOR_SYNC => HOR_SYNC,
    Active => pixelactive,
    VER_SYNC => VER_SYNC
    );
ImgGenMod: ImgGenModule port map (
    clk => clock ,
    VGA_RED => VGA_RED ,
    VGA_GREEN => VGA_GREEN ,
    timeup => stopsignal,
    input3 => input3,
    VGA_BLUE => VGA_BLUE ,
    ypixels => pixelsony ,
    xpixels => pixelsonx,
    Active => pixelactive,
    tw => Twins,
    ew => Ewins,
    Move => s_action
    );

keyboard: KeyboardInput
      GENERIC MAP( clk_freq => clk_freq,
                ps2_debounce_counter_size => ps2_debounce_counter_size)
      PORT MAP( clk => clock,
              ps2_clk => ps2_clk,
              ps2_data => ps2_data,
              ascii_new_out => s_ascii_out,
              Action => s_action);
counterMod: counter
Port Map(
          input1 => input1,
          input2=> input2,
```

```
        input3 => input3,
        tw => Twins,
        ew => Ewins,
        clk => clk ,
        topDispA => topDispA,
        topDispB => topDispB,
        topDispC => topDispC,
        topDispD => topDispD,
        topSegA => topSegA,
        topSegB => topSegB,
        topSegC => topSegC,
        topSegD => topSegD,
        topSegE => topSegE,
        topSegF => topSegF,
        topSegG => topSegG,
        timeup => stopsignal);
end Behavioral;
```

Sync Module:

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

entity SyncModule is
Port (
   clk : in STD_LOGIC;
   Yaxiscontrol: out STD_LOGIC_VECTOR (11 downto  0);
   Xaxiscontrol : out STD_LOGIC_VECTOR (11 downto 0);
   HOR_SYNC : out STD_LOGIC;
   VER_SYNC : out STD_LOGIC;
   Active : out std_logic
);
```

```
end SyncModule;

architecture Behavioral of SyncModule is


--***1280x1024@60Hz***--
constant FRAME_WIDTH : natural := 1280;
constant FRAME_HEIGHT : natural := 1024;
constant HOR_FP : natural := 48; --HOR front porch width (pixels)
constant HOR_PW : natural := 112; --HOR sync pulse width (pixels)
constant HOR_MAX : natural := 1688; --HOR total period (pixels)
constant VER_FP : natural := 1; --V front porch width (lines)
constant VER_PW : natural := 3; --V sync pulse width (lines)
constant VER_MAX : natural := 1066; --V total period (lines)
constant HOR_POL : std_logic := '1';
constant VER_POL : std_logic := '1';
 --signal active : std_logic;


-- Horizontal and Vertical counters
signal xpixels : std_logic_vector(11 downto 0) := (others =>'0');
signal ypixels : std_logic_vector(11 downto 0) := (others =>'0');

-- Horizontal and Vertical Sync
signal horizontalsync_reg : std_logic := not(HOR_POL);
signal verticalsync_reg : std_logic := not(VER_POL);

begin
-- Generate Horizontal, Vertical counters and the Sync signals

------------------------------------------------------------
Horizontalcounter: process (clk)
begin
```

```
if (rising_edge(clk)) then
   if (xpixels = (HOR_MAX - 1)) then
      xpixels <= (others =>'0');
   else
      xpixels <= xpixels + 1;
   end if;
end if;
end process;
Verticalcounter: process (clk)
begin
if (rising_edge(clk)) then
   if ((xpixels = (HOR_MAX - 1)) and (ypixels = (VER_MAX - 1))) then
      ypixels <= (others =>'0');
   elsif (xpixels = (HOR_MAX - 1)) then
   ypixels <= ypixels + 1;
   end if;
end if;
end process;

-- Horizontalsync
process (clk)
begin
if (rising_edge(clk)) then
   if (xpixels >= (HOR_FP + FRAME_WIDTH - 1)) and (xpixels < (HOR_FP + FRAME_WIDTH +
HOR_PW - 1)) then
         horizontalsync_reg <= HOR_POL;
      else
         horizontalsync_reg <= not(HOR_POL);
   end if;
end if;
end process;
-- Vertical sync
process (clk) begin
```

```
if (rising_edge(clk)) then
   if (ypixels >= (VER_FP + FRAME_HEIGHT - 1)) and (ypixels < (VER_FP + FRAME_HEIGHT +
VER_PW - 1)) then
      verticalsync_reg <= VER_POL;
   else
      verticalsync_reg <= not(VER_POL);
   end if;
end if;
end process;
-- active signal
active <= '1' when xpixels < FRAME_WIDTH and ypixels < (FRAME_HEIGHT) else '0';
Yaxiscontrol <= ypixels ;
Xaxiscontrol <= xpixels ;
HOR_SYNC <= horizontalsync_reg ;
VER_SYNC <= verticalsync_reg ;

end Behavioral;
```

Image Generation Module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

--use work.Arrays.all;

entity ImgGenModule is
  Port (
  clk: in std_logic;
  xpixels: in std_logic_vector (11 downto 0); -- pixels on x
  ypixels: in std_logic_vector (11 downto 0); -- pixels on y
  VGA_BLUE: out std_logic_vector (3 downto 0);
```

```
  VGA_Red: out std_logic_vector (3 downto 0);
  VGA_Green: out std_logic_vector (3 downto 0);
  timeup:in std_logic;
  tw:out std_logic;
  ew: out std_logic;
  input3: in std_logic;
  Active : in std_logic;
  Move: in std_logic_vector (3 downto 0));

  --wallpixels: out myarray
--   );
end ImgGenModule;


architecture Behavioral of ImgGenModule is

signal x: std_logic_vector (11 downto 0); -- a variable declared to ease in process of writing
signal y: std_logic_vector (11 downto 0);
signal z: integer;
signal t: integer;
signal p: std_logic; -- same as above
signal VGA_RED_COMB: std_logic_vector (3 downto 0);
signal VGA_BLUE_COMB: std_logic_vector (3 downto 0);
signal VGA_GREEN_COMB: std_logic_vector (3 downto 0);
signal VGA_RED_TEMP: std_logic_vector (3 downto 0);
signal VGA_BLUE_TEMP: std_logic_vector (3 downto 0);
signal VGA_GREEN_TEMP: std_logic_vector (3 downto 0);
signal pixelon: std_logic;
Signal Tyc: std_logic_vector (11 downto 0):="000000111111" ;
Signal Txc: std_logic_vector (11 downto 0):="000011111111" ;
Signal Eyc: std_logic_vector (11 downto 0):="001000111111" ;
Signal Exc: std_logic_vector (11 downto 0):="000011111111" ;
signal count: integer := 0;
```

```
signal objectpixelon: std_logic:= '0';
signal timwins: std_logic := '0';
signal elifwins: std_logic := '0';
signal tup: std_logic := '0';
signal o: integer;
signal reset: std_logic;
signal sync: std_logic := '1';

begin
x <= xpixels;
y <= ypixels;
reset <= input3;

--process ( Sync)
--begin
--    if ( rising_edge(Sync)) then
--        if input3 = '1' then
--            Tyc <= "000000111111";
--            Txc <= "000011111111";
--            Eyc <= "001000111111";
--            Exc <="000011111111";
--            timwins <= '0';
--            elifwins <= '0';
--         end if;
--     end if;
--end process;

process (x,y, Timwins, Elifwins, timeup)
begin
z <= 0;
o<= 50;
t <= 512;
--if count = 0 then
```

```
--tw <=timwins;
--ew <= elifwins;
--end if;
--t <= 512;
--IF input3 = '1' then
-- Tyc<="000000111111" ;
-- Txc<="000011111111" ;
-- Eyc<="001000111111" ;
-- Exc<="000011111111" ;
--end if;
if Timeup = '1' then
--time up notification
   --writing T
   if (x > 214 and x < 316) and (y > z+55+50 and y < z+107+50) then
     pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   elsif (x > 251 and x < 279)and (y > z+105+50 and y <z+257+50) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   --writing I
   elsif (x > 401 and x < 429) and (y > z+55+50 and y < z+257+50) then
     pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   --writing M
   elsif (x > 514 and x < 616)and (y > z+55+50 and y <z+56+27+50) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
```

```
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  elsif (x > 514 and x < 542)and (y > z+55+26+50 and y <z+257+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  elsif (x > 588 and x < 616)and (y > z+55+26+50 and y <z+257+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  elsif (x > 557 and x < 573)and (y > z+81+50 and y <z+132+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  --writing E


  elsif (x > 664 and x <766)and ((y > z+55+50 and y < z+83+50) or (y>z+142+50 and y<z+170+50) or
(y>z+229+50 and y<z+257+50)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  elsif (x > 664 and x < 692) and((y>z+81+50 and y<z+144+50)or (y>z+168 and y<z+231+50)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
   --wrting '
  elsif (x > 851 and x <879)and (y > z+55+50 and y < z+83+50)  then
    pixelon <= '1';
```

```
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   --writing S
  elsif (x > 964 and x < 1066)and ((y > z+55+50 and y < z+83+50) or (y>z+142+50 and y<z+170+50) or
(y>z+229+50 and y<z+257+50)) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   elsif (x > 964 and x < 992)  and(y>z+81+50 and y<z+144+50)  then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
    elsif(x>1038 and x<1066) and (y>z+168+50 and y<z+231+50) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
    --writing UP
    --writing U
     elsif (x > 514 and x < 542)  and(y>t+55 and y<t+201)  then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
    elsif(x>588 and x<616) and (y>t+55 and y<t+201) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
    elsif(x>514 and x<616) and (y>t+200 and y<t+257) then
```

```
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
 --writing P
elsif(x>664 and x<766) and ((y>t+55 and y<t+83) or (y>t+129 and y<t+157)) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif(x>664 and x<692) and (y>t+81 and y<t+131) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif(x>738 and x<766) and (y>t+81 and y<t+131) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif(x>664 and x<692) and (y>t+155 and y<t+257) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
else
    pixelon <= '0';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
 end if;
```

```
elsif Timwins = '1' then
--notify that tim wins = player1

--player 1 notification
   --writing Y
   if ((x > 139 and x < 167) or (x>213 and x<241)) and (y > o+56 and y < o+56+74) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   elsif (x > 139 and x < 241) and (y > o+129 and y < o+157) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   elsif (x > 140+36 and x < 140+63)and (y > o+155 and y <o+257) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   --writing O
   elsif (x > 289 and x < 317) and (y > o+55+52 and y < o+149+56) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   elsif (x>363 and x<391) and (y > o+55+52 and y < o+149+56) then
      pixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
   elsif (x > 289 and x < 391) and ((y > o+55 and y < o+109) or (y>o+203 and y<o+257)) then
      pixelon <= '1';
```

```
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
--writing U
elsif (x > 439 and x < 467) and (y > o+55 and y < o+207) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x>513 and x<541) and (y > o+55 and y < o+207) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 439 and x < 541) and (y > o+205 and y < o+257) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
--writing W
elsif (x > 639 and x < 667) and (y > o+55 and y < o+231) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x>713 and x<741) and (y > o+55 and y < o+231) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 639 and x < 741) and (y > o+229 and y < o+257) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
```

```
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 676 and x < 704) and (y > o+199 and y < o+231) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing I
    elsif (x > 826 and x < 854) and (y > o+55 and y < o+257) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing n
    elsif (x>939 and x<967) and (y > o+107 and y < o+257) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 1013 and x < 1041) and (y > o+107 and y < o+257) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 939 and x < 1041) and (y > o+55 and y < o+109) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
--player 2 notification
    --writing Y
    elsif ((x > 139 and x < 167) or (x>213 and x<241)) and (y > t+56+50 and y < t+56+74+50) then
        pixelon <= '1';
```

```
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 139 and x < 241) and (y > t+129+50 and y < t+157+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 140+36 and x < 140+63)and (y > t+155+50 and y < t+257+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing O
    elsif (x > 289 and x < 317) and (y > t+55+52+50 and y < t+149+56+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x>363 and x<391) and (y > t+55+52+50 and y < t+149+56+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 289 and x < 391) and ((y > t+55+50 and y < t+109+50) or (y>t+203+50 and y<t+257+50))
then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing U
    elsif (x > 439 and x < 467) and (y > t+55+50 and y < t+207+50) then
        pixelon <= '1';
```

```
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x>513 and x<541) and (y > t+55+50 and y < t+207+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 439 and x < 541) and (y > t+205+50 and y < t+257+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
--writing L
elsif (x > 589 and x < 617) and (y > t+55+50 and y < t+205+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 589 and x < 691) and (y > t+203+50 and y < t+257+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 --writing O
elsif (x > 739 and x < 767) and (y > t+55+52+50 and y < t+149+56+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x>813 and x<841) and (y > t+55+52+50 and y < t+149+56+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
```

```
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 739 and x < 841) and ((y > t+55+50 and y < t+109+50) or (y>t+203+50 and y<t+257+50))
then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing S
    elsif (x > 889 and x < 991)and ((y > t+55+50 and y < t+83+50) or (y>t+142+50 and y<t+170+50) or
(y>t+229+50 and y<t+257+50)) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 889 and x < 917)  and(y>t+81+50 and y<t+144+50)  then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
     elsif(x>963 and x<991) and (y>t+168+50 and y<t+231+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing E
    elsif (x > 1039 and x <1139)and ((y > t+55+50 and y < t+83+50) or (y>t+142+50 and y<t+170+50) or
(y>t+229+50 and y<t +257+50)) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 1039 and x < 1067) and((y>t+81+50 and y<t+144+50)or (y>t+168+50 and y<t+231+50))
then
```

```
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x>0 and x<1281) and (y>529 and y<537) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    else
        pixelon <= '0';
    end if;



elsif Elifwins = '1' then
--notfy that Elif wins = player 2
    --writing Y
    if ((x > 139 and x < 167) or (x>213 and x<241)) and (y > t+56 and y < t+56+74) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 139 and x < 241) and (y > t+129 and y < t+157) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 140+36 and x < 140+63)and (y > t+155 and y <t+257) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
```

```
--writing O
elsif (x > 289 and x < 317) and (y > t+55+52 and y < t+149+56) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x>363 and x<391) and (y > t+55+52 and y < t+149+56) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 289 and x < 391) and ((y > t+55 and y < t+109) or (y>t+203 and y<t+257)) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
--writing U
elsif (x > 439 and x < 467) and (y > t+55 and y < t+207) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x>513 and x<541) and (y > t+55 and y < t+207) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 439 and x < 541) and (y > t+205 and y < t+257) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
--writing W
```

```
elsif (x > 639 and x < 667) and (y > t+55 and y < t+231) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x>713 and x<741) and (y > t+55 and y < t+231) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 639 and x < 741) and (y > t+229 and y < t+257) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 676 and x < 704) and (y > t+199 and y < t+231) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
--writing I
elsif (x > 826 and x < 854) and (y > t+55 and y < t+257)  then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
--writing n
elsif (x>939 and x<967) and (y > t+107 and y < t+257) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 1013 and x < 1041) and (y > t+107 and y < t+257) then
```

```
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 939 and x < 1041) and (y > t+55 and y < t+109) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
--player 2 notification
    --writing Y
    elsif ((x > 139 and x < 167) or (x>213 and x<241)) and (y > o+56+50 and y < o+56+74+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 139 and x < 241) and (y > o+129+50 and y < o+157+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 140+36 and x < 140+63)and (y > o+155+50 and y < o+257+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing O
    elsif (x > 289 and x < 317) and (y > o+55+52+50 and y < o+149+56+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x>363 and x<391) and (y > o+55+52+50 and y < o+149+56+50) then
```

```
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 289 and x < 391) and ((y > o+55+50 and y < o+109+50) or (y>o+203+50 and y<o+257+50))
then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing U
    elsif (x > 439 and x < 467) and (y > o+55+50 and y < o+207+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x>513 and x<541) and (y > o+55+50 and y < o+207+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 439 and x < 541) and (y > o+205+50 and y < o+257+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    --writing L
    elsif (x > 589 and x < 617) and (y > o+55+50 and y < o+205+50) then
        pixelon <= '1';
        VGA_RED_TEMP <= "1111";
        VGA_Blue_TEMP <= "1111";
        VGA_Green_TEMP <= "1111";
    elsif (x > 589 and x < 691) and (y > o+203+50 and y < o+257+50) then
```

```
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 --writing O
elsif (x > 739 and x < 767) and (y > o+55+52+50 and y < o+149+56+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x>813 and x<841) and (y > o+55+52+50 and y < o+149+56+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 739 and x < 841) and ((y > o+55+50 and y < o+109+50) or (y>o+203+50 and y<o+257+50))
then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
--writing S
elsif (x > 889 and x < 991)and ((y > o+55+50 and y < o+83+50) or (y>o+142+50 and y<o+170+50) or
(y>o+229+50 and y<o+257+50)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 889 and x < 917)  and(y>o+81+50 and y<o+144+50)  then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
```

```
 elsif(x>963 and x<991) and (y>o+168+50 and y<o+231+50) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  --writing E
  elsif (x > 1039 and x <1139)and ((y > o+55+50 and y < o+83+50) or (y>o+142+50 and y<o+170+50)
or (y>o+229+50 and y<o +257+50)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  elsif (x > 1039 and x < 1067) and((y>o+81+50 and y<o+144+50)or (y>o+168+50 and y<o+231+50))
then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  elsif (x>0 and x<1281) and (y>529 and y<537) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
  else
    pixelon <= '0';
  end if;



else
--drawing maze for player 1
if (x > 0 and x<1281) and (y > z+0 and y < z+8) then
  pixelon <= '1';
  VGA_RED_TEMP <= "1111";
```

```
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif ((x > 0 and x < 160) or (x>477 and x<796)) and (y > (z+84) and y < (z+8+84)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif ((x > 477 and x < 637) or (x>795 and x<1114)) and (y > z+168 and y < z+8+168) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif ((x > 159 and x < 319) or (x>636 and x<796)) and (y > z+252 and y < z+8+252) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 elsif ((x > 318 and x < 478) or (x>795 and x<955)) and (y > z+336 and y < z+8+336) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 elsif ((x > 159 and x < 637) or (x>954 and x<1114)) and (y > z+420 and y < z+8+420) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 elsif ((x > 0 and x<1281)) and (y > z+504 and y < z+8+504) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
```

```
elsif (x > 0 and x < 8) and (y > z+0 and y < z+512) then --draw first column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 159 and x < 167) and (y > z+168 and y < z+336) then --draw second column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 318 and x < 326) and ((y > z+0 and y < z+260) or (y > z+336 and y < z+420)) then --draw third
column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 477 and x < 485) and (y > z+84 and y < z+344) then --draw 4th column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 636 and x < 644) and ((y > z+0 and y < z+84) or (y > z+336 and y < z+428)) then --draw 5th
column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 795 and x < 803) and (y > z+168 and y < Z+504) then --draw 6th column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
```

```
elsif (x > 954 and x < 962) and ((y > z+84 and y < z+168) or (y > z+252 and y < z+344)) then --draw 7th column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 1113 and x < 1121) and ((y > z+0 and y < z+84) or (y > z+168 and y < z+428)) then --draw 8th column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 1272 and x < 1280) and ((y > z+0 and y < z+345)) then --draw 9th column
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";



--drawing Tim
   elsif ((x > Txc - 11 and x<Txc + 11)) and (y > Tyc - 11 and y < Tyc + 11) then --draw tim
      pixelon <= '1';
      objectpixelon <= '1';
      VGA_RED_TEMP <= "1111";
      VGA_Blue_TEMP <= "1111";
      VGA_Green_TEMP <= "1111";
elsif ((x > 0 and x<1281)) and (y > t+0 and y < t+8) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif ((x > 0 and x < 160) or (x>477 and x<796)) and (y > (t+84) and y < (t+8+84)) then
```

```
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif ((x > 477 and x < 637) or (x>795 and x<1114)) and (y > t+168 and y < t+8+168) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif ((x > 159 and x < 319) or (x>636 and x<796)) and (y > t+252 and y < t+8+252) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 elsif ((x > 318 and x < 478) or (x>795 and x<955)) and (y > t+336 and y < t+8+336) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 elsif ((x > 159 and x < 637) or (x>954 and x<1114)) and (y > t+420 and y < t+8+420) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
 elsif (x > 0 and x<1281) and (y > t+504 and y < t+8+504) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";



elsif (x > 0 and x < 8) and (y > t+0 and y < t+512) then
    pixelon <= '1';
```

```
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 159 and x < 167) and (y > t+168 and y < t+336) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 318 and x < 326) and ((y > t+0 and y < t+260) or (y > t+336 and y < t+420)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 477 and x < 485) and (y > t+84 and y < t+344) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 636 and x < 644) and ((y > t+0 and y < t+84) or (y > t+336 and y < t+428)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 795 and x < 803) and (y > t+168 and y < t+504) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
elsif (x > 954 and x < 962) and ((y > t+84 and y < z+168) or (y > t+252 and y < t+344)) then
    pixelon <= '1';
    VGA_RED_TEMP <= "1111";
    VGA_Blue_TEMP <= "1111";
    VGA_Green_TEMP <= "1111";
```

```
elsif (x > 1113 and x < 1121) and ((y > t+0 and y < z+84) or (y > t+168 and y < t+428)) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif (x > 1272 and x < 1280) and (y > t+0 and y < t+345) then
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
elsif ((x > exc - 10 and x<exc + 10)) and (y > eyc - 10 and y < eyc + 10) then --draw elif
   objectpixelon <= '1';
   pixelon <= '1';
   VGA_RED_TEMP <= "1111";
   VGA_Blue_TEMP <= "1111";
   VGA_Green_TEMP <= "1111";
   else
      pixelon <= '0';
      objectpixelon <= '0';
   end if;
end if;
--if pixelon = '1' and objectpixelon = '0' and count < 3339 then --wall exists
--    wallpx(count) <= x; --store x pixel value of wall in myarray at position count
--    wallpy(count) <= y; --store y pixel value of wall in myarray at position count
--    count <= count + 1; --increment count so we move to next block of myarray
--end if;
--When "0110" =>-
--if ((x > 0 and x < 160) or (x>477 and x<796)) and (y > (z+84) and y < (z+8+84)) then
  --  pixelon <= '1';
   --VGA_RED_TEMP <= "1111";
   --VGA_Blue_TEMP <= "1111";
   --VGA_Green_TEMP <= "1111";
--end if;
```

```
--When "0111" =>

--if ((x > 477 and x < 637) or (x>795 and x<1114)) and (y > z+168 and y < z+8+168) then
   -- pixelon <= '1';
    --VGA_RED_TEMP <= "1111";
    --VGA_Blue_TEMP <= "1111";
    --VGA_Green_TEMP <= "1111";
--end if;
---When others => null;
    --pixelon <= '0';
    --objectpixelon <= '0';
--end case;



end process;

Process (Move,clk,x,y)
variable Wallfound: integer;
begin
Wallfound := 0;
t<= 512;




IF(rising_edge(clk)) and (x>1280) and (y>1024) then
count <= count +1;

        if input3 = '1' then
            Tyc <= "000000111111";
            Txc <= "000011111111";
            Eyc <= "001000111111";
            Exc <="000011111111";
```

```
        timwins <= '0';
        elifwins <= '0';
        tw <= '0';
        ew <= '0';
      end if;


if count = 12500 then
count <= 0;
Case Move IS
When "0000" => --A/a

if (Txc-10 = 7) and (Tyc > 0 and Tyc < 512)then --1st column
wallfound := 1;
elsif (Txc-10 = 166) and ((Tyc>75 and Tyc<101) or (Tyc > 158 and Tyc < 345)) then --2nd column
wallfound := 1;
elsif (Txc-10 = 325 and ((Tyc> 0 and Tyc< 269) or (Tyc>336 and Tyc< 420))) then --3rd column
wallfound := 1;
elsif (Txc-10 = 484 and (Tyc> 74 and Tyc< 353)) then --4th column
wallfound := 1;
elsif (Txc-10 = 643 and ((Tyc> 0 and Tyc< 86)or (Tyc>159 and Tyc<185) or(Tyc>327 and Tyc<436)))
then --5th column
wallfound := 1;
elsif (Txc-10 = 802) and ((Tyc>75 and Tyc<101)or (Tyc> 168 and Tyc< 504)) then --6th column
wallfound := 1;
elsif (Txc-10 = 961 and ((Tyc> 75 and Tyc< 168)or(Tyc>243 and Tyc<353))) then --7th column
wallfound := 1;
elsif (Txc-10 = 1120 and ((Tyc> 0 and Tyc< 93)or(Tyc>159 and Tyc<437))) then --8th column
wallfound := 1;

end if;

if wallfound = 0 then
```

```
Txc <= Txc -1;
end if;

When "0001" => --D

if (Txc+10 = 160) and ((Tyc>75 and Tyc< 101) or (Tyc > 157 and Tyc < 345) or (Tyc>411 and
Tyc<437))then --1st column
wallfound := 1;
elsif (Txc+10 = 319 and ((Tyc> 0 and Tyc< 269) or (Tyc>327 and Tyc< 420))) then --2nd column
wallfound := 1;
elsif (Txc+10 = 478 and (Tyc> 75 and Tyc< 353)) then
wallfound := 1;
elsif (Txc+10 = 637 and ((Tyc> 0 and Tyc< 94) or(Tyc>243 and Tyc<269) or(Tyc>327 and Tyc<436)))
then --4th column
wallfound := 1;
elsif (Txc+10 = 796 and (Tyc> 159 and Tyc< 513)) then --5th column
wallfound := 1;
elsif (Txc+10 = 955 and ((Tyc> 75 and Tyc< 168)or(Tyc>243 and Tyc<353) or (Tyc>411 and
Tyc<437))) then --6th column
wallfound := 1;
elsif (Txc+10 = 1114) and ((Tyc> 0 and Tyc< 93)or(Tyc>159 and Tyc<437)) then --7th column
wallfound := 1;
elsif (Txc+10 = 1278) and (Tyc> 0 and Tyc< 354) then --8th column
wallfound := 1;
end if;

if wallfound = 0 then
Txc <= Txc +1 ;
end if;

if (Txc+10 = 1273 and (Tyc>334 and Tyc< 515)) then
timwins <= '1';
tw <= '1';
end if;
```

```
When "0010" => --W/w


if (Tyc-10 = 8) and (Txc > 0 and Txc<1281) then
wallfound := 1;
elsif (Tyc-10 = 92 and ((Txc > 0 and Txc < 169) or (Txc>477 and Txc<812) or (Txc>1103 and
Txc<1131))) then
wallfound := 1;
elsif (Tyc-10 = 176 and ((Txc> 468 and Txc< 646) or (Txc>786 and Txc< 1123))) then
wallfound := 1;
elsif (Tyc-10 = 260 and ((Txc> 159 and Txc< 319) or (Txc>627 and Txc<796))) then
wallfound := 1;
elsif (Tyc-10 = 344 and ((Txc>150 and Txc<175) or (Txc> 309 and Txc< 478) or(Txc>795 and
Txc<956))) then
wallfound := 1;
elsif (Tyc-10 = 428 and ((Txc> 150 and Txc< 637)or(Txc>945 and Txc<1114))) then
wallfound := 1;
end if;

if wallfound = 0 then
Tyc <= Tyc -1;
end if;



When "0011" => --S/s
if (Tyc+10 = 85) and ((Txc > 0 and Txc < 179) or (Txc>477 and Txc<796)) then
wallfound := 1;
elsif (Tyc+10 = 168 and ((Txc>150 and Txc<176) or (Txc> 468 and Txc< 653) or (Txc>786 and Txc<
1130))) then
wallfound := 1;
elsif (Tyc+10 = 253 and ((Txc> 159 and Txc< 319) or (Txc>627 and Txc<796)or (Txc>945 and
Txc<971))) then
wallfound := 1;
```

```
elsif (Tyc+10 = 337) and ((Txc> 309 and Txc< 478)or (Txc>627 and Txc<653) or(Txc>795 and Txc<956)) then
wallfound := 1;
elsif (Tyc+10 = 421 and ((Txc> 150 and Txc< 637)or(Txc>945 and Txc<1114))) then
wallfound := 1;
elsif (Tyc+10 = 505 and (Txc> 0 and Txc< 1281)) then
wallfound := 1;
end if;
if wallfound = 0 then
Tyc <= Tyc + 1 ;
end if;

--Elif's movement
When "0101" => --L/l

if (Exc-10 = 7) and (Eyc > 0+t and Eyc < 512+t)then --1st column
wallfound := 1;
elsif (Exc-10 = 166) and ((Eyc>75+t and Eyc<101+t) or (Eyc > 158+t and Eyc < 345+t)) then --2nd column
wallfound := 1;
elsif (Exc-10 = 325 and ((Eyc> 0+t and Eyc< 269+t) or (Eyc>336+t and Eyc< 420+t))) then --3rd column
wallfound := 1;
elsif (Exc-10 = 484 and (Eyc> 74+t and Eyc< 353+t)) then --4th column
wallfound := 1;
elsif (Exc-10 = 643 and ((Eyc> 0+t and Eyc< 86+t)or (Eyc>159+t and Eyc<185+t) or(Eyc>327+t and Eyc<436+t))) then --5th column
wallfound := 1;
elsif (Exc-10 = 802) and ((Eyc>75+t and Eyc<101+t)or (Eyc> 168+t and Eyc< 504+t)) then --6th column
wallfound := 1;
elsif (Exc-10 = 961 and ((Eyc> 75+t and Eyc< 168+t)or(Eyc>243+t and Eyc<353+t))) then --7th column
wallfound := 1;
elsif (Exc-10 = 1120 and ((Eyc> 0+t and Eyc< 93+t)or(Eyc>159+t and Eyc<437+t))) then --8th column
wallfound := 1;
```

```
end if;

if wallfound = 0 then
Exc <= Exc -1;
end if;

When "0110" => --'

if (Exc+10 = 160) and ((Eyc>75+t and Eyc< 101+t) or (Eyc > 157+t and eyc < 345+t) or (Eyc>411+t and Eyc<437+t))then --1st column
wallfound := 1;
elsif (Exc+10 = 319 and ((Eyc> 0+t and Eyc< 269+t) or (Eyc>327+t and Eyc< 420+t))) then --2nd column
wallfound := 1;
elsif (Exc+10 = 478 and (Eyc> 75+t and Eyc< 353+t)) then
wallfound := 1;
elsif (Exc+10 = 637 and ((Eyc> 0+t and Eyc< 94+t) or(Eyc>243+t and Eyc<269+t) or(Eyc>327+t and Eyc<436+t))) then --4th column
wallfound := 1;
elsif (Exc+10 = 796 and (Eyc> 159+t and Eyc< 513+t)) then --5th column
wallfound := 1;
elsif (Exc+10 = 955 and ((Eyc> 75+t and Eyc< 168+t)or(Eyc>243+t and Eyc<353+t) or (Eyc>411+t and Eyc<437+t))) then --6th column
wallfound := 1;
elsif (Exc+10 = 1114) and ((Eyc> 0+t and Eyc< 93+t)or(Eyc>159+t and Eyc<437+t)) then --7th column
wallfound := 1;
elsif (Exc+10 = 1278) and (Eyc> 0+t and Eyc< 354+t) then --8th column
wallfound := 1;
end if;

if wallfound = 0 then
Exc <= Exc +1 ;
end if;
```

```
if (Exc+10 = 1273 and (Eyc>334+t and Eyc< 515+t)) then
elifwins <= '1';
ew <= '1';
end if;

When "0100" => --P/p

if (Eyc-10 = 8+t) and (Exc > 0 and Txc<1281) then
wallfound := 1;
elsif (Eyc-10 = 92+t and ((Exc > 0 and Exc < 169) or (Exc>477 and Exc<812) or (Exc>1103 and
Exc<1131))) then
wallfound := 1;
elsif (Eyc-10 = 176+t and ((Exc> 468 and Exc< 646) or (Exc>786 and Exc< 1123))) then
wallfound := 1;
elsif (Eyc-10 = 260+t and ((Exc> 159 and Exc< 335) or (Exc>627 and Exc<796))) then
wallfound := 1;
elsif (Eyc-10 = 343+t and ((Exc>150 and Exc<176) or (Exc> 309 and Exc< 478) or(Exc>795 and
Exc<956))) then
wallfound := 1;
elsif (Eyc-10 = 428+t and ((Exc> 150 and Exc< 637)or(Exc>945 and Exc<1130))) then
wallfound := 1;
end if;

if wallfound = 0 then
Eyc <= Eyc -1;
end if;



When "0111" => --;
if (Eyc+10 = 85) and ((Exc > 0 and Exc < 179) or (Exc>468 and Exc<806)) then
wallfound := 1;
elsif (Eyc+10 = 168+t and ((Exc>150 and Exc<176) or (Exc> 468 and Exc< 653) or (Exc>786 and Exc<
1130))) then
```

```
wallfound := 1;
elsif (Eyc+10 = 253+t) and ((Exc> 159 and Exc< 319) or (Exc>627 and Exc<796)or(Exc>945 and Exc<971)) then
wallfound := 1;
elsif (Eyc+10 = 337+t) and ((Exc> 309 and Exc< 478)or (Exc>627 and Exc<653) or(Exc>795 and Exc<956)) then
wallfound := 1;
elsif (Eyc+10 = 421+t and ((Exc> 150 and Exc< 637)or(Exc>945 and Exc<1114))) then
wallfound := 1;
elsif (Eyc+10 = 505+t and (Exc> 0 and Exc< 1281)) then
wallfound := 1;
end if;
if wallfound = 0 then
Eyc <= Eyc + 1 ;
end if;

WHEN OTHERS => null;

end case;
end if;
end if;

end process;



p <= pixelon;
VGA_RED_COMB <= (p & p & p & p) and VGA_RED_TEMP;
VGA_BLUE_COMB <= (p & p & p & p) and VGA_BLUE_TEMP;
VGA_GREEN_COMB <= (p & p & p & p) and VGA_GREEN_TEMP;

-- VGA OUTPUT
process (clk)
```

```
   begin
      if (rising_edge(clk)) then
         VGA_RED <= VGA_RED_COMB;
         VGA_BLUE <= VGA_BLUE_COMB;
         VGA_GREEN <= VGA_GREEN_COMB;


       end if;
end process;
end Behavioral;
```

Keyboard Module:

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



entity KeyboardInput is
GENERIC(
    clk_freq                : INTEGER := 100_000_000; --system clock frequency in Hz
    ps2_debounce_counter_size : INTEGER := 9);         --set such that 2^size/clk_freq = 5us (size = 8 for
50MHz)
   Port (
         clk : in STD_LOGIC;  -- System Clock
         ps2_clk : in STD_LOGIC;  --Clock Signal from ps2 keyboard
         ps2_data : in STD_LOGIC;  --Data signal from ps2 keyboard
        ascii_new_out : out STD_LOGIC;
         Action : out STD_LOGIC_VECTOR (3 downto 0)); --Action to be taken when a certain key is
pressed.
end KeyboardInput;


architecture Behavioral of KeyboardInput is
COMPONENT ps2_keyboard_to_ascii IS
   GENERIC(
    clk_freq             : INTEGER;  --system clock frequency in Hz
```

```
    debounce_counter_size : INTEGER); --set such that 2^size/clk_freq = 5us (size = 8 for 50MHz)
  PORT(
    clk         : IN  STD_LOGIC;                    --system clock
    ps2_clk     : IN  STD_LOGIC;                    --clock signal from PS2 keyboard
    ps2_data    : IN  STD_LOGIC;                    --data signal from PS2 keyboard
    ascii_new   : OUT STD_LOGIC;                    --output flag indicating new ASCII value
    ascii_code : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)); --ASCII value
 END COMPONENT;
 signal ascii_new : STD_LOGIC;
 signal ascii_code : STD_LOGIC_VECTOR(6 DOWNTO 0);
begin
  ps2_keyboard_0:  ps2_keyboard_to_ascii
  GENERIC MAP(clk_freq => clk_freq, debounce_counter_size => ps2_debounce_counter_size)
  PORT MAP(clk => clk, ps2_clk => ps2_clk, ps2_data => ps2_data, ascii_new => ascii_new,
ascii_code => ascii_code);

  Process(clk)
  BEGIN
  if (rising_edge(clk)) then
    If( ascii_new = '1') Then
      CASE ascii_code IS
        WHEN (x"41") => --A
        Action <= "0000" ; -- tim moves left
        WHEN (x"61") => --a
        Action <= "0000" ; -- tim moves left
        WHEN x"44" => -- D
        Action <= "0001"; -- tim moves right
        WHEN x"64" => -- d
        Action <= "0001"; -- tim moves right
        WHEN x"53" => --S
        Action <= "0011"; --tim moves down
        WHEN x"73" => --s
        Action <= "0011"; --tim moves down
```

```
            WHEN x"57" => --W
            Action <= "0010"; --tim moves up
            WHEN x"77" => --w
            Action <= "0010"; --tim moves up
            WHEN x"50" => --P
            Action <= "0100"; -- elif moves up
            WHEN x"70" => --p
            Action <= "0100"; -- elif moves up
            WHEN x"4C" => --L
            Action <= "0101"; --elif moves left
            WHEN x"6C" => --l
            Action <= "0101"; --elif moves left
            WHEN x"22" => --"
            Action <= "0110"; --elif moves right
            WHEN x"27" =>--'
            Action <= "0110"; --elif moves right
            WHEN x"3B" => --;
            Action <= "0111"; --elif moves down
            WHEN x"3A" => --:
            Action <= "0111"; --elif moves down
            WHEN OTHERS => null;
          END CASE;
        ELSE
          Action <= "1111";

      END IF;
    END IF;
    END PROCESS;
    ascii_new_out <= ascii_new;

end Behavioral;
```

Ps2 to Keyboard Ascii Module:

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;


ENTITY ps2_keyboard_to_ascii IS
  GENERIC(
    clk_freq                  : INTEGER := 100_000_000; --system clock frequency in Hz
    ps2_debounce_counter_size : INTEGER := 9);         --set such that 2^size/clk_freq = 5us (size = 8 for 50MHz)
  PORT(
    clk        : IN  STD_LOGIC;                    --system clock input
    ps2_clk    : IN  STD_LOGIC;                    --clock signal from PS2 keyboard
    ps2_data   : IN  STD_LOGIC;                    --data signal from PS2 keyboard
    ascii_new  : OUT STD_LOGIC;                    --output flag indicating new ASCII value
    ascii_code : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)); --ASCII value
END ps2_keyboard_to_ascii;


ARCHITECTURE behavior OF ps2_keyboard_to_ascii IS


TYPE machine IS(ready, new_code, translate, output);              --needed states
  SIGNAL state             : machine;                            --state machine
  SIGNAL ps2_code_new      : STD_LOGIC;                          --new PS2 code flag from ps2_keyboard component
  SIGNAL ps2_code          : STD_LOGIC_VECTOR(7 DOWNTO 0);       --PS2 code input form ps2_keyboard component
  SIGNAL prev_ps2_code_new : STD_LOGIC := '1';                   --value of ps2_code_new flag on previous clock
  SIGNAL break             : STD_LOGIC := '0';                   --'1' for break code, '0' for make code
  SIGNAL e0_code           : STD_LOGIC := '0';                   --'1' for multi-code commands, '0' for single code commands
  SIGNAL caps_lock         : STD_LOGIC := '0';                   --'1' if caps lock is active, '0' if caps lock is inactive
  SIGNAL control_r         : STD_LOGIC := '0';                   --'1' if right control key is held down, else '0'
  SIGNAL control_l         : STD_LOGIC := '0';                   --'1' if left control key is held down, else '0'
  SIGNAL shift_r           : STD_LOGIC := '0';                   --'1' if right shift is held down, else '0'
```

```
  SIGNAL shift_l          : STD_LOGIC := '0';                   --'1' if left shift is held down, else '0'
  SIGNAL ascii            : STD_LOGIC_VECTOR(7 DOWNTO 0) := x"FF"; --internal value of ASCII translation

  --declare PS2 keyboard interface component
  COMPONENT ps2_keyboard IS
    GENERIC(
      clk_freq              : INTEGER;  --system clock frequency in Hz
      debounce_counter_size : INTEGER); --set such that 2^size/clk_freq = 5us (size = 8 for 50MHz)
    PORT(
      clk          : IN  STD_LOGIC;                   --system clock
      ps2_clk      : IN  STD_LOGIC;                   --clock signal from PS2 keyboard
      ps2_data     : IN  STD_LOGIC;                   --data signal from PS2 keyboard
      ps2_code_new : OUT STD_LOGIC;                   --flag that new PS/2 code is available on ps2_code bus
      ps2_code     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --code received from PS/2
  END COMPONENT;

BEGIN

  --instantiate PS2 keyboard interface logic
  ps2_keyboard_0:  ps2_keyboard
    GENERIC MAP(clk_freq => clk_freq, debounce_counter_size => ps2_debounce_counter_size)
    PORT MAP(clk => clk, ps2_clk => ps2_clk, ps2_data => ps2_data, ps2_code_new => ps2_code_new, ps2_code => ps2_code);

PROCESS(clk)

BEGIN

IF(clk'EVENT AND clk = '1') THEN

prev_ps2_code_new <= ps2_code_new; --keep track of previous ps2_code_new values to determine low-to-high transitions
```

```
CASE state IS

--ready state: wait for a new PS2 code to be received

WHEN ready =>

IF(prev_ps2_code_new = '0' AND ps2_code_new = '1') THEN --new PS2 code received

ascii_new <= '0'; --reset new ASCII code indicator

state <= new_code; --proceed to new_code state

ELSE --no new PS2 code received yet

state <= ready; --remain in ready state

END IF;

--new_code state: determine what to do with the new PS2 code

WHEN new_code =>

IF(ps2_code = x"F0") THEN --code indicates that next command is break

break <= '1'; --set break flag

state <= ready; --return to ready state to await next PS2 code

ELSIF(ps2_code = x"E0") THEN --code indicates multi-key command

e0_code <= '1'; --set multi-code command flag
```

```
state <= ready; --return to ready state to await next PS2 code

ELSE --code is the last PS2 code in the make/break code

ascii(7) <= '1'; --set internal ascii value to unsupported code (for verification)

state <= translate; --proceed to translate state

END IF;


--translate state: translate PS2 code to ASCII value

WHEN translate =>

break <= '0'; --reset break flag

e0_code <= '0'; --reset multi-code command flag

--handle codes for control, shift, and caps lock

CASE ps2_code IS

WHEN x"58" => --caps lock code

IF(break = '0') THEN --if make command

caps_lock <= NOT caps_lock; --toggle caps lock

END IF;

WHEN x"14" => --code for the control keys
```

```
IF(e0_code = '1') THEN --code for right control

control_r <= NOT break; --update right control flag

ELSE --code for left control

control_l <= NOT break; --update left control flag

END IF;

WHEN x"12" => --left shift code

shift_l <= NOT break; --update left shift flag

WHEN x"59" => --right shift code

shift_r <= NOT break; --update right shift flag

WHEN OTHERS => NULL;

END CASE;

--translate control codes (these do not depend on shift or caps lock)

IF(control_l = '1' OR control_r = '1') THEN

CASE ps2_code IS

WHEN x"1E" => ascii <= x"00"; --^@ NUL

WHEN x"1C" => ascii <= x"01"; --^A SOH

WHEN x"32" => ascii <= x"02"; --^B STX
```

```
WHEN x"21" => ascii <= x"03"; --^C ETX

WHEN x"23" => ascii <= x"04"; --^D EOT

WHEN x"24" => ascii <= x"05"; --^E ENQ

WHEN x"2B" => ascii <= x"06"; --^F ACK

WHEN x"34" => ascii <= x"07"; --^G BEL

WHEN x"33" => ascii <= x"08"; --^H BS

WHEN x"43" => ascii <= x"09"; --^I HT

WHEN x"3B" => ascii <= x"0A"; --^J LF

WHEN x"42" => ascii <= x"0B"; --^K VT

WHEN x"4B" => ascii <= x"0C"; --^L FF

WHEN x"3A" => ascii <= x"0D"; --^M CR

WHEN x"31" => ascii <= x"0E"; --^N SO

WHEN x"44" => ascii <= x"0F"; --^O SI

WHEN x"4D" => ascii <= x"10"; --^P DLE

WHEN x"15" => ascii <= x"11"; --^Q DC1

WHEN x"2D" => ascii <= x"12"; --^R DC2
```

```
WHEN x"1B" => ascii <= x"13"; --^S DC3

WHEN x"2C" => ascii <= x"14"; --^T DC4

WHEN x"3C" => ascii <= x"15"; --^U NAK

WHEN x"2A" => ascii <= x"16"; --^V SYN

WHEN x"1D" => ascii <= x"17"; --^W ETB

WHEN x"22" => ascii <= x"18"; --^X CAN

WHEN x"35" => ascii <= x"19"; --^Y EM

WHEN x"1A" => ascii <= x"1A"; --^Z SUB

WHEN x"54" => ascii <= x"1B"; --^[ ESC

WHEN x"5D" => ascii <= x"1C"; --^\ FS

WHEN x"5B" => ascii <= x"1D"; --^] GS

WHEN x"36" => ascii <= x"1E"; --^^ RS

WHEN x"4E" => ascii <= x"1F"; --^_ US

WHEN x"4A" => ascii <= x"7F"; --^? DEL

WHEN OTHERS => NULL;

END CASE;

ELSE --if control keys are not pressed
```

```
--translate characters that do not depend on shift, or caps lock

CASE ps2_code IS

WHEN x"29" => ascii <= x"20"; --space

WHEN x"66" => ascii <= x"08"; --backspace (BS control code)

WHEN x"0D" => ascii <= x"09"; --tab (HT control code)

WHEN x"5A" => ascii <= x"0D"; --enter (CR control code)

WHEN x"76" => ascii <= x"1B"; --escape (ESC control code)

WHEN x"71" =>

IF(e0_code = '1') THEN --ps2 code for delete is a multi-key code

ascii <= x"7F"; --delete/

END IF;

WHEN OTHERS => NULL;

END CASE;

--translate letters (these depend on both shift and caps lock)

IF((shift_r = '0' AND shift_l = '0' AND caps_lock = '0') OR

((shift_r = '1' OR shift_l = '1') AND caps_lock = '1')) THEN --letter is lowercase
```

```
CASE ps2_code IS

WHEN x"1C" => ascii <= x"61"; --a

WHEN x"32" => ascii <= x"62"; --b

WHEN x"21" => ascii <= x"63"; --c

WHEN x"23" => ascii <= x"64"; --d

WHEN x"24" => ascii <= x"65"; --e

WHEN x"2B" => ascii <= x"66"; --f

WHEN x"34" => ascii <= x"67"; --g

WHEN x"33" => ascii <= x"68"; --h

WHEN x"43" => ascii <= x"69"; --i

WHEN x"3B" => ascii <= x"6A"; --j

WHEN x"42" => ascii <= x"6B"; --k

WHEN x"4B" => ascii <= x"6C"; --l

WHEN x"3A" => ascii <= x"6D"; --m

WHEN x"31" => ascii <= x"6E"; --n

WHEN x"44" => ascii <= x"6F"; --o

WHEN x"4D" => ascii <= x"70"; --p
```

```
WHEN x"15" => ascii <= x"71"; --q

WHEN x"2D" => ascii <= x"72"; --r

WHEN x"1B" => ascii <= x"73"; --s

WHEN x"2C" => ascii <= x"74"; --t

WHEN x"3C" => ascii <= x"75"; --u

WHEN x"2A" => ascii <= x"76"; --v

WHEN x"1D" => ascii <= x"77"; --w

WHEN x"22" => ascii <= x"78"; --x

WHEN x"35" => ascii <= x"79"; --y

WHEN x"1A" => ascii <= x"7A"; --z

WHEN OTHERS => NULL;

END CASE;

ELSE --letter is uppercase

CASE ps2_code IS

WHEN x"1C" => ascii <= x"41"; --A

WHEN x"32" => ascii <= x"42"; --B
```

```
WHEN x"21" => ascii <= x"43"; --C

WHEN x"23" => ascii <= x"44"; --D

WHEN x"24" => ascii <= x"45"; --E

WHEN x"2B" => ascii <= x"46"; --F

WHEN x"34" => ascii <= x"47"; --G

WHEN x"33" => ascii <= x"48"; --H

WHEN x"43" => ascii <= x"49"; --I

WHEN x"3B" => ascii <= x"4A"; --J

WHEN x"42" => ascii <= x"4B"; --K

WHEN x"4B" => ascii <= x"4C"; --L

WHEN x"3A" => ascii <= x"4D"; --M

WHEN x"31" => ascii <= x"4E"; --N

WHEN x"44" => ascii <= x"4F"; --O

WHEN x"4D" => ascii <= x"50"; --P

WHEN x"15" => ascii <= x"51"; --Q

WHEN x"2D" => ascii <= x"52"; --R

WHEN x"1B" => ascii <= x"53"; --S
```

```
WHEN x"2C" => ascii <= x"54"; --T

WHEN x"3C" => ascii <= x"55"; --U

WHEN x"2A" => ascii <= x"56"; --V

WHEN x"1D" => ascii <= x"57"; --W

WHEN x"22" => ascii <= x"58"; --X

WHEN x"35" => ascii <= x"59"; --Y

WHEN x"1A" => ascii <= x"5A"; --Z

WHEN OTHERS => NULL;

END CASE;

END IF;

--translate numbers and symbols (these depend on shift but not caps lock)

IF(shift_l = '1' OR shift_r = '1') THEN --key's secondary character is desired

CASE ps2_code IS

WHEN x"16" => ascii <= x"21"; --!

WHEN x"52" => ascii <= x"22"; --"

WHEN x"26" => ascii <= x"23"; --#
```

```
WHEN x"25" => ascii <= x"24"; --$

WHEN x"2E" => ascii <= x"25"; --%

WHEN x"3D" => ascii <= x"26"; --&

WHEN x"46" => ascii <= x"28"; --(

WHEN x"45" => ascii <= x"29"; --)

WHEN x"3E" => ascii <= x"2A"; --*

WHEN x"55" => ascii <= x"2B"; --+

WHEN x"4C" => ascii <= x"3A"; --:

WHEN x"41" => ascii <= x"3C"; --<

WHEN x"49" => ascii <= x"3E"; -->

WHEN x"4A" => ascii <= x"3F"; --?

WHEN x"1E" => ascii <= x"40"; --@

WHEN x"36" => ascii <= x"5E"; --^

WHEN x"4E" => ascii <= x"5F"; --_

WHEN x"54" => ascii <= x"7B"; --{

WHEN x"5D" => ascii <= x"7C"; --|

WHEN x"5B" => ascii <= x"7D"; --}
```

```
WHEN x"0E" => ascii <= x"7E"; --~

WHEN OTHERS => NULL;

END CASE;

ELSE --key's primary character is desired

CASE ps2_code IS

WHEN x"45" => ascii <= x"30"; --0

WHEN x"16" => ascii <= x"31"; --1

WHEN x"1E" => ascii <= x"32"; --2

WHEN x"26" => ascii <= x"33"; --3

WHEN x"25" => ascii <= x"34"; --4

WHEN x"2E" => ascii <= x"35"; --5

WHEN x"36" => ascii <= x"36"; --6

WHEN x"3D" => ascii <= x"37"; --7

WHEN x"3E" => ascii <= x"38"; --8

WHEN x"46" => ascii <= x"39"; --9

WHEN x"52" => ascii <= x"27"; --'
```

```
WHEN x"41" => ascii <= x"2C"; --,

WHEN x"4E" => ascii <= x"2D"; ---

WHEN x"49" => ascii <= x"2E"; --.

WHEN x"4A" => ascii <= x"2F"; --/

WHEN x"4C" => ascii <= x"3B"; --;

WHEN x"55" => ascii <= x"3D"; --=

WHEN x"54" => ascii <= x"5B"; --[

WHEN x"5D" => ascii <= x"5C"; --\

WHEN x"5B" => ascii <= x"5D"; --]

WHEN x"0E" => ascii <= x"60"; --`

WHEN OTHERS => NULL;

END CASE;

END IF;

END IF;

IF(break = '0') THEN --the code is a make

state <= output; --proceed to output state

ELSE --code is a break
```

```
state <= ready; --return to ready state to await next PS2 code

END IF;

--output state: verify the code is valid and output the ASCII value

WHEN output =>

IF(ascii(7) = '0') THEN --the PS2 code has an ASCII output

ascii_new <= '1'; --set flag indicating new ASCII output

ascii_code <= ascii(6 DOWNTO 0); --output the ASCII value

END IF;

state <= ready; --return to ready state to await next PS2 code


END CASE;

END IF;

END PROCESS;


END behavioral;
```

Ps2 Keyboard:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ps2_keyboard IS
```

```
 GENERIC(
   clk_freq              : INTEGER := 50_000_000; --system clock frequency in Hz
   debounce_counter_size : INTEGER := 8);         --set such that (2^size)/clk_freq = 5us (size = 8 for 50MHz)
 PORT(
   clk         : IN  STD_LOGIC;                   --system clock
   ps2_clk     : IN  STD_LOGIC;                   --clock signal from PS/2 keyboard
   ps2_data    : IN  STD_LOGIC;                   --data signal from PS/2 keyboard
   ps2_code_new : OUT STD_LOGIC;                  --flag that new PS/2 code is available on ps2_code bus
   ps2_code    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --code received from PS/2
END ps2_keyboard;

ARCHITECTURE logic OF ps2_keyboard IS
 SIGNAL sync_ffs     : STD_LOGIC_VECTOR(1 DOWNTO 0);       --synchronizer flip-flops for PS/2 signals
 SIGNAL ps2_clk_int  : STD_LOGIC;                          --debounced clock signal from PS/2 keyboard
 SIGNAL ps2_data_int : STD_LOGIC;                          --debounced data signal from PS/2 keyboard
 SIGNAL ps2_word     : STD_LOGIC_VECTOR(10 DOWNTO 0);      --stores the ps2 data word
 SIGNAL error        : STD_LOGIC;                          --validate parity, start, and stop bits
 SIGNAL count_idle   : INTEGER RANGE 0 TO clk_freq/18_000; --counter to determine PS/2 is idle

 --declare debounce component for debouncing PS2 input signals
 COMPONENT debounce IS
  GENERIC(
    counter_size : INTEGER); --debounce period (in seconds) = 2^counter_size/(clk freq in Hz)
  PORT(
    clk    : IN  STD_LOGIC;  --input clock
    button : IN  STD_LOGIC;  --input signal to be debounced
    result : OUT STD_LOGIC); --debounced signal
 END COMPONENT;
BEGIN
 --synchronizer flip-flops
 PROCESS(clk)
 BEGIN
```

```
  IF(clk'EVENT AND clk = '1') THEN  --rising edge of system clock
    sync_ffs(0) <= ps2_clk;          --synchronize PS/2 clock signal
    sync_ffs(1) <= ps2_data;         --synchronize PS/2 data signal
  END IF;
END PROCESS;
--debounce PS2 input signals
debounce_ps2_clk: debounce
  GENERIC MAP(counter_size => debounce_counter_size)
  PORT MAP(clk => clk, button => sync_ffs(0), result => ps2_clk_int);
debounce_ps2_data: debounce
  GENERIC MAP(counter_size => debounce_counter_size)
  PORT MAP(clk => clk, button => sync_ffs(1), result => ps2_data_int);
--input PS2 data
PROCESS(ps2_clk_int)
BEGIN
  IF(ps2_clk_int'EVENT AND ps2_clk_int = '0') THEN    --falling edge of PS2 clock
    ps2_word <= ps2_data_int & ps2_word(10 DOWNTO 1);   --shift in PS2 data bit
  END IF;
END PROCESS;

--verify that parity, start, and stop bits are all correct
error <= NOT (NOT ps2_word(0) AND ps2_word(10) AND (ps2_word(9) XOR ps2_word(8) XOR
     ps2_word(7) XOR ps2_word(6) XOR ps2_word(5) XOR ps2_word(4) XOR ps2_word(3) XOR
     ps2_word(2) XOR ps2_word(1)));
--determine if PS2 port is idle (i.e. last transaction is finished) and output result
PROCESS(clk)
BEGIN
  IF(clk'EVENT AND clk = '1') THEN          --rising edge of system clock

    IF(ps2_clk_int = '0') THEN               --low PS2 clock, PS/2 is active
      count_idle <= 0;                     --reset idle counter
    ELSIF(count_idle /= clk_freq/18_000) THEN  --PS2 clock has been high less than a half clock period
(<55us)
```

```
        count_idle <= count_idle + 1;           --continue counting
      END IF;

      IF(count_idle = clk_freq/18_000 AND error = '0') THEN  --idle threshold reached and no errors
detected
        ps2_code_new <= '1';                          --set flag that new PS/2 code is available
        ps2_code <= ps2_word(8 DOWNTO 1);                   --output new PS/2 code
      ELSE                                      --PS/2 port active or error detected
        ps2_code_new <= '0';                          --set flag that PS/2 transaction is in progress
      END IF;

    END IF;
  END PROCESS;

END logic;
```

Debounce Ps2 Clock:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY debounce IS
  GENERIC(
    counter_size  :  INTEGER := 19); --counter size (19 bits gives 10.5ms with 50MHz clock)
  PORT(
    clk     : IN  STD_LOGIC;  --input clock
    button  : IN  STD_LOGIC;  --input signal to be debounced
    result  : OUT STD_LOGIC); --debounced signal
END debounce;
ARCHITECTURE logic OF debounce IS
  SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
  SIGNAL counter_set : STD_LOGIC;                   --sync reset to zero
  SIGNAL counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (OTHERS => '0'); --
counter output
BEGIN
```

```
  counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/reset counter

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
      flipflops(0) <= button;
      flipflops(1) <= flipflops(0);
      If(counter_set = '1') THEN                --reset counter because input is changing
        counter_out <= (OTHERS => '0');
      ELSIF(counter_out(counter_size) = '0') THEN --stable input time is not yet met
        counter_out <= counter_out + 1;
      ELSE                                      --stable input time is met
        result <= flipflops(1);
      END IF;
    END IF;
  END PROCESS;
END logic;
```

Debounce Ps2 Data:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY debounce IS
  GENERIC(
    counter_size  :  INTEGER := 19); --counter size (19 bits gives 10.5ms with 50MHz clock)
  PORT(
    clk     : IN  STD_LOGIC;  --input clock
    button  : IN  STD_LOGIC;  --input signal to be debounced
    result  : OUT STD_LOGIC); --debounced signal
END debounce;
ARCHITECTURE logic OF debounce IS
  SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
  SIGNAL counter_set : STD_LOGIC;                    --sync reset to zero
```

```
  SIGNAL counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (OTHERS => '0'); --
counter output
BEGIN
  counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/reset counter

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
      flipflops(0) <= button;
      flipflops(1) <= flipflops(0);
      If(counter_set = '1') THEN                --reset counter because input is changing
        counter_out <= (OTHERS => '0');
      ELSIF(counter_out(counter_size) = '0') THEN --stable input time is not yet met
        counter_out <= counter_out + 1;
      ELSE                                      --stable input time is met
        result <= flipflops(1);
      END IF;
    END IF;
  END PROCESS;
END logic;
```

Counter Module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use Ieee.numeric_std.all ;


entity counter is
```

```
    Port (
        input1 : in std_logic;
        input2:in std_logic;
        input3:in std_logic;
        tw: in std_logic;
        ew: in std_logic;
        clk : in STD_LOGIC ;
        topDispA : out std_logic ;
        topDispB : out std_logic ;
        topDispC : out std_logic ;
        topDispD : out std_logic ;
        topSegA : out std_logic ;
        topSegB : out std_logic ;
        topSegC : out std_logic ;
        topSegD : out std_logic ;
        topSegE : out std_logic ;
        topSegF : out std_logic ;
        timeup: out std_logic;
        topSegG : out std_logic  );

end counter;



architecture Behavioral of counter is

component SegmentDriver is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
    B: in STD_LOGIC_VECTOR (3 downto 0);
    C: in STD_LOGIC_VECTOR (3 downto 0);
    D: in STD_LOGIC_VECTOR (3 downto 0);
        seg_a : out STD_LOGIC;
```

```
        seg_b : out STD_LOGIC;
        seg_c : out STD_LOGIC;
        seg_d : out STD_LOGIC;
        seg_e : out STD_LOGIC;
        seg_f : out STD_LOGIC;
        seg_g : out STD_LOGIC;
        display_A : out STD_LOGIC;
        display_B : out STD_LOGIC;
        display_c : out STD_LOGIC;
        display_D : out STD_LOGIC;
        clk : in STD_LOGIC);
end component;

component timer is
    Port ( Clock : in STD_LOGIC; -- 100 MHz
        seconds1, seconds2 :  out std_logic_vector(3 downto 0);
        Switchon, Switch2on,Switch3on, Timwins, Elifwins: in STD_LOGIC;
-- Switch4on
        tu: out std_logic;
        clock_start: in STD_LOGIC
        );
end component;



signal Ai,Aj,Ak : std_logic_vector(3 downto 0) ;
signal Bi,Bj,Bk : std_logic_vector(3 downto 0) ;
signal Ci,Cj,Ck : std_logic_vector(3 downto 0) ;
signal Di,Dj,Dk : std_logic_vector(3 downto 0) ;

signal sensor_meters,sensor_meters2,sm3 : std_logic_vector(3 downto 0) ;
signal sensor_decimeters,sensor_decimeters2,sd3  : std_logic_vector(3 downto 0) ;
signal sensor_centimeters,sensor_centimeters2 , sc3: std_logic_vector(3 downto 0);
signal stopsignal: std_logic := '0';
```

```
begin

uut : Segmentdriver port map(
      A => Ai,
      B => Bi,
      C => Ci,
      D => Di ,
       seg_a => topsegA,
       seg_b => topsegB ,
       seg_c => topsegC,
       seg_d => topsegD,
       seg_e => topsegE,
       seg_f => topsegF ,
       seg_g => topsegG ,
      display_A => topdispA ,
      display_B => topdispB,
      display_c => topdispC,
      display_D => topDispD,
      clk => clk
      ) ;


      uut3 : timer port map(

          Clock => clk ,
           seconds1 => Ai ,
           seconds2 => Bi ,
--           timcount => Ci,
--           elifcount => Di,
           Switchon => input1,
           Switch2on => input2,
           Switch3on => input3,
           Timwins => tw,
```

```
        Elifwins => ew,
        tu => stopsignal,
        clock_start => input1
    ) ;
timeup <= stopsignal;

end Behavioral;
```

Segment Driver:

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

entity SegmentDriver is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
        B : in STD_LOGIC_VECTOR (3 downto 0);
        C : in STD_LOGIC_VECTOR (3 downto 0);
        D : in STD_LOGIC_VECTOR (3 downto 0);
        seg_a : out STD_LOGIC;
        seg_b : out STD_LOGIC;
        seg_c : out STD_LOGIC;
        seg_d : out STD_LOGIC;
        seg_e : out STD_LOGIC;
        seg_f : out STD_LOGIC;
        seg_g : out STD_LOGIC;
        display_A : out STD_LOGIC;
        display_B : out STD_LOGIC;
        display_c : out STD_LOGIC;
        display_D : out STD_LOGIC;
        clk : in STD_LOGIC);
```

```
end SegmentDriver;

 architecture Behavioral of SegmentDriver is

component clkdivider
    Port ( clk : in STD_LOGIC;
           enable : in STD_LOGIC;
           reset : in STD_LOGIC;
           data_clk : out std_logic_vector (15 downto 0));
end component;

component SegmentDecoder
    Port ( Digit : in STD_LOGIC_VECTOR (3 downto 0);
           segment_a : out STD_LOGIC;
           segment_b : out STD_LOGIC;
           segment_c : out STD_LOGIC;
           segment_d : out STD_LOGIC;
           segment_e : out STD_LOGIC;
           segment_f : out STD_LOGIC;
           segment_g : out STD_LOGIC );
end component;

signal temp : std_logic_vector(3 downto 0) ;
signal clock_word : std_logic_vector(15 downto 0) ;
signal slowclk : std_logic ;

begin
 u2 : clkdivider port map(
       clk => clk ,
      enable => '1' ,
       reset => '0' ,
       data_clk => clock_word ) ;
```

```
u1 : SegmentDecoder port map(
      Digit  => temp ,
       segment_a => seg_a ,
        segment_b => seg_b ,
        segment_c => seg_c ,
         segment_d => seg_d,
         segment_e => seg_e ,
         segment_f => seg_f ,
         segment_g => seg_g );

slowclk <= clock_word(15) ;

process (slowclk)

variable display_selection : std_logic_vector(1 downto 0 ) ;

begin

if slowclk'event and slowclk = '1'  then

case display_selection is

when "00" => temp <= A ;

      Display_A <= '0' ;
      Display_B <= '1' ;
      Display_C <= '1' ;
      Display_D <= '1' ;
display_selection := display_selection + '1' ;
when "01" => temp <= B ;

      Display_A <= '1' ;
      Display_B <= '0' ;
```

```
      Display_C <= '1' ;
      Display_D <= '1' ;
display_selection := display_selection + '1' ;
when "11" => temp <= C ;

      Display_A <= '1' ;
      Display_B <= '1' ;
      Display_C <= '0' ;
    Display_D <= '1' ;
display_selection := display_selection + '1' ;
when others => temp <= D ;

    Display_A <= '1' ;
      Display_B <= '1' ;
     Display_C <= '1' ;
      Display_D <= '0' ;
display_selection := display_selection + '1' ;

end case ;
end if;
end process ;

end Behavioral;
```

Clock Divider:

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

entity SegmentDriver is
```

```
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC_VECTOR (3 downto 0);
           D : in STD_LOGIC_VECTOR (3 downto 0);
           seg_a : out STD_LOGIC;
           seg_b : out STD_LOGIC;
           seg_c : out STD_LOGIC;
           seg_d : out STD_LOGIC;
           seg_e : out STD_LOGIC;
           seg_f : out STD_LOGIC;
           seg_g : out STD_LOGIC;
           display_A : out STD_LOGIC;
           display_B : out STD_LOGIC;
           display_c : out STD_LOGIC;
           display_D : out STD_LOGIC;
           clk : in STD_LOGIC);
end SegmentDriver;

 architecture Behavioral of SegmentDriver is

component clkdivider
    Port ( clk : in STD_LOGIC;
           enable : in STD_LOGIC;
           reset : in STD_LOGIC;
           data_clk : out std_logic_vector (15 downto 0));
end component;

component SegmentDecoder
    Port ( Digit : in STD_LOGIC_VECTOR (3 downto 0);
           segment_a : out STD_LOGIC;
           segment_b : out STD_LOGIC;
           segment_c : out STD_LOGIC;
           segment_d : out STD_LOGIC;
```

```
        segment_e : out STD_LOGIC;
        segment_f : out STD_LOGIC;
        segment_g : out STD_LOGIC );
end component;

signal temp : std_logic_vector(3 downto 0) ;
signal clock_word : std_logic_vector(15 downto 0) ;
signal slowclk : std_logic ;

begin
 u2 : clkdivider port map(
       clk => clk ,
      enable => '1' ,
       reset => '0' ,
       data_clk => clock_word ) ;

u1 : SegmentDecoder port map(
      Digit  => temp ,
       segment_a => seg_a ,
        segment_b => seg_b ,
        segment_c => seg_c ,
         segment_d => seg_d,
         segment_e => seg_e ,
         segment_f => seg_f ,
         segment_g => seg_g );

slowclk <= clock_word(15) ;

process (slowclk)

variable display_selection : std_logic_vector(1 downto 0 ) ;

begin
```

```
if slowclk'event and slowclk = '1'  then

case display_selection is

when "00" => temp <= A ;

     Display_A <= '0' ;
     Display_B <= '1' ;
     Display_C <= '1' ;
     Display_D <= '1' ;
display_selection := display_selection + '1' ;
when "01" => temp <= B ;

     Display_A <= '1' ;
     Display_B <= '0' ;
     Display_C <= '1' ;
     Display_D <= '1' ;
display_selection := display_selection + '1' ;
when "11" => temp <= C ;

     Display_A <= '1' ;
     Display_B <= '1' ;
     Display_C <= '0' ;
   Display_D <= '1' ;
display_selection := display_selection + '1' ;
when others => temp <= D ;

   Display_A <= '1' ;
     Display_B <= '1' ;
    Display_C <= '1' ;
     Display_D <= '0' ;
display_selection := display_selection + '1' ;
```

```
end case ;
end if;
end process ;

end Behavioral;
```

Segment Decoder:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 04/12/2018 02:23:48 AM
-- Design Name:
-- Module Name: SegmentDecoder - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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

```
entity SegmentDecoder is
    Port ( Digit : in STD_LOGIC_VECTOR (3 downto 0);
           segment_a : out STD_LOGIC;
           segment_b : out STD_LOGIC;
           segment_c : out STD_LOGIC;
           segment_d : out STD_LOGIC;
           segment_e : out STD_LOGIC;
           segment_f : out STD_LOGIC;
           segment_g : out STD_LOGIC );
end SegmentDecoder;

architecture Behavioral of SegmentDecoder is

begin
process(Digit)
variable decodedata : std_logic_vector(6 downto 0) ;
begin

case Digit is
        when "0000" => decodedata := "1111110" ;
       when "0001" => decodedata := "0110000" ;
       when "0010" => decodedata := "1101101" ;
       when "0011" => decodedata := "1111001" ;
       when "0100" => decodedata := "0110011" ;
       when "0101" => decodedata := "1011011" ;
       when "0110" => decodedata := "1011111" ;
       when "0111" => decodedata := "1110000" ;
       when "1000" => decodedata := "1111111" ;
       when "1001" => decodedata := "1111011" ;
       when others => decodedata := "1000111" ;
        end case ;

        segment_a <= NOT decodedata(6) ;
```

```
        segment_b <= NOT decodedata(5) ;
        segment_c <= NOT decodedata(4) ;
        segment_d <= NOT decodedata(3) ;
        segment_e <= NOT decodedata(2) ;
        segment_f <= NOT decodedata(1) ;
        segment_g <= NOT decodedata(0) ;
            end process ;
end Behavioral;
```

Timer Module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use Ieee.numeric_std.all ;
entity timer is
    Port ( Clock : in STD_LOGIC; -- 100 MHz
           seconds1, seconds2 : out std_logic_vector(3 downto 0);
           Switchon,Switch2on,Switch3on, Timwins, Elifwins: in STD_LOGIC;
           --Switch4on
           tu:out std_logic;
           clock_start: in STD_LOGIC
           );
end timer;

architecture Behavioral of timer is

    SIGNAL Sync : STD_LOGIC := '1';
```

```
    SIGNAL int : INTEGER range 0 to 49999999 := 0;
    SIGNAL sec, sec2, tsec :  std_logic_vector(3 downto 0);
    signal tbsec, tbsec2: std_logic_vector(3 downto 0); --stores best time of tim to complete game
    signal ebsec, ebsec2: std_logic_vector(3 downto 0); --stores best time of tim to complete game
    SIGNAL hour : INTEGER range 0 to 24 := 0;
    signal stopcounting: std_logic := '0';
    signal timcount,elifcount: integer := 0;

begin

    process ( Clock)
    begin
       --frequency = 1 Hz
       if ( rising_edge(Clock)) then
          if ( int < 49999999) then
             int <= int + 1;
          else
             int <= 0;
             Sync <= NOT Sync;
          end if;
       end if;
    end process;
    process ( Sync)
    begin
       if ( rising_edge(Sync)) then
            if Switch3on = '1' then
                tu <='0';
                sec<= "0000";
                sec2<= "0000";
            elsif (Switchon ='1') and (Timwins = '0') and (Elifwins = '0') and (clock_start <= '1') then
                if ( sec < 9) and ( sec2 < 6) then
                   sec <= sec + 1;
                else
```

```
                sec <= "0000";
                if ( sec2 < 6) then
                   sec2 <= sec2 + 1;
                else
                   tu<='1';
                 end if;
              end if;
        elsif Switchon = '0' then
            sec<= "0000";
            sec2<= "0000";
            tu <= '0';
        end if;
end if;
end process;
   Seconds1 <= sec;
   seconds2 <= sec2 ;
end Behavioral;
```