Youtube Video Link

https://youtu.be/kSBFRjKAO70

Abstract / Objective

The purpose of this project was to design and implement a code breaking game called “mastermind” on an FPGA using VHDL hardware description language. In order to achieve this, concepts of clock division, seven segment decoding and driving, push button debouncing, VGA, and graphics design were covered. First the design of the game graphics were made, then the design of the game mechanics took place. Finally, the implementation was done on a BASYS3 FPGA board with VHDL.

Design Specification Plan

First of all, a clear and thorough description of the game shall be given. “Mastermind” is a code breaking game. It is played on a decoding board, and the player tries to guess a hidden code. The code consists as a combination of colors chosen from a given color set. The player has a limited amount of guesses to find the code. For each of his guesses, the player receives certain hints about the similarity between the guess and the code to be cracked. If the guess has a correct color element in a correct position, then for each correct element, the current guess receives a ‘full correct’ key pin next to it. Else if the guess has a correct color element in an incorrect position, then for each correct element, the current guess receives a ‘half correct’ key pin next to it. Using the information provided by the key pins, the player makes further guesses and tries to crack the code. If the player successfully cracks the code without running out of guesses, he wins the game. Else, the player loses.

To implement this game, the decoding board was displayed on a monitor through the VGA interface, and the control inputs were taken from the pushbuttons on the BASYS3. A game score element was added to the game and was displayed both on screen and on the built in seven segment display of the BASYS3. Also, three switches were used, on for resetting the game, and two for changing the color scheme of the game.

The Design Methodology

First of all, I had to understand the basics of VGA to proceed with my project, so I did some research. Through this process, I have seen that the VGA interface has five signals in total: two for horizontal and vertical synchronization, and three for color determination. The synchronization signals are used to stabilize the display on the screen, and a pixel clock is used to scan through the pixels on the screen and color them in accordance with the color signals. The color signal traces through the screen, but it goes off the edges for some distance, during which the signal is stabilized. These distances are called as front porch, back porch, and sync pulse, and when these points are traced, no color should be displayed. These numbers change with screen resolution, and I have determined the screen resolution I wanted to use while implementing this game as 640x480. The corresponding porch and pulse numbers can be seen in my elements.vhd package. Furthermore, the display was working only in certain range of pixel frequencies, and for my screen resolution, I needed a pixel clock at 25 MHz. Since BASYS3 has a clock at 100 MHz, I also needed a clock divider module to decrease the clock frequency to the appropriate level.

After having an idea of VGA, I have designed the decoding board on which the game would be played based on the screen resolution. I used GIMP image processor for this purpose. My initial design is given below.

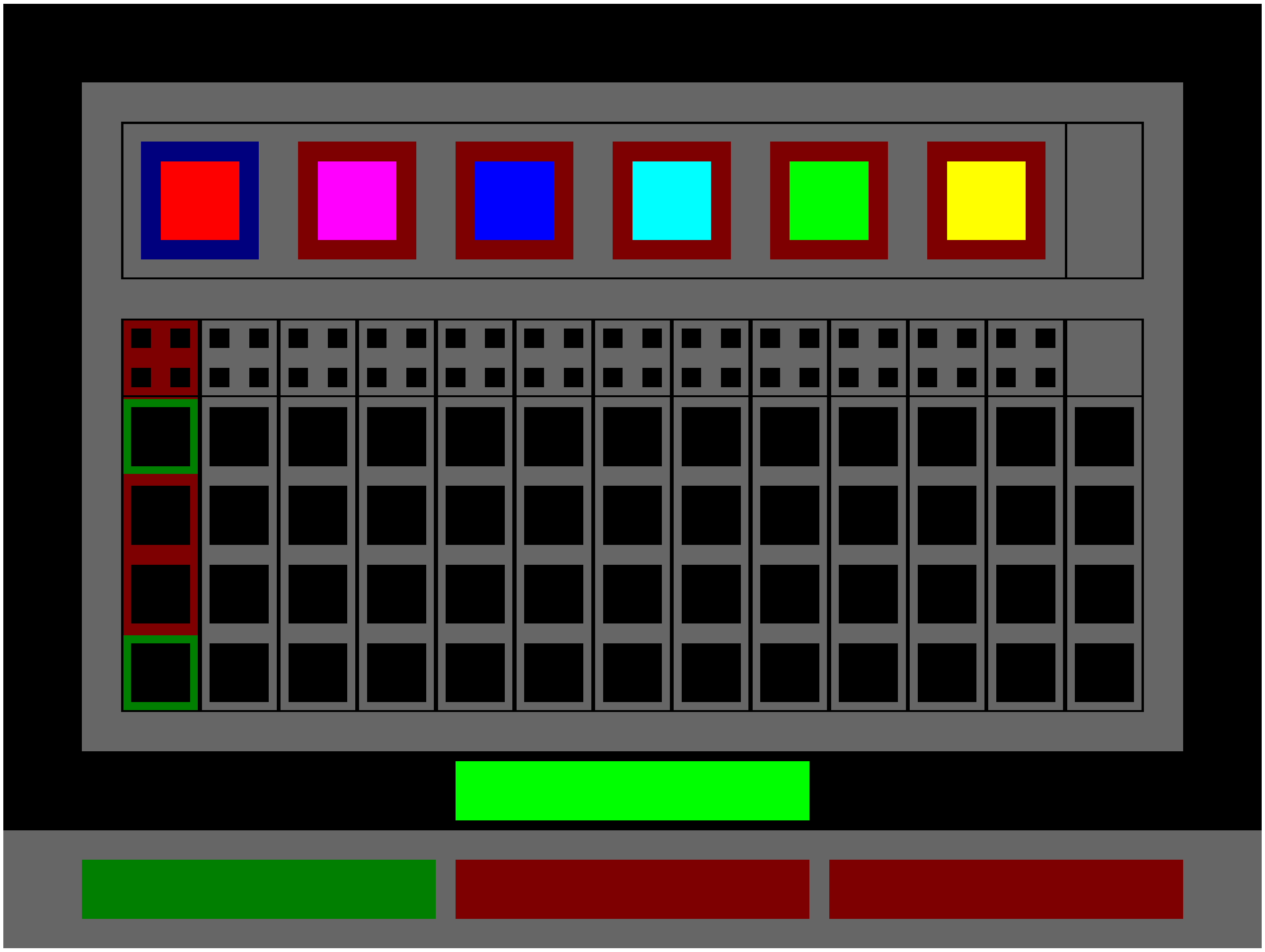

Illustration 1: my initial design for the decoding board. The menu at the bottom was not implemented in the final game, but a score indicator was displayed instead.

Then, from this graphic design, I have measured some distances and coordinates which I would be using to display the game graphics on the screen. These measurements can be seen in my elements.vhd package. Using these measurements, I was now ready for graphical implementation.

Afterwards, I have added the game controls. I used pushbuttons and switches to control the game. Pushbuttons were used to navigate between the cells. Selected cells are highlighted with a colored frame around them. The player selects a color cell with left and right pushbuttons, a guess cell with up and down pushbuttons, and assigns the selected color to the selected guess cell by pressing the center pushbutton. If the player pushes the center pushbutton at the key pin position (four small squares at top of each guess grid in illustration 1), a guess is submitted, and the hints for that guess is displayed. Since the pushbuttons were not very stable, they often debounced, I had to use an appropriate clock to determine the button presses. I have used a clock signal at 10 Hz, and only taken button press signals at the rising edge of the clock. This ensured that debouncing of the pushbuttons would not cause unwanted behavior.

I have assigned a switch for resetting the game, and two switches to select different color schemes for the game, such as monochrome color set for color blind players. When the game is reset, score and all cells are reset to their initial condition, and a random game code is generated for the new game.

When the game is won, the game code is revealed, and four non-black colored key pins show up at the top of the game code. If the game is lost, then the code is still revealed, but four black colored pins show up at the top of the game code instead.

I have also implemented a score system in the game to make it more entertaining. The player starts with a score of 500 to each game, and the score decreases by 2 for each passing second, and by 10 for each submitted guess. Therefore, players get higher scores for winning the game faster and with less number of guesses. The game score is indicated on the seven segment displays of the BASYS3, and as a red bar under the decoding board on the display.

I have designed five modules and one package in total for this project.

top_module: As is evident from its name, this was my top module, and it connected my other modules between themselves. The constraints file only dealt with this module.

clock_divider: Decreases the system clock frequency to appropriate frequencies for usage. I have used a 25 MHz clock for VGA synchronization, 200 Hz clock for driving the seven segment displays, and 10 Hz clock for pushbutton button press detection and updating the score.

vga_sync: Most of my work went into this module. It defines the game board, stores the colors of the cells, describes the game logic and responses to the pushbutton and switch inputs, and drives the VGA interface of the project.

elements: This package defines the useful VGA synchronization, game graphics, and game mechanics constants and types.

segment_driver & segment_decoder: These modules were used to display the game score on the seven segment displays. Segment decoder converts four bit binary inputs to seven segment code, and segment driver selects different seven segments sequentially to display them, to allow the visibility of different digits at the same time by human eye.

Results

Various game screens and the controls of the game can be seen in the following illustrations.

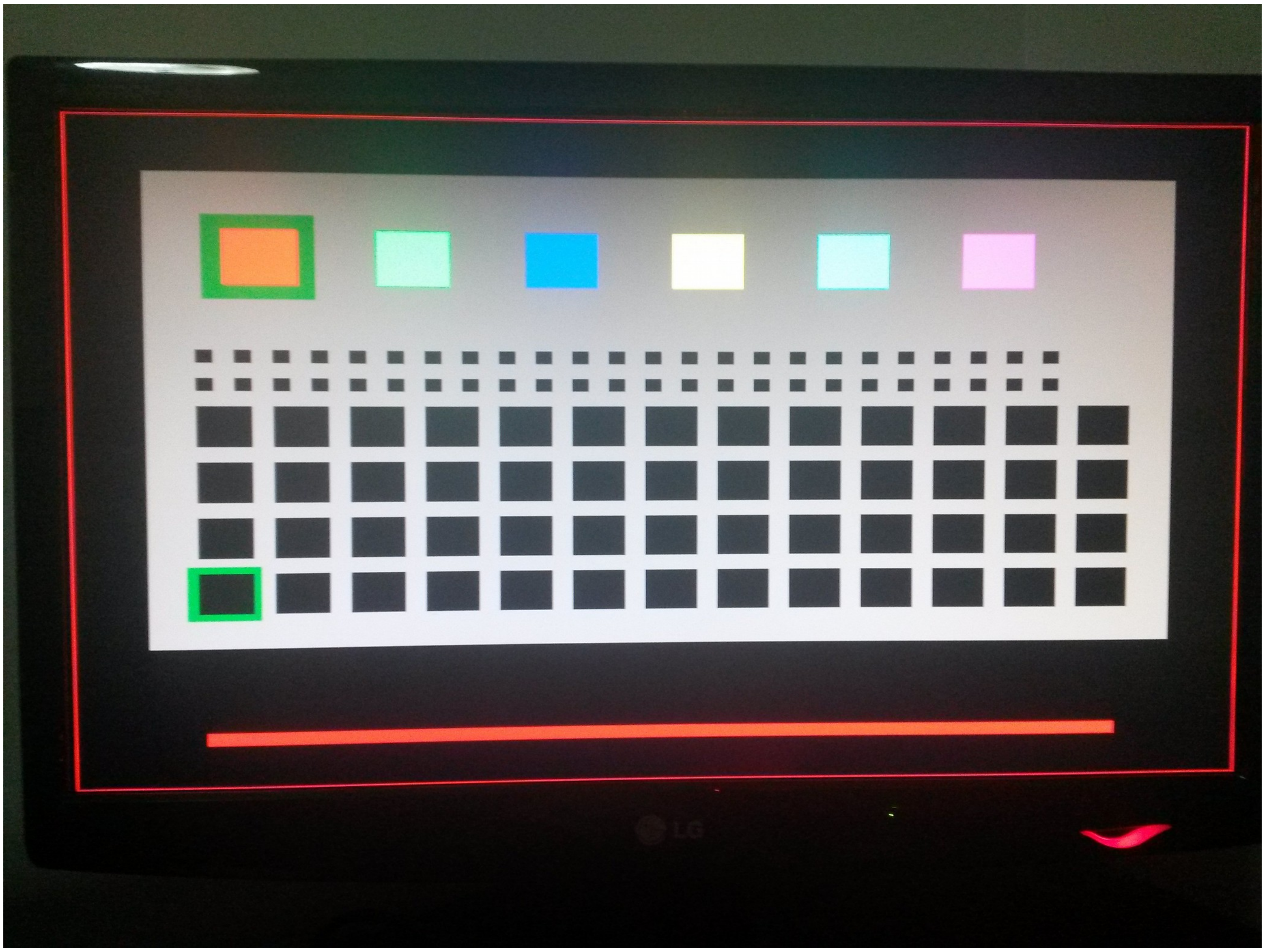

Illustration 2: The new game screen. The game opens with this screen and returns to this screen after every reset.

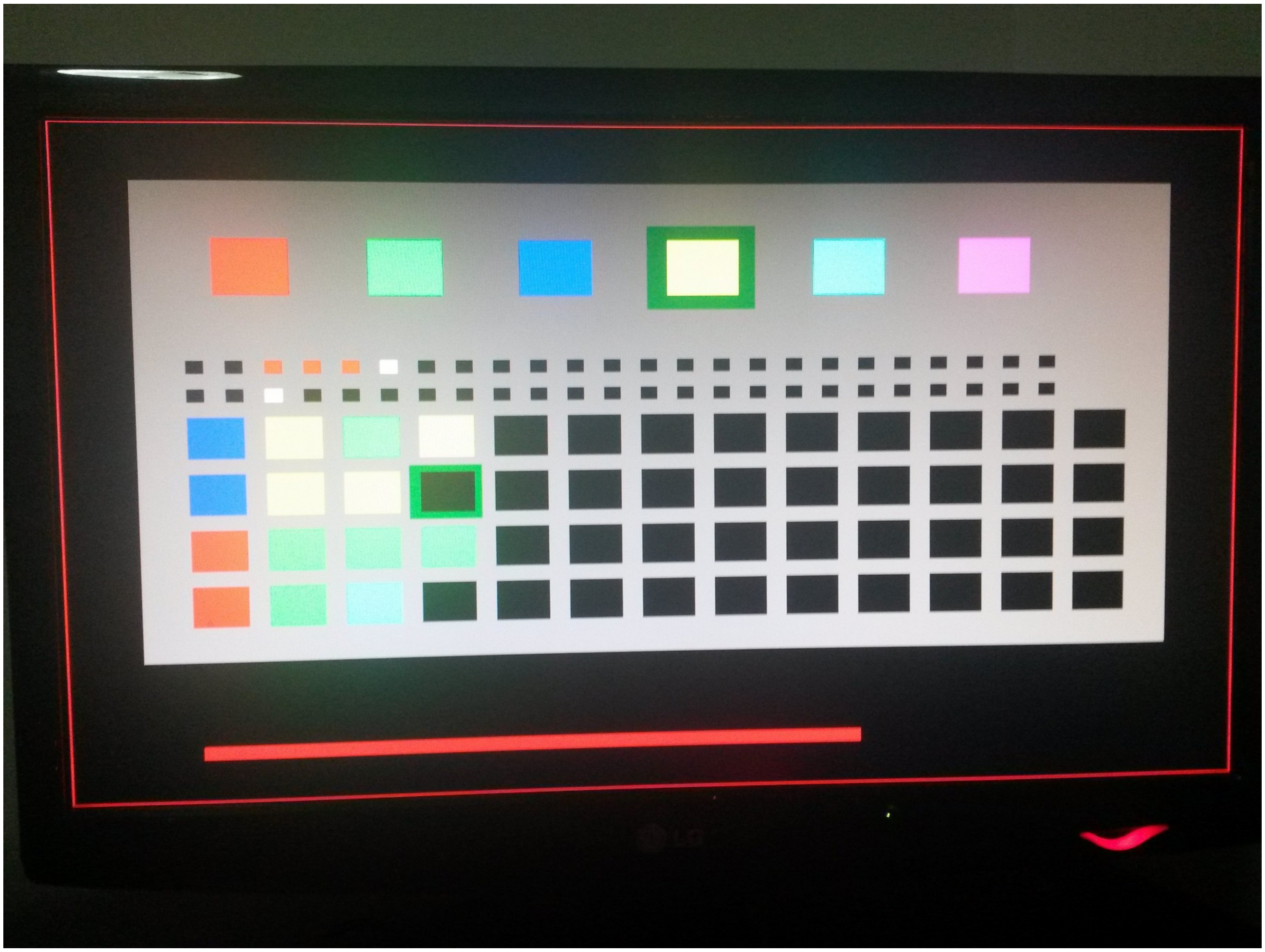

Illustration 3: A game in progress. Colors are picked from the top row, and guesses are formed. The key pins display hints about the current guess. The score bar at the bottom decreases with time and each submitted guess.

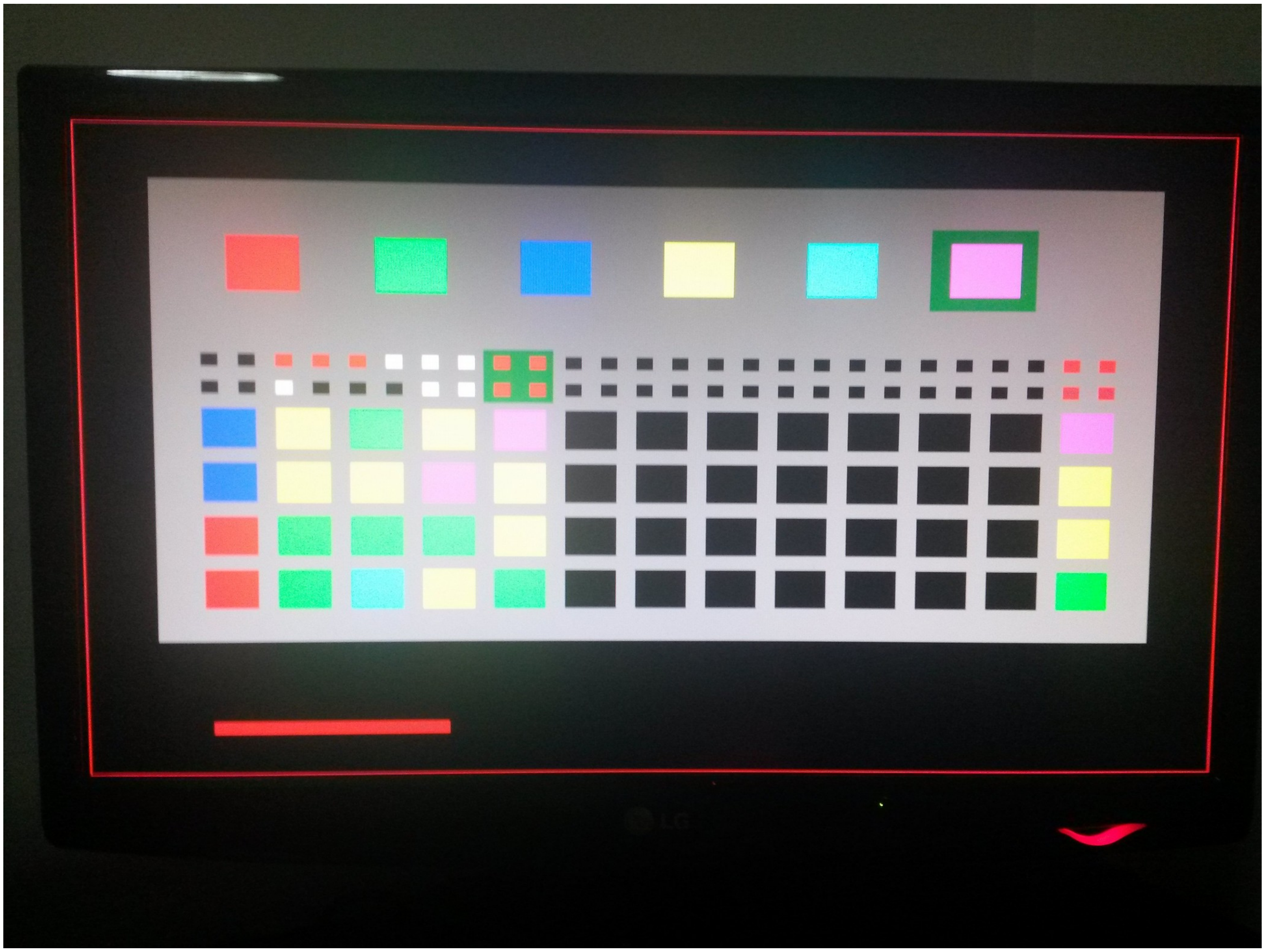

Illustration 4: A game won. No further change in the game can be done without resetting, and the score of the player is indicated with the bar at the bottom. Four red key pins indicate that the game has been won.

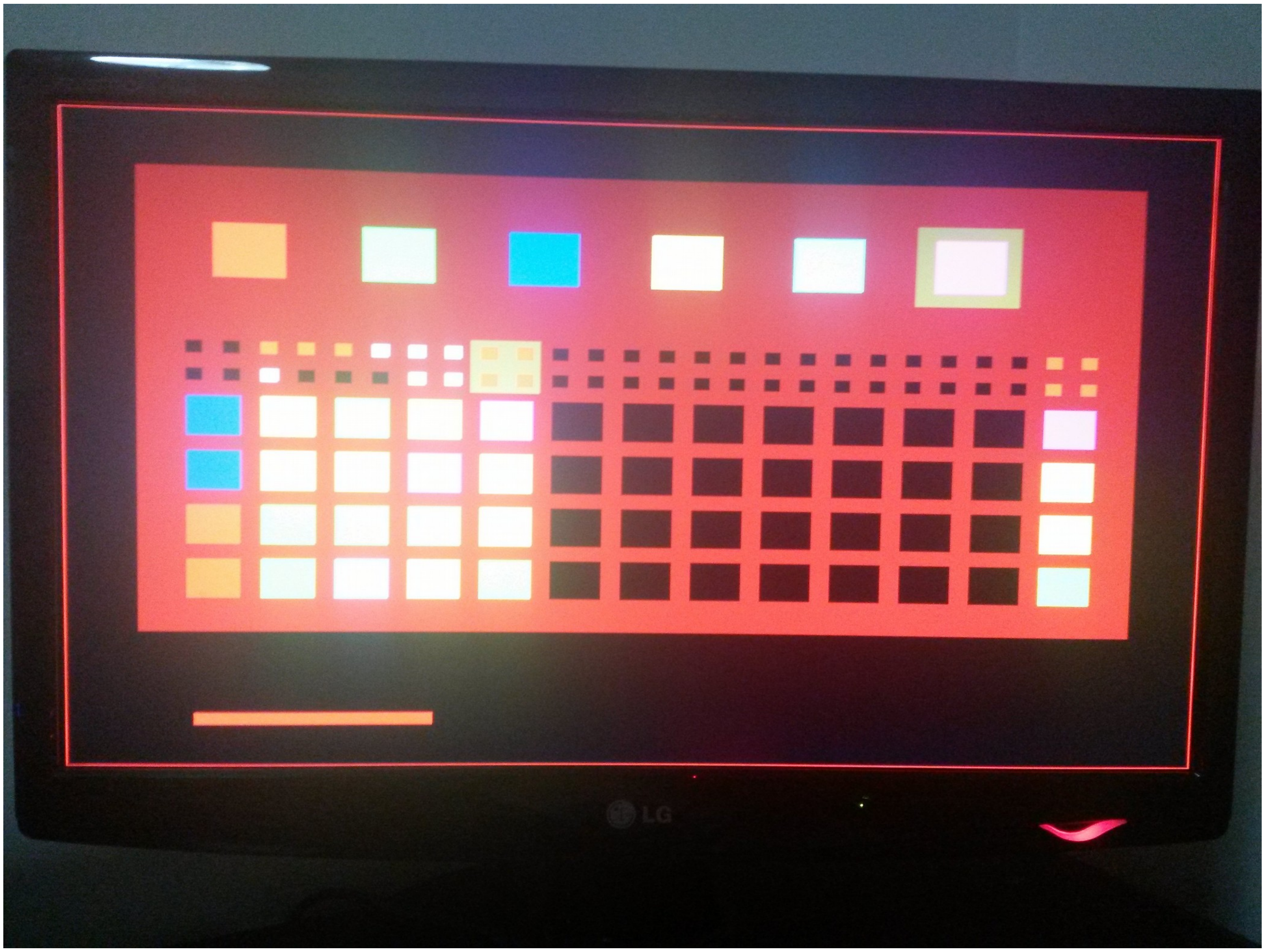

Illustration 5: A different color scheme for the game. Chosen with the switches.

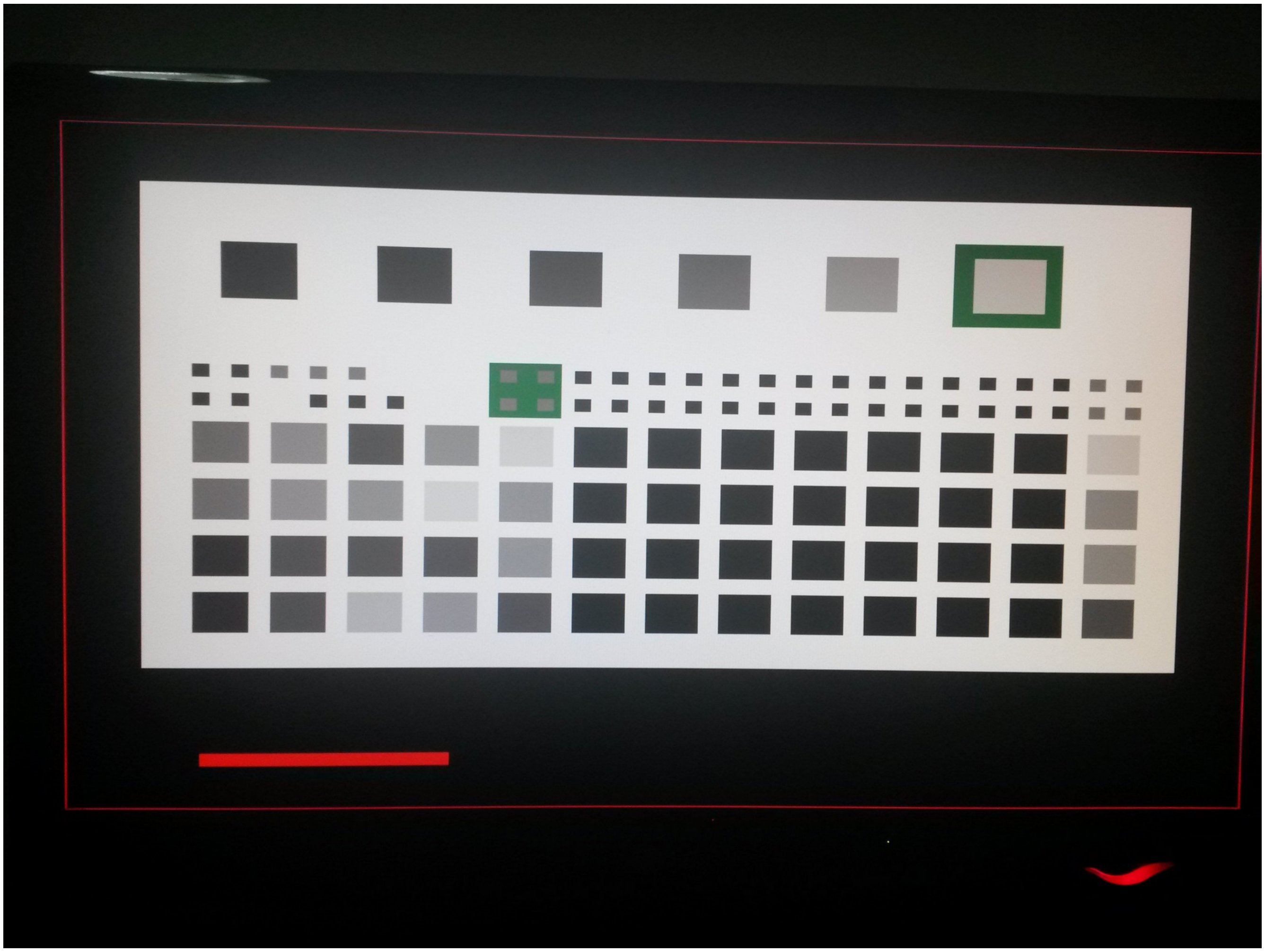

Illustration 6: Another color scheme for the game. Made for the color blind players.

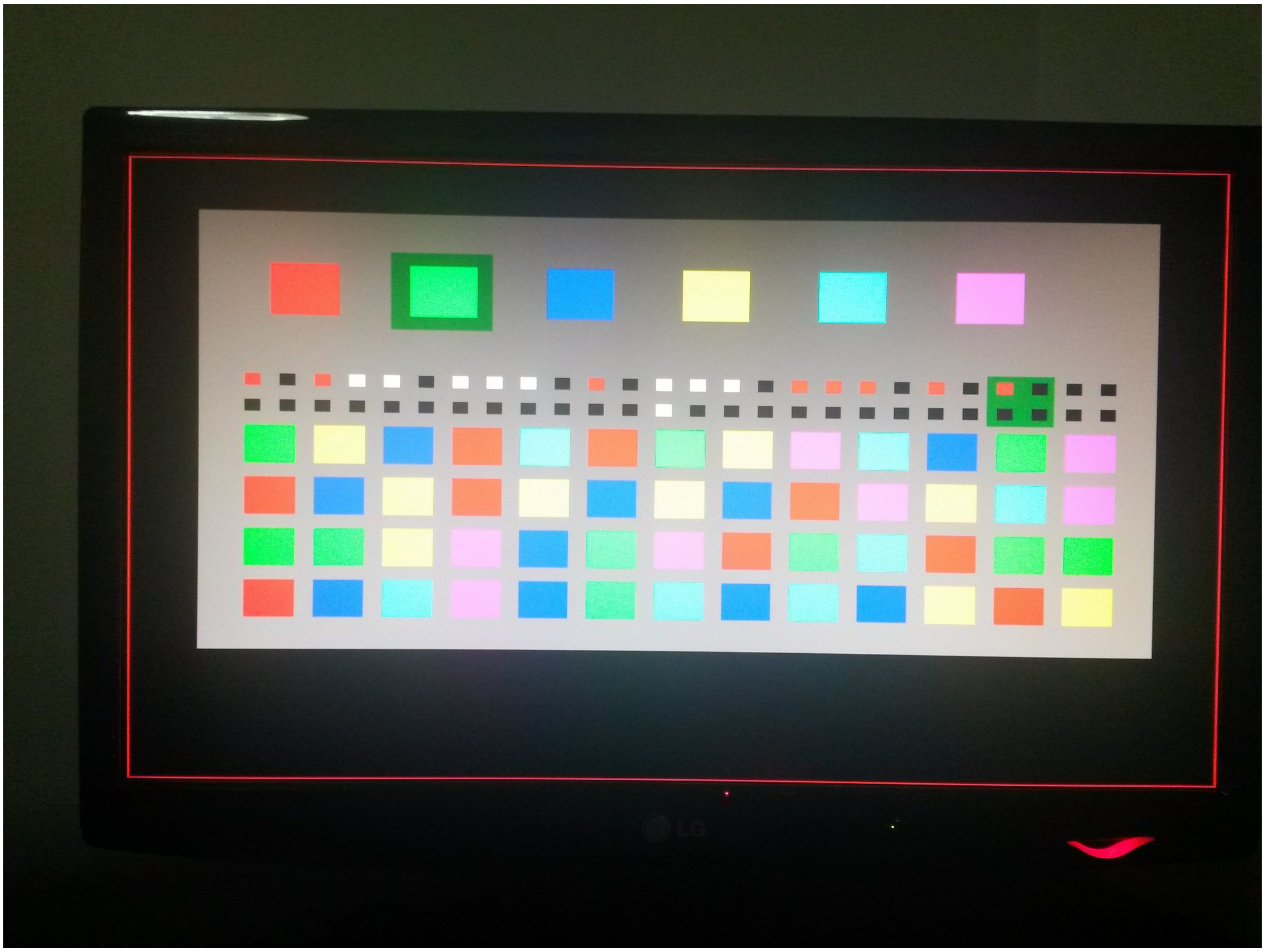

Illustration 7: A game lost. Four black key pins at the top of the revealed game code indicate that the game has been lost. The score bar is no longer visible.

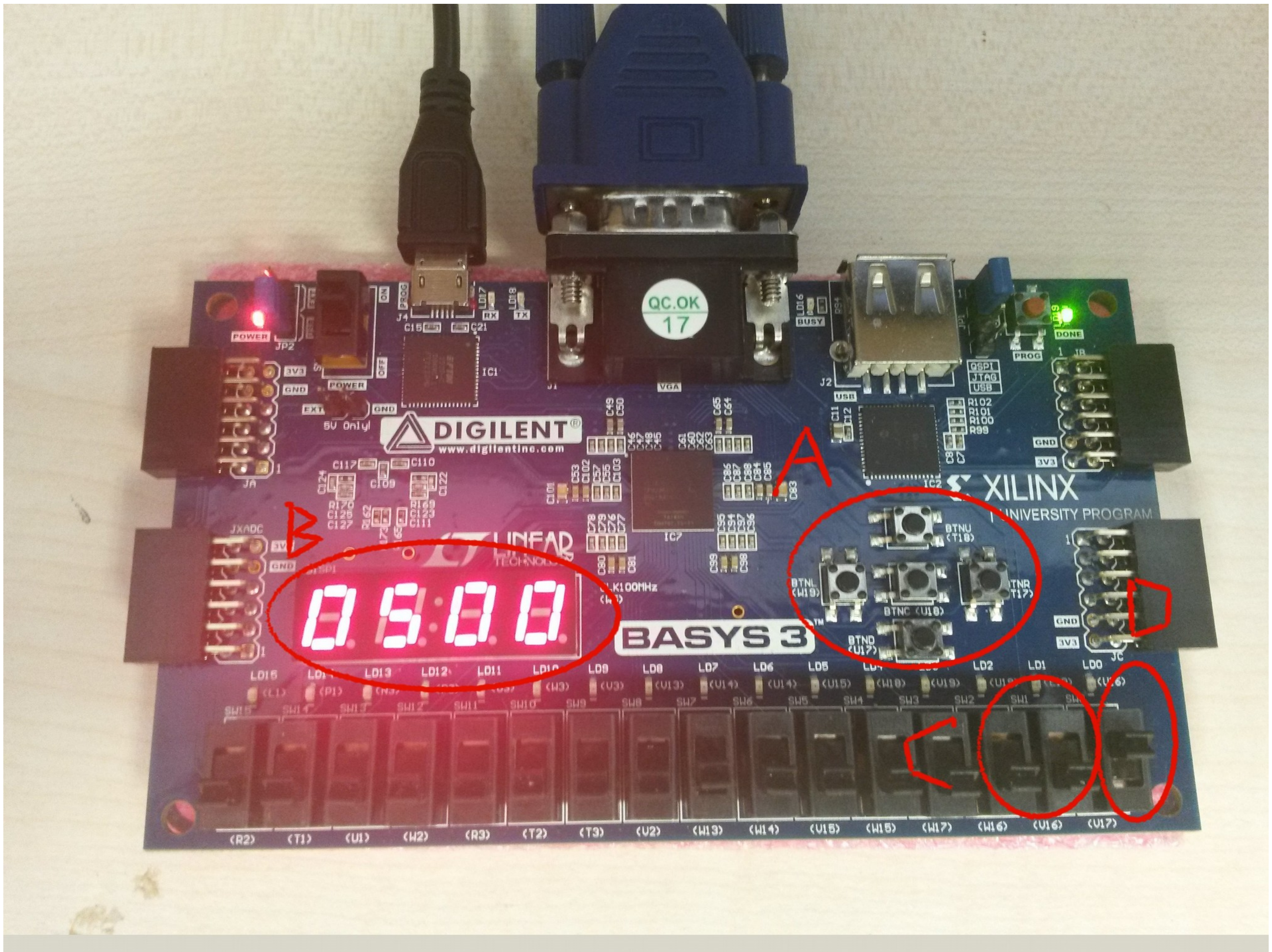

Illustration 8: The BASYS3 FPGA board on which the project was implemented. (A) the game is controlled by the pushbuttons. (B) the score is displayed on the seven segment displays. (C) the color scheme of the game can be changed by these two switches. (D) the game is reset by activating this switch. Deactivating this switch enables the player to play the game again.

Conclusion

Through this project, I have learned many things about concepts such as VHDL language, VGA interface, pushbutton debouncing, and seven segment displays. I have seen that making a design first can simplify the implementation process in graphical design. I have used for loops extensively in my VHDL code to draw the cells on the screen and to implement the game logic, and it saved me from writing many more lines of code. I have realized that VHDL is not like other high level programming languages, in the sense that it has the time notion embedded in it, and timing is very crucial in hardware design. Also, everything written on the VHDL language corresponds to a hardware component, though it may not be very obvious at all times, and this thought is fascinating for me. Furthermore, I have learned that by knowing the characteristics of an interface, I can drive different external devices from my FPGA. Overall, this project has been very informative for me and it made me understand some digital design concepts much better, so it was a good experience.

Appendices – VHDL Code

top_module.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--use IEEE.NUMERIC_STD.ALL;

ENTITY top_module IS
   PORT
   (
      clk, btnC, btnU, btnL, btnR, btnD : IN STD_LOGIC;
      sw                           : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      Hsync, Vsync                 : OUT STD_LOGIC;
      vgaRed, vgaGreen, vgaBlue    : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
      an                           : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
      seg                          : OUT STD_LOGIC_VECTOR (0 TO 6)
   );
END top_module;

ARCHITECTURE MAIN OF top_module IS

   COMPONENT clock_divider IS
      PORT
      (
         CLK_IN  : IN STD_LOGIC;
```

```
        CLK_25M : OUT STD_LOGIC;
        clk_200 : OUT STD_LOGIC;
        CLK_10  : OUT STD_LOGIC;
        clk_1   : OUT STD_LOGIC
    );
END COMPONENT clock_divider;


COMPONENT vga_sync IS
    PORT
    (
        Clk_25m, clk_10, clk_1         : IN STD_LOGIC;
        Left, Right, up, down, center : IN STD_LOGIC;
        switch                        : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        HSYNC, VSYNC                  : OUT STD_LOGIC;
        R, G, B                       : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
        game_score                    : OUT INTEGER RANGE 0 TO 300
    );
END COMPONENT vga_sync;


COMPONENT segment_driver
    PORT
    (
        input_int      : IN INTEGER;
        clk_200        : IN STD_LOGIC;
        segment        : OUT STD_LOGIC_VECTOR (0 TO 6);
        select_display : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
    );
    END COMPONENT segment_driver;


    SIGNAL clk_25m : std_logic;
    SIGNAL clk_200 : std_logic;
    SIGNAL clk_10  : std_logic;
    SIGNAL clk_1   : std_logic;


    SIGNAL score   : INTEGER;


BEGIN
    C1 : clock_divider
    PORT MAP
    (
        CLK_IN  => clk,
        CLK_25M => clk_25m,
        clk_200 => clk_200,
        clk_10  => clk_10,
        clk_1   => clk_1
    );
    C2 : vga_sync
    PORT MAP
    (
        CLK_25m    => clk_25m,
        clk_10     => clk_10,
        clk_1      => clk_1,
        LEFT       => btnL,
        RIGHT      => btnR,
        up          => btnU,
        down        => btnD,
        center      => btnC,
        switch      => sw,
        HSYNC       => Hsync,
        VSYNC       => Vsync,
        R           => vgaRed,
        G           => vgaGreen,
        B           => vgaBlue,
        game_score => score
    );

    C3 : segment_driver
    PORT MAP
    (
        input_int       => score,
        clk_200         => clk_200,
        segment         => seg,
        select_display => an
    );

END MAIN;
```

clock_divider.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--use IEEE.NUMERIC_STD.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY clock_divider IS
   PORT
   (
      CLK_IN  : IN STD_LOGIC;
      CLK_25M : OUT STD_LOGIC;
      CLK_200 : OUT STD_LOGIC;
      CLK_10  : OUT STD_LOGIC;
      CLK_1   : OUT STD_LOGIC
   );
END clock_divider;

ARCHITECTURE MAIN OF clock_divider IS
   SIGNAL clk_25MHz     : std_logic := '0';
   SIGNAL clk_200Hz     : std_logic := '0';
   SIGNAL clk_10Hz      : std_logic := '0';
   SIGNAL clk_1Hz       : std_logic := '0';
   SIGNAL clk_25MHz_int : INTEGER;
   SIGNAL clk_200Hz_int : INTEGER;
   SIGNAL clk_10Hz_int  : INTEGER;
   SIGNAL clk_1Hz_int   : INTEGER;
```

```
BEGIN
   PROCESS (CLK_IN)
   BEGIN
      IF rising_edge(CLK_IN) THEN

         clk_25MHz_int <= clk_25MHz_int + 1;
         clk_200Hz_int <= clk_200Hz_int + 1;
         clk_10Hz_int  <= clk_10Hz_int + 1;
         clk_1Hz_int   <= clk_1Hz_int + 1;

         IF clk_25MHz_int = 4/2 - 1 THEN
            clk_25MHz     <= NOT clk_25MHz;
            clk_25MHz_int <= 0;
         END IF;

         IF clk_200Hz_int = 500000/2 - 1 THEN
            clk_200Hz     <= NOT clk_200Hz;
            clk_200Hz_int <= 0;
         END IF;

         IF clk_10Hz_int = 10000000/2 - 1 THEN
            clk_10Hz     <= NOT clk_10Hz;
            clk_10Hz_int <= 0;
         END IF;

         IF clk_1Hz_int = 100000000/2 - 1 THEN
```

```
                clk_1Hz     <= NOT clk_1Hz;
                clk_1Hz_int <= 0;
            END IF;

        END IF;

    END PROCESS;

    CLK_25M <= clk_25MHz;
    CLK_200 <= clk_200Hz;
    CLK_10  <= clk_10Hz;
    CLK_1   <= clk_1Hz;

END MAIN;
```

vga_sync.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE work.elements.ALL;

ENTITY vga_sync IS
   PORT
   (
      clk_25m, clk_10, clk_1        : IN STD_LOGIC;
      left, right, up, down, center : IN STD_LOGIC;
      switch                        : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      game_score                    : OUT INTEGER RANGE 0 TO 300;
      HSYNC, VSYNC                  : OUT STD_LOGIC;
      R, G, B                       : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
   );
END vga_sync;

ARCHITECTURE MAIN OF vga_sync IS

   --synchronization signals
   SIGNAL HPOS_cur  : INTEGER RANGE 1 TO TOT_H := 1;
   SIGNAL VPOS_cur  : INTEGER RANGE 1 TO TOT_V := 1;
   SIGNAL HPOS_next : INTEGER RANGE 1 TO TOT_H;
   SIGNAL VPOS_next : INTEGER RANGE 1 TO TOT_V;
   SIGNAL rgb_cur   : std_logic_vector (11 DOWNTO 0);
```

```
    SIGNAL rgb_next  : std_logic_vector (11 DOWNTO 0);


    --game status signals
    SIGNAL reset    : std_logic := '0';
    SIGNAL in_game  : std_logic := '1';
    SIGNAL game_won : std_logic := '0';
    SIGNAL score    : INTEGER := 500;


    --color scheme signals
    SIGNAL color_scheme_select : INTEGER;
    SIGNAL color_scheme        : COLOR_SCHEMES (0 TO 3);


    --signal to determine if a color layer is to be displayed
    SIGNAL show_color : std_logic_vector (1 TO 15);


    --cell colors stored as integers
     --color codes for color cells are shifted by COLOR_INDEX_SHIFT and key cells are shifted by KEYPIN_INDEX_SHIFT
    --this was done to keep all game colors in one array
     SIGNAL guess_cell_colors  : INT_ARRAY_2D (0 TO 12, 0 TO 3) := (OTHERS => (OTHERS => 4));
      SIGNAL key_cell_colors     : INT_ARRAY_2D (0 TO 12, 0 TO 3) := (12 => (OTHERS => 0), OTHERS => (OTHERS => 11));
     SIGNAL color_panel_colors : INT_ARRAY_2D (0 TO 5, 0 TO 0) := (OTHERS => (OTHERS => 0));


    --the game code to be solved
    SIGNAL game_code : INT_ARRAY_2D (0 TO 0, 0 TO 3) := (0 => (5, 6, 7, 8));
```

```
   --definition of the game board cell
   SIGNAL game_board : CELL := (8,
      NULL_H + SCREEN_BORDER, NULL_V + SCREEN_BORDER,
            BOARD_H, BOARD_V
   );


   --definition of the pin selection cell
   SIGNAL pin_selection : SELECTION_CELL := (3,
          NULL_H + SCREEN_BORDER + BOARD_BORDER, NULL_V + SCREEN_BORDER +
BOARD_BORDER + COLOR_PANEL_V + BOARD_BORDER + 4 * CELL_SIDE,
            CELL_SIDE, CELL_SIDE, 0, 4
   );


   --definition of the color selection cell
   SIGNAL color_selection : SELECTION_CELL := (3,
       NULL_H + SCREEN_BORDER + 2 * BOARD_BORDER - COLOR_SELECTION_BORDER,
NULL_V + SCREEN_BORDER + 2 * BOARD_BORDER - COLOR_SELECTION_BORDER,
            COLOR_SELECTION_SIDE, COLOR_SELECTION_SIDE, 0, 0
   );


   --definition of the score bar cell
   SIGNAL score_bar : CELL := (14,
            NULL_H + SCREEN_BORDER + 30, TOT_V - SCREEN_BORDER,
            score, KEYPIN_SIDE
   );

BEGIN
```

```
--a process for initializing the color selection panel and color schemes
--could have been done manually
initialization : PROCESS IS
BEGIN
   FOR i IN 0 TO 5 LOOP
      color_panel_colors(i, 0) <= i + 5;
   END LOOP;
   --assign some color schemes that can be used
   color_scheme(0) <= color_set_0;
   color_scheme(1) <= color_set_1;
   color_scheme(2) <= color_set_2;
   --other color sets
   FOR j IN 0 TO 15 LOOP
      color_scheme(3)(j) <= NOT color_set_0(j);
   END LOOP;
   WAIT;
END PROCESS;


--process to update the game on clock signals
update : PROCESS (clk_25m, clk_10, clk_1, reset) IS
   --variables to be used in the game logic
   VARIABLE num_of_red_key_pins    : INTEGER RANGE 0 TO 4;
   VARIABLE num_of_white_key_pins  : INTEGER RANGE 0 TO 4;
   VARIABLE counted_guess_elements : std_logic_vector(0 TO 3);
   VARIABLE counted_code_elements  : std_logic_vector(0 TO 3);


   VARIABLE clk_count_10 : INTEGER := 0;
```

```
BEGIN
    IF rising_edge(clk_10) THEN
        --score control
        IF reset = '0' AND in_game = '1' AND score > 0 THEN
            clk_count_10 := clk_count_10 + 1;


            --decrement score by guess submission
            IF center = '1' AND pin_selection(6) = 0 THEN
                IF score > 10 THEN
                    score <= score - 10;
                ELSE
                    score <= 0;
                END IF;
            --decrement score by time
            ELSIF clk_count_10 = 10 THEN
                clk_count_10 := 0;
                score <= score - 2;
            END IF;


        --if game lost, reset score to 0
        ELSIF in_game = '0' AND game_won = '0' THEN
            score <= 0;
        --if game reset, reset score to 500
        ELSIF reset = '1' THEN
            clk_count_10 := 0;
            score <= 500;
        END IF;
```

```
--reset control
IF reset = '1' THEN
    --generate a random code
        game_code <= (0 => ((HPOS_cur MOD 6) + 5, (VPOS_cur MOD 6) + 5, ((HPOS_cur * VPOS_cur) MOD 6) + 5, ((HPOS_cur/VPOS_cur) MOD 6) + 5));
    --reset cell colors, selections and game status
    guess_cell_colors <= (OTHERS => (OTHERS => 4));
    key_cell_colors   <= (12 => (OTHERS => 0), OTHERS => (OTHERS => 11));
    pin_selection     <= (3,
        NULL_H + SCREEN_BORDER + BOARD_BORDER,
                NULL_V + SCREEN_BORDER + BOARD_BORDER + COLOR_PANEL_V + BOARD_BORDER + 4 * CELL_SIDE,
        CELL_SIDE, CELL_SIDE, 0, 4);
    color_selection <= (3,
                        NULL_H + SCREEN_BORDER + 2 * BOARD_BORDER - COLOR_SELECTION_BORDER,
                        NULL_V + SCREEN_BORDER + 2 * BOARD_BORDER - COLOR_SELECTION_BORDER,
        COLOR_SELECTION_SIDE, COLOR_SELECTION_SIDE, 0, 0);
    in_game <= '1';


ELSIF in_game = '1' THEN
    --control left and right
    IF left = '1' AND right = '0' AND color_selection(5) > 0 THEN
        color_selection(1) <= color_selection(1) - 2 * CELL_SIDE;
        color_selection(5) <= color_selection(5) - 1;
    ELSIF left = '0' AND right = '1' AND color_selection(5) < 5 THEN
    color_selection(1) <= color_selection(1) + 2 * CELL_SIDE;
    color_selection(5) <= color_selection(5) + 1;


--control up and down
ELSIF up = '1' AND down = '0' AND pin_selection(6) > 0 THEN
    pin_selection(2) <= pin_selection(2) - CELL_SIDE;
    pin_selection(6) <= pin_selection(6) - 1;
ELSIF up = '0' AND down = '1' AND pin_selection(6) < 4 THEN
    pin_selection(2) <= pin_selection(2) + CELL_SIDE;
    pin_selection(6) <= pin_selection(6) + 1;


--control center button
ELSIF center = '1' THEN
    --submit guess
    IF pin_selection(6) = 0 AND pin_selection(5) < TOT_GUESS_NUMBER THEN
        num_of_white_key_pins  := 0;
        num_of_red_key_pins    := 0;
        counted_guess_elements := "0000";
        counted_code_elements  := "0000";
        --determine number of key pins
        FOR i IN 0 TO 3 LOOP
            IF guess_cell_colors(pin_selection(5), i) = game_code(0, i) THEN
                num_of_red_key_pins       := num_of_red_key_pins + 1;
                counted_guess_elements(i) := '1';
                counted_code_elements(i)  := '1';
            END IF;
        END LOOP;
```

```
FOR i IN 0 TO 3 LOOP
   FOR j IN 0 TO 3 LOOP
      IF counted_guess_elements(i) = '0' AND counted_code_elements(j) = '0' AND
       guess_cell_colors(pin_selection(5), i) = game_code(0, j) THEN

         num_of_white_key_pins     := num_of_white_key_pins + 1;

         counted_guess_elements(i) := '1';
         counted_code_elements(j)  := '1';

         EXIT;
      END IF;
   END LOOP;
END LOOP;

--update key pin colors
FOR j IN 0 TO 3 LOOP
   IF j < num_of_red_key_pins THEN
      key_cell_colors(pin_selection(5), j) <= 13;
   ELSIF j < num_of_red_key_pins + num_of_white_key_pins THEN
      key_cell_colors(pin_selection(5), j) <= 12;
   END IF;
END LOOP;

--determine if game is over or continuing
--game won
```

```
IF num_of_red_key_pins = 4 THEN
  FOR j IN 0 TO 3 LOOP
    guess_cell_colors(TOT_GUESS_NUMBER, j) <= game_code(0, j);
    key_cell_colors(TOT_GUESS_NUMBER, j)   <= 13;
  END LOOP;

  game_won <= '1';
  in_game  <= '0';

--game lost
 ELSIF pin_selection(5) = TOT_GUESS_NUMBER - 1 AND num_of_red_key_pins /= 4 THEN
  FOR j IN 0 TO 3 LOOP

    -- for j in 0 to 3 loop

    guess_cell_colors(TOT_GUESS_NUMBER, j) <= game_code(0, j);

    key_cell_colors(TOT_GUESS_NUMBER, j)   <= 11;

    -- end loop;

  END LOOP;

  game_won <= '0';
  in_game  <= '0';

  --game ongoing
```

```
            ELSIF pin_selection(5) < TOT_GUESS_NUMBER - 1 THEN

              pin_selection(1) <= pin_selection(1) + CELL_SIDE;

              pin_selection(2) <= pin_selection(2) + 4 * CELL_SIDE;

              pin_selection(5) <= pin_selection(5) + 1;

              pin_selection(6) <= 4;


            END IF;


          --change the currently selected element color

          ELSE

            guess_cell_colors (pin_selection(5), pin_selection(6) - 1) <= color_selection(5) + 5;

          END IF;

        END IF;

      END IF;

    END IF;


    --

    IF rising_edge(clk_25m) THEN


      --determine if inside a color cell

      loop1 : FOR i IN 0 TO 12 LOOP

        FOR j IN 0 TO 3 LOOP

                 IF (HPOS_cur > NULL_H + SCREEN_BORDER + BOARD_BORDER + i *
CELL_SIDE + CELL_BORDER AND

          HPOS_cur < NULL_H + SCREEN_BORDER + BOARD_BORDER + i * CELL_SIDE +
CELL_BORDER + ELEMENT_SIDE) AND
```

```
                                    (VPOS_cur > NULL_V + SCREEN_BORDER + BOARD_BORDER +
COLOR_PANEL_V + BOARD_BORDER + CELL_SIDE + j * CELL_SIDE + CELL_BORDER
AND
                                    VPOS_cur < NULL_V + SCREEN_BORDER + BOARD_BORDER +
COLOR_PANEL_V + BOARD_BORDER + CELL_SIDE + j * CELL_SIDE + CELL_BORDER +
ELEMENT_SIDE) THEN


                    FOR k IN 4 TO 10 LOOP
                        IF guess_cell_colors(i, j) = k THEN
                            show_color(k) <= '1';
                        ELSE
                            show_color(k) <= '0';
                        END IF;
                    END LOOP;


                    EXIT loop1; --exit if inside a cell already


                        ELSIF (HPOS_cur > NULL_H + SCREEN_BORDER + BOARD_BORDER + i *
CELL_SIDE + CELL_BORDER + (j MOD 2) * (KEYPIN_SIDE + KEYPIN_BORDER) AND
                    HPOS_cur < NULL_H + SCREEN_BORDER + BOARD_BORDER + i * CELL_SIDE
+ CELL_BORDER + (j MOD 2) * (KEYPIN_SIDE + KEYPIN_BORDER) + KEYPIN_SIDE) AND
                                    (VPOS_cur > NULL_V + SCREEN_BORDER + BOARD_BORDER +
COLOR_PANEL_V + BOARD_BORDER + CELL_BORDER + (j / 2) * (KEYPIN_SIDE +
KEYPIN_BORDER) AND
                                    VPOS_cur < NULL_V + SCREEN_BORDER + BOARD_BORDER +
COLOR_PANEL_V + BOARD_BORDER + CELL_BORDER + (j / 2) * (KEYPIN_SIDE +
KEYPIN_BORDER) + KEYPIN_SIDE) THEN


                    FOR k IN 11 TO 13 LOOP
                        IF key_cell_colors(i, j) = k THEN
```

```
                show_color(k) <= '1';
              ELSE
                show_color(k) <= '0';
              END IF;
            END LOOP;

            EXIT loop1; --exit if inside a cell already

          ELSE
            show_color(4 TO 13) <= (OTHERS => '0');
          END IF;
        END LOOP;

        IF (i < 6) THEN
               IF (HPOS_cur > NULL_H + SCREEN_BORDER + 2 * BOARD_BORDER + i *
(CELL_SIDE + 2 * BOARD_BORDER) AND
                  HPOS_cur < NULL_H + SCREEN_BORDER + 2 * BOARD_BORDER + i *
(CELL_SIDE + 2 * BOARD_BORDER) + CELL_SIDE) AND
           (VPOS_cur > NULL_V + SCREEN_BORDER + 2 * BOARD_BORDER AND
            VPOS_cur < NULL_V + SCREEN_BORDER + 2 * BOARD_BORDER + CELL_SIDE)
THEN

            FOR k IN 5 TO 10 LOOP
              IF color_panel_colors(i, 0) = k THEN
                show_color(k) <= '1';
              ELSE
                show_color(k) <= '0';
              END IF;
```

```
                END LOOP;


                EXIT loop1; --exit if inside a cell already


              ELSE
                show_color(5 TO 10) <= (OTHERS => '0');
              END IF;
            END IF;
          END LOOP;


          HPOS_cur <= HPOS_next;
          VPOS_cur <= VPOS_next;
          rgb_cur  <= rgb_next;
        END IF;
      END PROCESS;


      reset            <= switch(0);


      color_scheme_select <= conv_integer(switch(2 DOWNTO 1));


      --background
      show_color(1) <= '1' WHEN (HPOS_cur > NULL_H + 1 AND HPOS_cur < TOT_H - 1) AND
              (VPOS_cur > NULL_V + 1 AND VPOS_cur < TOT_V) ELSE
              '0';
      --board
       show_color(2) <= '1' WHEN (HPOS_cur > game_board(1) AND HPOS_cur < game_board(1) +
    game_board(3)) AND
              (VPOS_cur > game_board(2) AND VPOS_cur < game_board(2) + game_board(4)) ELSE
```

```
                '0';


   --selection
    show_color(3) <= '1' WHEN ((HPOS_cur > pin_selection(1) AND HPOS_cur < pin_selection(1) + pin_selection(3)) AND
                (VPOS_cur > pin_selection(2) AND VPOS_cur < pin_selection(2) + pin_selection(4)))
                OR
                        ((HPOS_cur > color_selection(1) AND HPOS_cur < color_selection(1) + color_selection(3)) AND
                         (VPOS_cur > color_selection(2) AND VPOS_cur < color_selection(2) + color_selection(4))) ELSE
                '0';


   --score bar
    show_color(14) <= '1' WHEN (HPOS_cur > score_bar(1) AND HPOS_cur < score_bar(1) + score) AND
                 (VPOS_cur > score_bar(2) AND VPOS_cur < score_bar(2) + score_bar(4)) ELSE
                 '0';


   --frame
     show_color(15) <= '1' WHEN (((VPOS_cur = NULL_V + 1) OR (VPOS_cur = TOT_V)) AND (HPOS_cur > NULL_H AND HPOS_cur <= TOT_H)) OR
                   (((HPOS_cur = NULL_H + 1) OR (HPOS_cur = TOT_H - 1)) AND (VPOS_cur > NULL_V AND VPOS_cur <= TOT_V)) ELSE
                 '0';


   --scanning through pixels
   HPOS_next <= HPOS_cur + 1 WHEN HPOS_cur < TOT_H ELSE
              1;
```

```
VPOS_next <= VPOS_cur + 1 WHEN HPOS_cur = TOT_H AND VPOS_cur < TOT_V ELSE
            1 WHEN HPOS_cur = TOT_H AND VPOS_cur = TOT_V ELSE
            VPOS_cur;


--rgb setting
--colors are arranged as layers
--layers of colors are in a way put on top of each other to produce the rgb signal
rgb_next <= color_scheme(color_scheme_select)(15) WHEN show_color(15) = '1' ELSE
           color_scheme(color_scheme_select)(14) WHEN show_color(14) = '1' ELSE
           color_scheme(color_scheme_select)(13) WHEN show_color(13) = '1' ELSE
           color_scheme(color_scheme_select)(12) WHEN show_color(12) = '1' ELSE
           color_scheme(color_scheme_select)(11) WHEN show_color(11) = '1' ELSE
           color_scheme(color_scheme_select)(10) WHEN show_color(10) = '1' ELSE
           color_scheme(color_scheme_select)(9) WHEN show_color(9) = '1' ELSE
           color_scheme(color_scheme_select)(8) WHEN show_color(8) = '1' ELSE
           color_scheme(color_scheme_select)(7) WHEN show_color(7) = '1' ELSE
           color_scheme(color_scheme_select)(6) WHEN show_color(6) = '1' ELSE
           color_scheme(color_scheme_select)(5) WHEN show_color(5) = '1' ELSE
           color_scheme(color_scheme_select)(4) WHEN show_color(4) = '1' ELSE
           color_scheme(color_scheme_select)(3) WHEN show_color(3) = '1' ELSE
           color_scheme(color_scheme_select)(2) WHEN show_color(2) = '1' ELSE
           color_scheme(color_scheme_select)(1) WHEN show_color(1) = '1' ELSE
           COLOR_BLACK;


--syncronization signals
HSYNC      <= '0' WHEN (HPOS_cur > FP_H) AND (HPOS_cur < FP_H + SP_H + 1) ELSE '1';
VSYNC      <= '0' WHEN (VPOS_cur > FP_V) AND (VPOS_cur < FP_V + SP_V + 1) ELSE '1';
```

```
    R         <= rgb_cur(11 DOWNTO 8);
    G         <= rgb_cur(7 DOWNTO 4);
    B         <= rgb_cur(3 DOWNTO 0);

    game_score <= score;

END MAIN;
```

segment_driver.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY segment_driver IS
  PORT
  (
    input_int      : IN INTEGER;
    clk_200        : IN STD_LOGIC;
    segment        : OUT STD_LOGIC_VECTOR (0 TO 6);
    select_display : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
  );
END segment_driver;

ARCHITECTURE Behavioral OF segment_driver IS

  COMPONENT segment_decoder
    PORT
    (
      digit    : IN std_logic_vector(3 DOWNTO 0);
      segments : OUT std_logic_vector(0 TO 6)
    );
  END COMPONENT;

  SIGNAL temporary_data               : std_logic_vector(3 DOWNTO 0);
```

```
    SIGNAL digit3, digit2, digit1, digit0 : std_logic_vector(3 DOWNTO 0);

BEGIN
    digit3 <= conv_std_logic_vector(input_int / 1000, 4);
    digit2 <= conv_std_logic_vector((input_int MOD 1000) / 100, 4);
    digit1 <= conv_std_logic_vector((input_int MOD 100) / 10, 4);
    digit0 <= conv_std_logic_vector((input_int MOD 10), 4);

    uut0_1 : segment_decoder
    PORT MAP
    (
        digit    => temporary_data,
        segments => segment
    );

    PROCESS (clk_200)
    VARIABLE display_selection : std_logic_vector(1 DOWNTO 0);
    BEGIN
        IF rising_edge(clk_200) THEN
            CASE display_selection IS
                WHEN "00" => temporary_data <= digit0;
                select_display              <= "1110";

                WHEN "01" => temporary_data <= digit1;
                select_display              <= "1101";
        WHEN "10" => temporary_data <= digit2;
        select_display          <= "1011";

        WHEN "11" => temporary_data <= digit3;
        select_display          <= "0111";

        WHEN OTHERS => temporary_data    <= digit3;
        select_display          <= "1111";

      END CASE;

      display_selection := display_selection + 1;

    END IF;

  END PROCESS;

END Behavioral;
```

segment_decoder.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY segment_decoder IS
   PORT
   (
      digit    : IN std_logic_vector(3 DOWNTO 0);
      segments : OUT std_logic_vector(0 TO 6)
   );
END segment_decoder;

ARCHITECTURE Behavioral OF segment_decoder IS

BEGIN
   PROCESS (digit)

   BEGIN
      CASE digit IS
         WHEN "0000" => segments <= "0000001"; -- "0"
         WHEN "0001" => segments <= "1001111"; -- "1"
         WHEN "0010" => segments <= "0010010"; -- "2"
         WHEN "0011" => segments <= "0000110"; -- "3"
         WHEN "0100" => segments <= "1001100"; -- "4"
         WHEN "0101" => segments <= "0100100"; -- "5"
         WHEN "0110" => segments <= "0100000"; -- "6"
         WHEN "0111" => segments <= "0001111"; -- "7"
```

```
          WHEN "1000" => segments <= "0000000"; -- "8"
          WHEN "1001" => segments <= "0000100"; -- "9"
          WHEN "1010" => segments <= "0001000"; -- "A"
          WHEN "1011" => segments <= "1100000"; -- "b"
          WHEN "1100" => segments <= "0110001"; -- "C"
          WHEN "1101" => segments <= "1000010"; -- "d"
          WHEN "1110" => segments <= "0110000"; -- "E"
          WHEN "1111" => segments <= "0111000"; -- "F"
          WHEN OTHERS => segments <= "1111111"; -- null
       END CASE;

    END PROCESS;

END Behavioral;
```

elements.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE elements IS

    --selection cell
    --selection_cell(0) - color code
    --selection_cell(1) - display x coordinate
    --selection_cell(2) - display y coordinate
    --selection_cell(3) - x width
    --selection_cell(4) - y height
    --selection_cell(5) - selection x coordinate
    --selection_cell(6) - selection y coordinate
    TYPE SELECTION_CELL IS ARRAY (0 TO 6) OF INTEGER RANGE 0 TO 800;

    --cell array
    --cell(0) - color code
    --cell(1) - display x coordinate
    --cell(2) - display y coordinate
    --cell(3) - x width
    --cell(4) - y height
    TYPE CELL IS ARRAY (0 TO 4) OF INTEGER RANGE 0 TO 800;

    --int array for holding the color information of cells
        TYPE INT_ARRAY_2D IS ARRAY (INTEGER RANGE <>, INTEGER RANGE <>) OF INTEGER RANGE 0 TO 15;
```

```
--an array of colors to be used as a color scheme
TYPE COLORS IS ARRAY (0 TO 15) OF std_logic_vector (11 DOWNTO 0);


--an array of color schemes that can be used in the game
TYPE COLOR_SCHEMES IS ARRAY (INTEGER RANGE <>) OF COLORS;


--some basic colors
CONSTANT COLOR_NULL       : std_logic_vector (11 DOWNTO 0) := "000000000000";
CONSTANT COLOR_BLACK      : std_logic_vector (11 DOWNTO 0) := "000000000000";
CONSTANT COLOR_RED        : std_logic_vector (11 DOWNTO 0) := "111100000000";
CONSTANT COLOR_GREEN      : std_logic_vector (11 DOWNTO 0) := "000011110000";
CONSTANT COLOR_BLUE       : std_logic_vector (11 DOWNTO 0) := "000000001111";
CONSTANT COLOR_YELLOW     : std_logic_vector (11 DOWNTO 0) := "111111110000";
CONSTANT COLOR_CYAN       : std_logic_vector (11 DOWNTO 0) := "000011111111";
CONSTANT COLOR_MAGENTA    : std_logic_vector (11 DOWNTO 0) := "111100001111";
CONSTANT COLOR_WHITE      : std_logic_vector (11 DOWNTO 0) := "111111111111";
CONSTANT COLOR_GRAY       : std_logic_vector (11 DOWNTO 0) := "011101110111";

CONSTANT COLOR_DARK_GREEN : std_logic_vector (11 DOWNTO 0) := "000001110000";
CONSTANT COLOR_OLIVE      : std_logic_vector (11 DOWNTO 0) := "100010000000";

CONSTANT COLOR_MAROON     : std_logic_vector (11 DOWNTO 0) := "100000000000";
CONSTANT COLOR_DARK_GRAY  : std_logic_vector (11 DOWNTO 0) := "001000100010";

--a color set that can be used to determine the game colors
--other color sets were determined based on this one in particular
--###
```

```
--color_set legend
--color_set(0) : null (nonexistent element) - will not be referenced as rgb signal, insignificant
--color_set(1) : background color
--color_set(2) : board color
--color_set(3) : selection color
--color_set(4) : cell color 0
--color_set(5) : cell color 1
--color_set(6) : cell color 2
--color_set(7) : cell color 3
--color_set(8) : cell color 4
--color_set(9) : cell color 5
--color_set(10) : cell color 6
--color_set(11) : key cell color, empty
--color_set(12) : key cell color, half true
--color_set(13) : key cell color, full true
--color_set(14) : TBD
--color_set(15) : border color
CONSTANT color_set_0 : COLORS := (
   COLOR_NULL, --0
   COLOR_BLACK, --1
   COLOR_GRAY, --2
   COLOR_DARK_GREEN, --3
   COLOR_BLACK, --4
   COLOR_RED, --5
   COLOR_GREEN, --6
   COLOR_BLUE, --7
   COLOR_YELLOW, --8
```

```
    COLOR_CYAN, --9
    COLOR_MAGENTA, --10
    COLOR_BLACK, --11
    COLOR_WHITE, --12
    COLOR_RED, --13
    COLOR_RED, --14
    COLOR_RED --15
);

CONSTANT color_set_1 : COLORS := (
    COLOR_NULL, --0
    COLOR_BLACK, --1
    COLOR_MAROON, --2
    COLOR_OLIVE, --3
    COLOR_BLACK, --4
    COLOR_RED, --5
    COLOR_GREEN, --6
    COLOR_BLUE, --7
    COLOR_YELLOW, --8
    COLOR_CYAN, --9
    COLOR_MAGENTA, --10
    COLOR_BLACK, --11
    COLOR_WHITE, --12
    COLOR_RED, --13
    COLOR_RED, --14
    COLOR_RED --15
);
```

```
--color blind color set
CONSTANT color_set_2 : COLORS := (
   COLOR_NULL, --0
   COLOR_BLACK, --1
   COLOR_WHITE, --2
   COLOR_DARK_GREEN, --3
   "000000000000", --4
   "001000100010", --5
   "010001000100", --6
   "011001100110", --7
   "100010001000", --8
   "101010101010", --9
   "110011001100", --10
   COLOR_BLACK, --11
   COLOR_WHITE, --12
   COLOR_GRAY, --13
   COLOR_RED, --14
   COLOR_RED --15
);

SIGNAL color_set_3 : COLORS;
--constants for horizontal synchronization
CONSTANT VIS_H  : INTEGER := 640;
CONSTANT FP_H   : INTEGER := 16;
CONSTANT SP_H   : INTEGER := 96;
CONSTANT BP_H   : INTEGER := 48;
CONSTANT TOT_H  : INTEGER := VIS_H + FP_H + SP_H + BP_H; --800
CONSTANT NULL_H : INTEGER := FP_H + SP_H + BP_H; --horizontal black rgb area


--constants for vertical synchronization
CONSTANT VIS_V  : INTEGER := 480;
CONSTANT FP_V   : INTEGER := 10;
CONSTANT SP_V   : INTEGER := 2;
CONSTANT BP_V   : INTEGER := 33;
CONSTANT TOT_V  : INTEGER := VIS_V + FP_V + SP_V + BP_V; --525
CONSTANT NULL_V : INTEGER := FP_V + SP_V + BP_V; --vertical black rgb area


--game mechanics constants
CONSTANT TOT_GUESS_NUMBER   : INTEGER := 12;
CONSTANT TOT_COLOR_NUMBER   : INTEGER := 6;


CONSTANT COLOR_INDEX_SHIFT  : INTEGER := 4;
CONSTANT KEYPIN_INDEX_SHIFT : INTEGER := COLOR_INDEX_SHIFT + TOT_COLOR_NUMBER + 1;


--game visuals constants
CONSTANT SCREEN_BORDER          : INTEGER := 40;


CONSTANT BOARD_V                : INTEGER := 340;
CONSTANT BOARD_H                : INTEGER := 560;
CONSTANT BOARD_BORDER           : INTEGER := 20;


CONSTANT COLOR_PANEL_H          : INTEGER := 520;
CONSTANT COLOR_PANEL_V          : INTEGER := 80;
```

```
    CONSTANT COLOR_SELECTION_SIDE   : INTEGER := 60;
    CONSTANT COLOR_SELECTION_BORDER : INTEGER := 10;

    CONSTANT GUESS_PANEL_H          : INTEGER := 520;
    CONSTANT GUESS_PANEL_V          : INTEGER := 200;

    CONSTANT CELL_SIDE              : INTEGER := 40;
    CONSTANT CELL_BORDER            : INTEGER := 5;

    CONSTANT ELEMENT_SIDE           : INTEGER := 30;
    CONSTANT KEYPIN_SIDE            : INTEGER := 10;

    CONSTANT KEYPIN_BORDER          : INTEGER := 10;

    CONSTANT SELECTION_H            : INTEGER := 40;
    CONSTANT SELECTION_V            : INTEGER := 200;

END elements;

PACKAGE BODY elements IS
END PACKAGE BODY;
```

Constraints.xdc

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]


## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]


#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]

##Buttons
set_property PACKAGE_PIN U18 [get_ports btnC]
	set_property IOSTANDARD LVCMOS33 [get_ports btnC]
set_property PACKAGE_PIN T18 [get_ports btnU]
	set_property IOSTANDARD LVCMOS33 [get_ports btnU]
set_property PACKAGE_PIN W19 [get_ports btnL]
	set_property IOSTANDARD LVCMOS33 [get_ports btnL]
set_property PACKAGE_PIN T17 [get_ports btnR]
	set_property IOSTANDARD LVCMOS33 [get_ports btnR]
set_property PACKAGE_PIN U17 [get_ports btnD]
	set_property IOSTANDARD LVCMOS33 [get_ports btnD]

##VGA Connector
set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
set_property PACKAGE_PIN P19 [get_ports Hsync]
        set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
set_property PACKAGE_PIN R19 [get_ports Vsync]
        set_property IOSTANDARD LVCMOS33 [get_ports Vsync]
```

RTL Schematic

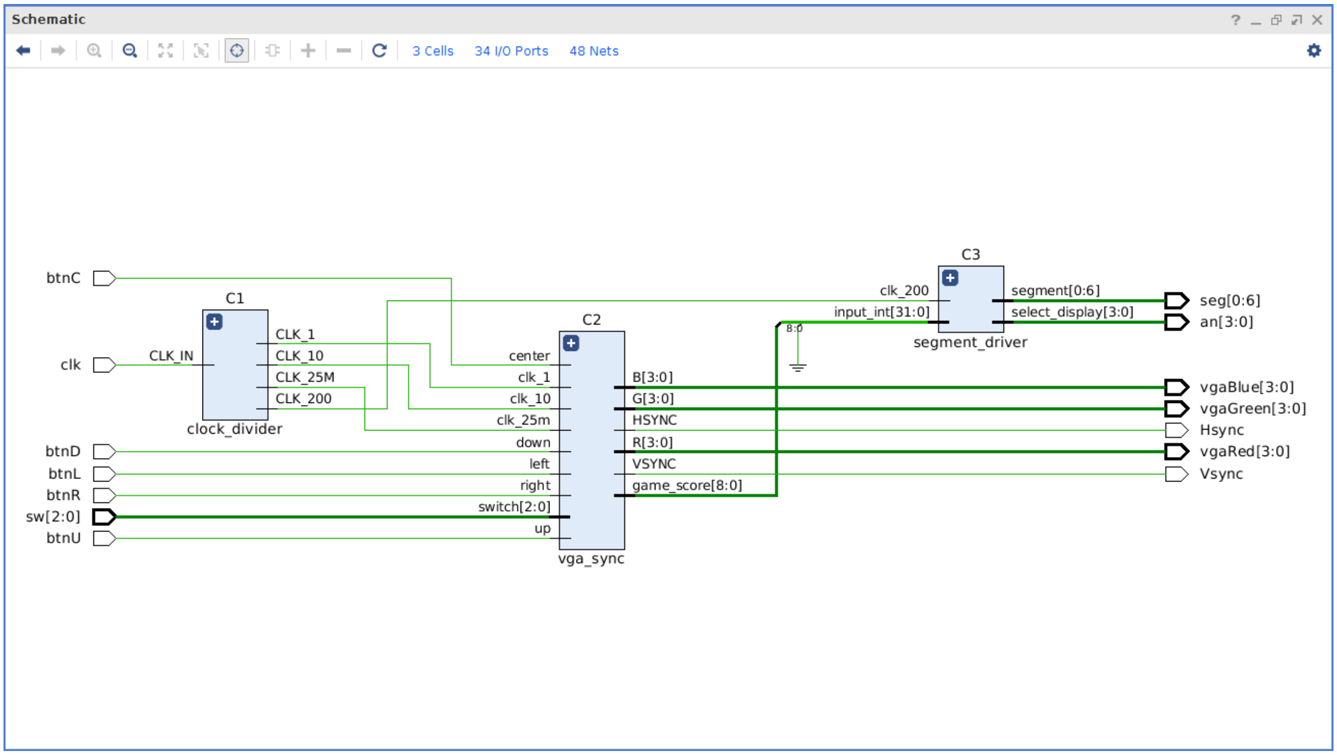

Illustration 9: The RTL schematic of the project. Segment decoder module is inside the segment driver module.

References

Digilent Basys3TM FPGA Board Reference Manual, Revised August 12, 2014.