Term Project Final Phase Report
Simon Says Game with VGA
Hasan Emre Erdemoğlu | 21401462 | EEE 102 – 2
December 15th 2015, Tuesday

YouTube Video Link

https://www.youtube.com/watch?v=Cz_4qf8CKN4

Abstract

This project serves to many purposes. The main purpose of this project was to integrate the syllabus of EEE 102 Introduction to Digital Circuit Design with a large-scale circuit design. The project consists of a combination of combinatorial and sequential circuits. It uses many finite state machines and memory to receive and display data. Additionally pulse width modulation is used to generate auditory sounds from digital signals to give the user both visual and auditory feedback. The most challenging part of the project is that the comparator circuitry within the project is also a finite state machine that takes its inputs from two other FSMs in order to create the next state logic. Therefore functions of many circuitries within the project have the risk to create output conflicts or result with an error state if not handled properly. The project uses top down design. The design specification plan and the design methodology will discuss the project in detail.

Design Specification Plan

In this project an initial problem was given. The problem was to implement a memory game circuit that generates a sequence and request user to remember and enter the sequence in order as shown by the machine. The feedback would be supplied to the player by using a buzzer and a VGA output.

This initial problem is implemented on a game that is called as Simon Says. The game consists of four colors (Red, Blue, Green, Yellow). The machine randomly creates a sequence of colors and displays a portion of the sequence in each level. For each level the sequence should increase, therefore the user is more susceptible to make mistakes.

A device that resolves this problem should have 4 buttons that represent the colors available (2 bits for 1 color). The device should have one finite state machine that deals with the user interaction with BASYS2 and should update the visual and auditory feedback. The device should also have a finite state machine that deals with the calculation of a pseudo-random sequence and comparison of this pseudo-random number with user input. These two state machines should interact in such a way that for each level the outputs, the comparison of the sequences should change. The device should have a VGA and a buzzer driver that controls the outputs. The device should also have components that control the outputs for each possible state of the device.

A top-down design may make things easier since it consists of building a whole system from individual blocks. If the individual blocks work as expected, the basic behavioral and structural modeling of the general circuit can be expressed easily in VHDL allowing the designer to design and debug the hardware description more easily.

Design Methodology

The following RTL schematic consists of the top-module and the modules within this top-module. There is no structural coding between the individual modules, all modules are designed in a way that they function as a whole when properly connected. The modules within the top module also have inner modules, mainly the finite state machines that create specific functions for the given module.

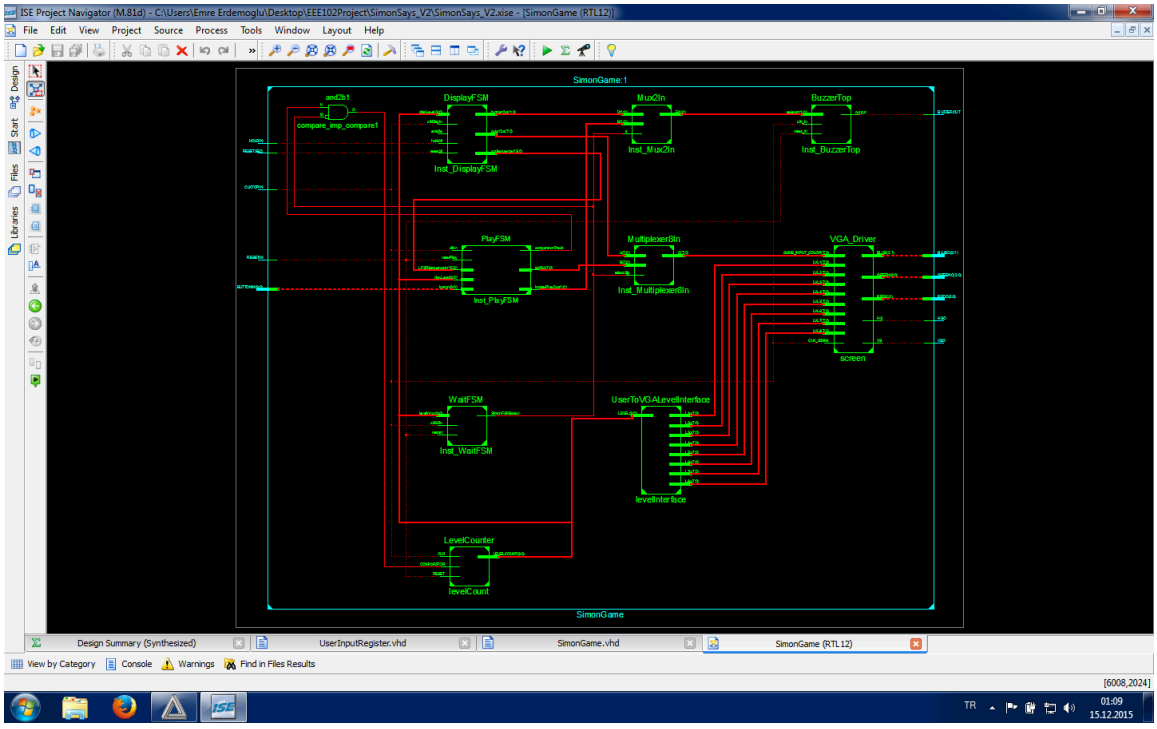

Figure 1. The top module for the game.

The schematic that is shown above consists of a single module with 10 compartments, which expand to 20 individual compartments. Details of these can be found within the VHDL code of the project within the appendices. The main modules are the VGA driver, the pulse width modulation driver, the level counter ROM, the linear feedback shift register for pseudo-random sequence generation, shift register for comparison mechanism, comparator to provide next state logic, the user input register for button inputs and the wait FSM that switches between two individual FSM of the circuit that decides whether the computer or the user to play the game.

The game proceeds as follows:

When the game starts, the game waits for a brief moment of time for user to prepare, the player toggles a switch to hold a value from linear feedback shift register. Then after another brief moment of time, the game shows a sequence. After this sequence the game display state toggles off and the user have to put input to the system within a certain amount of time. Failing at remembering the sequence or passing the time limit causes the game to end. Circuit wise, there is a counter within the comparator circuitry, the comparison returns false if a mistake has been made or the time limit has been reached. In such cases game goes to game over state and stays there until a restart. The level counter keeps

the states for the levels using the comparison output of the play finite state machine. The level counter was used, as a backbone to determine which portion of the sequences should be matches and how much button presses that the user can make. The level counter also is a ROM for the VGA output. It manages the colors that the level section of the VGA should have for each level. When the state of the circuit changes the level output changes. This change is independent from display and play machines of the circuit. Since only one buzzer and VGA exists in the design, the play and display machines share the same VGA and buzzer. A wait state machine written to switch back and forth between these machines automatically, when a certain amount of time has passed (either user time limit, or display timing).

Results

At phase two, the circuit logic was ready and was tested using test benches. A unit testing was used, as well as the entire circuitry. I was able to generate correct waveforms for play and display machines individually however I encountered an output conflict whenever I merged those two machines together. I have seen that in the initial design I have missed the wait/switch state between play and display states. Since both states share the same output devices that are the VGA and the buzzer the output conflict caused buzzer to burn and the VGA produced an output that changes if any object that can function as an antenna was moved close to the FPGA. Therefore after phase two the entire design has been changed, the VGA was modified in such way that gets input from outside ROM for each state of the device whereas in the old design the possible color patterns were embedded within the driver. I have added some interfaces for the buzzer and the VGA for output validation and several multiplexers to pick between the play and display state. Additionally I have created a wait state machine that chooses between the play and the display states.

In the final design I have used the components that I have written before. Initially the system worked fine. After the test benches I have combined the system together. I have resolved the output conflict problem however the wait finite state machine was not able to switch between the states as the game starts. This is due to the initial condition of the comparator. Since comparator returns the std_logic signal ‘0’ after the display state, the game state directly goes to game over state ending the game. In such case, the wait state machine is never triggered, therefore the game is unable to change between the states, and the user is unable to add any sequence for the computer to compare.

Conclusion

This was the hardest assignment that I have done in EEE 102 course. I have encountered many errors; I had to change my design several times to achieve better results however at the end, I was unable to produce a totally working design. The hardest part of the project was the amount of components that the project has within it. Even though the components worked fine individually, within a complex system, the combination of components did not efficiently produced the outcomes that I have demanded. Since many FSMs lapsed with each other I also had difficulty designing the circuit, writing the state diagrams

and Karnaugh Maps. Many mistakes that I did with this project was because of this problem.
This project forced me to use nearly every single content that the course have provided to me. This application helped me to evaluate myself, to work on my weaknesses and appreciate my strengths. The project changed my thinking. At the beginning of the semester I was looking VHDL as I was looking to Java, which created many problems since VHDL is not a programming language, therefore most of the content do not work the same. Currently I am able to create many sequences within a module to make the designed circuit do a specific work, I can add many modules together to alter the logical flow of a circuit and I can trace and read RTL schematics. These skills are very practical and I can use them in my further courses.
For the errors that I have encountered now, I have to reorganize the circuitry described and alter some of the inputs, outputs and some of the processes of the modules in order to resolve them. Since the system consists many components the entire framework of the design has to be changed which requires time and extra planning. The problems within this final form of the described circuit are discussed above.

References

- Template Resource for VGA: http://blog.tkjelectronics.dk/2011/03/generating-a-vga-signal-with-an-fpga/
- VGA Support: http://www.edaboard.com/thread119404.html
- VGA Controller: https://eewiki.net/pages/viewpage.action?pageId=15925278
- VHDL Support: https://www.ics.uci.edu/~jmoorkan/vhdlref/vhdl.html#wait
- Linear Feedback Shift Registers: http://www-math.ucdenver.edu/~wcherowi/courses/m5410/m5410fsr.html
- Brown, Stephen D., and Zvonko G. Vranesic. Fundamentals of Digital Logic with VHDL Design. Boston: McGraw-Hill, 2000. Print.
- Wakerly, John F. Digital Design: Principles and Practices. Upper Saddle River, N.J.: Prentice Hall, 1990. Print.
- Digilent. “Digilent BASYS2 Board Reference Manual.” Pullman: Digilent, 11 November 2011. <https://www.digilentinc.com/Data/Products/BASYS2/Basys2_rm.pdf>

Appendices

The following appendices have the VHDL codes, circuit schematics, and the pictures of the device with the outputs, some waveforms for the circuits that are described using the codes provided in this report. Additional information can be provided in or after the demo session.

VHDL Codes:

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

entity SimonGame is
    port ( CLKTOPIN          : IN STD_LOGIC;
           RESETIN,RESETSEQ, HOLDIN          : IN STD_LOGIC;
           BUTTONIN                          : IN
STD_LOGIC_VECTOR(3 DOWNTO 0);
           VSO, HSO                          : OUT
STD_LOGIC;
           REDO,GREENO                       : OUT
STD_LOGIC_VECTOR(2 DOWNTO 0);
           BLUEO                                 : OUT
STD_LOGIC_VECTOR(2 DOWNTO 1);
           BUZZEROUT                 : OUT STD_LOGIC); --
shared by both user input and sequence reader.

                                  -- use mux to choose between wait and play
states.
end SimonGame;

architecture Behavioral of SimonGame is

    -- COMPONENT DECLERATIONS

    COMPONENT DisplayFSM
    PORT(
        dispLevel : IN std_logic_vector(3 downto 0);
        clkDispIn : IN std_logic;
        resetAll : IN std_logic;
        holdAll : IN std_logic;
        outSequence : OUT std_logic_vector(15 downto 0);
        colorOut : OUT std_logic_vector(7 downto 0);
        buzzerOut : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;

    component VGA_Driver is
    port ( CLK_50MHz: in std_logic;
  VS: out std_logic;
           HS: out std_logic;
           RED: out std_logic_vector(2 downto 0);
           GREEN: out std_logic_vector(2 downto 0);
           BLUE: out std_logic_vector(2 downto 1);
```

```
                GAME_INPUT_COLOR: IN STD_LOGIC_VECTOR (7 DOWNTO
0);
                LVL1,LVL2,LVL3,LVL4,LVL5,LVL6,LVL7,LVL8 : IN
STD_LOGIC_VECTOR(7 DOWNTO 0));
    end component;


    component LevelCounter is
        port ( CLK                  : IN STD_LOGIC;
                LEVELCOUNT          :          OUT
STD_LOGIC_VECTOR(3 DOWNTO 0);
                COMPARATOR  : IN STD_LOGIC;
                RESET               : IN STD_LOGIC);
    end component;

    component   UserToVGALevelInterface is
         Port ( LEVEL : in  STD_LOGIC_VECTOR (3 downto 0);
                L1o : out  STD_LOGIC_VECTOR (7 downto 0);
                L2o : out  STD_LOGIC_VECTOR (7 downto 0);
                L3o : out  STD_LOGIC_VECTOR (7 downto 0);
                L4o : out  STD_LOGIC_VECTOR (7 downto 0);
                L5o : out  STD_LOGIC_VECTOR (7 downto 0);
                L6o : out  STD_LOGIC_VECTOR (7 downto 0);
                L7o : out  STD_LOGIC_VECTOR (7 downto 0);
                L8o : out  STD_LOGIC_VECTOR (7 downto 0));
    end component;

    COMPONENT BuzzerTop
    PORT(
        clk_in : IN std_logic;
        reset_in : IN std_logic;
        selector : IN std_logic_vector(1 downto 0);
        output : OUT std_logic
        );
    END COMPONENT;

    COMPONENT PlayFSM
    PORT(
        LFSRSequenceIn : IN std_logic_vector(15 downto 0);
        playLevel : IN std_logic_vector(3 downto 0);
        button : IN std_logic_vector(3 downto 0);
        clkIn : IN std_logic;
        resetPlay : IN std_logic;
        toVGA : OUT std_logic_vector(7 downto 0);
        comparatorCheck : OUT std_logic;
        buzzerPlayOut : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;
```

```
	COMPONENT Multiplexer8In
	PORT(
		A : IN std_logic_vector(7 downto 0);
		B : IN std_logic_vector(7 downto 0);
		selectSig : IN std_logic;
		C : OUT std_logic_vector(7 downto 0)
		);
	END COMPONENT;

	COMPONENT Mux2In
	PORT(
		D : IN std_logic_vector(1 downto 0);
		E : IN std_logic_vector(1 downto 0);
		s : IN std_logic;
		F : OUT std_logic_vector(1 downto 0)
		);
	END COMPONENT;


	COMPONENT WaitFSM
	PORT(
		clk50In : IN std_logic;
		levelInput : IN std_logic_vector(3 downto 0);
		restart : IN std_logic;
		SimonFSMSelect : OUT std_logic
		);
	END COMPONENT;

	-- SIGNAL DECLERATIONS
	signal inputCol : std_logic_vector(7 downto 0); -- input color both from display and play FSM.
	signal l1,l2,l3,l4,l5,l6,l7,l8 : std_logic_vector(7 downto 0);
--	signal currLevel : std_logic_vector (3 downto 0);
	signal counted : std_logic_vector (3 downto 0);
	signal comparatorTrue,compare : std_logic;
	signal buzzerSelector : std_logic_vector(1 downto 0);
	signal lfsrGenerated : std_logic_vector(15 downto 0);
	signal displayPaint,playPaint : std_logic_vector(7 downto 0);
	signal displayBuzzer,playBuzzer : std_logic_vector(1 downto 0);
	signal waitFSMsel : std_logic;


begin

	-- SIGNAL INSTANTIATIONS
	-- COMPONENT INSTANTIATIONS
```

```
        screen : VGA_Driver
                port map ( CLK_50MHz => CLKTOPIN,
                                        VS => VSO,
                                        HS => HSO,
                                        RED => REDO,
                                        GREEN => GREENO,
                                        BLUE => BLUEO,
                                        GAME_INPUT_COLOR => inputCol,
                                        LVL1 => l1,
                                        LVL2 => l2,
                                        LVL3 => l3,
                                        LVL4 => l4,
                                        LVL5 => l5,
                                        LVL6 => l6,
                                        LVL7 => l7,
                                        LVL8 => l8);

        levelInterface : UserToVGALevelInterface
                port map ( LEVEL => counted,
                                        L1o => l1,
                                        L2o => l2,
                                        L3o => l3,
                                        L4o => l4,
                                        L5o => l5,
                                        L6o => l6,
                                        L7o => l7,
                                        L8o => l8);

compare <= comparatorTrue AND ( NOT waitFSMsel);


        levelCount : levelCounter
                port map ( CLK => CLKTOPIN,
                                        LEVELCOUNT => counted,
                                        comparator => compare,
                                        reset => RESETIN);

        Inst_BuzzerTop: BuzzerTop PORT MAP(
                clk_in =>  CLKTOPIN,
                reset_in => RESETIN,
                selector => buzzerSelector,
                output => BUZZEROUT);

        Inst_DisplayFSM: DisplayFSM PORT MAP(
                dispLevel => counted,
                clkDispIn => CLKTOPIN,
                outSequence => lfsrGenerated,
                colorOut => displayPaint,
```

```
            buzzerOut => displayBuzzer,
            resetAll => RESETSEQ,
            holdAll => HOLDIN);


    Inst_PlayFSM: PlayFSM PORT MAP(
            LFSRSequenceIn => lfsrGenerated,
            playLevel => counted,
            button => BUTTONIN,
            clkIn => CLKTOPIN,
            resetPlay => RESETIN,
            toVGA => playPaint,
            comparatorCheck => comparatorTrue,
            buzzerPlayOut => playBuzzer);

    Inst_Multiplexer8In: Multiplexer8In PORT MAP(
            A => displayPaint,
            B => playPaint,
            selectSig => waitFSMsel,
            C => inputCol);

    Inst_Mux2In: Mux2In PORT MAP(
            D => displayBuzzer,
            E => playBuzzer,
            s => waitFSMsel,
            F => buzzerSelector);


    Inst_WaitFSM: WaitFSM PORT MAP(
            clk50In => CLKTOPIN,
            levelInput => counted,
            restart => RESETIN,
            SimonFSMSelect => waitFSMsel);

end Behavioral;

----------------------------------------------------------------------------------
-- Engineer:
--
-- Create Date:    21:11:04 11/02/2015
-- Module Name:    LFSR_NumberGen - Behavioral

--Template Resource: http://blog.tkjelectronics.dk/2011/03/generating-a-vga-signal-with-an-fpga/
----------------------------------------------------------------------------------
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 VGA_Driver is
        port ( CLK_50MHz: in std_logic;
      VS: out std_logic;
                        HS: out std_logic;
                        RED: out std_logic_vector(2 downto 0);
                        GREEN: out std_logic_vector(2 downto 0);
                        BLUE: out std_logic_vector(2 downto 1);
                        GAME_INPUT_COLOR: IN STD_LOGIC_VECTOR (7 DOWNTO
0);
                        LVL1,LVL2,LVL3,LVL4,LVL5,LVL6,LVL7,LVL8 : IN
STD_LOGIC_VECTOR(7 DOWNTO 0) -- levels and colors for them.
 );
end VGA_Driver;

architecture Behavioral of VGA_Driver is
 -- VGA Definitions starts
 constant HDisplayArea: integer:= 640; -- horizontal display area
 constant HLimit: integer:= 800; -- maximum horizontal amount (limit)
 constant HFrontPorch: integer:= 16; -- h. front porch
 constant HBackPorch: integer:= 48;        -- h. back porch
 constant HSyncWidth: integer:= 96;        -- h. pulse width
 constant VDisplayArea: integer:= 480; -- vertical display area
 constant VLimit: integer:= 525; -- maximum vertical amount (limit)
 constant VFrontPorch: integer:= 10;       -- v. front porch
 constant VBackPorch: integer:= 33;        -- v. back porch
 constant VSyncWidth: integer:= 2;-- v. pulse width

 signal Clk_25MHz: std_logic := '0';
 signal HBlank, VBlank, Blank: std_logic := '0';

 signal CurrentHPos: std_logic_vector(10 downto 0) := (others => '0'); -- goes to
1100100000 = 800
 signal CurrentVPos: std_logic_vector(10 downto 0) := (others => '0'); -- goes to
1000001101 = 525
 signal ScanlineX, ScanlineY: std_logic_vector(10 downto 0);

 signal ColorOutput: std_logic_vector(7 downto 0);

 -- VGA Definitions end

begin

 Generate25MHz: process (CLK_50MHz)
 begin
```

```
  if rising_edge(CLK_50MHz) then
          Clk_25MHz <= not Clk_25MHz;
         end if;
 end process Generate25MHz;


 VGAPosition: process (Clk_25MHz)
 begin
  if rising_edge(Clk_25MHz) then -- 25.175 MHz need, 25MHz given. Creates
noise.
          if CurrentHPos < HLimit-1 then
                  CurrentHPos <= CurrentHPos + 1;
                 else
                  if CurrentVPos < VLimit-1 then
                   CurrentVPos <= CurrentVPos + 1;
                  else
                   CurrentVPos <= (others => '0'); -- reset Vertical Position
                  end if;
                  CurrentHPos <= (others => '0'); -- reset Horizontal Position
                 end if;
         end if;
 end process VGAPosition;

 -- Timing definition for HSync, VSync and Blank (http://tinyvga.com/vga-
timing/640x480@60Hz)
   HS <= '0' when CurrentHPos < HSyncWidth else
              '1';

         VS <= '0' when CurrentVPos < VSyncWidth else
             '1';

         HBlank <= '0' when (CurrentHPos >= HSyncWidth + HFrontPorch) and
         (CurrentHPos < HSyncWidth + HFrontPorch + HDisplayArea) else
               '1';

         VBlank <= '0' when (CurrentVPos >= VSyncWidth + VFrontPorch) and
         (CurrentVPos < VSyncWidth + VFrontPorch + VDisplayArea) else
               '1';

         Blank <= '1' when HBlank = '1' or VBlank = '1' else
               '0';

         ScanlineX <= CurrentHPos - HSyncWidth - HFrontPorch when Blank = '0'
else
                  (others => '0');

         ScanlineY <= CurrentVPos - VSyncWidth - VFrontPorch when Blank = '0'
else
```

```
                (others => '0');

    RED <= ColorOutput(7 downto 5) when Blank = '0' else
        "000";
    GREEN <= ColorOutput(4 downto 2) when Blank = '0' else
        "000";
    BLUE <= ColorOutput(1 downto 0) when Blank = '0' else
        "00";


 -- Print to VGA Display

        -- to divide this into 3 subcategories a mux will be needed. color output has the level print,
        -- the upGrid and downGrid to print.
        displayVGA: process (Clk_25mhz, ScanlineX, ScanlineY, LVL1, LVL2, LVL3, LVL4, LVL5, LVL6, LVL7, LVL8, GAME_INPUT_COLOR)
                begin
                        case Clk_25Mhz is
                                when '0' =>
                                        if ( ScanlineX >= 3 and ScanlineX < 75 and ScanlineY < 96 ) then
                                                ColorOutput <= LVL1;
                                        elsif (ScanlineX >= 80 and ScanlineX < 155 and ScanlineY < 96) then
                                                ColorOutput <= LVL2;
                                        elsif (ScanlineX >= 160 and ScanlineX < 235 and ScanlineY < 96) then
                                                ColorOutput <= LVL3;
                                        elsif (ScanlineX >= 240 and ScanlineX < 315 and ScanlineY < 96) then
                                                ColorOutput <= LVL4;
                                        elsif (ScanlineX >= 320 and ScanlineX < 395 and ScanlineY < 96) then
                                                ColorOutput <= LVL5;
                                        elsif (ScanlineX >= 400 and ScanlineX < 475 and ScanlineY < 96) then
                                                ColorOutput <= LVL6;
                                        elsif (ScanlineX >= 480 and ScanlineX < 555 and ScanlineY < 96) then
                                                ColorOutput <= LVL7;
                                        elsif (ScanlineX >= 560 and ScanlineX < 637 and ScanlineY < 96) then
                                                ColorOutput <= LVL8;
                                        else
                                                ColorOutput <= "00000000";
                                        end if;
```

```
                        when others =>
                                if ( ScanlineX > 3 and ScanlineX < 637 and ScanlineY
>=  101 and ScanlineY < 477) then
                                        ColorOutput <= GAME_INPUT_COLOR;
                                        else
                                        ColorOutput <= "00000000";
                                end if;
                        end case;

        end process;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity UserToVGALevelInterface is
   Port ( LEVEL : in  STD_LOGIC_VECTOR (3 downto 0);
        L1o : out  STD_LOGIC_VECTOR (7 downto 0);
                        L2o : out  STD_LOGIC_VECTOR (7 downto 0);
        L3o : out  STD_LOGIC_VECTOR (7 downto 0);
        L4o : out  STD_LOGIC_VECTOR (7 downto 0);
        L5o : out  STD_LOGIC_VECTOR (7 downto 0);
        L6o : out  STD_LOGIC_VECTOR (7 downto 0);
        L7o : out  STD_LOGIC_VECTOR (7 downto 0);
        L8o : out  STD_LOGIC_VECTOR (7 downto 0));
end UserToVGALevelInterface;

architecture Behavioral of UserToVGALevelInterface is -- A ROM TO HOLD
STATES OF OUTPUT FOR EACH LEVEL

signal L1,L2,L3,L4,L5,L6,L7,L8 : STD_LOGIC_VECTOR (7 downto 0);

begin

        L1o <= L1;
        L2o <= L2;
        L3o <= L3;
        L4o <= L4;
        L5o <= L5;
        L6o <= L6;
        L7o <= L7;
        L8o <= L8;

        process (LEVEL,L1,L2,L3,L4,L5,L6,L7,L8) begin
                if LEVEL = "0000"  then -- LEVEL 1
                        L1 <= "11111100";
```

```
            L2 <= "00000000";
            L3 <= "00000000";
            L4 <= "00000000";
            L5 <= "00000000";
            L6 <= "00000000";
            L7 <= "00000000";
            L8 <= "00000000";
        elsif LEVEL = "0001"  then -- LEVEL 2
            L1 <= "00011100";
            L2 <= "11111100";
            L3 <= "00000000";
            L4 <= "00000000";
            L5 <= "00000000";
            L6 <= "00000000";
            L7 <= "00000000";
            L8 <= "00000000";
        elsif LEVEL = "0010"  then -- LEVEL 3
            L1 <= "00011100";
            L2 <= "00011100";
            L3 <= "11111100";
            L4 <= "00000000";
            L5 <= "00000000";
            L6 <= "00000000";
            L7 <= "00000000";
            L8 <= "00000000";
        elsif LEVEL = "0011"  then -- LEVEL 4
            L1 <= "00011100";
            L2 <= "00011100";
            L3 <= "00011100";
            L4 <= "11111100";
            L5 <= "00000000";
            L6 <= "00000000";
            L7 <= "00000000";
            L8 <= "00000000";
        elsif LEVEL = "0100"  then -- LEVEL 5
            L1 <= "00011100";
            L2 <= "00011100";
            L3 <= "00011100";
            L4 <= "00011100";
            L5 <= "11111100";
            L6 <= "00000000";
            L7 <= "00000000";
            L8 <= "00000000";
        elsif LEVEL = "0101"  then -- LEVEL 6
            L1 <= "00011100";
            L2 <= "00011100";
            L3 <= "00011100";
            L4 <= "00011100";
```

```
                L5 <= "00011100";
                L6 <= "11111100";
                L7 <= "00000000";
                L8 <= "00000000";
            elsif LEVEL = "0110"  then -- LEVEL 7
                L1 <= "00011100";
                L2 <= "00011100";
                L3 <= "00011100";
                L4 <= "00011100";
                L5 <= "00011100";
                L6 <= "00011100";
                L7 <= "11111100";
                L8 <= "00000000";
            elsif LEVEL = "0111"  then -- LEVEL 8
                L1 <= "00011100";
                L2 <= "00011100";
                L3 <= "00011100";
                L4 <= "00011100";
                L5 <= "00011100";
                L6 <= "00011100";
                L7 <= "00011100";
                L8 <= "11111100";
            elsif LEVEL = "1000" then -- WIN STATE
                L1 <= "00011100";
                L2 <= "00011100";
                L3 <= "00011100";
                L4 <= "00011100";
                L5 <= "00011100";
                L6 <= "00011100";
                L7 <= "00011100";
                L8 <= "00011100";
            else                                                    -- LOSE STATE
                L1 <= "11100000";
                L2 <= "11100000";
                L3 <= "11100000";
                L4 <= "11100000";
                L5 <= "11100000";
                L6 <= "11100000";
                L7 <= "11100000";
                L8 <= "11100000";
            end if;
        end process;
end Behavioral;

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

```
entity LevelCounter is
	port ( CLK 			: IN STD_LOGIC;
			LEVELCOUNT 		: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
			COMPARATOR 	: IN STD_LOGIC;
			RESET 			: IN STD_LOGIC);

end LevelCounter;

architecture Behavioral of LevelCounter is

signal Q : std_logic_vector(3 downto 0);

begin

	LEVELCOUNT <= Q;

process (CLK, RESET, COMPARATOR) begin
	if RESET = '1' then
		Q <= "0000"; --level 1
	elsif rising_edge(CLK) then
		if COMPARATOR = '1' then -- written all cases, since some cases lead to win lose error cases.
			if Q = "0000" then
				Q <= "0001";
			elsif Q = "0001" then
				Q <= "0010";
			elsif Q = "0010" then
				Q <= "0011";
			elsif Q = "0011" then
				Q <= "0100";
			elsif Q = "0100" then
				Q <= "0101";
			elsif Q = "0101" then
				Q <= "0110";
			elsif Q = "0110" then
				Q <= "0111";
			elsif Q = "0111" then
				Q <= "1000"; -- WIN STATE
			else
				Q <= "0000"; -- RESET
			end if;
		else
			Q <= "1001"; -- LOSE STATE
		end if;
	end if;
end process;
```

```
end Behavioral;


----------------------------------------------------------------------------------
-- Engineer:
--
-- Create Date:    21:11:04 11/02/2015
-- Module Name:    LFSR_NumberGen - Behavioral
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity BuzzerTop is
	port( clk_in, reset_in: in std_logic;
			selector: in std_logic_vector(1 downto 0);
			output: out std_logic);
end BuzzerTop;

architecture Behavioral of BuzzerTop is

component Buzzer1
	port ( CLK, RESET : IN STD_LOGIC;
			PWM_OUT: OUT STD_LOGIC);
end component;

component Buzzer2
	port ( CLK, RESET : IN STD_LOGIC;
			PWM_OUT: OUT STD_LOGIC);
end component;

component Buzzer3
	port ( CLK, RESET : IN STD_LOGIC;
			PWM_OUT: OUT STD_LOGIC);
end component;

component Buzzer4
	port ( CLK, RESET : IN STD_LOGIC;
			PWM_OUT: OUT STD_LOGIC);
end component;


signal temp: std_logic_vector(3 downto 0);
begin

pwm11: component Buzzer1
	port map ( CLK => clk_in,
```

```
                RESET => reset_in,
                PWM_OUT => temp(0) );

pwm22: component Buzzer2
        port map ( CLK => clk_in,
                RESET => reset_in,
                PWM_OUT => temp(1) );

pwm33: component Buzzer3
        port map ( CLK => clk_in,
                RESET => reset_in,
                PWM_OUT => temp(2) );

pwm44: component Buzzer4
        port map ( CLK => clk_in,
                RESET => reset_in,
                PWM_OUT => temp(3) );


process (selector, temp)
        begin
        case selector is
                when "00" => output <= temp(0);
                when "01" => output <= temp(1);
                when "10" => output <= temp(2);
                when others => output <= temp(3);
        end case;
end process;
end Behavioral;


----------------------------------------------------------------------------------
-- Engineer:
--
-- Create Date:    21:11:04 11/02/2015
-- Module Name:    LFSR_NumberGen - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity Buzzer1 is
```

```
        port ( CLK, RESET : IN STD_LOGIC; -- BASYS2 has 50MHz internal clock, constants fixed accordingly.
                        PWM_OUT: OUT STD_LOGIC);
end Buzzer1;

architecture Behavioral of Buzzer1 is
signal c1: integer;
signal pwm: std_logic;

begin

process (CLK, RESET)
        begin
                if RESET = '1' then
                        c1 <= 0;
                        pwm <= '0';
                elsif rising_edge(CLK) then
                        if c1 = 500000 then
                                c1 <= 1;
                                pwm <= not pwm;
                        else
                                c1 <= c1 + 1;
                        end if;
                end if;
        end process;

pwm_out <= pwm;
end Behavioral;


----------------------------------------------------------------------------------
-- Engineer:
--
-- Create Date:    21:11:04 11/02/2015
-- Module Name:    LFSR_NumberGen - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity Buzzer2 is
        port ( CLK, RESET : IN STD_LOGIC; -- BASYS2 has 50MHz internal clock, constants fixed accordingly.
```

```
                        PWM_OUT: OUT STD_LOGIC);
end Buzzer2;

architecture Behavioral of Buzzer2 is
signal c1: integer;
signal pwm: std_logic;

begin

process (CLK, RESET)
        begin
                if RESET = '1' then
                        c1 <= 0;
                        pwm <= '0';
                elsif rising_edge(CLK) then
                        if c1 = 500000 then
                                c1 <= 1;
                                pwm <= not pwm;
                        else
                                c1 <= c1 + 1;
                        end if;
                end if;
        end process;

pwm_out <= pwm;
end Behavioral;


----------------------------------------------------------------------------------
-- Engineer:
--
-- Create Date:    21:11:04 11/02/2015
-- Module Name:    LFSR_NumberGen - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity Buzzer3 is
        port ( CLK, RESET : IN STD_LOGIC; -- BASYS2 has 50MHz internal clock,
constants fixed accordingly.
                        PWM_OUT: OUT STD_LOGIC);
end Buzzer3;
```

```
architecture Behavioral of Buzzer3 is
signal c1: integer;
signal pwm: std_logic;

begin

process (CLK, RESET)
	begin
		if RESET = '1' then
			c1 <= 0;
			pwm <= '0';
		elsif rising_edge(CLK) then
			if c1 = 500000 then
				c1 <= 1;
				pwm <= not pwm;
			else
				c1 <= c1 + 1;
			end if;
		end if;
	end process;

pwm_out <= pwm;
end Behavioral;


----------------------------------------------------------------------------------
-- Engineer:
--
-- Create Date:    21:11:04 11/02/2015
-- Module Name:    LFSR_NumberGen - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity Buzzer4 is
	port ( CLK, RESET : IN STD_LOGIC; -- BASYS2 has 50MHz internal clock,
constants fixed accordingly.
			 PWM_OUT: OUT STD_LOGIC);
end Buzzer4;

architecture Behavioral of Buzzer4 is
```

```
signal c1: integer;
signal pwm: std_logic;

begin

process (CLK, RESET)
	begin
		if RESET = '1' then
			c1 <= 0;
			pwm <= '0';
		elsif rising_edge(CLK) then
			if c1 = 500000 then
				c1 <= 1;
				pwm <= not pwm;
			else
				c1 <= c1 + 1;
			end if;
		end if;
	end process;

pwm_out <= pwm;
end Behavioral;

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 DisplayFSM is
	port (  enable    : in std_logic;
				dispLevel : in std_logic_vector( 3 downto 0); -- comes
from level counter.
				clkDispIn : in std_logic; -- 50 mhz clk
				outSequence : out std_logic_vector( 15 downto 0); --
comes from LFSR
				colorOut: out std_logic_vector(7 downto 0); -- paint
the output VGA
				buzzerOut: out std_logic_vector(1 downto 0);
				resetAll, holdAll : in std_logic); -- for LFSR
end DisplayFSM;

architecture Behavioral of DisplayFSM is

--component declerations
component RandomSequenceGenerator is
	port ( CLK, RESET, HOLD: IN STD_LOGIC;
```

```
                    OUTPUT : out STD_LOGIC_VECTOR (15 downto 0)); -- 16
bit number will be created.
end component;

component ClockDivider is
   Port ( clkIn : in  STD_LOGIC;
          clkOut1Hz : out  STD_LOGIC);
end component;

COMPONENT ShiftRegister
        PORT(
                input : IN std_logic;
                regclock : IN std_logic;
                output : out std_logic_vector(1 downto 0)
                );
        END COMPONENT;


--signal declerations
signal sequence : std_logic_vector (15 downto 0);
signal seqPortion : std_logic_vector(1 downto 0);
signal color : std_logic_vector (7 downto 0);
signal buzzer: std_logic_vector(1 downto 0);
signal clkclk : std_logic;

type state is (idle, active);
signal currentstate : state;

begin
        --component instantiation
        sequenceDisp : RandomSequenceGenerator
                port map (              CLK => clkDispIn,
                                                RESET => resetAll,
                                                HOLD => holdAll,
                                                OUTPUT => sequence);

        Inst_ClockDivider: ClockDivider
                PORT MAP(
                        clkIn => clkDispIn,
                        clkOut1Hz => clkclk);

        -- shift register to shift and feed color system
        -- shifts every clock cycle, that is 1 Hz.
        Inst_ShiftRegister: ShiftRegister PORT MAP(
                        input => sequence(0),
                        regclock => clkclk,
                        output => seqPortion);
```

```
-- signal instantiations
colorOut <= color;
buzzerOut <= buzzer;
outSequence <= sequence;

--processes
-- storage that holds the color patterns for each sequence portion.
colorPanel : process (seqPortion) begin
	if currentState = active then
		case seqPortion is
			when "00" => color <= "11100000"; --red
			when "01" => color <= "00011100"; --green
			when "10" => color <= "00000011"; -- blue
			when others => color <= "11111100"; -- yellow
		end case;
	else
		color <= "00000000";
	end if;
end process;

-- sends buzzer selector signal
buzzerPanel : process (seqPortion) begin
	if currentState = active then
		case seqPortion is
			when "00" => buzzer <= "00"; --red
			when "01" => buzzer <= "01"; --green
			when "10" => buzzer <= "10"; -- blue
			when others => buzzer <= "11"; -- yellow
		end case;
	else
		buzzer <= "XX";
	end if;
end process;


-- FSM process
FSM : process ( resetAll, enable, clkclk ) begin
	if resetAll = '1' then
		currentstate <= active;

	elsif rising_edge(clkclk) then
		if currentstate <= idle then
--			sequence <= "0000000000000000";
			if enable <= '1' then
				currentstate <= active;
			end if;
		end if;
	end if;
```

```
	end process;

-- counter process

end Behavioral;

----------------------------------------------------------------------------------
-- Engineer:
--
-- Create Date:    21:11:04 11/02/2015
-- Module Name:    LFSR_NumberGen - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity RandomSequenceGenerator is
	port ( CLK, RESET, HOLD: IN STD_LOGIC;
			output : OUT STD_LOGIC_VECTOR (15 downto 0)); -- 16 bit
number will be created.
end RandomSequenceGenerator;

architecture Behavioral of RandomSequenceGenerator is

signal newVector,curVector,selected: std_logic_vector ( 15 downto 0) :=
"0000000000000000";
constant key: std_logic_vector ( 15 downto 0) := "0010110101001011";

-- these signals will be used to hold and replace states as needed.

--  Characteristic Polynomial (decided randomly): P(x) = x15+ x^12 + x^5 + x + 1
--  defines XOR positions for the linear feedback. determines next output (1 bit -
shift)
--  initial key (decided randomly) : 0010110101001011 (16 bits)

begin

generateLFSR: process (CLK, RESET)
	begin
		if rising_edge (CLK) then
			if RESET = '1' then -- do a reset if required. then the initial
key is logged in.
				curVector <= key;
			else
				curVector <= newVector;
```

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

update: process(curVector) -- use polynomial xor statement, update first bit accordingly. shift other entries.
        begin
                        newVector(15 downto 1) <= curVector (14 downto 0); -- copy shift elements to new vector.
                        newVector(0) <= ((((curVector (0) XOR curVector (1)) XOR curVector (5)) XOR curVector (12) XOR curVector(15)));
                        -- use xor statement given in polynomial.
end process;-- set to output

selectOutput: process( CLK, HOLD, curVector)
        begin
                if rising_edge(CLK) then
                if HOLD = '1' then
                        selected <= curVector;
--              else
--                      selected <= "0000000000000000"; -- Latches the value, required.
                end if;
                end if;
        end process;
OUTPUT <= selected;

end Behavioral;


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

entity ClockDivider is
   Port ( clkIn : in  STD_LOGIC;
          clkOut1Hz : out  STD_LOGIC);
end ClockDivider;

architecture Behavioral of ClockDivider is

signal count : std_logic_vector (25 downto 0);
signal clk1Hz : std_logic := '0'; -- for ease, we need a reset signal actually.
begin

        clkOut1Hz <= clk1Hz;
```

```
	divider1Hz : process (clkIn) begin
		if rising_edge(clkIn) then
			if count = "10111110101111000010000000" then -- 50000000
				count <= "00000000000000000000000000";
				clk1Hz <= not clk1Hz;
			else
				count <= count + "00000000000000000000000001";
			end if;
		end if;
	end process;

end Behavioral;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity ShiftRegister is
   Port ( input : in  STD_LOGIC;
	  regclock : in  STD_LOGIC;
	  output : out  STD_LOGIC_vector(1 downto 0));
end ShiftRegister;

architecture Behavioral of ShiftRegister is

signal Q : std_logic_vector (15 downto 0);

begin

	process (regclock) begin
		if rising_edge(regclock) then
			Q(0) <= input;
			Q(1) <= Q(0);
			Q(2) <= Q(1);
			Q(3) <= Q(2);
			Q(4) <= Q(3);
			Q(5) <= Q(4);
			Q(6) <= Q(5);
			Q(7) <= Q(6);
			Q(8) <= Q(7);
			Q(9) <= Q(8);
			Q(10) <= Q(9);
			Q(11) <= Q(10);
			Q(12) <= Q(11);
```

```
                        Q(13) <= Q(12);
                        Q(14) <= Q(13);
                        Q(15) <= Q(14);
                end if;
        end process;

        output <= Q(15) & Q(14);

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity PlayFSM is
        port(           LFSRSequenceIn : in std_logic_vector(15 downto 0);
                                playLevel               : in  std_logic_vector(3  downto
0);
                                button                  : in  std_logic_vector(3  downto
0);
                                clkIn                           : in std_logic;
                                resetPlay               : in std_logic;
                                toVGA                           :  out  std_logic_vector(7
downto 0);
                                comparatorCheck: out std_logic;
                                buzzerPlayOut           : out std_logic_vector(1 downto
0)
                                );


end PlayFSM;

architecture Behavioral of PlayFSM is

--component declarations
component UserInputRegister is
        PORT ( BUTTONS          : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                        CLK             : IN STD_LOGIC;
                        LEVEL                   :  IN  STD_LOGIC_VECTOR(3  DOWNTO
0);
                        RESET                   : IN STD_LOGIC;
                        REG_OUT         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
                        VGA_OUT                 :       OUT       STD_LOGIC_VECTOR(7
DOWNTO 0));
end component;

component Comparator is
```

```
        PORT ( CLK                      : IN STD_LOGIC;
                        GENERATEDSIGNAL     :     IN     STD_LOGIC_VECTOR(15
DOWNTO 0);
                        USERINPUT           :     IN     STD_LOGIC_VECTOR(15
DOWNTO 0);
                        LEVEL                     :   IN   STD_LOGIC_VECTOR(3
DOWNTO 0);
                        CORRECT                   : OUT STD_LOGIC); -- partial
comparison using levels. The rest of

                         -- circuit is not compared in lower levels.

end component;

--signal declarations
signal colorOut : std_logic_vector(7 downto 0);
signal registerO : std_logic_vector(15 downto 0);
signal check : std_logic; -- for correct signal of comparator.
signal buzzer : std_logic_vector(1 downto 0);

begin

--component declerations
UIR : UserInputRegister
        port map ( BUTTONS => button,
                                CLK => clkIn,
                                LEVEL => playLevel,
                                RESET => resetPlay,
                                REG_OUT => registerO,
                                VGA_OUT => colorOut );

comp : Comparator
        port map ( GENERATEDSIGNAL => LFSRSequenceIn,
                                USERINPUT => registerO,
                                LEVEL => playLevel,
                                CORRECT => check,
                                CLK => clkIn);
--signal declerations
toVGA <= colorOut;
comparatorCheck <= check;
buzzerPlayOut <= buzzer;

--processes
buzzerprocess : process (button) begin
        case button is
                when "1000" => buzzer <= "00";
                when "0100" => buzzer <= "01";
                when "0010" => buzzer <= "10";
```

```
		when "0001" => buzzer <= "11";
		when others => buzzer <= "--"; -- ERROR CASE, should not go here.
	end case;
end process;
end Behavioral;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity UserInputRegister is
	PORT ( BUTTONS	: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
			CLK		: IN STD_LOGIC;
			LEVEL		: IN STD_LOGIC_VECTOR(3 DOWNTO
0);
			RESET		: IN STD_LOGIC;
			REG_OUT	: OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
			VGA_OUT		: OUT STD_LOGIC_VECTOR(7
DOWNTO 0)
			);
end UserInputRegister;

architecture Behavioral of UserInputRegister is

signal REG1, REG2, REG3, REG4, REG5, REG6, REG7, REG8 :
STD_LOGIC_VECTOR(1 DOWNTO 0);
begin


	REG_OUT <= REG1 & REG2 & REG3& REG4 & REG5 & REG6 & REG7&
REG8;

	REGISTER_STORAGE: process (CLK, RESET, BUTTONS) begin
		if RESET = '1'  then
			REG1 <= "00";
			REG2 <= "00";
			REG3 <= "00";
			REG4 <= "00";
			REG5 <= "00";
			REG6 <= "00";
			REG7 <= "00";
			REG8 <= "00";
		elsif rising_edge(CLK) then
			if LEVEL = "0000" then -- level 1;
				case (BUTTONS) is
					when "1000" => REG1 <= "00";
					when "0100" => REG1 <= "01";
```

```
                when "0010" => REG1 <= "10";
                when "0001" => REG1 <= "11";
                when others => REG1 <= "XX";
        end case;
elsif LEVEL = "0001" then -- level 2;
                case (BUTTONS) is

                        when "1000" => REG2 <= "00";
                        when "0100" => REG2 <= "01";
                        when "0010" => REG2 <= "10";
                        when "0001" => REG2 <= "11";
                        when others => REG2 <= "XX";
                end case;
                elsif LEVEL = "0010" then -- level 3;
                case (BUTTONS) is

                        when "1000" => REG3 <= "00";
                        when "0100" => REG3 <= "01";
                        when "0010" => REG3 <= "10";
                        when "0001" => REG3 <= "11";
                        when others => REG3 <= "XX";
                end case;
elsif LEVEL = "0011" then -- level 4;
                case (BUTTONS) is

                        when "1000" => REG4 <= "00";
                        when "0100" => REG4 <= "01";
                        when "0010" => REG4 <= "10";
                        when "0001" => REG4 <= "11";
                        when others => REG4 <= "XX";
                end case;
elsif LEVEL = "0100" then -- level 5;
                case (BUTTONS) is

                        when "1000" => REG5 <= "00";
                        when "0100" => REG5 <= "01";
                        when "0010" => REG5 <= "10";
                        when "0001" => REG5 <= "11";
                        when others => REG5 <= "XX";
                end case;
elsif LEVEL = "0101" then -- level 6;
                case (BUTTONS) is

                        when "1000" => REG6 <= "00";
                        when "0100" => REG6 <= "01";
                        when "0010" => REG6 <= "10";
                        when "0001" => REG6 <= "11";
                        when others => REG6 <= "XX";
```

```
                        end case;
                elsif LEVEL = "0110" then -- level 7;
                        case (BUTTONS) is

                                when "1000" => REG7 <= "00";
                                when "0100" => REG7 <= "01";
                                when "0010" => REG7 <= "10";
                                when "0001" => REG7 <= "11";
                                when others => REG7 <= "XX";
                        end case;
                elsif LEVEL = "0111" then -- level 8;
                        case (BUTTONS) is

                                when "1000" => REG8 <= "00";
                                when "0100" => REG8 <= "01";
                                when "0010" => REG8 <= "10";
                                when "0001" => REG8 <= "11";
                                when others => REG8 <= "XX";
                        end case;
                end if;
        end if;

        end process;

        VGA_REFLECT: process (CLK, BUTTONS) begin
                if rising_edge(CLK) then
                        if BUTTONS = "1000" then
                                VGA_OUT <= "11100000"; -- RED
                        elsif BUTTONS = "0100" then
                                VGA_OUT <= "00011100"; -- GREEN
                        elsif BUTTONS = "0010" then
                                VGA_OUT <= "00000011"; -- BLUE
                        elsif BUTTONS = "0001" then
                                VGA_OUT <= "11111100"; -- YELLOW
                        else
                                VGA_OUT <= "00000000";
                        end if;
                end if;
        end process;

end Behavioral;


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

```
entity Comparator is
	PORT ( GENERATEDSIGNAL : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
			USERINPUT		: IN STD_LOGIC_VECTOR(15
DOWNTO 0);
			LEVEL			: IN STD_LOGIC_VECTOR(3
DOWNTO 0);
			CLK			: IN STD_LOGIC;
			CORRECT			: OUT STD_LOGIC); -- partial
comparison using levels. The rest of

			 -- circuit is not compared in lower levels.

end Comparator;

architecture Behavioral of Comparator is

signal state : std_logic_vector(3 downto 0); -- 1 lose state, 1 win state + 6 staging
states, rest -> dont care.
signal temp : std_logic := '0';

begin

correct <= temp;

	comparator : process( GENERATEDSIGNAL, USERINPUT, state, LEVEL,
clk) begin
		if rising_edge(CLK) then
			case state&LEVEL is -- comparators are checked only for
the levels that the user is in.
				when "00000000" =>
					if GENERATEDSIGNAL(15 downto 13) =
USERINPUT(15 downto 13) then
						state <= "0001"; -- next level
						temp <= '1';
					else
						state <= "1001"; -- error state
						temp <= '0';
					end if;
				when "00010000" | "00010001" =>
					if GENERATEDSIGNAL(13 downto 11) =
USERINPUT(13 downto 11) then
						state <= "0010";
						temp <= '1';
					else
						state <= "1001";
						temp <= '0';
					end if;
				when "00100000" | "00100001" | "00100010" =>
```

```
                        if GENERATEDSIGNAL(11 downto 9) =
USERINPUT(11 downto 9) then
                                state <= "0011";
                                temp <= '1';
                        else
                                state <= "1001";
                                temp <= '0';
                        end if;
                when "00110000" | "00110001" | "00110010" |
"00110011"  =>
                        if GENERATEDSIGNAL(9 downto 7) =
USERINPUT(9 downto 7) then
                                state <= "0100";
                                temp <= '1';
                        else
                                state <= "1001";
                                temp <= '0';
                        end if;
                when "01000000" | "01000001" | "01000010" |
"01000011" | "01000100" =>
                        if GENERATEDSIGNAL(7 downto 5) =
USERINPUT(7 downto 5) then
                                state <= "0101";
                                temp <= '1';
                        else
                                state <= "1001";
                                temp <= '0';
                        end if;
                when "01010000" | "01010001" | "01010010" |
"01010011" | "01010100" | "01010101"  =>
                        if GENERATEDSIGNAL(5 downto 3) =
USERINPUT(5 downto 3) then
                                state <= "0110";
                                temp <= '1';
                        else
                                state <= "1001";
                                temp <= '0';
                        end if;
                when "01100000" | "01100001" | "01100010" |
"01100011" | "01100100" | "01100101" | "01100110" =>
                        if GENERATEDSIGNAL(3 downto 1) =
USERINPUT(3 downto 1) then
                                state <= "0111";
                                temp <= '1';
                        else
                                state <= "1001";
                                temp <= '0';
                        end if;
```

```
                when "01110000" | "01110001" | "01110010" |
"01110011" | "01110100" | "01110101" | "01110110" | "01110111" =>
                        if GENERATEDSIGNAL(1 downto 0) =
USERINPUT(1 downto 0) then
                                state <= "1000";
                                temp <= '1';
                        else
                                state <= "1001"; -- WIN
                                temp <= '0';
                        end if;
                when others =>
                        state <= "1010"; -- ERROR STATE
                        temp <= '0';
            end case;
        end if;
    end process;



--      levelout : process(LEVEL,
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity Multiplexer8In is
    port ( A, B : in std_logic_vector(7 downto 0);
                selectSig : in std_logic;
                C : out std_logic_vector (7 downto 0));
end Multiplexer8In;

architecture Behavioral of Multiplexer8In is

begin

    process (A,B, selectSig) begin
        if selectSig = '0' then
            C <= A;
        else
            C <= B;
        end if;
    end process;

end Behavioral;
```

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


entity Mux2In is
	port ( D, E : in std_logic_vector(1 downto 0);
			s : in std_logic;
			F : out std_logic_vector (1 downto 0));
end Mux2In;

architecture Behavioral of Mux2In is

begin

	process (D,E, s) begin
		if s = '0' then
			F<= D;
		else
			F <= E;
		end if;
	end process;

end Behavioral;

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


entity WaitFSM is
	port ( clk50In : in std_logic;
			levelInput : in std_logic_vector(3 downto 0);
			restart : in std_logic;
			SimonFSMSelect : out std_logic);
end WaitFSM;

architecture Behavioral of WaitFSM is

	-- component declaration
	component ClockDivider is
		Port ( clkIn : in  STD_LOGIC;
				clkOut1Hz : out  STD_LOGIC);
	end component;


	-- signal declaration
```

```
	signal clkUsed : std_logic;
	signal selectFSM : std_logic;
	signal count : std_logic_vector ( 4 downto 0 ) := "00000"; -- 20 seconds
max, this allocates 32 seconds.
	type state_type is (play, display);
	signal state,nextState : state_type;

	begin

		-- component initialization
		divider : ClockDivider
			port map ( clkIn => clk50In,
					clkOut1Hz => clkUsed);
		-- signal initialization
		SimonFSMSelect <= selectFSM;

		-- processes
		counter : process(clkUsed ,count) begin
			if rising_edge(clkUsed) then
				count <= count + "00001";
			end if;
		end process;

		mapping : process ( clkUsed,state ) begin
			if rising_edge(clkUsed) then
				case state is
					when display =>
						selectFSM <= '0';
					when others =>
						selectFSM <= '1';
				end case;
			end if;
		end process;


		start : process ( state, restart, clkUsed) begin
			if rising_edge(clkUsed) then
				if restart = '1' then
					state <= display;
				else
					state <= play;
				end if;
			end if;
		end process;

		waitFSM : process ( count, state, nextState, levelInput) begin
			case state is
				when display =>
```

```
                    if levelInput = "0000" then
                        if count = "00111" then -- 5 + 2
seconds
                            nextState <= play;
                        end if;
                    elsif levelInput = "0001" then
                        if count = "01001" then -- 5 + 4
seconds
                            nextState <= play;
                        end if;
                    elsif levelInput = "0010" then
                        if count = "01011" then -- 5 + 6
seconds
                            nextState <= play;
                        end if;

                    elsif levelInput = "0011" then
                        if count = "01101" then -- 5 + 8
seconds
                            nextState <= play;
                        end if;

                    elsif levelInput = "0100" then
                        if count = "01111" then -- 5 + 10
seconds
                            nextState <= play;
                        end if;

                    elsif levelInput = "0101" then
                        if count = "10011" then -- 5 + 14
seconds
                            nextState <= play;
                        end if;

                    elsif levelInput = "0110" then
                        if count = "10101" then -- 5 + 16
seconds
                            nextState <= play;
                        end if;

                    elsif levelInput = "0111" then
                        if count = "10111" then -- 5 + 18
seconds
                            nextState <= play;
                        end if;
                    end if;

                when others =>
```

```
                    if count = "10101" then -- 20 + 1 seconds
                        nextState <= display;
                    end if;
                end case;
        end process;



end Behavioral;

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

entity ClockDivider is
   Port ( clkIn : in  STD_LOGIC;
          clkOut1Hz : out  STD_LOGIC);
end ClockDivider;

architecture Behavioral of ClockDivider is

signal count : std_logic_vector (25 downto 0);
signal clk1Hz : std_logic := '0'; -- for ease, we need a reset signal actually.
begin

        clkOut1Hz <= clk1Hz;

        divider1Hz : process (clkIn) begin
                if rising_edge(clkIn) then
                        if count = "10111110101111000010000000" then --
50000000
                                count <= "00000000000000000000000000";
                                clk1Hz <= not clk1Hz;
                        else
                                count <= count +
"00000000000000000000000001";
                        end if;
                end if;
        end process;

end Behavioral;
```

UCF File:

```
NET "BLUEO[1]" LOC = H13;
NET "BLUEO[2]" LOC = J13;
```

```
NET "BUTTONIN[3]" LOC = A7;
NET "BUTTONIN[2]" LOC = M4;
NET "BUTTONIN[1]" LOC = C11;
NET "BUTTONIN[0]" LOC = G12;
NET "BUZZEROUT" LOC = B2;
NET "CLKTOPIN" LOC = B8;
NET "GREENO[0]" LOC = F14;
NET "GREENO[1]" LOC = G13;
NET "GREENO[2]" LOC = G14;
NET "HOLDIN" LOC = N3;
NET "HSO" LOC = J14;
NET "REDO[0]" LOC = C14;
NET "REDO[1]" LOC = D13;
NET "REDO[2]" LOC = F13;
NET "RESETIN" LOC = P11;
NET "VSO" LOC = K13;
net "RESETSEQ" LOC = E2;
```

The pictures of the device/equipment:

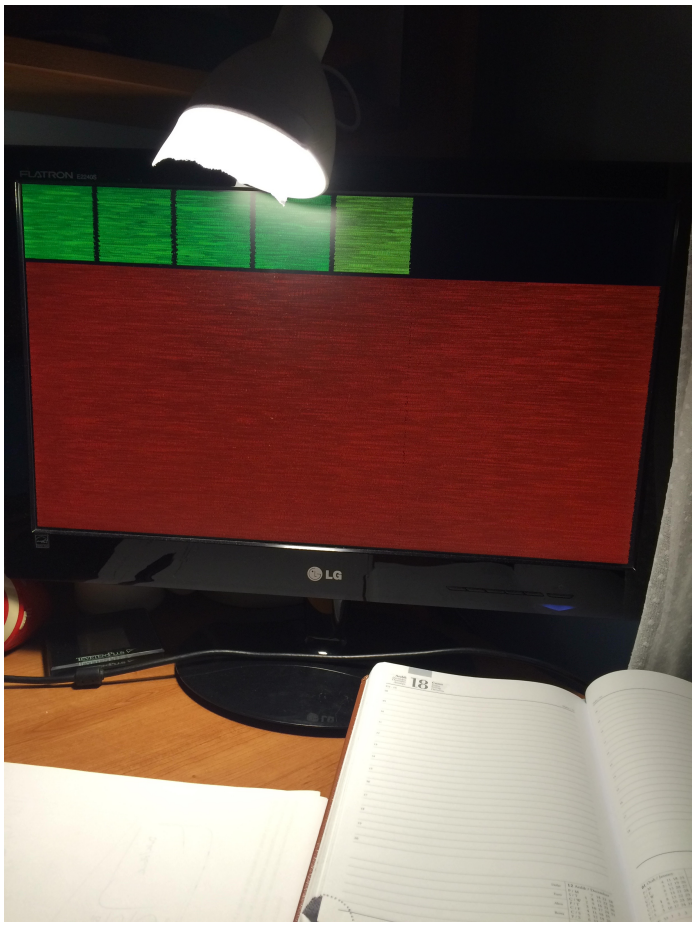

Figure 2. Level 5 of the game, working. This is from the design where wait states were switched using slideswitches.

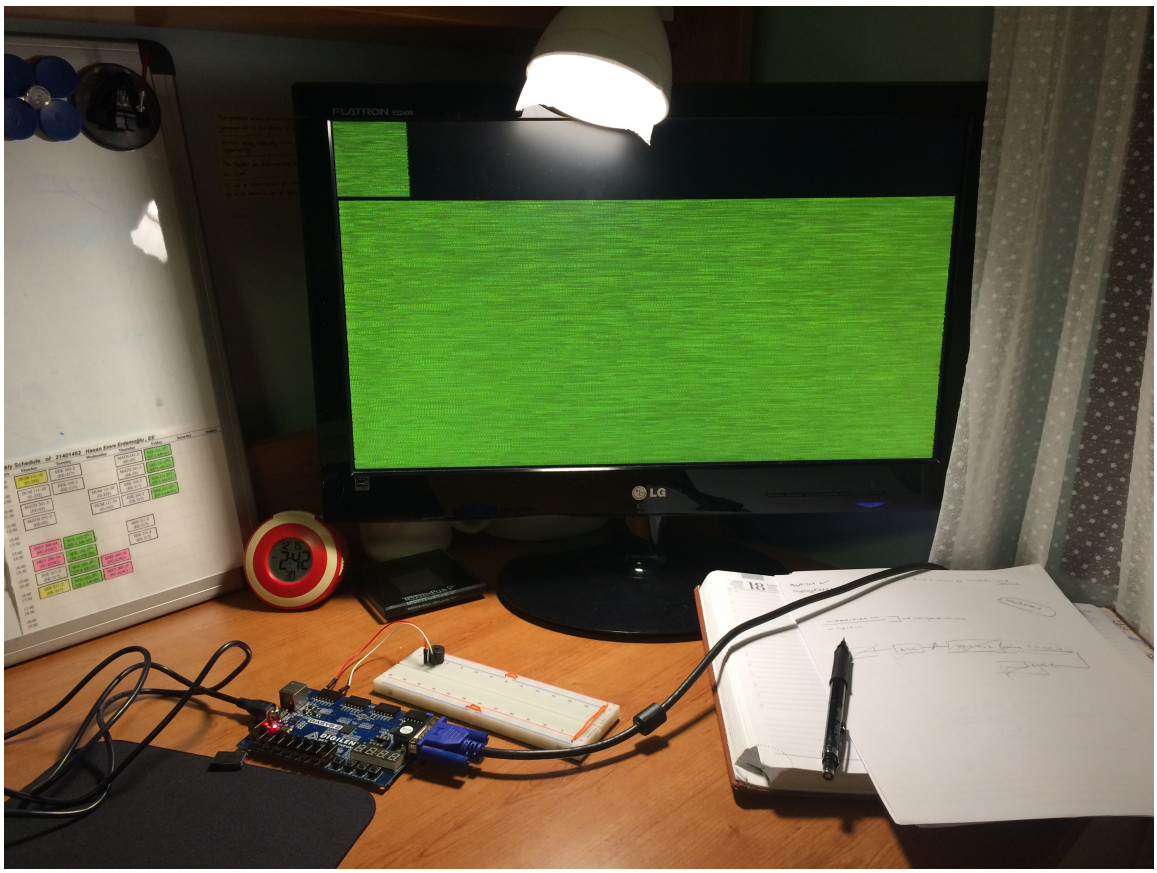

Figure 4. Another picture of the device. Note that the crystal of BASYS 2 provides 50 MHz clock, that is divided to 25 MHz clock for the VGA. Actually for a still image, the VGA needs 25.125 MHz clock, which is not possible without a proper quartz crystal or any equivalent equipment.

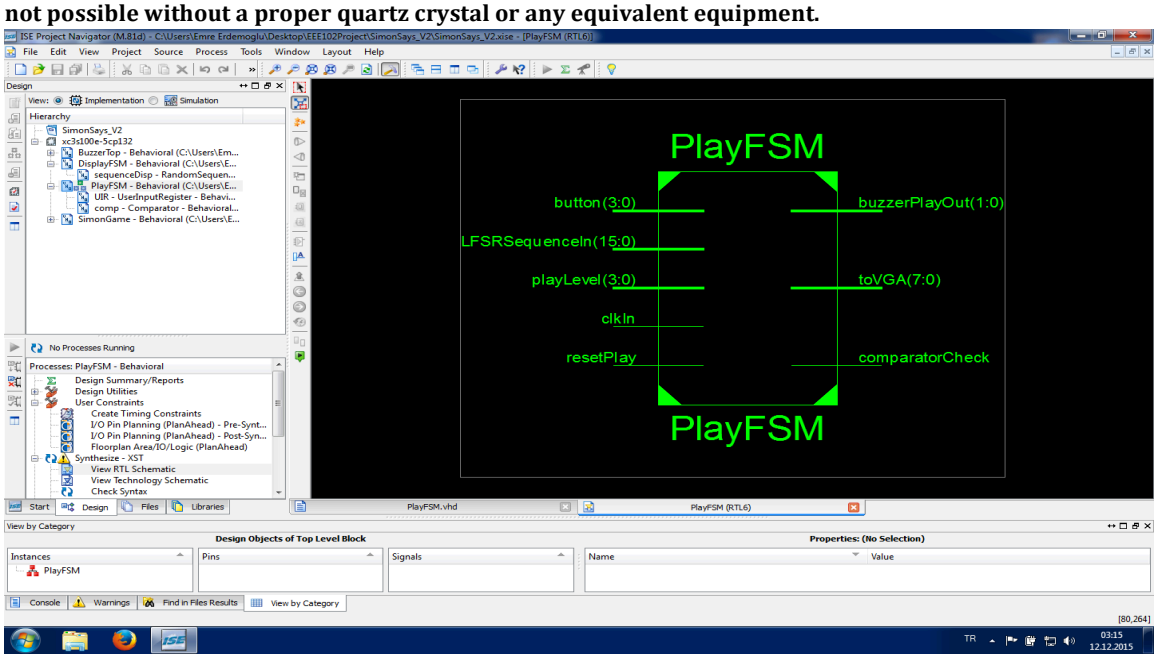

Figure 3. Black box diagram for PlayFSM

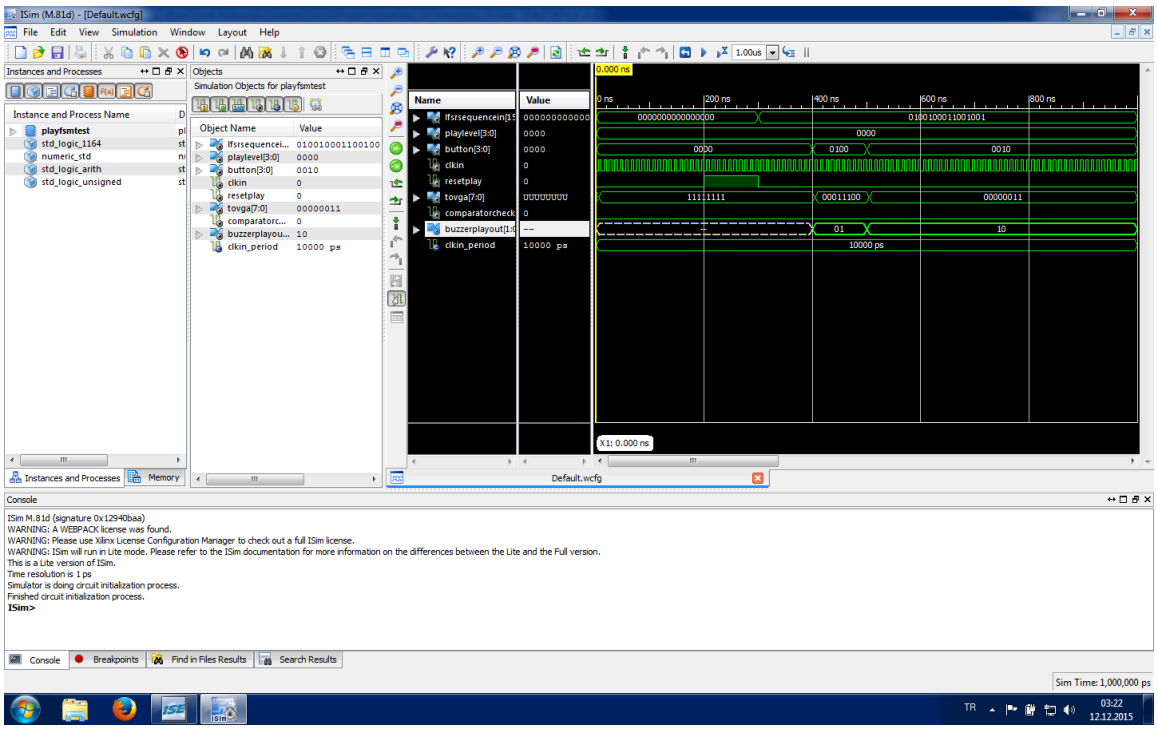

Figure 5. Test bench for Play FSM

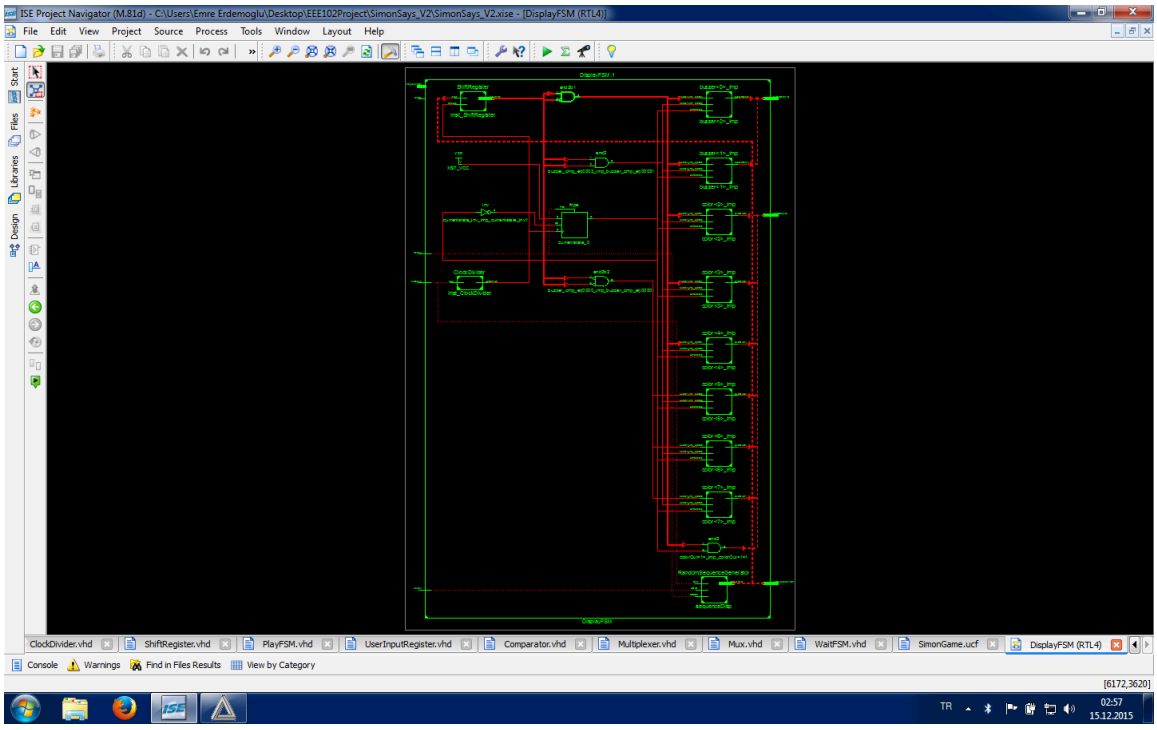

Figure 6. RTL Schematic for Display FSM

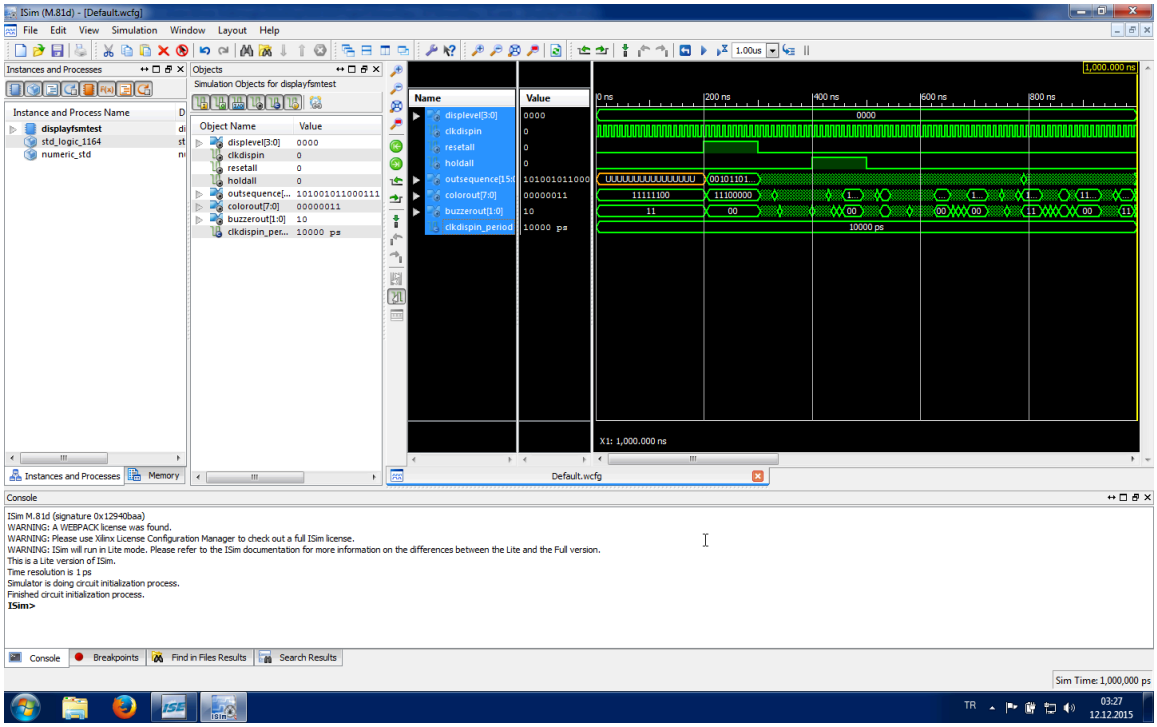

Figure 8. The test bench for Display FSM

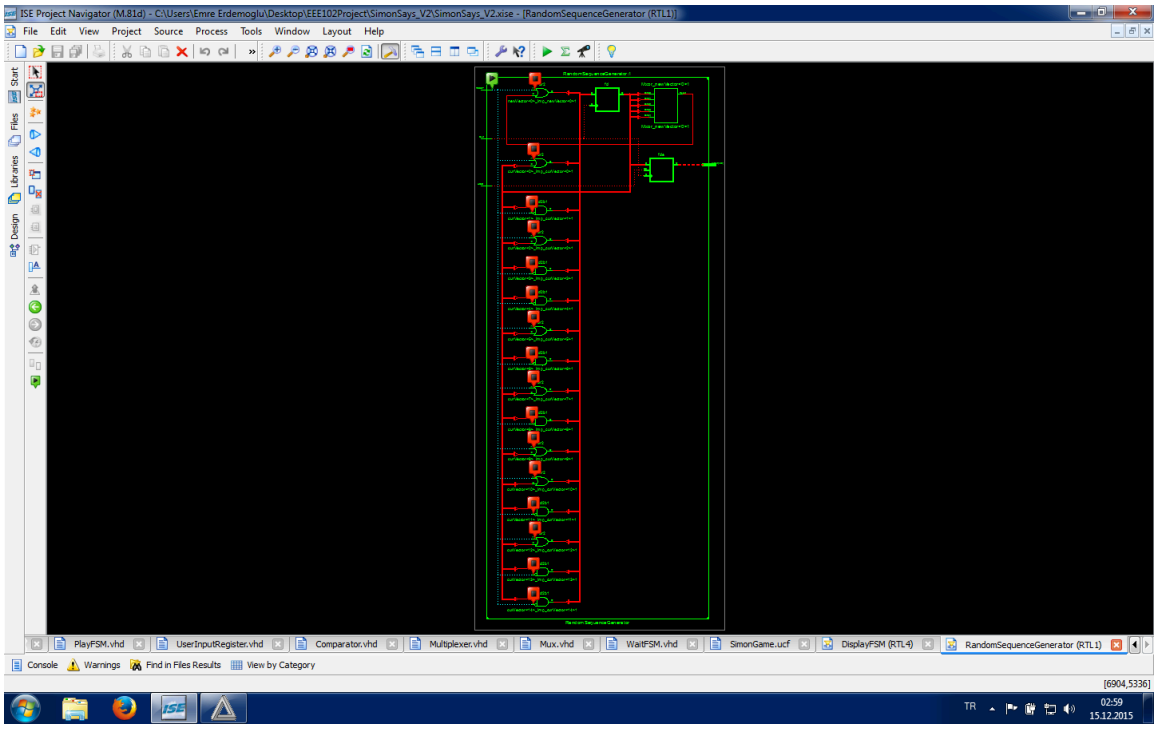

Figure 7. RTL Schematic of LFSR

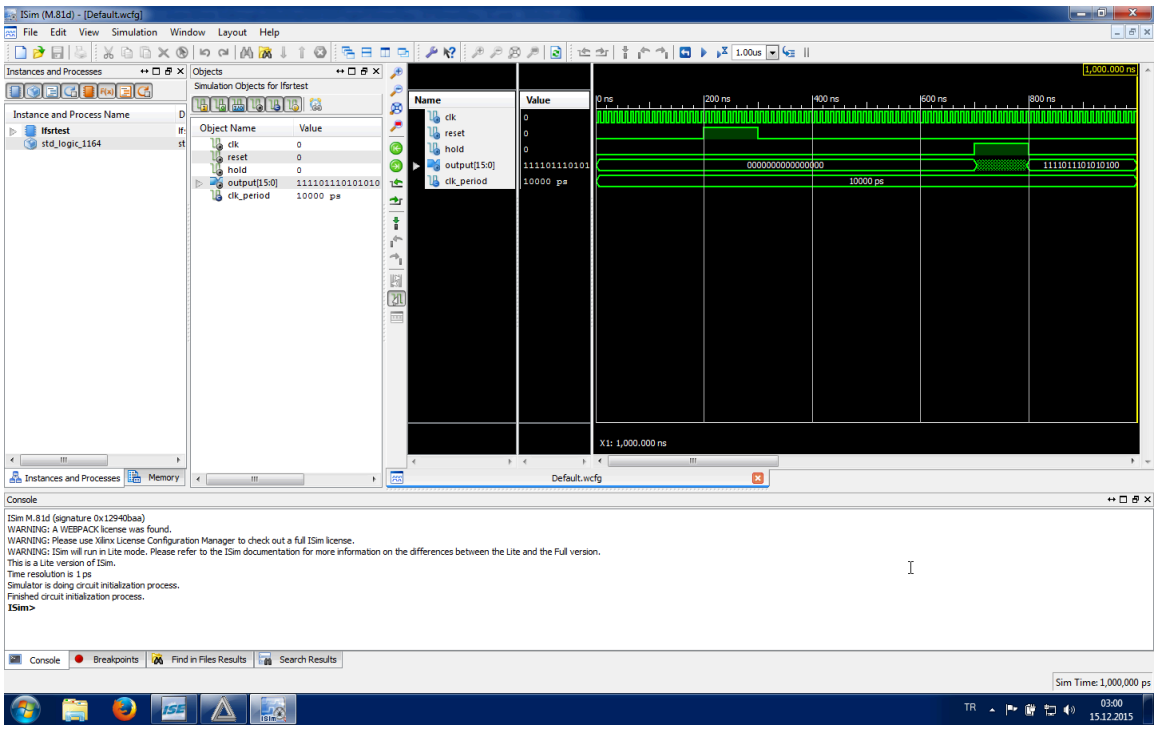

Figure 9. Test Bench for LFSR

The unit testing provided positive feedback on the workability of the whole circuitry. However this was not the case because of the reasons discussed above.