CalPolyPomona

College of Engineering

Electrical and Computer Engineering

ECE 3300L - Digital Circuit Design Using Verilog

Professor Mohammed El-Hadedy

Final Project - The Art Thief

Group B:
Noah Aldridge
Quinn Bell
Mike Plata
Eduardo Vargas

The Art Thief

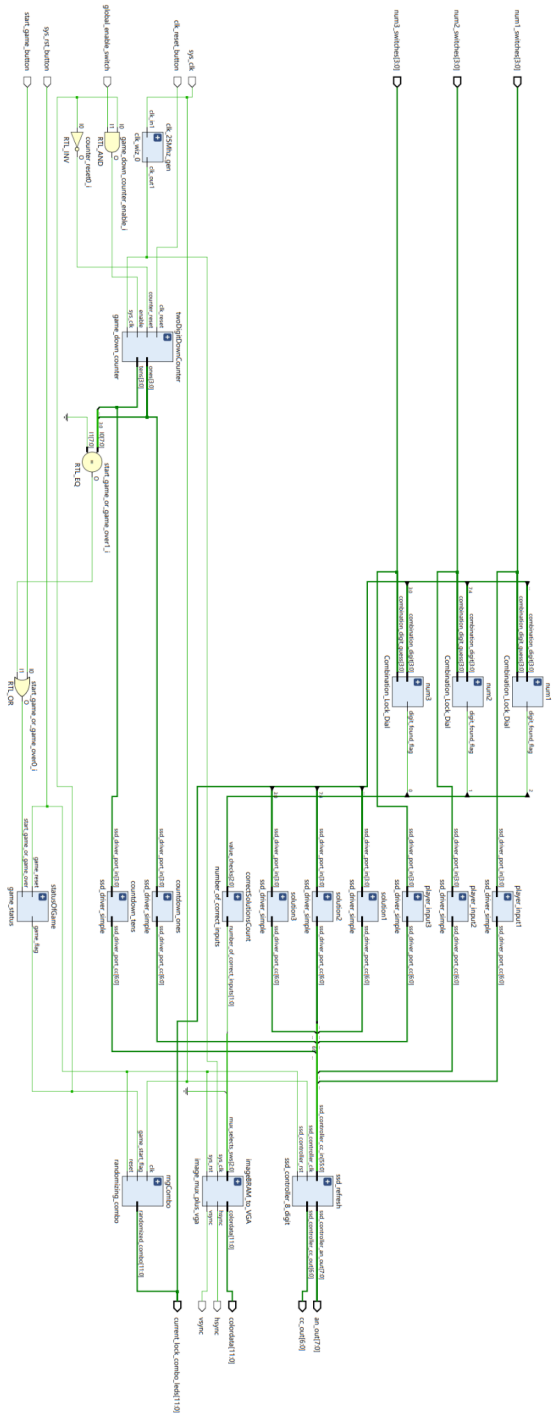
Abstract:
This project, THE ART THIEF, represents the amalgamation of theoretical concepts from ECE 3300 with practical application using the Nexys A7 FPGA board. Employing Verilog principles, it showcases Verilog's practicality through decoding a 3-digit padlock within a time limit to reveal an image. Verilog's integral role in managing game progression, initiating timers, interpreting switch inputs, and synchronizing game elements highlights its real-time operational strength. As the backbone, Verilog governs logic circuits controlling switches, LEDs, displays, and timers, ensuring precise game control and board functionality. This project underscores optimized design efficiency by streamlining redundant modules and consolidating functionalities. This approach guarantees meeting diverse design requirements without functional compromise. The experience with image conversion stresses proper data transformation's necessity for accurate image display, emphasizing data compatibility's pivotal role. THE ART THIEF showcases Verilog application in digital systems, illustrating the integration of theory and practice ingrained throughout the course.

Introduction:
This project represents a culmination of acquired knowledge and practical skills obtained through ECE 3300, encompassing the principles and applications of Verilog within digital design. Studied throughout the course were the designs of various systems, encompassing fundamental elements like adders, counters, and shift registers. Additionally, the course covered finite state machines, timing considerations, and calculation of maximum clock rates while ensuring adherence to setup and hold time specifications.
This project merges theoretical concepts with practical implementation through the development of a game utilizing the Nexys A7 FPGA board. The goal is to showcase Verilog's implementation in game design and development. This project highlights the practical application of Verilog principles in creating an engaging and interactive gaming experience. The game's design, functionality, and execution represent the culmination of skills acquired during this comprehensive course on digital systems and Verilog programming.

Project Description:
The objective of the game is for a player to decipher a random 3-digit padlock combination within 30 seconds to reveal an obscured image on the screen. Failure to complete the task within the time limit stops the game, waiting for the player to reset the game. Successfully deciphering the combination unveils the previously obscured image. Once the game is initiated, a blurred image will be displayed on the screen alongside a pair of RGB LEDs—glowing red to let the player know they have not won yet. A button is incorporated to activate the game. When pressed, a countdown timer initiates, starting from 29 and decrementing to 0. To decipher the 3-digit padlock combination, players utilize an array of switches, with every 4 switches representing a hexadecimal (0-F) digit. The digits are displayed on a seven-segment display.

Successful guesses illuminate an LED beneath the corresponding digit, progressively revealing the image by unblurring it. Upon correctly guessing all three padlock digits, the timer halts, and the RGB LEDs transition to green. Simultaneously, the image on the screen becomes fully clear, indicating successful completion.
In this game's context, Verilog operates as the fundamental coding framework responsible for managing the game's progression. It handles the initiation of the game, management of the countdown timer, interpretation of switch inputs, and dynamic alteration of LED displays. Moreover, Verilog's real-time operation capabilities provide for synchronizing of crucial game elements. This ensures precision in the countdown timer, LED illumination, and image display, all contingent upon player interaction.

Design Specifications:
Verilog stands as the foundation for the game's functionality, driving the logic circuits which govern switches, LEDs, the seven-segment display, and the timer. These components are implemented using Verilog to ensure precise control and operation within the game environment and on the FPGA board. The language, as implemented on the FPGA board, showcases its effectiveness in orchestrating real-time digital system operations for seamless game functionality.
Various components of the Nexys 17 board were utilized. The 100MHz clock signal ensures synchronous operations. The global enable switch at pin J15 provides centralized control, while switches across pins R17, T18, U18, R13, T8, U8, R16, T13, H6, U12, U11, and V10 facilitate player input. LEDs at pins R18, V17, U17, U16, V16, T15, U14, T16, V15, V14, V12, and V11 confirm the lock combination. And, the 7-segment display array showcases both the combination and players' inputs. Repurposing the CPU reset button at pin C12 initiates the game, with dedicated buttons at pins N17 and N18 serving as the system and clock reset, respectively. Integrating the VGA connector extends visual features, displaying images on an external monitor.
At the game’s core is the Combination Generator, employing a counter and barrel shifters to craft a randomized 3-digit code. This code, matched against player inputs by the Combination Lock Dial module, triggers visual feedback through the Number of Correct Inputs module, directing the game's progression. Each successful guess incrementally unveils an obscured image, facilitated by the Image Mux and ROMs, strategically selecting and displaying images based on the player's advancements.
Driving the visual display, the VGA Driver manages image generation, utilizing register values to procure color data from the Image Mux and ROMs. Concurrently, the Countdown module acts as the timer, initiating a 30-second countdown upon game activation. The Seven Segment Display Driver illuminates the seven-segment display to showcase guessed digits, aiding the player in deciphering the padlock code. Lastly, the Seven Segment Display Controller manages the entire display, depicting player inputs, the game solution, and the countdown timer, providing a holistic view of the game's progress.



Project Implementation Overview:

Specification	Utilization
LUT	0.25%
FF	0.08%
BRAM	8.89%
IO	25.24%
PLL	16.67%

Project Timing Results:

Timing Category	Time
Worst Negative Slack	6.19 ns
Total Negative Slack	0 ns
Worst Hold Slack	0.175 ns
Total Hold Slack	0 ns

Power Consumption and Temperature:

Total On-Chip Power	0.214 W
Static Power	0.098 W
Dynamic Power	0.116 W
Junction Temperature	26°C

Verilog Code:

The following is a comprehensive explanation of the Verilog code, featuring the Combination Generator, Combination Lock Dial, Number of Correct Inputs, Image Mux and ROMs, VGA Driver, Countdown, Seven Segment Display Driver, and Seven Segment Display Controller modules. Accompanying the Verilog code are comments delineating their functionalities.

Combination Generator:

This module features an always-enabled counter and multiple barrel shifters to generate three pseudo-random hexadecimal numbers, constituting the padlock passcode. These numbers are then cross-referenced with switch input values. Upon game initiation, the module locks the current passcode numbers until the game restarts. Its essential ports include clock, reset, and game_start_flag inputs. The clock controls the counter, while the game_start_flag input captures

the passcode on its positive edge. Internally, the module comprises two up-counters and three barrel shifters. The up-counters and barrel shifters collaborate, where the second up-counter influences the selection process for the barrel shifters. Adding complexity, each barrel shifter receives selections from the same bus but in a distinct wiring sequence.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/07/2023 06:06:41 PM
// Design Name:
// Module Name: Combination_Generator
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module Combination_Generator(
              input[11:0] randomization_factors,
              input clock,

              output[11:0] lock_combination
       );

       wire[11:0] lock_combo_bus;
       wire rng_always_enabled;
       wire rng_never_resets;

       assign rng_always_enabled = 1'b1;
       assign rng_never_resets = 1'b0;

       RNG_0_TO_F RandomDigit1(
              .clock(clock),
              .enable(rng_always_enabled),
              .reset(rng_never_resets),
              .count_direction(randomization_factors[0]),
              .shift_or_rotate(randomization_factors[1]),
              .number_of_shifts_or_rotations(randomization_factors[3:2]),
```

```
        .random_number_out(lock_combo_bus[3:0])
    );

    RNG_0_TO_F RandomDigit2(
        .clock(clock),
        .enable(rng_always_enabled),
        .reset(rng_never_resets),
        .count_direction(randomization_factors[4]),
        .shift_or_rotate(randomization_factors[5]),
        .number_of_shifts_or_rotations(randomization_factors[7:6]),

        .random_number_out(lock_combo_bus[7:4])
    );

    RNG_0_TO_F RandomDigit3(
        .clock(clock),
        .enable(rng_always_enabled),
        .reset(rng_never_resets),
        .count_direction(randomization_factors[8]),
        .shift_or_rotate(randomization_factors[9]),
        .number_of_shifts_or_rotations(randomization_factors[11:10]),

        .random_number_out(lock_combo_bus[11:8])
    );

    assign lock_combination = lock_combo_bus;
endmodule
```

Combination Lock Dial:

Three instances of this module facilitate the comparison between the player's inputs and the current game's passcode. When a switch number matches the random number, it triggers a '1' output to the digit_found_flag; otherwise, a '0' is produced. Each module instance manages four of the total 12 switches, interpreting the player's inputs. The module interfaces through a 4-bit passcode digit input, a 4-bit player input, and an output for the digit_found_flag.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/07/2023 08:32:27 PM
// Design Name:
// Module Name: Combination_Lock_Dial
// Project Name:
```

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module Combination_Lock_Dial(
        input[3:0] combination_digit,
        input[3:0] combination_digit_guess,

        output reg digit_found_flag
    );


    always @(combination_digit_guess)
        begin
            if(combination_digit_guess == combination_digit)
                digit_found_flag <= 1'b1;
            else
                digit_found_flag <= 1'b0;
        end
endmodule
```

Number of Correct Inputs:

This intermediary module connects the combination lock dial modules with the subsequent module, the image mux. Featuring two ports, it accepts a 3-bit input derived from the lock dials' outputs, functioning as a bus. Its output, a 2-bit signal, denotes the count of "1's" identified in the input bus. For instance, inputs like 001, 100, or 010 yield a 2'd1 output, while inputs like 011, 110, or 101 result in a 2'd2 output. The image mux utilizes this output to determine and present an image corresponding to the player's progress.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/08/2023 11:45:44 AM
// Design Name:
// Module Name: number_of_correct_inputs
// Project Name:
```

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/* This module is used to organize the mux from most blurry to least blurry (no
blur). */

module number_of_correct_inputs(
        input [2:0] value_checks,                       //Output of "value check"
modules taken together as a bus
        output reg [1:0] number_of_correct_inputs       //Outputs to image mux.
Considers # of '1's from value check's, does not care which inputs are correct.
    );

always@(value_checks)
    begin
            case(value_checks)
                3'b000: number_of_correct_inputs <= 2'b00;   // No correct inputs
                3'b001: number_of_correct_inputs <= 2'b01;   // 1 correct input
                3'b010: number_of_correct_inputs <= 2'b01;   // 1 correct input
                3'b011: number_of_correct_inputs <= 2'b10;   // 2 correct inputs
                3'b100: number_of_correct_inputs <= 2'b01;   // 1 correct input
                3'b101: number_of_correct_inputs <= 2'b10;   // 2 correct inputs
                3'b110: number_of_correct_inputs <= 2'b10;   // 2 correct inputs
                3'b111: number_of_correct_inputs <= 2'b11;   // 3 correct inputs
                default: number_of_correct_inputs <= 2'd0;    // Standard default
            endcase
    end
endmodule
```

Image Mux and ROMs:

The image multiplexer module initializes four images within the board's block memory (BRAM), configured as a read-only memory (ROM). These images feature various blur levels applied to the same base image. Each memory output connects to the 12-bit wide inputs of a 4-to-1 multiplexer. The multiplexer employs the output from the "number of correct inputs" module as selection bits. Upon image selection, this module outputs the color data corresponding to the current pixel of the chosen image to the board's VGA port. This process operates simultaneously with the VGA Driver module, ensuring parallel functionality.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/08/2023 11:32:17 PM
// Design Name:
// Module Name: image_mux_plus_vga
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module image_mux_plus_vga(
            input sys_clk,  //25MHz
            input sys_rst,
            input [2:0] mux_selects_sws,
            output reg [11:0] colordata,
            output vsync,
            output hsync
    );

wire clk_25MHz;
wire [9:0] xCount;  // Current column #
wire [9:0] yCount;  // Current row #
reg [9:0] imageX;   // Location on screen that image should begin displaying (left
of image)
reg [9:0] imageY;   // Location on screen that image should begin displaying (top
of image)
wire displayArea;   // Check if row and column are currently in the display area
before generating pixel.
reg write_image;    // "Enable" for generating a pixel to the screen.
wire [11:0] colordata_temp;     // Connects output of mux (data from BRAM) to the
output.

vgaTimings outputVGA(
    .VGA_clk(sys_clk),
    .xCount(xCount),
```

```
    .yCount(yCount),
    .displayArea(displayArea),
    .VGA_hSync(hsync),
    .VGA_vSync(vsync)
);

image_mux muxForImages(
    .clk(sys_clk),
    .mux_selects(mux_selects_sws),
    .row(yCount),
    .col(xCount),
    .colordata(colordata_temp)
);

always@(posedge sys_clk)
            begin
        if(sys_rst)
            begin
                colordata <= 0;
                write_image <= 0;
                imageX <= 0;
                imageY <= 0;
            end

        else
                write_image <= (
                    (xCount > imageX & xCount < (imageX + 50)) &    // (imageX + #)
: # is the width of images
                    (yCount > imageY & yCount < (imageY + 100))     // (imageY + #)
: # is the height of images
                                );

    if(write_image & displayArea)
            begin
                colordata <= colordata_temp;

            end

            else
                begin
                    colordata <= 0;
                end

            end
endmodule
```

VGA Driver:

This module coordinates the generation of a 640x480 resolution image via the board's VGA port. Leveraging the 25MHz clock and several counters, it produces the Vsync and Hsync signals transmitted to the board. Additionally, it generates xCount and yCount register values, indicating the column and row numbers of the current pixel, respectively. These register values interface with the image mux and ROMs to retrieve the corresponding color data for the currently generated pixel.

```
module VGA_Driver(
        input VGA_clk,
        input rst,
        output reg [9:0] xCount, yCount,
        output reg displayArea,
        output VGA_hSync, VGA_vSync
    );

    reg p_hSync, p_vSync;

    integer porchHF = 640; //start of horizontal front porch
    integer syncH = 656; //start of horizontal sync
    integer porchHB = 752; //start of horizontal back porch
    integer maxH = 800; //total length of column


    integer porchVF = 480; //start of vertical front porch
    integer syncV = 490; //start of vertical sync
    integer porchVB = 492; //start of vertical back porch
    integer maxV = 525; //total length of row

//    always@(rst)
//        begin xCount = 0; yCount = 0; end

    always@(posedge VGA_clk)
    begin
        if(xCount == maxH)
            xCount <= 0;
        else
            xCount <= xCount + 1'b1;
    end

    always@(posedge VGA_clk)
    begin
        if(xCount == maxH)
            begin
```

```
                if (yCount == maxV)
                    yCount <= 0;
                else
                    yCount <= yCount + 1'b1;
            end
    end

    always@(posedge VGA_clk)
        begin
            displayArea <= ((xCount < porchHF) && (yCount < porchVF));
        end

    always@(posedge VGA_clk)
        begin
            p_hSync <= ((xCount >= syncH) && (xCount < porchHB));
            p_vSync <= ((yCount >= syncV) && (yCount < porchVB));
        end

    assign VGA_vSync = ~p_vSync;
    assign VGA_hSync = ~p_hSync;
endmodule
```

Countdown:

The 2-digit down counter operates with a clock signal and reset, solely functioning as the player's time limit indicator. Spanning from 29 to 0, it represents the allocated time for code guessing. Its output, two 7-segment display codes, interfaces with two seven-segment display drivers, visually presenting the player's remaining time on the board. Alongside this display, it emits a one-bit game-over signal to alert other modules, halting the game's progression. After this point, the game enters an idle state until the player initiates a new game by resetting and pressing the start-game button.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/09/2023 10:03:52 AM
// Design Name:
// Module Name: game_down_counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module game_down_counter(
        input sys_clk,
        input enable,
        input counter_reset,
        input clk_reset,
        output [3:0] tens,
        output [3:0] ones
    );

wire [4:0] locked_speed = 5'b11000;   //RTL Value
//wire [4:0] locked_speed = 5'b00000;     //TB Value
wire slow_clk;
wire tens_enable;

assign tens_enable = enable & (ones == 0);

clk_counter slow_clock(
    .sys_clk(sys_clk),
    .sys_rst(clk_reset),
    .speed_selector(locked_speed),
    .block_clk(slow_clk)
);

game_downcounter_tens tens_counter(
    .dcbcd_clk(slow_clk),
    .dcbcd_rst(counter_reset),
    .dcbcd_en(tens_enable),
    .dcbcd_q(tens)
);

downcounterbcd ones_counter(
    .dcbcd_clk(slow_clk),
    .dcbcd_rst(counter_reset),
    .dcbcd_en(enable),
    .dcbcd_q(ones)

);
endmodule
```

Seven Segment Display Driver:
This module accepts a 4-bit number input and generates a corresponding 7-bit code to drive a seven-segment display, showcasing the symbol representing that specific number. With just two ports, it features a 4-bit input and a 7-bit output.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/07/2023 04:39:38 PM
// Design Name:
// Module Name: ssd_driver_simple
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module ssd_driver_simple(
    input [3:0] ssd_driver_port_in,
    output [6:0] ssd_driver_port_cc
    );

    reg [6:0] ssd_driver_temp_cc;
    wire [3:0] ssd_driver_digits;
    reg [7:0] ssd_driver_an_temp;
    assign ssd_driver_digits = ssd_driver_port_in;

    always @(ssd_driver_digits)
        begin:SEG_EN

            case(ssd_driver_digits)
                4'h0: ssd_driver_temp_cc = 7'b1000000;
                4'h1: ssd_driver_temp_cc = 7'b1111001;
                4'h2: ssd_driver_temp_cc = 7'b0100100;
                4'h3: ssd_driver_temp_cc = 7'b0110000;
                4'h4: ssd_driver_temp_cc = 7'b0011001;
                4'h5: ssd_driver_temp_cc = 7'b0010010;
                4'h6: ssd_driver_temp_cc = 7'b0000010;
```

```
                4'h7: ssd_driver_temp_cc = 7'b1111000;
                4'h8: ssd_driver_temp_cc = 7'b0000000;
                4'h9: ssd_driver_temp_cc = 7'b0010000;
                4'hA: ssd_driver_temp_cc = 7'b0001000;
                4'hB: ssd_driver_temp_cc = 7'b0000011;
                4'hC: ssd_driver_temp_cc = 7'b1000110;
                4'hD: ssd_driver_temp_cc = 7'b0100001;
                4'hE: ssd_driver_temp_cc = 7'b0000110;
                4'hF: ssd_driver_temp_cc = 7'b0001110;
                default: ssd_driver_temp_cc = 7'hzz;
            endcase
        end
    assign ssd_driver_port_cc = ssd_driver_temp_cc;
endmodule
```

Seven Segment Display Controller:

The controller processes all seven-segment display codes across the project, directing them to the board's 8-digit display. It operates with the board's 100MHz clock and a counter, refreshing each display digit every 2ms for simultaneous appearance of all numbers. Among the eight digits, the initial three exhibit the player's ongoing inputs. Following these, the subsequent three digits present the current game solution—serving as a demonstrative aid for observing the game's progression. Finally, the last two digits showcase the current countdown timer value.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/07/2023 04:41:16 PM
// Design Name:
// Module Name: ssd_controller_8_digit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module ssd_controller_8_digit(
    input ssd_controller_clk,
```

```
    input ssd_controller_rst,
    input [55:0] ssd_controller_cc_in,
    output reg [6:0] ssd_controller_cc_out,
    output reg [7:0] ssd_controller_an_out
    );

reg [2:0] digit_sel;
reg [15:0] digit_refresh_counter;

always@(posedge ssd_controller_clk)
    begin
        if (ssd_controller_rst)
            begin: RST
            digit_sel <= 0;
            digit_refresh_counter <= 0;
            end: RST
        else if (digit_refresh_counter == 16'd50000)    // 2ms = 100MHz / 50,000. 2
ms = display_on time
            begin: DISP_REFRESH
                digit_refresh_counter <= 0;
                digit_sel <= digit_sel + 1;
            end: DISP_REFRESH
        else //Continue counting
            digit_refresh_counter <= digit_refresh_counter + 1;
    end

always@(*)
    begin: AN_SELECT
        case(digit_sel)                                 //Digit from right to left
            3'b000: begin
                    ssd_controller_an_out = 8'b11111110; //0 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[6:0];
                  end
            3'b001: begin
                    ssd_controller_an_out = 8'b11111101; //1 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[13:7];
                    end
            3'b010: begin
                    ssd_controller_an_out = 8'b11111011; //2 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[20:14];
                    end
            3'b011: begin
                    ssd_controller_an_out = 8'b11110111; //3 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[27:21];
                    end
```

```
            3'b100: begin
                    ssd_controller_an_out = 8'b11101111; //4 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[34:28];
                    end
            3'b101: begin
                    ssd_controller_an_out = 8'b11011111; //5 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[41:35];
                    end
            3'b110: begin
                    ssd_controller_an_out = 8'b10111111; //6 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[48:42];
                    end
            3'b111: begin
                    ssd_controller_an_out = 8'b01111111; //7 digit
                    ssd_controller_cc_out = ssd_controller_cc_in[55:49];
                    end

            default: ssd_controller_an_out = 8'b00000000;
        endcase
    end
endmodule
```

XDC Constraint File:

An XDC file in FPGA programming, specifically in Xilinx's Vivado software, is crucial for mapping the logical elements defined in the Verilog code to the physical pins on the FPGA. Creating a custom XDC file requires understanding the FPGA's pin-to-location mapping and electrical standards, which can be challenging for beginners due to the intricate constraints imposed by the physical board layout and its associated electrical properties. Below is the XDC file provided by Xilinx, which was edited to implement the project's code on the Nexys A7 board.

```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal
names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { sys_clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {sys_clk}];


##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports {
global_enable_switch }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }];
```

```
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { num3_switches[0]
}]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { num3_switches[1]
}]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { num3_switches[2]
}]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { num3_switches[3]
}]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { num2_switches[0]
}]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { num2_switches[1]
}]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { num2_switches[2]
}]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { num2_switches[3]
}]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { num1_switches[0]
}]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { num1_switches[1]
}]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { num1_switches[2]
}]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { num1_switches[3]
}]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
#set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
#set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[0] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[1] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[2] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[3] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[4] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports {
```

```
current_lock_combo_leds[5] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[6] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[7] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[8] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[9] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[10] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports {
current_lock_combo_leds[11] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

## RGB LEDs
#set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { LED16_B }];
#IO_L5P_T0_D06_14 Sch=led16_b
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { LED16_G }];
#IO_L10P_T1_D14_14 Sch=led16_g
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { LED16_R }];
#IO_L11P_T1_SRCC_14 Sch=led16_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { LED17_B }];
#IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
#set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports { LED17_G }];
#IO_0_14 Sch=led17_g
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { LED17_R }];
#IO_L11N_T1_SRCC_14 Sch=led17_r

##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { cc_out[0] }];
#IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { cc_out[1] }];
#IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { cc_out[2] }];
#IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { cc_out[3] }];
#IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { cc_out[4] }];
#IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { cc_out[5] }];
#IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { cc_out[6] }];
#IO_L4P_T0_D04_14 Sch=cg
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }];
#IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { an_out[0] }];
#IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { an_out[1] }];
#IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { an_out[2] }];
```

```
#IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { an_out[3] }];
#IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { an_out[4] }];
#IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { an_out[5] }];
#IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { an_out[6] }];
#IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { an_out[7] }];
#IO_L23N_T3_A02_D18_14 Sch=an[7]

##CPU Reset Button
set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { start_game_button
}]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { sys_rst_button }];
#IO_L9P_T1_DQS_14 Sch=btnc
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { BTNU }];
#IO_L4N_T0_D05_14 Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { BTNL }];
#IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { BTNR }];
#IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { clk_reset_button
}]; #IO_L9N_T1_DQS_D13_14 Sch=btnd


##Pmod Headers
##Pmod Header JA
#set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { JA[1] }];
#IO_L20N_T3_A19_15 Sch=ja[1]
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { JA[2] }];
#IO_L21N_T3_DQS_A18_15 Sch=ja[2]
#set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { JA[3] }];
#IO_L21P_T3_DQS_15 Sch=ja[3]
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { JA[4] }];
#IO_L18N_T2_A23_15 Sch=ja[4]
#set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { JA[7] }];
#IO_L16N_T2_A27_15 Sch=ja[7]
#set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { JA[8] }];
#IO_L16P_T2_A28_15 Sch=ja[8]
#set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports { JA[9] }];
#IO_L22N_T3_A16_15 Sch=ja[9]
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { JA[10] }];
#IO_L22P_T3_A17_15 Sch=ja[10]

##Pmod Header JB
#set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { JB[1] }];
```

```
#IO_L1P_T0_AD0P_15 Sch=jb[1]
#set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { JB[2] }];
#IO_L14N_T2_SRCC_15 Sch=jb[2]
#set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { JB[3] }];
#IO_L13N_T2_MRCC_15 Sch=jb[3]
#set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { JB[4] }];
#IO_L15P_T2_DQS_15 Sch=jb[4]
#set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { JB[7] }];
#IO_L11N_T1_SRCC_15 Sch=jb[7]
#set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { JB[8] }];
#IO_L5P_T0_AD9P_15 Sch=jb[8]
#set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { JB[9] }];
#IO_0_15 Sch=jb[9]
#set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { JB[10] }];
#IO_L13P_T2_MRCC_15 Sch=jb[10]

##Pmod Header JC
#set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { JC[1] }];
#IO_L23N_T3_35 Sch=jc[1]
#set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33 } [get_ports { JC[2] }];
#IO_L19N_T3_VREF_35 Sch=jc[2]
#set_property -dict { PACKAGE_PIN J2    IOSTANDARD LVCMOS33 } [get_ports { JC[3] }];
#IO_L22N_T3_35 Sch=jc[3]
#set_property -dict { PACKAGE_PIN G6    IOSTANDARD LVCMOS33 } [get_ports { JC[4] }];
#IO_L19P_T3_35 Sch=jc[4]
#set_property -dict { PACKAGE_PIN E7    IOSTANDARD LVCMOS33 } [get_ports { JC[7] }];
#IO_L6P_T0_35 Sch=jc[7]
#set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { JC[8] }];
#IO_L22P_T3_35 Sch=jc[8]
#set_property -dict { PACKAGE_PIN J4    IOSTANDARD LVCMOS33 } [get_ports { JC[9] }];
#IO_L21P_T3_DQS_35 Sch=jc[9]
#set_property -dict { PACKAGE_PIN E6    IOSTANDARD LVCMOS33 } [get_ports { JC[10] }];
#IO_L5P_T0_AD13P_35 Sch=jc[10]

##Pmod Header JD
#set_property -dict { PACKAGE_PIN H4    IOSTANDARD LVCMOS33 } [get_ports { JD[1] }];
#IO_L21N_T3_DQS_35 Sch=jd[1]
#set_property -dict { PACKAGE_PIN H1    IOSTANDARD LVCMOS33 } [get_ports { JD[2] }];
#IO_L17P_T2_35 Sch=jd[2]
#set_property -dict { PACKAGE_PIN G1    IOSTANDARD LVCMOS33 } [get_ports { JD[3] }];
#IO_L17N_T2_35 Sch=jd[3]
#set_property -dict { PACKAGE_PIN G3    IOSTANDARD LVCMOS33 } [get_ports { JD[4] }];
#IO_L20N_T3_35 Sch=jd[4]
#set_property -dict { PACKAGE_PIN H2    IOSTANDARD LVCMOS33 } [get_ports { JD[7] }];
#IO_L15P_T2_DQS_35 Sch=jd[7]
#set_property -dict { PACKAGE_PIN G4    IOSTANDARD LVCMOS33 } [get_ports { JD[8] }];
#IO_L20P_T3_35 Sch=jd[8]
#set_property -dict { PACKAGE_PIN G2    IOSTANDARD LVCMOS33 } [get_ports { JD[9] }];
#IO_L15N_T2_DQS_35 Sch=jd[9]
#set_property -dict { PACKAGE_PIN F3    IOSTANDARD LVCMOS33 } [get_ports { JD[10] }];
```

```
#IO_L13N_T2_MRCC_35 Sch=jd[10]

##Pmod Header JXADC
#set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports { XA_N[1] }];
#IO_L9N_T1_DQS_AD3N_15 Sch=xa_n[1]
#set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports { XA_P[1] }];
#IO_L9P_T1_DQS_AD3P_15 Sch=xa_p[1]
#set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports { XA_N[2] }];
#IO_L8N_T1_AD10N_15 Sch=xa_n[2]
#set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports { XA_P[2] }];
#IO_L8P_T1_AD10P_15 Sch=xa_p[2]
#set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports { XA_N[3] }];
#IO_L7N_T1_AD2N_15 Sch=xa_n[3]
#set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports { XA_P[3] }];
#IO_L7P_T1_AD2P_15 Sch=xa_p[3]
#set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports { XA_N[4] }];
#IO_L10N_T1_AD11N_15 Sch=xa_n[4]
#set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { XA_P[4] }];
#IO_L10P_T1_AD11P_15 Sch=xa_p[4]

##VGA Connector
set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { colordata[8] }];
#IO_L8N_T1_AD14N_35 Sch=vga_r[0]
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { colordata[9] }];
#IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { colordata[10] }];
#IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { colordata[11] }];
#IO_L8P_T1_AD14P_35 Sch=vga_r[3]
set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { colordata[4] }];
#IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { colordata[5] }];
#IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { colordata[6] }];
#IO_L2N_T0_AD12N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { colordata[7] }];
#IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]
set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { colordata[0] }];
#IO_L2P_T0_AD12P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { colordata[1] }];
#IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { colordata[2] }];
#IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { colordata[3] }];
#IO_L4P_T0_35 Sch=vga_b[3]
set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { hsync }];
#IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { vsync }];
#IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

```
##Micro SD Connector
#set_property -dict { PACKAGE_PIN E2    IOSTANDARD LVCMOS33 } [get_ports { SD_RESET }];
#IO_L14P_T2_SRCC_35 Sch=sd_reset
#set_property -dict { PACKAGE_PIN A1    IOSTANDARD LVCMOS33 } [get_ports { SD_CD }];
#IO_L9N_T1_DQS_AD7N_35 Sch=sd_cd
#set_property -dict { PACKAGE_PIN B1    IOSTANDARD LVCMOS33 } [get_ports { SD_SCK }];
#IO_L9P_T1_DQS_AD7P_35 Sch=sd_sck
#set_property -dict { PACKAGE_PIN C1    IOSTANDARD LVCMOS33 } [get_ports { SD_CMD }];
#IO_L16N_T2_35 Sch=sd_cmd
#set_property -dict { PACKAGE_PIN C2    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[0] }];
#IO_L16P_T2_35 Sch=sd_dat[0]
#set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[1] }];
#IO_L18N_T2_35 Sch=sd_dat[1]
#set_property -dict { PACKAGE_PIN F1    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[2] }];
#IO_L18P_T2_35 Sch=sd_dat[2]
#set_property -dict { PACKAGE_PIN D2    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[3] }];
#IO_L14N_T2_SRCC_35 Sch=sd_dat[3]

##Accelerometer
#set_property -dict { PACKAGE_PIN E15   IOSTANDARD LVCMOS33 } [get_ports { ACL_MISO }];
#IO_L11P_T1_SRCC_15 Sch=acl_miso
#set_property -dict { PACKAGE_PIN F14   IOSTANDARD LVCMOS33 } [get_ports { ACL_MOSI }];
#IO_L5N_T0_AD9N_15 Sch=acl_mosi
#set_property -dict { PACKAGE_PIN F15   IOSTANDARD LVCMOS33 } [get_ports { ACL_SCLK }];
#IO_L14P_T2_SRCC_15 Sch=acl_sclk
#set_property -dict { PACKAGE_PIN D15   IOSTANDARD LVCMOS33 } [get_ports { ACL_CSN }];
#IO_L12P_T1_MRCC_15 Sch=acl_csn
#set_property -dict { PACKAGE_PIN B13   IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[1] }];
#IO_L2P_T0_AD8P_15 Sch=acl_int[1]
#set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[2] }];
#IO_L20P_T3_A20_15 Sch=acl_int[2]

##Temperature Sensor
#set_property -dict { PACKAGE_PIN C14   IOSTANDARD LVCMOS33 } [get_ports { TMP_SCL }];
#IO_L1N_T0_AD0N_15 Sch=tmp_scl
#set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 } [get_ports { TMP_SDA }];
#IO_L12N_T1_MRCC_15 Sch=tmp_sda
#set_property -dict { PACKAGE_PIN D13   IOSTANDARD LVCMOS33 } [get_ports { TMP_INT }];
#IO_L6N_T0_VREF_15 Sch=tmp_int
#set_property -dict { PACKAGE_PIN B14   IOSTANDARD LVCMOS33 } [get_ports { TMP_CT }];
#IO_L2N_T0_AD8N_15 Sch=tmp_ct

##Omnidirectional Microphone
#set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33 } [get_ports { M_CLK }];
#IO_25_35 Sch=m_clk
#set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 } [get_ports { M_DATA }];
#IO_L24N_T3_35 Sch=m_data
#set_property -dict { PACKAGE_PIN F5    IOSTANDARD LVCMOS33 } [get_ports { M_LRSEL }];
#IO_0_35 Sch=m_lrsel
```

```
##PWM Audio Amplifier
#set_property -dict { PACKAGE_PIN A11   IOSTANDARD LVCMOS33 } [get_ports { AUD_PWM }];
#IO_L4N_T0_15 Sch=aud_pwm
#set_property -dict { PACKAGE_PIN D12   IOSTANDARD LVCMOS33 } [get_ports { AUD_SD }];
#IO_L6P_T0_15 Sch=aud_sd

##USB-RS232 Interface
#set_property -dict { PACKAGE_PIN C4    IOSTANDARD LVCMOS33 } [get_ports { UART_TXD_IN }];
#IO_L7P_T1_AD6P_35 Sch=uart_txd_in
#set_property -dict { PACKAGE_PIN D4    IOSTANDARD LVCMOS33 } [get_ports { UART_RXD_OUT }];
#IO_L11N_T1_SRCC_35 Sch=uart_rxd_out
#set_property -dict { PACKAGE_PIN D3    IOSTANDARD LVCMOS33 } [get_ports { UART_CTS }];
#IO_L12N_T1_MRCC_35 Sch=uart_cts
#set_property -dict { PACKAGE_PIN E5    IOSTANDARD LVCMOS33 } [get_ports { UART_RTS }];
#IO_L5N_T0_AD13N_35 Sch=uart_rts

##USB HID (PS/2)
#set_property -dict { PACKAGE_PIN F4    IOSTANDARD LVCMOS33 } [get_ports { PS2_CLK }];
#IO_L13P_T2_MRCC_35 Sch=ps2_clk
#set_property -dict { PACKAGE_PIN B2    IOSTANDARD LVCMOS33 } [get_ports { PS2_DATA }];
#IO_L10N_T1_AD15N_35 Sch=ps2_data

##SMSC Ethernet PHY
#set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 } [get_ports { ETH_MDC }];
#IO_L11P_T1_SRCC_16 Sch=eth_mdc
#set_property -dict { PACKAGE_PIN A9    IOSTANDARD LVCMOS33 } [get_ports { ETH_MDIO }];
#IO_L14N_T2_SRCC_16 Sch=eth_mdio
#set_property -dict { PACKAGE_PIN B3    IOSTANDARD LVCMOS33 } [get_ports { ETH_RSTN }];
#IO_L10P_T1_AD15P_35 Sch=eth_rstn
#set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { ETH_CRSDV }];
#IO_L6N_T0_VREF_16 Sch=eth_crsdv
#set_property -dict { PACKAGE_PIN C10   IOSTANDARD LVCMOS33 } [get_ports { ETH_RXERR }];
#IO_L13N_T2_MRCC_16 Sch=eth_rxerr
#set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33 } [get_ports { ETH_RXD[0] }];
#IO_L13P_T2_MRCC_16 Sch=eth_rxd[0]
#set_property -dict { PACKAGE_PIN D10   IOSTANDARD LVCMOS33 } [get_ports { ETH_RXD[1] }];
#IO_L19N_T3_VREF_16 Sch=eth_rxd[1]
#set_property -dict { PACKAGE_PIN B9    IOSTANDARD LVCMOS33 } [get_ports { ETH_TXEN }];
#IO_L11N_T1_SRCC_16 Sch=eth_txen
#set_property -dict { PACKAGE_PIN A10   IOSTANDARD LVCMOS33 } [get_ports { ETH_TXD[0] }];
#IO_L14P_T2_SRCC_16 Sch=eth_txd[0]
#set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { ETH_TXD[1] }];
#IO_L12N_T1_MRCC_16 Sch=eth_txd[1]
#set_property -dict { PACKAGE_PIN D5    IOSTANDARD LVCMOS33 } [get_ports { ETH_REFCLK }];
#IO_L11P_T1_SRCC_35 Sch=eth_refclk
#set_property -dict { PACKAGE_PIN B8    IOSTANDARD LVCMOS33 } [get_ports { ETH_INTN }];
#IO_L12P_T1_MRCC_16 Sch=eth_intn

##Quad SPI Flash
#set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[0] }];
```

```
#IO_L1P_T0_D00_MOSI_14 Sch=qspi_dq[0]
#set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[1] }];
#IO_L1N_T0_D01_DIN_14 Sch=qspi_dq[1]
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[2] }];
#IO_L2P_T0_D02_14 Sch=qspi_dq[2]
#set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[3] }];
#IO_L2N_T0_D03_14 Sch=qspi_dq[3]
#set_property -dict { PACKAGE_PIN L13   IOSTANDARD LVCMOS33 } [get_ports { QSPI_CSN }];
#IO_L6P_T0_FCS_B_14 Sch=qspi_csn
```

Python Code:

In addition to the Verilog code, we implemented a python to aid in the building of this project. The Python script processes an image, extracting color data from individual pixels and generating a module where this data is stored in either row or column registers. Initially, the script processes the chosen image using the CV2 library, generating a file named based on the image file's name. It automates the writing of color data for each pixel within a case statement within an always block. The script configures this to trigger on a row or column change, which occurs on the positive edge of the clock, enabling retrieval of the recorded color data inserted into the block by the Python script. The resulting module length typically reaches around 5000 lines (at 100x50 resolution), but may vary depending on the image's resolution, potentially resulting in longer or shorter modules.

```
# converts image to Verilog HDL that infers a ROM using Xilinx Block RAM
# note: 12-bit color map word is r3, r2, r1, r0, g3, g2, g1, g0, b3, b2,
b1, b0

import math
import cv2
import os

# returns string of 12-bit color at row x, column y of image
def get_color_bits(im, y, x):
    # convert color components to byte string and slice needed upper bits
    b  = format(im[y][x][0], 'b').zfill(8)
    rx = b[0:4]
    b  = format(im[y][x][1], 'b').zfill(8)
    gx = b[0:4]
    b  = format(im[y][x][2], 'b').zfill(8)
    bx = b[0:4]

    # return concatenation of RGB bits
    return str(rx+gx+bx)
```

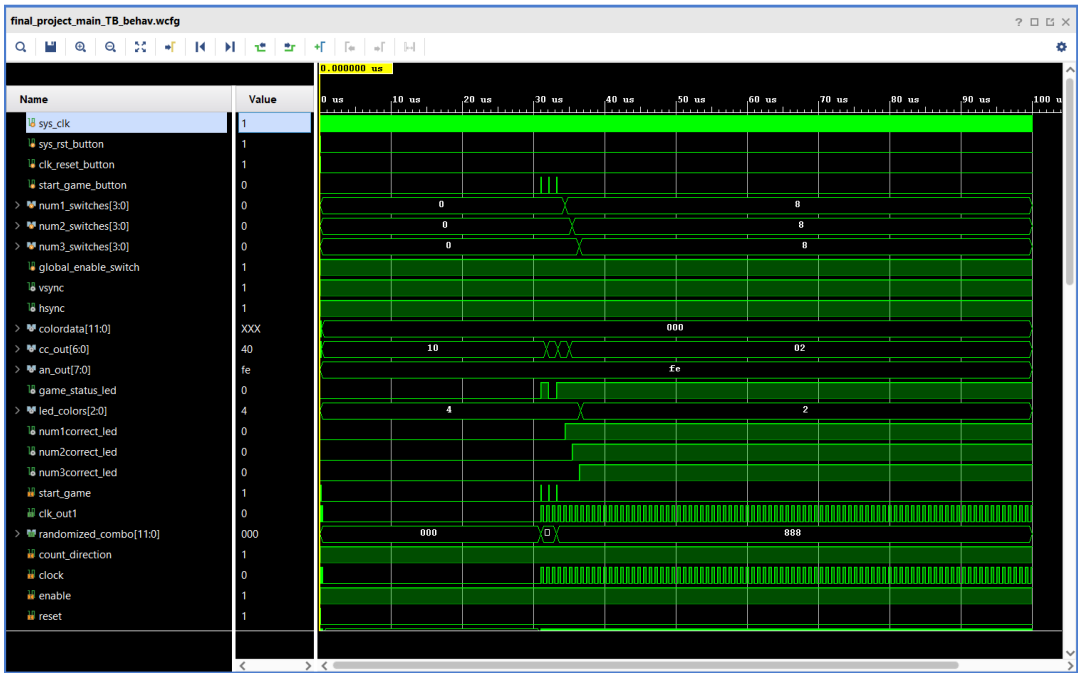
```
# write to file Verilog HDL
# takes name of file, image array,
# pixel coordinates of background color to mask as 0
def rom_12_bit(name, im, mask=False, rem_x=-1, rem_y=-1):

    # get colorbyte of background color
    # if coordinates left at default, map all data without masking
    if rem_x == -1 or rem_y == -1:
        a = "000000000000"

    # else set mask compare byte
    else:
        a = get_color_bits(im, rem_x, rem_y)

    # make output filename from input
    file_name = name.split('.')[0] + "_12_bit_rom.v"

    # open file
    f = open(file_name, 'w')

    # get image dimensions
    y_max, x_max, z = im.shape

    # get width of row and column case words
    row_width = math.ceil(math.log(y_max-1,2))
    col_width = math.ceil(math.log(x_max-1,2))

    # write beginning part of module up to case statements
    f.write("module " + name.split('.')[0] + "_rom\n\t(\n\t\t")
    f.write("input wire clk,\n\t\tinput wire [" + str(row_width-1) + ":0]
row,\n\t\t")
    f.write("input wire [" + str(col_width-1) + ":0] col,\n\t\t")
    f.write("output reg [11:0] color_data\n\t);\n\n\t")
    f.write("(* rom_style = \"block\" *)\n\n\t//signal declaration\n\t")
    f.write("reg [" + str(row_width-1) + ":0] row_reg;\n\t")
    f.write("reg [" + str(col_width-1) + ":0] col_reg;\n\n\t")
    f.write("always @(posedge clk)\n\t\tbegin\n\t\trow_reg <=
row;\n\t\tcol_reg <= col;\n\t\tend\n\n\t")
    f.write("always @*\n\tcase ({row_reg, col_reg})\n")
```

```
    # loops through y rows and x columns
    for y in range(y_max):
        for x in range(x_max):
            # write : color_data =
            case = format(y, 'b').zfill(row_width) + format(x,
'b').zfill(col_width)
            f.write("\t\t" + str(row_width + col_width) + "'b" + case + ":
color_data = " + str(12) + "'b")

            # if mask is set to false, just write color data
            if(mask == False):
                f.write(get_color_bits(im, y, x))
                f.write(";\n")

            elif(get_color_bits(im, y, x) != a):
                # write color bits to file
                f.write(get_color_bits(im, y, x))
                f.write(";\n")

            else:
                f.write("000000000000;\n")

        f.write("\n")

    # write end of module
    f.write("\t\tdefault: color_data =
12'b000000000000;\n\tendcase\nendmodule")
    # close file
    f.close()

# driver function
def generate(name, rem_x=-1, rem_y=-1):
    if not os.path.exists(name):
        print(f"Error: File '{name}' not found.")
        return

    im = cv2.imread(name,cv2.IMREAD_COLOR)
    print("width: " + str(im.shape[1]) + ", height: " + str(im.shape[0]))
    rom_12_bit(name, im)
# generate rom from full bitmap image
generate("monkey_blur_2_200_100.jpg")
```

Test Bench and Simulation:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/09/2023 10:24:27 AM
// Design Name:
// Module Name: game_down_counter_TB
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module game_down_counter_TB();
    reg sys_clk;
    reg enable;
    reg counter_reset;
    reg clk_reset;
    wire [3:0] tens;
    wire [3:0] ones;

game_down_counter DUT(
    .sys_clk(sys_clk),
    .enable(enable),
    .counter_reset(counter_reset),
    .clk_reset(clk_reset),
    .tens(tens),
    .ones(ones)
);

initial
    begin
        sys_clk = 1;
        enable = 0;
```

```
        counter_reset = 1;
        clk_reset = 1;

        #30 clk_reset = 0;
        #30 counter_reset = 0;
        #20 enable = 1;

    end

always
    begin
        #10 sys_clk = ~sys_clk;
    end
endmodule
```



Implementation on Nexys A7 FPGA Board:
Test benches were meticulously developed for key modules before their integration into the main project. Comprehensive assessments were conducted, observing module behaviors in both normal operations and corner cases. Subsequently, following the generation of a bitstream and board programming, extensive testing of the game's objectives included evaluating expected player actions and exploring unforeseen scenarios. This comprehensive testing aimed to uncover and address potential bugs or anomalies arising from actions beyond the defined scope of the game.

During physical testing, evaluations spanned routine operations. This included testing of the seven-segment displays, LEDs, various switches, and buttons used to interact with the game. As each element was tested, game states were noted, and outputs were evaluated to verify the fidelity of player inputs. Any errors encountered were subsequently dealt with. However, there were some unexpected operations which then became features of our design.
Exploration into these abnormal operations revealed unexpected scenarios, such as initiating the game while it was already active, which was later repurposed as a game halt feature. Also, altering correct player inputs to incorrect ones, or triggering the start game button before an active game, which remained unresolved due to time constraints. Additionally, a global enable switch, initially integrated for debugging and demonstration purposes, was incidentally disabled during an active game.

Results and Observations:
The division of tasks within the group led to a realization: optimizing our design's efficiency required the reduction of redundant modules. Specifically, these redundancies manifested in multiple versions of a module, each imported by individual group members when designing higher-level modules that instantiated similar ones (e.g., multiple counters). To address the challenge posed by redundant modules resulting from individual imports through the group's task division, a systematic identifying of overlapping functionalities across imported modules was applied. This initiative consolidated similar functionalities into singular, standardized modules, uniformly utilized across higher-level designs. Testing ensured that consolidated modules fulfilled diverse design requirements without compromising functionality.
Additionally, through this project, a key learning emerged. It encompassed the process of initializing the board's BRAM with data, particularly focusing on image conversion and its storage within BRAM. An error arose during the image-to-coefficient file (.coe) conversion phase. The issue stemmed from data offsetting caused by storing pixel data in 8-bit wide codes within the coefficient file, incompatible with the Nexys A7-100T's VGA port requirements, which necessitated 12-bit wide codes for proper functionality. This experience underscored the significance of effectively transforming an image into a format suitable for initialization and retrieval from BRAM, ultimately pivotal in accurately displaying the image on a monitor.

Conclusion:
This project demonstrates the integration of theoretical concepts from ECE 3300 with hands-on-application using the Nexys A7 FPGA board. Through the practical implementation of Verilog principles, the objective of decoding a 3-digit padlock within a time limit to reveal an image is evidence of Verilog's practical utilization. Its role in managing game progression, timer initiation, interpreting switch inputs, and synchronizing game elements emphasizes its real-time operational functionality. As the foundational framework, Verilog describes the logic circuits governing switches, LEDs, displays, and timers, facilitating control within the game and on the FPGA board. Notably, the comprehensive testing, module assessments, and evaluations uncovered unforeseen scenarios, which were then integrated into features or addressed. Moreover, this project emphasized the imperative of optimized design efficiency by streamlining redundant modules and consolidating functionalities into standardized modules. This approach

guaranteed meeting diverse design requirements without compromising functionality. And, the experience with image conversion underscored the criticality of proper data transformation for image display, emphasizing data compatibility's role for proper functionality. *The Art Thief* stands as an example of the progression from theoretical understanding to practical implementation, highlighting the skilled application of Verilog principles in digital systems. It emphasizes the iterative learning integral to FPGA-based game design, showcasing the fusion of theory and practicality fostered throughout this course.

References:

Python code retrieved from:
https://embeddedthoughts.com/2016/07/30/storing-image-data-in-block-ram-on-a-xilinx-fpga/

Nexys A7 reference:
https://digilent.com/reference/_media/programmable-logic/nexys-a7/nexys-a7-d3-sch.pdf