|  |
| --- |
| **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.

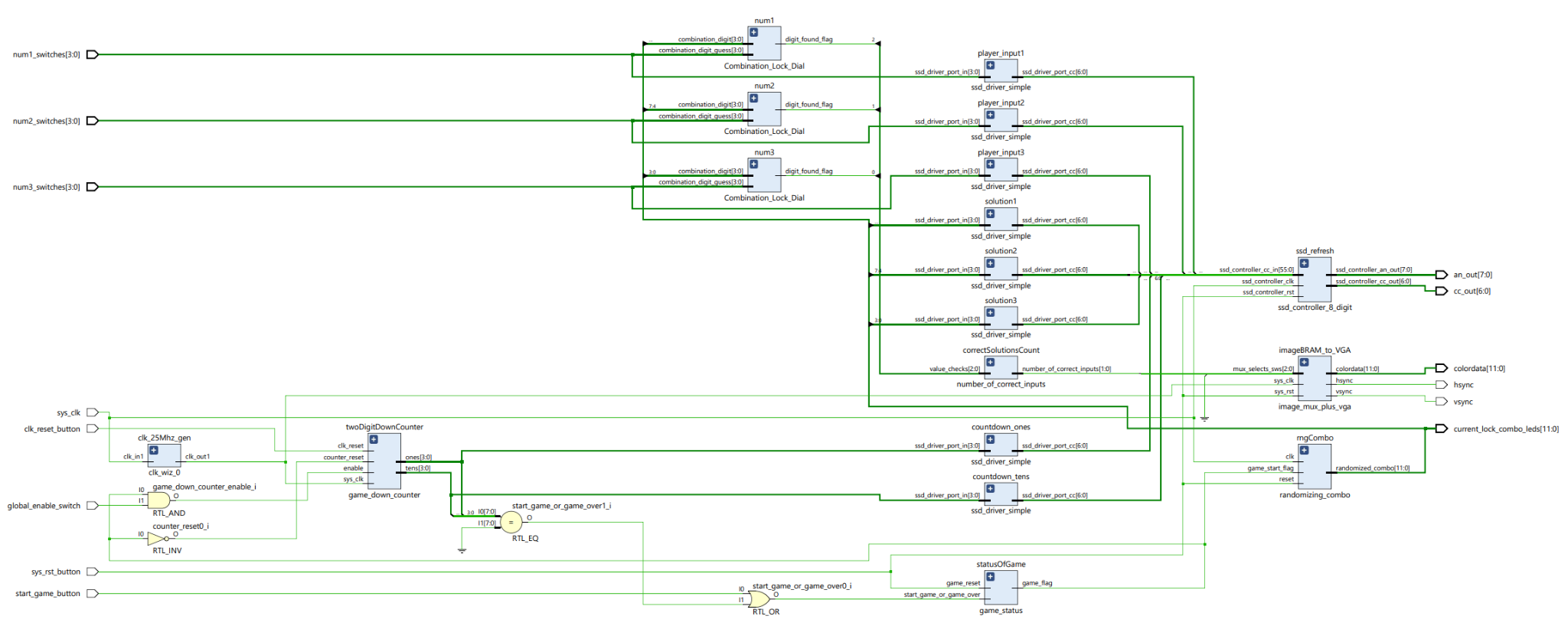
**Design Specifications:**

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 |
| --- |

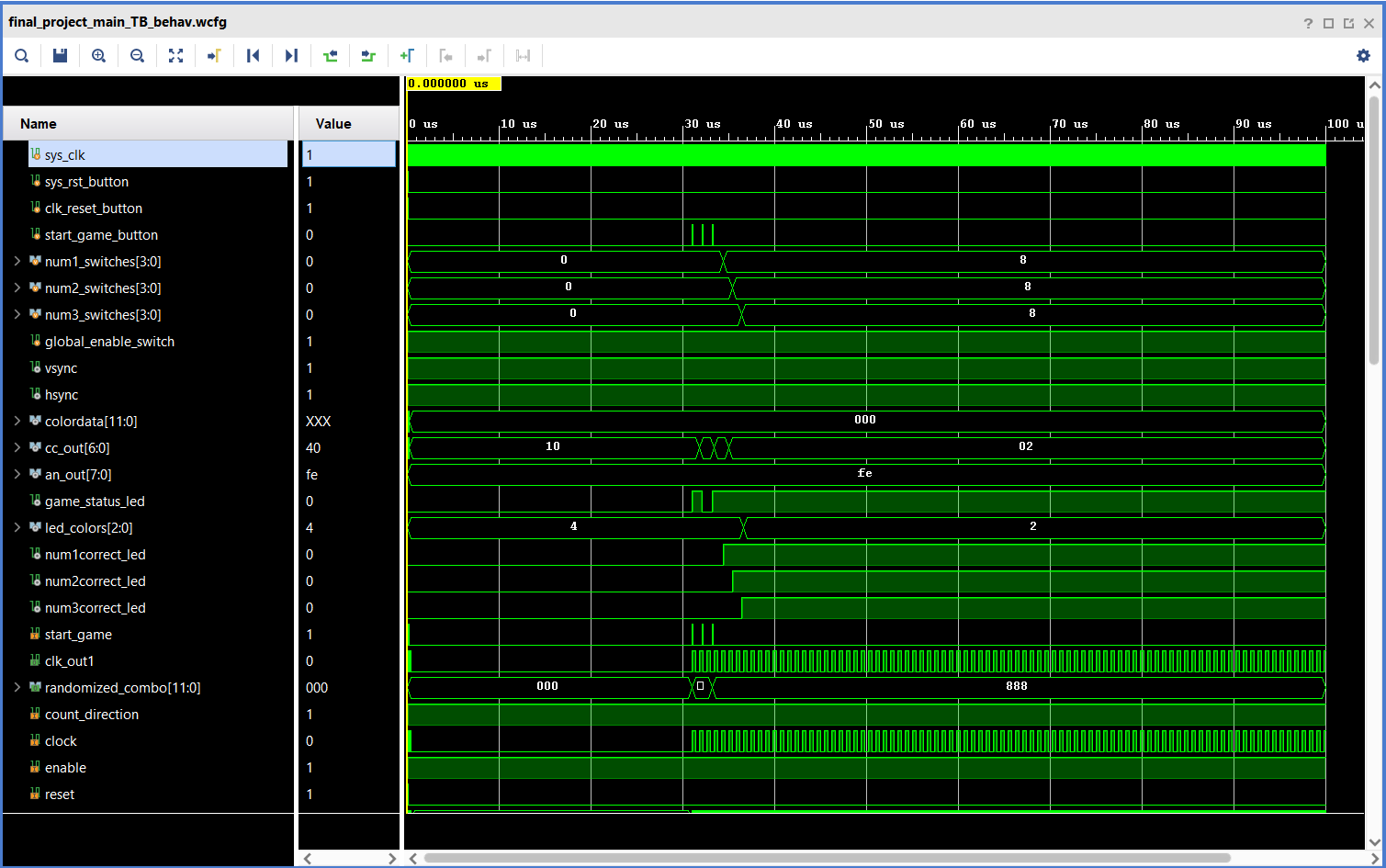
*Python Code:*

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>