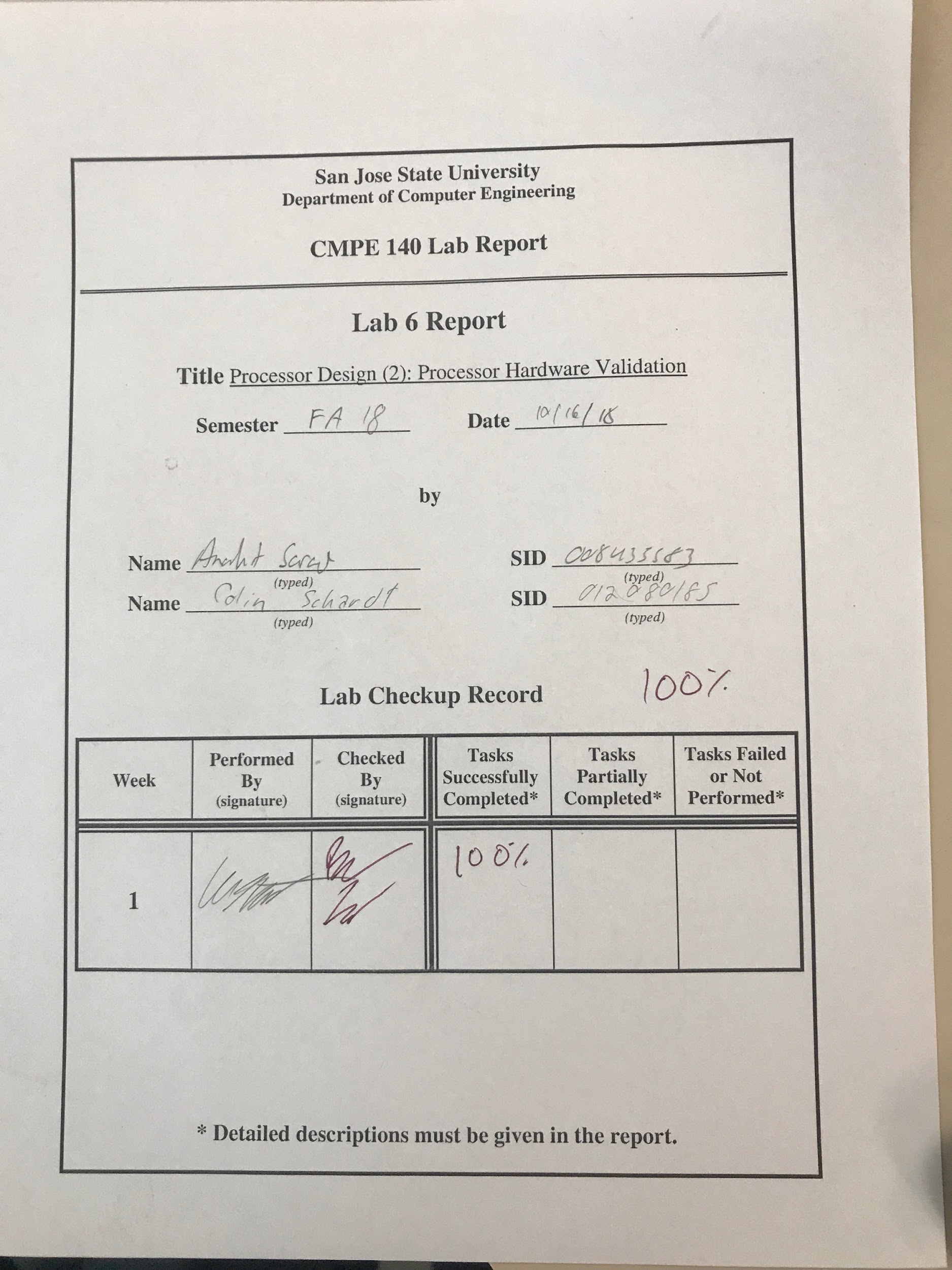
· 

***Design Methodology***

This is lab is an extension of lab 5, the lab requires full hardware validation testing for the single-process cpu. Additional files were added to the project such as the xdc constraints file and multiple design files. Provided below is a block diagram showing the testing environment. The learning outcome for this lab was to validate the MIPS processor design using the FPGA board. This allows the cpu to undergo software and hardware validation.

Tasks which were completed in lab:

- Diagram showing validation environment setup.

- Verify the hardware functionality using the FPGA.

- Compare the FPGA validation results with test log from Lab 2.

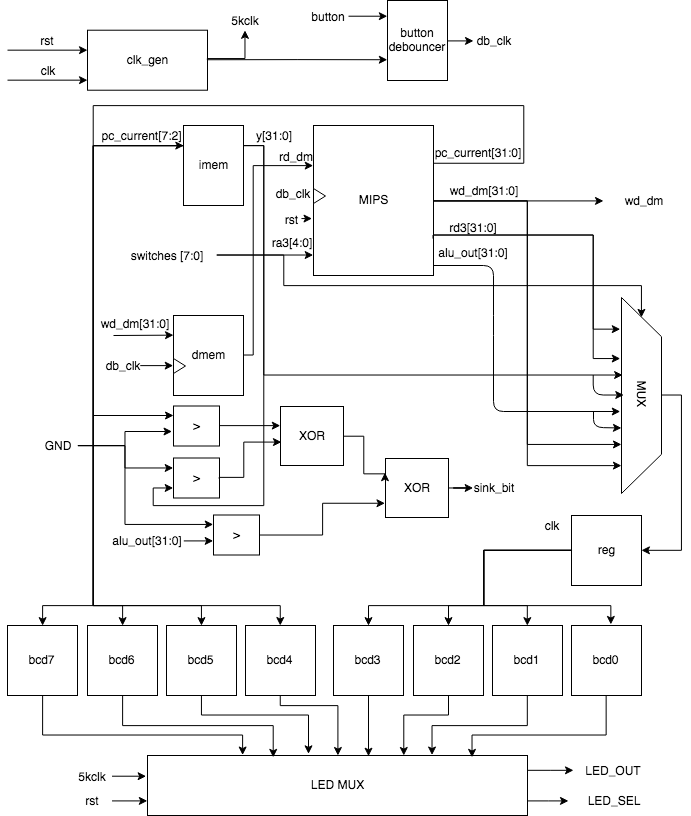


Figure 1. Validation Environment Setup

| Adr | Expected Machine Code | Actual Machine Code | PC | Registers | | | | | Memory Content | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $v0 | $v1 | $a0 | $a1 | $a3 | [80] | [84] |
| 00 | 20020005 | 20020005 | 0 | 5 |  |  |  |  |  |  |
| 04 | 2003000c | 2003000c | 4 | 5 | 12 |  |  |  |  |  |
| 08 | 2067fff7 | 2067fff7 | 8 | 5 | 12 |  |  | 3 |  |  |
| 0c | 00e22025 | 00e22025 | C | 5 | 12 | 7 |  | 3 |  |  |
| 10 | 00642824 | 00642824 | 10 | 5 | 12 | 7 | 4 | 3 |  |  |
| 14 | 00a42820 | 00a42820 | 14 | 5 | 12 | 7 | 11 | 3 |  |  |
| 18 | 10a7000a | 10a7000a | 18 | 5 | 12 | 7 | 11 | 3 |  |  |
| 1c | 0064202a | 0064202a | 1c | 5 | 12 | 0 | 11 | 3 |  |  |
| 20 | 10800001 | 10800001 | 20 | 5 | 12 | 0 | 11 | 3 |  |  |
| 24 | 20050000 | Not taken |  |  |  |  |  |  |  |  |
| 28 | 00e2202a | 00e2202a | 28 | 5 | 12 | 1 | 11 | 3 |  |  |
| 2c | 00853820 | 00853820 | 2c | 5 | 12 | 1 | 11 | 12 |  |  |
| 30 | 00e23822 | 00e23822 | 30 | 5 | 12 | 1 | 11 | 7 |  |  |
| 34 | ac670044 | ac670044 | 34 | 5 | 12 | 1 | 11 | 7 | 7 |  |
| 38 | 8c020050 | 8c020050 | 38 | 7 | 12 | 1 | 11 | 7 | 7 |  |
| 3c | 08000011 | 08000c11 | 3c | 7 | 12 | 1 | 11 | 7 | 7 |  |
| 40 | 20020001 | Not taken |  |  |  |  |  |  |  |  |
| 44 | ac020054 | ac020054 | 44 | 7 | 12 | 1 | 11 | 7 | 7 | 7 |
| 48 | 08000000 | 08000c00 | 48 | 7 | 12 | 1 | 11 | 7 | 7 | 7 |

Table 1. Test Results Log

***FPGA Test Plan***

For hardware validation the register values, machine code, and memory content were analyzed.

Using the FPGA to output all these values in every clock cycle. The values were compared to the output obtained in lab 2. Similar to the MIPS assembler the values did not change in memory till the next clock cycle.

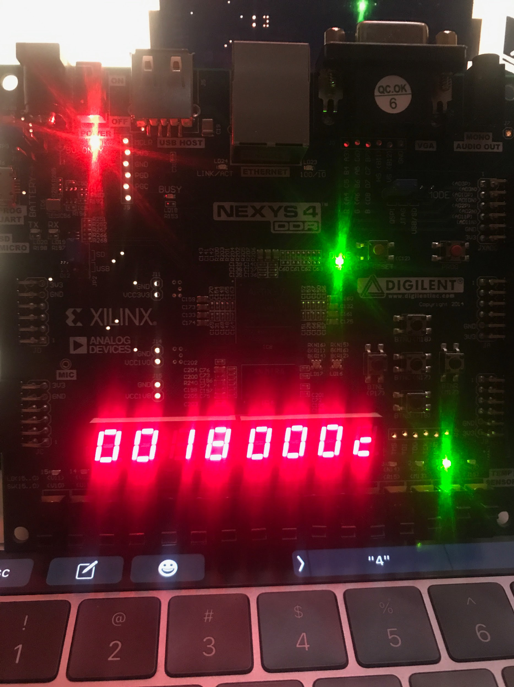


Figure 2. 7-Segment Display Showing Machine Code & Address

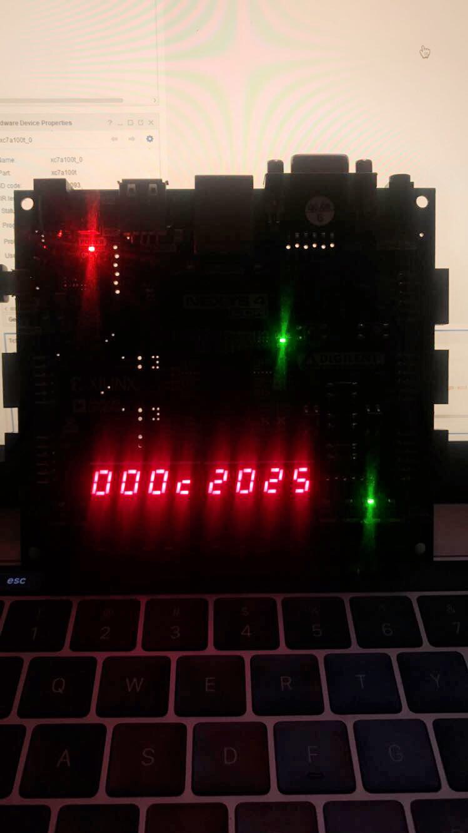


Figure 3. 7-Segment Display Showing Contents of Registers & Address

***Conclusion***

In conclusion the was easily finished as it was straightforward and easy to execute. The main functionality of this lab was to connect the data path and control unit then using a set of control signals to input instructions that would alter the output and control values. The control unit and data path along with the machine code and hardware unit helps understand the change in every instruction set as they are not all the same.

***Source Code***

| mips\_fpga.v |
| --- |
| module mips\_fpga (  input wire clk,  input wire rst,  input wire button,  input wire [7:0] switches,  output wire we\_dm,  output wire [7:0] LEDSEL,  output wire [7:0] LEDOUT  );  reg [15:0] reg\_hex;  wire clk\_sec;  wire clk\_5KHz;  wire clk\_pb;    wire [7:0] digit0;  wire [7:0] digit1;  wire [7:0] digit2;  wire [7:0] digit3;  wire [7:0] digit4;  wire [7:0] digit5;  wire [7:0] digit6;  wire [7:0] digit7;  wire [31:0] pc\_current;  wire [31:0] instr;  wire [31:0] alu\_out;  wire [31:0] wd\_dm;  wire [31:0] rd\_dm;  wire [31:0] dispData;  clk\_gen clk\_gen (  .clk100MHz (clk),  .rst (rst),  .clk\_4sec (clk\_sec),  .clk\_5KHz (clk\_5KHz)  );  button\_debouncer bd (  .clk (clk\_5KHz),  .button (button),  .debounced\_button (clk\_pb)  );  mips\_top mips\_top (  .clk (clk\_pb),  .rst (rst),  .ra3 (switches[4:0]),  .we\_dm (we\_dm),  .pc\_current (pc\_current),  .instr (instr),  .alu\_out (alu\_out),  .wd\_dm (wd\_dm),  .rd\_dm (rd\_dm),  .rd3 (dispData)  );  /\*  switches[4:0] are used as the 3rd read address (ra3) of the RF,  dispData is the register contents from the RF's 3rd read port (rd3).  \*/  hex\_to\_7seg hex7 (  .HEX (pc\_current[15:12]),  .s (digit7)  );  hex\_to\_7seg hex6 (  .HEX (pc\_current[11:8]),  .s (digit6)  );  hex\_to\_7seg hex5 (  .HEX (pc\_current[7:4]),  .s (digit5)  );  hex\_to\_7seg hex4 (  .HEX (pc\_current[3:0]),  .s (digit4)  );  hex\_to\_7seg hex3 (  .HEX (reg\_hex[15:12]),  .s (digit3)  );  hex\_to\_7seg hex2 (  .HEX (reg\_hex[11:8]),  .s (digit2)  );  hex\_to\_7seg hex1 (  .HEX (reg\_hex[7:4]),  .s (digit1)  );  hex\_to\_7seg hex0 (  .HEX (reg\_hex[3:0]),  .s (digit0)  );  led\_mux led\_mux (  .clk (clk\_5KHz),  .rst (rst),  .LED7 (digit7),  .LED6 (digit6),  .LED5 (digit5),  .LED4 (digit4),  .LED3 (digit3),  .LED2 (digit2),  .LED1 (digit1),  .LED0 (digit0),  .LEDSEL (LEDSEL),  .LEDOUT (LEDOUT)  );    /\*  switches[7:5] = 000 : Display lower half word of register selected by switches[4:0]  switches[7:5] = 001 : Display higher half word of register selected by switches[4:0]  switches[7:5] = 010 : Display lower half word of 'instr'  switches[7:5] = 011 : Display higher half word of 'instr'  switches[7:5] = 100 : Display lower half word of 'alu\_out'  switches[7:5] = 101 : Display higher half word of 'alu\_out'  switches[7:5] = 110 : Display lower half word of 'wd\_dm'  switches[7:5] = 111 : Display higher half word of 'wd\_dm'  \*/  always @ (posedge clk) begin  case ({switches[7:5]})  3'b000: reg\_hex = dispData[15:0];  3'b001: reg\_hex = dispData[31:16];  3'b010: reg\_hex = instr[15:0];  3'b011: reg\_hex = instr[31:16];  3'b100: reg\_hex = alu\_out[15:0];  3'b101: reg\_hex = alu\_out[31:16];  3'b110: reg\_hex = wd\_dm[15:0];  3'b111: reg\_hex = wd\_dm[31:16];  endcase  end  endmodule |

| lab6.asm |
| --- |
| main:  addi $2, $0, 5  addi $3, $0, 12  addi $7, $3, -9  or $4, $7, $2  and $5, $3, $4  add $5, $5, $4  beq $5, $7, end  slt $4, $3, $4  beq $4, $0, around  addi $5, $0, 0  around:  slt $4, $7, $2  add $7, $4, $5  sub $7, $7, $2  sw $7, 68($3)  lw $2, 80($0)  j end  addi $2, $0, 1  end:  sw $2, 84($0)  j main |

| mips\_fpga.xdc |
| --- |
| # Clock Signal  set\_property -dict {PACKAGE\_PIN E3 IOSTANDARD LVCMOS33} [get\_ports {clk}];  create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];  # Buttons  set\_property -dict {PACKAGE\_PIN N17 IOSTANDARD LVCMOS33} [get\_ports {button}]; # Center Button  set\_property -dict {PACKAGE\_PIN P17 IOSTANDARD LVCMOS33} [get\_ports {rst}]; # Left Button  # Switches  set\_property -dict {PACKAGE\_PIN J15 IOSTANDARD LVCMOS33} [get\_ports {switches[0]}]; # Switch 0  set\_property -dict {PACKAGE\_PIN L16 IOSTANDARD LVCMOS33} [get\_ports {switches[1]}]; # Switch 1  set\_property -dict {PACKAGE\_PIN M13 IOSTANDARD LVCMOS33} [get\_ports {switches[2]}]; # Switch 2  set\_property -dict {PACKAGE\_PIN R15 IOSTANDARD LVCMOS33} [get\_ports {switches[3]}]; # Switch 3  set\_property -dict {PACKAGE\_PIN R17 IOSTANDARD LVCMOS33} [get\_ports {switches[4]}]; # Switch 4  set\_property -dict {PACKAGE\_PIN T18 IOSTANDARD LVCMOS33} [get\_ports {switches[5]}]; # Switch 5  set\_property -dict {PACKAGE\_PIN U18 IOSTANDARD LVCMOS33} [get\_ports {switches[6]}]; # Switch 6  set\_property -dict {PACKAGE\_PIN R13 IOSTANDARD LVCMOS33} [get\_ports {switches[7]}]; # Switch 7  # LEDs  set\_property -dict {PACKAGE\_PIN H17 IOSTANDARD LVCMOS33} [get\_ports {we\_dm}]; # LED 0  # 7 Segment Display  set\_property -dict {PACKAGE\_PIN T10 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[0]}]; # CA  set\_property -dict {PACKAGE\_PIN R10 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[1]}]; # CB  set\_property -dict {PACKAGE\_PIN K16 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[2]}]; # CC  set\_property -dict {PACKAGE\_PIN K13 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[3]}]; # CD  set\_property -dict {PACKAGE\_PIN P15 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[4]}]; # CE  set\_property -dict {PACKAGE\_PIN T11 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[5]}]; # CF  set\_property -dict {PACKAGE\_PIN L18 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[6]}]; # CG  set\_property -dict {PACKAGE\_PIN H15 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[7]}]; # DP  set\_property -dict {PACKAGE\_PIN J17 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[0]}]; # AN0  set\_property -dict {PACKAGE\_PIN J18 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[1]}]; # AN1  set\_property -dict {PACKAGE\_PIN T9 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[2]}]; # AN2  set\_property -dict {PACKAGE\_PIN J14 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[3]}]; # AN3  set\_property -dict {PACKAGE\_PIN P14 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[4]}]; # AN4  set\_property -dict {PACKAGE\_PIN T14 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[5]}]; # AN5  set\_property -dict {PACKAGE\_PIN K2 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[6]}]; # AN6  set\_property -dict {PACKAGE\_PIN U13 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[7]}]; # AN7 |

| hex\_to\_7seg.v |
| --- |
| module hex\_to\_7seg (  input wire [3:0] HEX,  output reg [7:0] s  );  always @ (HEX) begin  case (HEX)  4'h0: s = 8'b11000000;  4'h1: s = 8'b11111001;  4'h2: s = 8'b10100100;  4'h3: s = 8'b10110000;  4'h4: s = 8'b10011001;  4'h5: s = 8'b10010010;  4'h6: s = 8'b10000010;  4'h7: s = 8'b11111000;  4'h8: s = 8'b10000000;  4'h9: s = 8'b10010000;  4'hA: s = 8'b10001000;  4'hB: s = 8'b10000000;  4'hC: s = 8'b11000110;  4'hD: s = 8'b11000000;  4'hE: s = 8'b10000110;  4'hF: s = 8'b10001110;  default: s = 8'b01111111;  endcase  end  endmodule |

| led\_mux.v |
| --- |
| module led\_mux (  input wire clk,  input wire rst,  input wire [7:0] LED7,  input wire [7:0] LED6,  input wire [7:0] LED5,  input wire [7:0] LED4,  input wire [7:0] LED3,  input wire [7:0] LED2,  input wire [7:0] LED1,  input wire [7:0] LED0,  output wire [7:0] LEDSEL,  output wire [7:0] LEDOUT  );  reg [2:0] index;  reg [15:0] led\_ctrl;  assign {LEDSEL, LEDOUT} = led\_ctrl;    always @ (posedge clk) index <= (rst) ? 3'b0 : (index + 3'd1);    always @ (index, LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7) begin  case (index)  3'd0: led\_ctrl <= {8'b11111110, LED0};  3'd1: led\_ctrl <= {8'b11111101, LED1};  3'd2: led\_ctrl <= {8'b11111011, LED2};  3'd3: led\_ctrl <= {8'b11110111, LED3};  3'd4: led\_ctrl <= {8'b11101111, LED4};  3'd5: led\_ctrl <= {8'b11011111, LED5};  3'd6: led\_ctrl <= {8'b10111111, LED6};  3'd7: led\_ctrl <= {8'b01111111, LED7};  default: led\_ctrl <= {8'b11111111, 8'hFF};  endcase  end    endmodule |

| clk\_gen.v |
| --- |
| module clk\_gen (  input wire clk100MHz,  input wire rst,  output reg clk\_4sec,  output reg clk\_5KHz  );  integer count1, count2;  always @ (posedge clk100MHz) begin  if (rst) begin  count1 = 0;  count2 = 0;  clk\_5KHz = 0;  clk\_4sec = 0;  end  else begin  if (count1 == 200000000) begin  clk\_4sec = ~clk\_4sec;  count1 = 0;  end  if (count2 == 10000) begin  clk\_5KHz = ~clk\_5KHz;  count2 = 0;  end  count1 = count1 + 1;  count2 = count2 + 1;  end  end  endmodule |

| button\_debouncer.v |
| --- |
| module button\_debouncer #(parameter depth = 16) (  input wire clk, /\* 5 KHz clock \*/  input wire button, /\* Input button from constraints \*/  output reg debounced\_button  );  localparam history\_max = (2\*\*depth)-1;  /\* History of sampled input button \*/  reg [depth-1:0] history;  always @ (posedge clk) begin  /\* Move history back one sample and insert new sample \*/  history <= { button, history[depth-1:1] };  /\* Assert debounced button if it has been in a consistent state throughout history \*/  debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;  end  endmodule |