***Design Methodology***

This is lab is an extension of lab 5, this lab requires full hardware validation testing for a single-process cpu . The files added to this project were the xdc constraints file and two extra source files. Provided below is a block diagram showing the testing environment in full. 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:

* Create a diagram to show the validation environment setup.
* Fully verify the functionality using the FPGA.
* Compare the FPGA validation results with test log obtained from Lab 2.

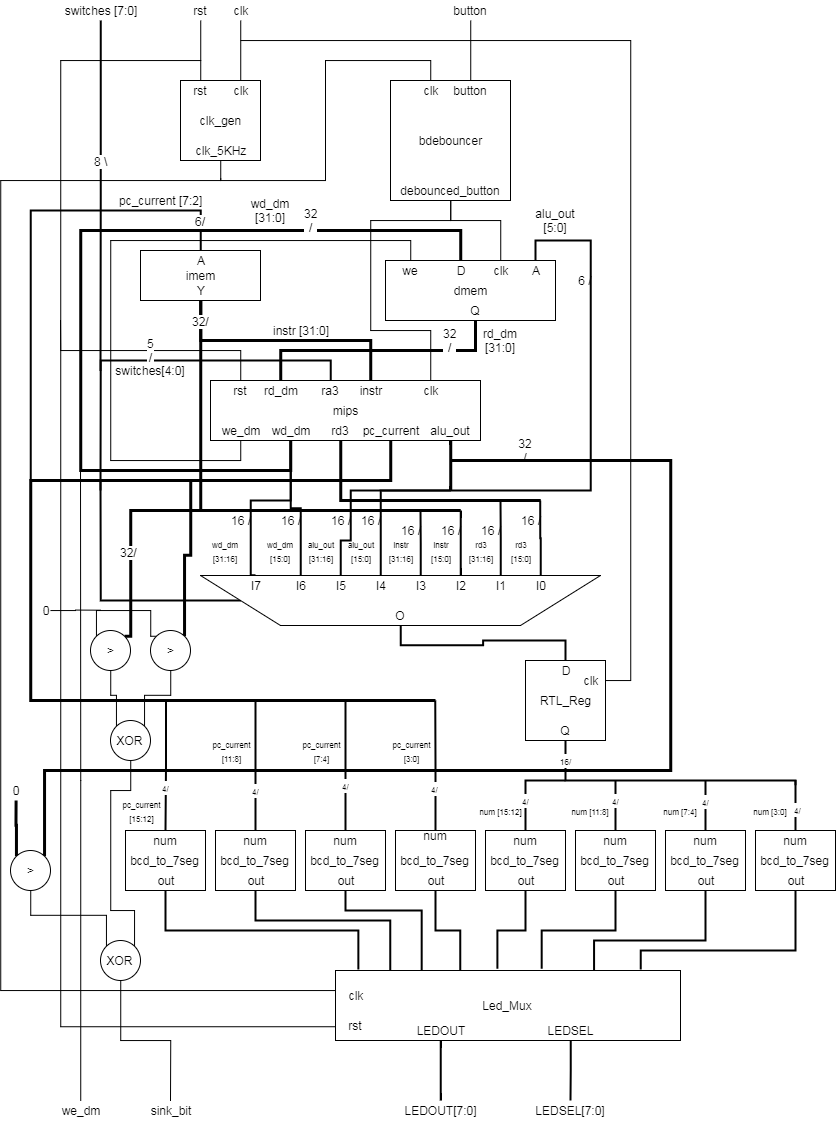


Figure 1. Validation Environment Setup.

Table 1. Test Results Log

| Adr | Expected Machine Code | Actual Machine Code | PC | Registers | | | | | Memory Content | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $v0 | $v1 | $a0 | $a1 | $a3 | [80] | [84] |
| 00 | 20020005 | 20020005 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 04 | 2003000c | 2003000c | 4 | 5 | c | 0 | 0 | 0 | 0 | 0 |
| 08 | 2067fff7 | 2067ffff | 8 | 5 | c | 0 | 0 | 3 | 0 | 0 |
| 0c | 00e22025 | 00e22025 | c | 5 | c | 7 | 0 | 3 | 0 | 0 |
| 10 | 00642824 | 00642824 | 10 | 5 | c | 7 | 4 | 3 | 0 | 0 |
| 14 | 00a42820 | 00a42820 | 14 | 5 | c | 7 | b | 3 | 0 | 0 |
| 18 | 10a7000a | 10e5000a | 18 | 5 | c | 7 | b | 3 | 0 | 0 |
| 1c | 0064202a | 0064202a | 1c | 5 | c | 0 | b | 3 | 0 | 0 |
| 20 | 10800001 | 10040001 | 20 | 5 | c | 0 | b | 3 | 0 | 0 |
| 24 | 20050000 | 20050000 |  |  |  |  |  |  |  |  |
| 28 | 00e2202a | 00e2202a | 28 | 5 | c | 1 | b | 3 | 0 | 0 |
| 2c | 00853820 | 00853820 | 2c | 5 | c | 1 | b | c | 0 | 0 |
| 30 | 00e23822 | 00e23822 | 30 | 5 | c | 1 | b | 7 | 0 | 0 |
| 34 | Ac670044 | Ac670044 | 34 | 5 | c | 1 | b | 7 | 7 | 0 |
| 38 | 8c020050 | 8c020050 | 38 | 7 | c | 1 | b | 7 | 7 | 0 |
| 3c | 08000011 | 08000011 | 3c | 7 | c | 1 | b | 7 | 7 | 0 |
| 40 | 20020001 | 20020001 |  |  |  |  |  |  |  |  |
| 44 | Ac020054 | Ac020054 | 44 | 7 | c | 1 | b | 7 | 7 | 7 |
| 48 | 08000000 | 08000000 | 48 | 7 | c | 1 | b | 7 | 7 | 7 |

***Screenshot***

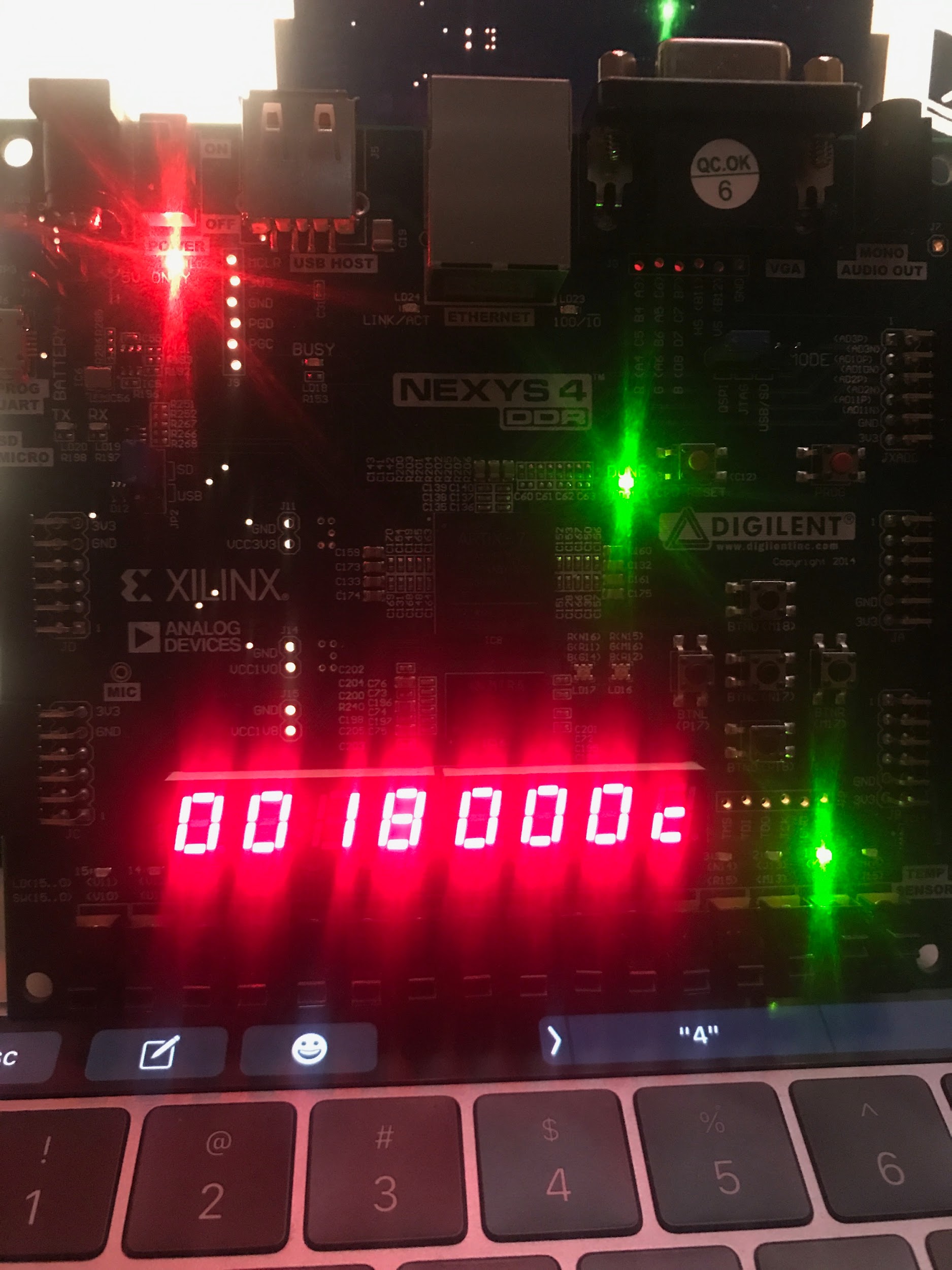
******

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

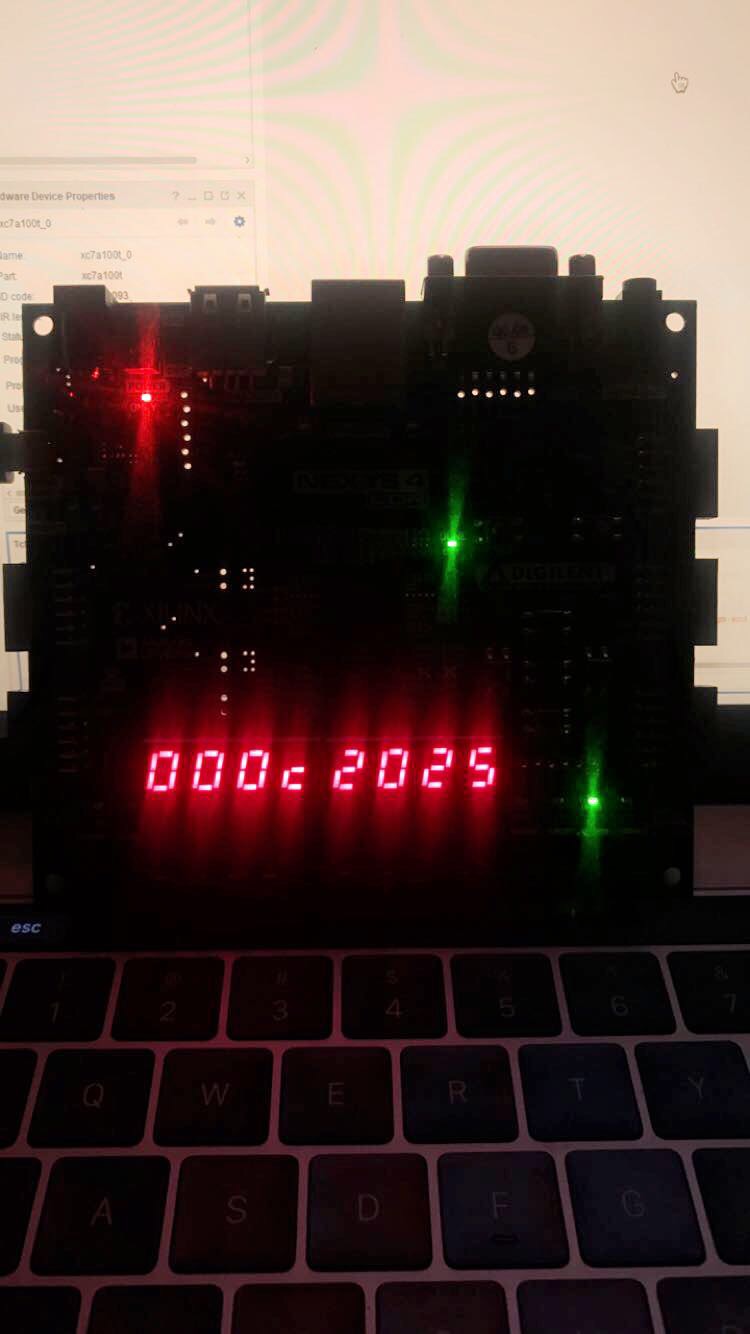


Figure 3. 7-Segment Display Showing Contents of Registers & Address

***Source Code:***

**mips\_top.v**

module mips\_top

(input clk, rst, output we\_dm, [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm);

wire [31:0] DONT\_USE;

mips mips (clk, rst, 0, instr, rd\_dm, we\_dm, pc\_current, alu\_out, wd\_dm, DONT\_USE);

imem imem (pc\_current[7:2], instr);

dmem dmem (clk, we\_dm, alu\_out[7:2], wd\_dm, rd\_dm);

endmodule

**mips.v**

module mips

(input clk, rst, [4:0] ra3, [31:0] instr, rd\_dm, output we\_dm, [31:0] pc\_current, alu\_out, wd\_dm, rd3);

wire pc\_src, jump, reg\_dst, we\_reg, alu\_src, dm2reg;

wire [2:0] alu\_ctrl;

datapath dp (clk, rst, pc\_src, jump, reg\_dst, we\_reg, alu\_src, dm2reg, alu\_ctrl, ra3, instr, rd\_dm, zero, pc\_current, alu\_out, wd\_dm, rd3);

controlunit cu (zero, instr[31:26], instr[5:0], pc\_src, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, alu\_ctrl);

endmodule

**Mem\_parts.v**

module imem

(input [5:0] a, output [31:0] y);

reg [31:0] rom [0:63];

initial begin

$readmemh ("memfile.dat", rom);

end

assign y = rom[a];

endmodule

module dmem

(input clk, we, [5:0] a, [31:0] d, output [31:0] q);

reg [31:0] ram [0:63];

integer n;

initial begin

for (n = 0; n < 64; n = n + 1) ram[n] = 32'hFFFFFFFF;

end

always @ (posedge clk)

begin

if (we) ram[a] <= d;

end

assign q = ram[a];

endmodule

**datapath.v**

module datapath

(input clk, rst, pc\_src, jump, reg\_dst, we\_reg, alu\_src, dm2reg, [2:0] alu\_ctrl, [4:0] ra3, [31:0] instr, rd\_dm, output zero, [31:0] pc\_current, alu\_out, wd\_dm, rd3);

wire [4:0] rf\_wa;

wire [31:0] pc\_plus4, pc\_pre, pc\_next, sext\_imm, ba, bta, jta, alu\_pa, alu\_pb, wd\_rf;

assign ba = {sext\_imm[29:0], 2'b00};

assign jta = {pc\_plus4[31:28], instr[25:0], 2'b00};

// --- PC Logic --- //

dreg pc\_reg (clk, rst, pc\_next, pc\_current);

adder pc\_plus\_4 (pc\_current, 4, pc\_plus4);

adder pc\_plus\_br (pc\_plus4, ba, bta);

mux2 #(32) pc\_src\_mux (pc\_src, pc\_plus4, bta, pc\_pre);

mux2 #(32) pc\_jmp\_mux (jump, pc\_pre, jta, pc\_next);

// --- RF Logic --- //

mux2 #(5) rf\_wa\_mux (reg\_dst, instr[20:16], instr[15:11], rf\_wa);

regfile rf (clk, we\_reg, instr[25:21], instr[20:16], ra3, rf\_wa, wd\_rf, alu\_pa, wd\_dm, rd3);

signext se (instr[15:0], sext\_imm);

// --- ALU Logic --- //

mux2 #(32) alu\_pb\_mux (alu\_src, wd\_dm, sext\_imm, alu\_pb);

alu alu (alu\_ctrl, alu\_pa, alu\_pb, zero, alu\_out);

// --- MEM Logic --- //

mux2 #(32) rf\_wd\_mux (dm2reg, alu\_out, rd\_dm, wd\_rf);

endmodule

**dp\_parts.v**

module mux2 #(parameter wide = 8)

(input sel, [wide-1:0] a, b, output [wide-1:0] y);

assign y = (sel) ? b : a;

endmodule

module adder

(input [31:0] a, b, output [31:0] y);

assign y = a + b;

endmodule

module signext

(input [15:0] a, output [31:0] y);

assign y = {{16{a[15]}}, a};

endmodule

module alu

(input [2:0] op, [31:0] a, b, output zero, reg [31:0] y);

assign zero = (y == 0);

always @ (op, a, b)

begin

case (op)

3'b000: y = a & b;

3'b001: y = a | b;

3'b010: y = a + b;

3'b110: y = a - b;

3'b111: y = (a < b) ? 1 : 0;

endcase

end

endmodule

module dreg

(input clk, rst, [31:0] d, output reg [31:0] q);

always @ (posedge clk, posedge rst)

begin

if (rst) q <= 0;

else q <= d;

end

endmodule

module regfile

(input clk, we, [4:0] ra1, ra2, ra3, wa, [31:0] wd, output [31:0] rd1, rd2, rd3);

reg [31:0] rf [0:31];

integer n;

initial begin

for (n = 0; n < 32; n = n + 1) rf[n] = 32'h0;

end

always @ (posedge clk)

begin

if (we) rf[wa] <= wd;

end

assign rd1 = (ra1 == 0) ? 0 : rf[ra1];

assign rd2 = (ra2 == 0) ? 0 : rf[ra2];

assign rd3 = (ra3 == 0) ? 0 : rf[ra3];

endmodule

**controlunit.v**

module controlunit

(input zero, [5:0] opcode, funct, output pc\_src, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, [2:0] alu\_ctrl);

wire [1:0] alu\_op;

assign pc\_src = branch & zero;

maindec md (opcode, branch, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, alu\_op);

auxdec ad (alu\_op, funct, alu\_ctrl);

endmodule

**cu\_parts.v**

module maindec

(input [5:0] opcode, output branch, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, [1:0] alu\_op);

reg [8:0] ctrl;

assign {branch, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, alu\_op} = ctrl;

always @ (opcode)

begin

case (opcode)

6'b00\_0000: ctrl = 9'b0\_0\_1\_1\_0\_0\_0\_10; // R-type

6'b00\_1000: ctrl = 9'b0\_0\_0\_1\_1\_0\_0\_00; // ADDI

6'b00\_0100: ctrl = 9'b1\_0\_0\_0\_0\_0\_0\_01; // BEQ

6'b00\_0010: ctrl = 9'b0\_1\_0\_0\_0\_0\_0\_00; // J

6'b10\_1011: ctrl = 9'b0\_0\_0\_0\_1\_1\_0\_00; // SW

6'b10\_0011: ctrl = 9'b0\_0\_0\_1\_1\_0\_1\_00; // LW

default: ctrl = 9'bx\_x\_x\_x\_x\_x\_x\_xx;

endcase

end

endmodule

module auxdec

(input [1:0] alu\_op, [5:0] funct, output [2:0] alu\_ctrl);

reg [2:0] ctrl;

assign {alu\_ctrl} = ctrl;

always @ (alu\_op, funct)

begin

case (alu\_op)

2'b00: ctrl = 3'b010; // add

2'b01: ctrl = 3'b110; // sub

default: case (funct)

6'b10\_0100: ctrl = 3'b000; // AND

6'b10\_0101: ctrl = 3'b001; // OR

6'b10\_0000: ctrl = 3'b010; // ADD

6'b10\_0010: ctrl = 3'b110; // SUB

6'b10\_1010: ctrl = 3'b111; // SLT

default: ctrl = 3'bxxx;

endcase

endcase

end

endmodule

**tb\_mips\_top.v**

module tb\_mips\_top;

reg clk, rst;

wire we\_dm;

wire [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm;

mips\_top DUT (clk, rst, we\_dm, pc\_current, instr, alu\_out, wd\_dm, rd\_dm);

task tick; begin #5 clk = 1; #5 clk = 0; end endtask

task rest; begin #5 rst = 1; #5 rst = 0; end endtask

initial begin

rest;

while(pc\_current != 32'h48) tick;

$finish;

end

Endmodule

**memfile.dat**

20020005

2003000C

2067FFF7

00E22025

00642824

00A42820

10E5000A

0064202A

10040001

20050000

00E2202A

00853820

00E23822

AC670044

8C020050

08000011

20020001

AC020054

08000000

**Mips.xdc**

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}];

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { button }];

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { rst }];

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { switches[0] }];

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { switches[1] }];

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { switches[2] }];

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { switches[3] }];

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { switches[4] }];

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { switches[5] }];

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { switches[6] }];

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { switches[7] }];

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { we\_dm }];

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { sink\_bit }];

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[0] }];

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[1] }];

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[2] }];

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[3] }];

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[4] }];

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[5] }];

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[6] }];

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[7] }];

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[0] }];

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[1] }];

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[2] }];

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[3] }];

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[4] }];

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[5] }];

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[6] }];

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[7] }];

**Mips\_fpga.v**

module mips\_fpga

(input clk, rst, button, [7:0] switches, output we\_dm, sink\_bit, [7:0] LEDSEL, LEDOUT);

reg [15:0] reg\_hex;

wire clk\_sec, clk\_5KHz, clk\_pb;

wire [7:0] digit0, digit1, digit2, digit3, digit4, digit5, digit6, digit7;

wire [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm, dispData;

clk\_gen clk\_gen (clk, rst, clk\_sec, clk\_5KHz);

bdebouncer bd (clk\_5KHz, button, clk\_pb);

mips mips (clk\_pb, rst, switches[4:0], instr, rd\_dm, we\_dm, pc\_current, alu\_out, wd\_dm, dispData);

/\*

switchs[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).

\*/

imem imem (pc\_current[7:2], instr);

dmem dmem (clk\_pb, we\_dm, alu\_out[7:2], wd\_dm, rd\_dm);

bcd\_to\_7seg bcd7 (pc\_current[15:12], digit7);

bcd\_to\_7seg bcd6 (pc\_current[11:8], digit6);

bcd\_to\_7seg bcd5 (pc\_current[7:4], digit5);

bcd\_to\_7seg bcd4 (pc\_current[3:0], digit4);

bcd\_to\_7seg bcd3 (reg\_hex[15:12], digit3);

bcd\_to\_7seg bcd2 (reg\_hex[11:8], digit2);

bcd\_to\_7seg bcd1 (reg\_hex[7:4], digit1);

bcd\_to\_7seg bcd0 (reg\_hex[3:0], digit0);

led\_mux led\_mux (clk\_5KHz, rst, digit7, digit6, digit5, digit4, digit3, digit2, digit1, digit0, LEDSEL, 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], switches[6], switches[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

assign sink\_bit = (pc\_current > 0) ^ (instr > 0) ^ (alu\_out > 0);

endmodule

**Utility.v**

module clk\_gen

(input clk100MHz, rst, output reg clk\_sec, clk\_5KHz);

integer count1, count2;

always @ (posedge clk100MHz)

begin

if (rst)

begin

count1 = 0; clk\_sec = 0;

count2 = 0; clk\_5KHz = 0;

end

else

begin

if (count1 == 50000000)

begin

clk\_sec = ~clk\_sec;

count1 = 0;

end

if (count2 == 10000)

begin

clk\_5KHz = ~clk\_5KHz;

count2 = 0;

end

count1 = count1 + 1;

count2 = count2 + 1;

end

end

endmodule

/\* 7-segment values \*/

`define D0 8'b10001000 /\* 0 \*/

`define D1 8'b11101101 /\* 1 \*/

`define D2 8'b10100010 /\* 2 \*/

`define D3 8'b10100100 /\* 3 \*/

`define D4 8'b11000101 /\* 4 \*/

`define D5 8'b10010100 /\* 5 \*/

`define D6 8'b10010000 /\* 6 \*/

`define D7 8'b10101101 /\* 7 \*/

`define D8 8'b10000000 /\* 8 \*/

`define D9 8'b10000100 /\* 9 \*/

`define DA 8'b10100000 /\* A \*/

`define DB 8'b11010000 /\* B \*/

`define DC 8'b11110010 /\* C \*/

`define DD 8'b11100000 /\* D \*/

`define DE 8'b10010010 /\* E \*/

`define DF 8'b10010011 /\* F \*/

`define DX 8'b01111111 /\* All segments off except decimal point \*/

module bcd\_to\_7seg

(input [3:0] num, output reg [7:0] out);

always @ (num)

begin

case (num)

4'h0: begin out=`D0; end

4'h1: begin out=`D1; end

4'h2: begin out=`D2; end

4'h3: begin out=`D3; end

4'h4: begin out=`D4; end

4'h5: begin out=`D5; end

4'h6: begin out=`D6; end

4'h7: begin out=`D7; end

4'h8: begin out=`D8; end

4'h9: begin out=`D9; end

4'hA: begin out=`DA; end

4'hB: begin out=`DB; end

4'hC: begin out=`DC; end

4'hD: begin out=`DD; end

4'hE: begin out=`DE; end

4'hF: begin out=`DF; end

default: begin out=`DX; end

endcase

end

endmodule

module led\_mux

(input clk, rst, [7:0] LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7, output [7:0] LEDSEL, LEDOUT);

reg [2:0] index;

reg [15:0] led\_ctrl;

assign {LEDSEL, LEDOUT} = led\_ctrl;

always@(posedge clk)

begin

index <= (rst) ? 3'd0 : (index + 3'd1);

end

always @(index, LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7)

begin

case(index)

3'd0: led\_ctrl <= {8'b11111110, LED7};

3'd1: led\_ctrl <= {8'b11111101, LED6};

3'd2: led\_ctrl <= {8'b11111011, LED5};

3'd3: led\_ctrl <= {8'b11110111, LED4};

3'd4: led\_ctrl <= {8'b11101111, LED3};

3'd5: led\_ctrl <= {8'b11011111, LED2};

3'd6: led\_ctrl <= {8'b10111111, LED1};

3'd7: led\_ctrl <= {8'b01111111, LED0};

default: led\_ctrl <= {8'b11111111, 8'hFF};

endcase

end

endmodule

module bdebouncer #(parameter depth = 16)

(input clk, button, output reg debounced\_button);

localparam history\_max = (2\*\*depth)-1;

reg [depth-1:0] history;

always @ (posedge clk)

begin

history <= {button, history[depth-1:1]};

debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;

end

endmodule