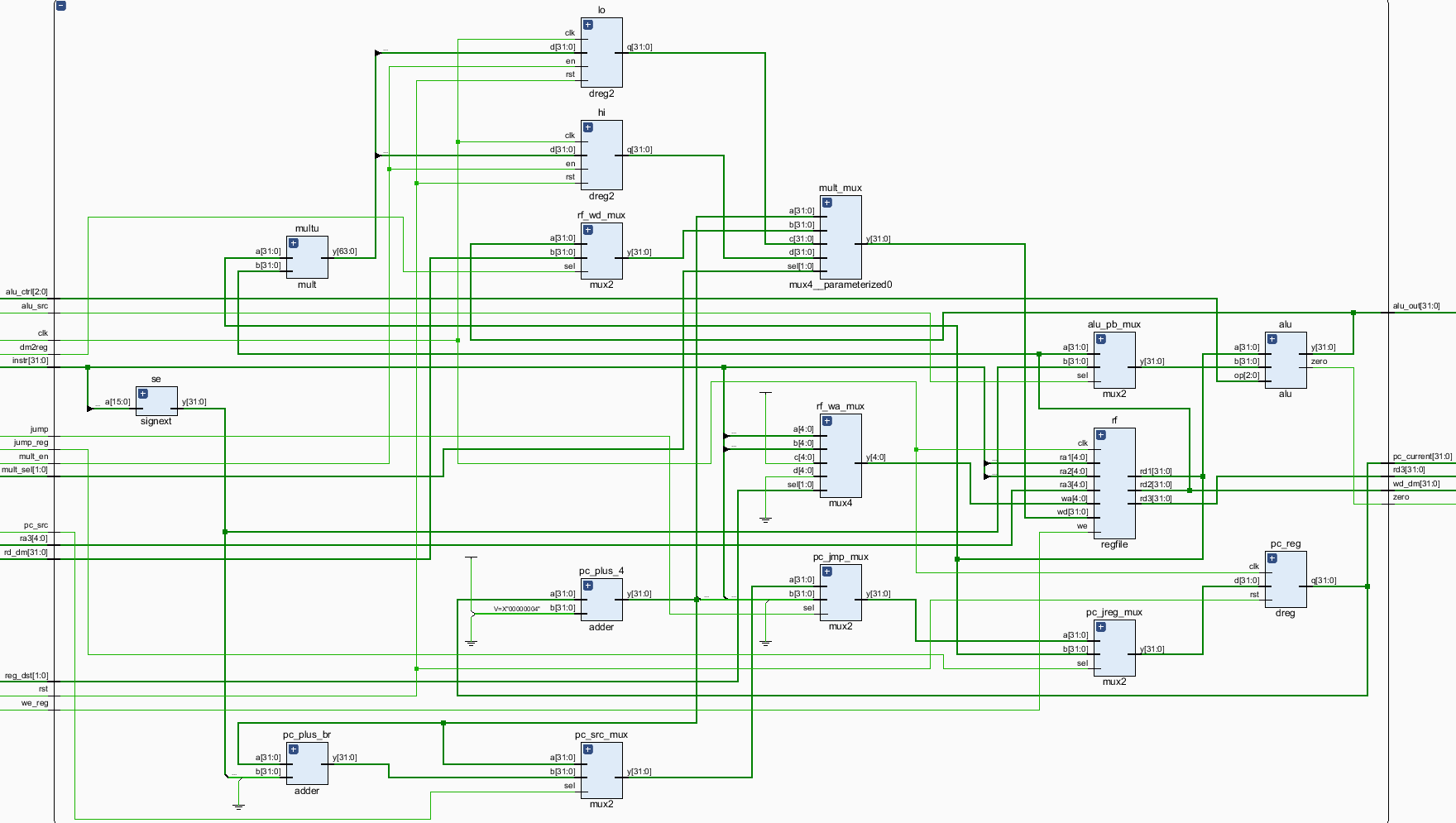
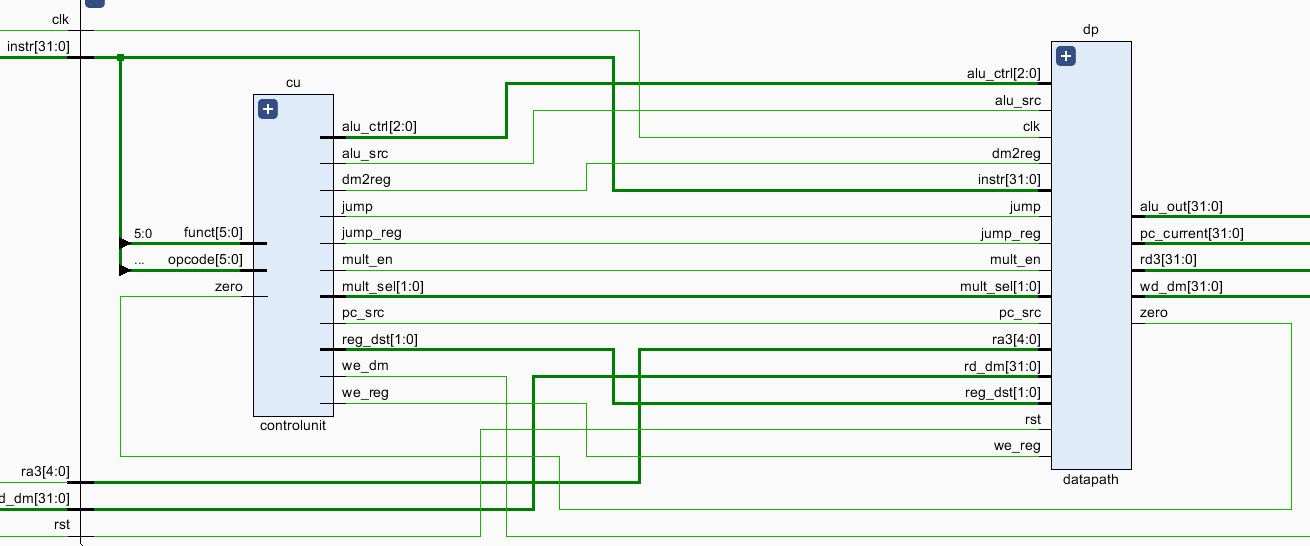


**Diagrams:**

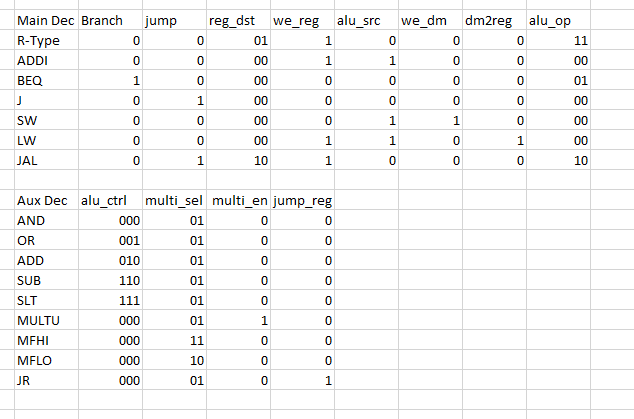
Data Path

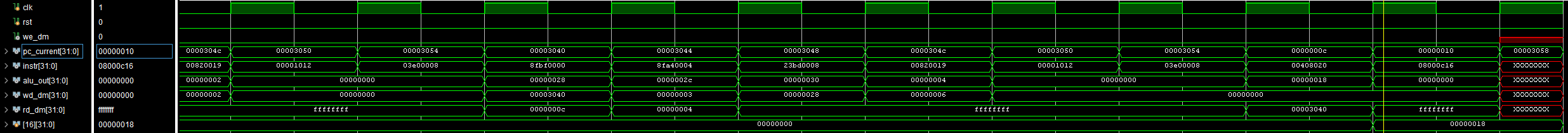
****

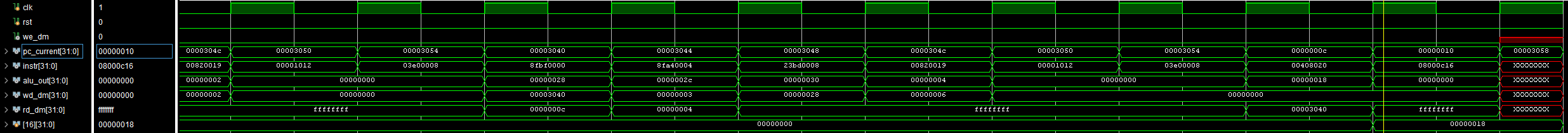
Control Unit – Data Path

****

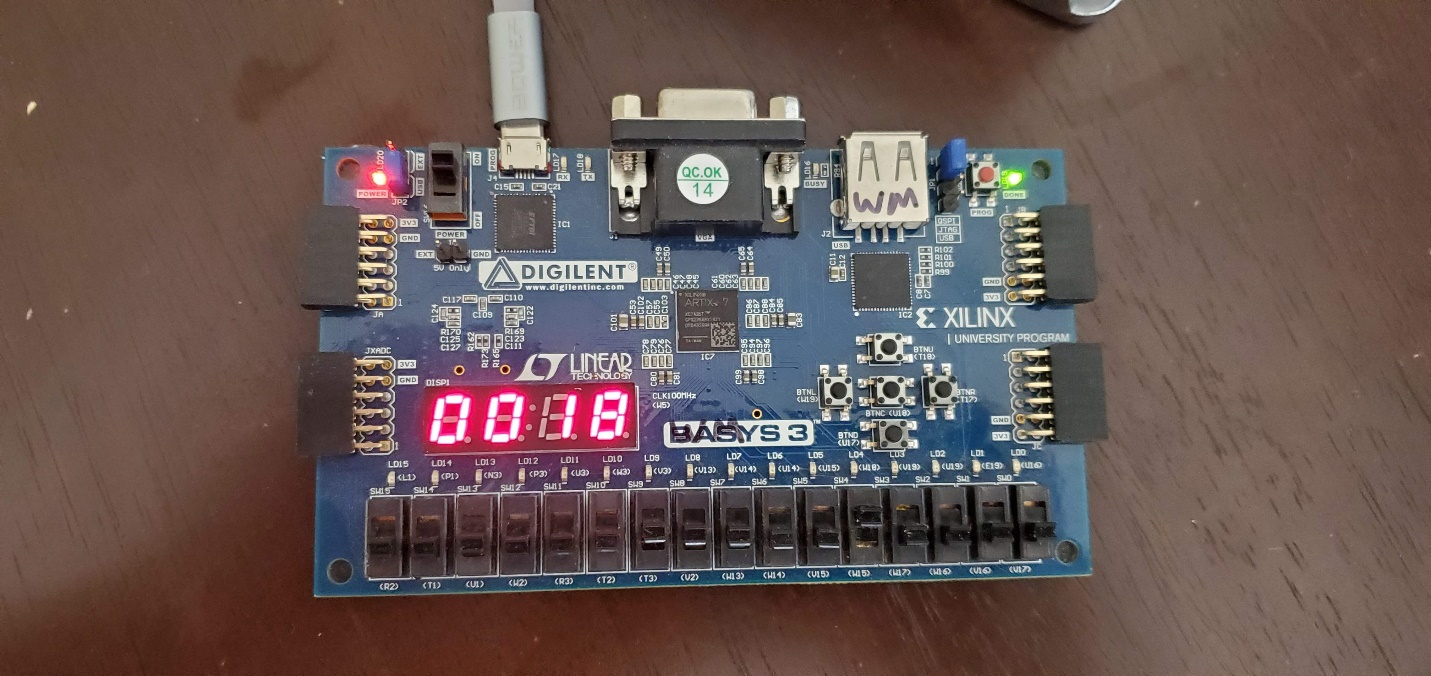
**Truth Table**

****

**Testbench waveform**

****

**Board Validation**

****

**Source Code**

module mips\_fpga

(

input wire clk,

input wire rst,

input wire button,

input wire [8:0] switches,

output wire we\_dm,

output wire [3:0] LEDSEL,

output wire [7:0] LEDOUT

);

reg [15:0] reg\_hex;

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

wire [7:0] digit0, digit1, digit2, digit3;

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

.rst (rst),

.ra3 (switches[4:0]),

.instr (instr),

.we\_dm (we\_dm),

.pc\_current (pc\_current),

.alu\_out (alu\_out),

.wd\_dm (wd\_dm),

.rd3 (dispData)

);

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),

.LED3 (digit3),

.LED2 (digit2),

.LED1 (digit1),

.LED0 (digit0),

.LEDSEL (LEDSEL),

.LEDOUT (LEDOUT)

);

always @ (posedge clk) begin

case ({switches[8:5]})

4'b0000: reg\_hex = dispData[15:0];

4'b0001: reg\_hex = dispData[31:16];

4'b0010: reg\_hex = instr[15:0];

4'b0011: reg\_hex = instr[31:16];

4'b0100: reg\_hex = alu\_out[15:0];

4'b0101: reg\_hex = alu\_out[31:16];

4'b0110: reg\_hex = wd\_dm[15:0];

4'b1000: reg\_hex = pc\_current[15:0];

4'b1001: reg\_hex = pc\_current[31:16];

default: reg\_hex = pc\_current[15:0];

endcase

end

endmodule

module mips(

input wire clk,

input wire rst,

input wire [4:0] ra3,

input wire [31:0] instr,

input wire [31:0] rd\_dm,

output wire we\_dm,

output wire [31:0] pc\_current,

output wire [31:0] alu\_out,

output wire [31:0] wd\_dm,

output wire [31:0] rd3

);

wire pc\_src, jump, we\_reg, alu\_src, dm2reg, mult\_en, jump\_reg;

wire [1:0] mult\_sel, reg\_dst;

wire [2:0] alu\_ctrl;

wire zero;

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

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

endmodule

module maindec

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

output we\_dm, output dm2reg, output [1:0] alu\_op);

reg [9:0] 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 = 10'b0\_0\_01\_1\_0\_0\_0\_11; // R-type

6'b00\_1000: ctrl = 10'b0\_0\_00\_1\_1\_0\_0\_00; // ADDI

6'b00\_0100: ctrl = 10'b1\_0\_00\_0\_0\_0\_0\_01; // BEQ

6'b00\_0010: ctrl = 10'b0\_1\_00\_0\_0\_0\_0\_00; // J

6'b10\_1011: ctrl = 10'b0\_0\_00\_0\_1\_1\_0\_00; // SW

6'b10\_0011: ctrl = 10'b0\_0\_00\_1\_1\_0\_1\_00; // LW

6'b00\_0011: ctrl = 10'b0\_1\_10\_1\_0\_0\_0\_10; // JAL

default: ctrl = 10'bx\_x\_xx\_x\_x\_x\_x\_xx;

endcase

end

endmodule

module auxdec

(input [1:0] alu\_op, input [5:0] funct, output [2:0] alu\_ctrl, output [1:0] multi\_sel, output multi\_en, output jump\_reg);

reg [6:0] ctrl;

assign {alu\_ctrl, multi\_sel, multi\_en, jump\_reg} = ctrl;

always @ (alu\_op, funct)

begin

case (alu\_op)

2'b00: ctrl = 7'b010\_01\_0\_0; // add

2'b01: ctrl = 7'b110\_01\_0\_0; // sub

2'b10: ctrl = 7'b000\_00\_0\_0; // jal

default: case (funct)

6'b10\_0100: ctrl = 7'b000\_01\_0\_0; // AND

6'b10\_0101: ctrl = 7'b001\_01\_0\_0; // OR

6'b10\_0000: ctrl = 7'b010\_01\_0\_0; // ADD

6'b10\_0010: ctrl = 7'b110\_01\_0\_0; // SUB

6'b10\_1010: ctrl = 7'b111\_01\_0\_0; // SLT

6'b01\_1001: ctrl = 7'b000\_01\_1\_0; // MULTU

6'b01\_0000: ctrl = 7'b000\_11\_0\_0; // MFHI

6'b01\_0010: ctrl = 7'b000\_10\_0\_0; // MFLO

6'b00\_1000: ctrl = 7'b000\_01\_0\_1; // JR

default: ctrl = 7'bxxx\_xx\_x\_x;

endcase

endcase

end

endmodule

module controlunit(

input zero,

input [5:0] opcode,

input [5:0] funct,

output pc\_src,

output jump,

output we\_reg,

output alu\_src,

output we\_dm,

output dm2reg,

output mult\_en,

output jump\_reg,

output [1:0] reg\_dst,

output [1:0] mult\_sel,

output [2:0] alu\_ctrl

);

wire [1:0] alu\_op;

wire branch;

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

auxdec ad (alu\_op, funct, alu\_ctrl, mult\_sel, mult\_en, jump\_reg);

assign pc\_src = branch & zero;

endmodule

module datapath(

input wire clk,

input wire mult\_en,

input wire jump\_reg,

input wire rst,

input wire pc\_src,

input wire jump,

input wire we\_reg,

input wire alu\_src,

input wire dm2reg,

input wire [2:0] alu\_ctrl,

input wire [1:0] reg\_dst,

input wire [1:0] mult\_sel,

input wire [4:0] ra3,

input wire [31:0] instr,

input wire [31:0] rd\_dm,

output wire zero,

output wire [31:0] pc\_current,

output wire [31:0] alu\_out,

output wire [31:0] wd\_dm,

output wire [31:0] 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;

wire [63:0] mult\_out;

wire [31:0] lo\_out, hi\_out, mult\_mux\_out, jr\_mux\_out;

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, jr\_mux\_out, 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);

mux2 #(32) pc\_jreg\_mux (jump\_reg, pc\_next, alu\_pa, jr\_mux\_out);

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

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

mux4 #(5) rf\_wa\_mux (reg\_dst, instr[20:16], instr[15:11], 31, 0, rf\_wa);

regfile rf (clk, we\_reg, instr[25:21], instr[20:16], ra3, rf\_wa, mult\_mux\_out, 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\_ctrl, alu\_pa, alu\_pb, zero, alu\_out);

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

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

// --- MULTU Logic --- //

mult multu (alu\_pa, wd\_dm, mult\_out);

dreg2 lo (clk, rst, mult\_en, mult\_out[31:0], lo\_out);

dreg2 hi (clk, rst, mult\_en, mult\_out[63:32], hi\_out);

mux4 #(32) mult\_mux(mult\_sel, pc\_plus4, wd\_rf, lo\_out, hi\_out, mult\_mux\_out);

endmodule

module mips\_top(

input wire clk,

input wire rst,

input wire [4:0] ra3,

input wire [31:0] instr,

output wire we\_dm,

output wire [31:0] rd\_dm,

output wire [31:0] pc\_current,

output wire [31:0] alu\_out,

output wire [31:0] wd\_dm,

output wire [31:0] rd3

);

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

**Discussion:**

The purpose of this lab was to further our understanding of single-cycle CPUs by implementing MULTU, MFHI, MFLO, JAL, JR into the previous labs. This was a two-week lab where in the first week we designed the connections of the system and the control logic, and in the second week we implemented the code in Verilog and validated our results onboard our BASYS 3 FPGA. The results of the labs are above. Overall as visible in the test bench the we were able to achieve the correct output from our design. Our group struggled with getting the overall layout correct and spent a lot of time chasing down loose connections, however, once all the connections were wired in correctly our control logic was able to correctly control the process.