WPI

ECE 505 Computer Architecture – Project Report

Worcester Polytechnic Institute

Fall 2022

RISC-V Single-Cycle CPU

Submitted by

Jonathan Lopez

Professor Huang

December 15th, 2022

Table of Contents

Assembly and Machine Code of Factorial 6	*2*
Assembly and Machine Code of My Program	*4*
Memory Contents of Factorial 6	*7*
Memory Contents of My Program	*8*
Modules and Testbench Output	*11*
Timing Constraints	*34*
FPGA Resources	*35*
Appendix	*37*

Assembly and Machine Code of Factorial 6

```
        addi    a0, x0, 6
        jal     ra, fact
        sw      a0, 0(x0)
        halt
fact:
        addi    sp, sp, -8
        sw      ra, 4(sp)
        sw      a0, 0(sp)
        addi    a0, a0, -1
        bne     a0, x0, else
        addi    a0, x0, 1
        addi    sp, sp, 8
        jalr    x0, 0(ra)
else:
        jal     ra, fact
        addi    t0, a0,0
        lw      a0, 0(sp)
        lw      ra, 4(sp)
        addi    sp, sp, 8
        mul     a0, a0, t0
        jalr    x0, 0(ra)
```

```
addi a0, x0, 6
jal ra, fact
sw a0, 0(x0)
NOP
fact:
addi sp, sp, -8
sw  ra, 4(sp)
sw  a0, 0(sp)
addi a0, a0, -1
bne a0, x0, else
addi a0, x0, 1
addi sp, sp, 8
jalr x0, 0(ra)
else:
jal ra, fact
addi t0, a0,0
lw  a0, 0(sp)
lw  ra, 4(sp)
addi sp, sp, 8
mul a0, a0, t0
jalr x0, 0(ra)
```

Figure 1: Assembly Code in Venus to Convert to Hexidecimal for reg_rom

```
//factorial 6 //part b

file[0] = 32'h00600513; //addi a0, x0, 6
file[1] = 32'h00c000ef; //jal ra, fact (12)
file[2] = 32'h00a02023; //sw a0, 0(x0)
file[3] = 32'h0000007f; //HALT
//fact:
file[4] = 32'hff810113; //addi sp, sp, -8
file[5] = 32'h00112223; //sw  ra, 4(sp)
file[6] = 32'h00a12023; //sw x10 0(x2)
file[7] = 32'hfff50513; //addi x10 x10 -1
file[8] = 32'h00051863; //bne a0, x0, else (16)
file[9] = 32'h00100513; //addi a0, x0, 1
file[10] = 32'h00810113; //addi sp, sp, 8
file[11] = 32'h00008067; //jalr x0, 0(ra)
//else:
file[12] = 32'hfe1ff0ef; //jal ra, fact(-32)
file[13] = 32'h00050293; //addi t0, a0,0
file[14] = 32'h00012503; //lw  a0, 0(sp)
file[15] = 32'h00412083; //lw  ra, 4(sp)
file[16] = 32'h00810113; //addi sp, sp, 8
file[17] = 32'h02550533; //mul a0, a0, t0
file[18] = 32'h00008067; //jalr x0, 0(ra)
```

Figure 2: Machine Code for Factorial 6

Assembly and Machine Code of My Program

```
//My Program
file [0] = 32'h00100113;
file [1] = 32'h402080b3;
file [2] = 32'h00102223;
file [3] = 32'hfe001ce3;
file [4] = 32'h0000007f;
```

Figure 3: My accumulation Program

```
ADDI x2, x0, 1

loop:
   SUB x1, x1, x2
   SW  x1, 4(x0)
   BNE zero, x0, loop
   NOP
```

Figure 4: Venus For BNE program

Memory Contents of Factorial 6

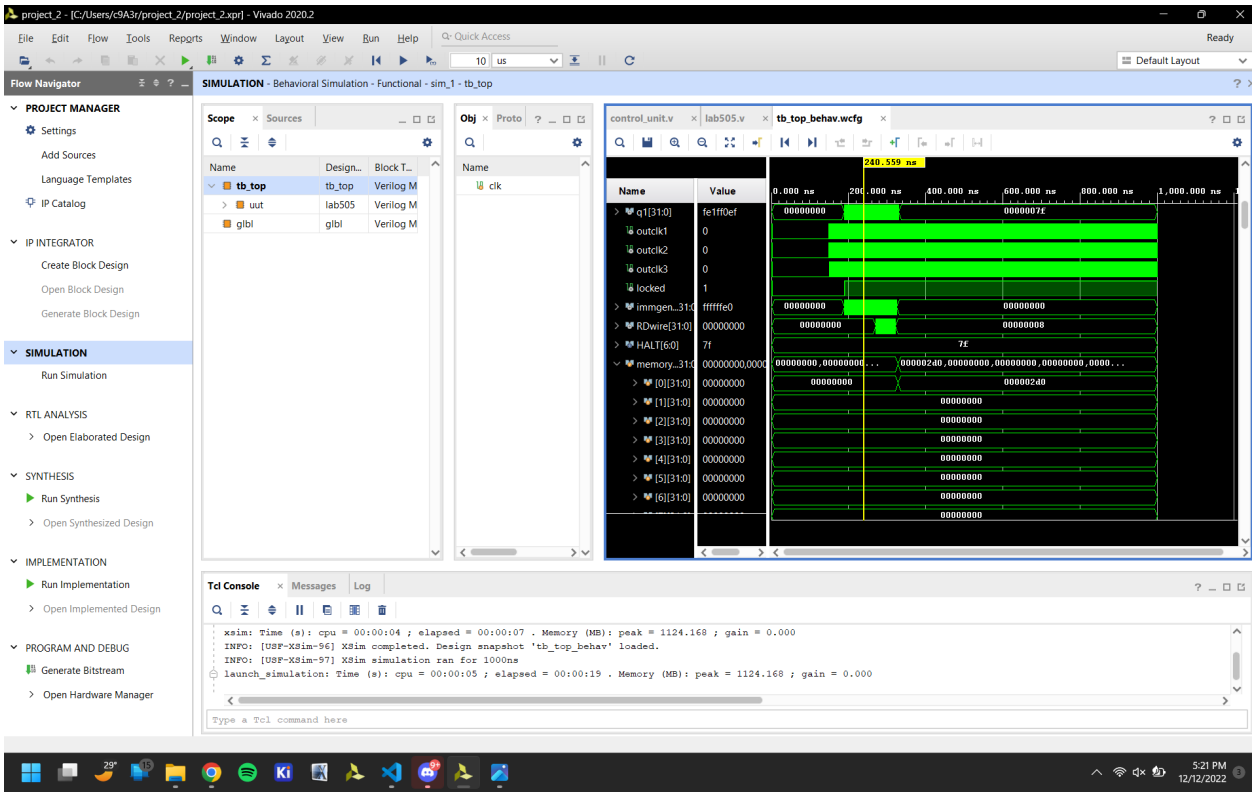

Figure 5: Screenshot of Working Factorial 6

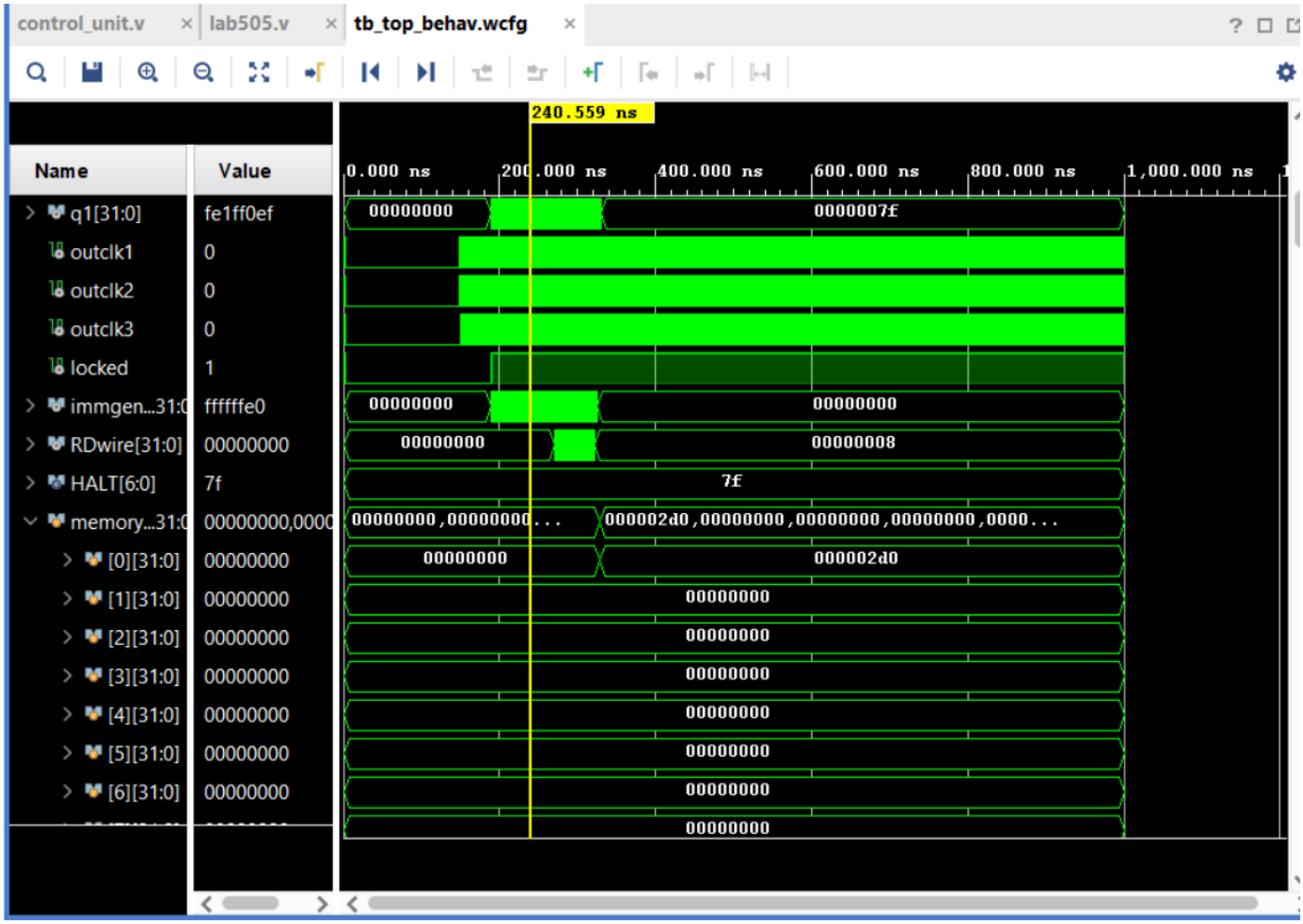

Figure 6: Zoom in on 2d0 in mem[0]

This matches the screenshot the TA gave us as shown in Figure 5

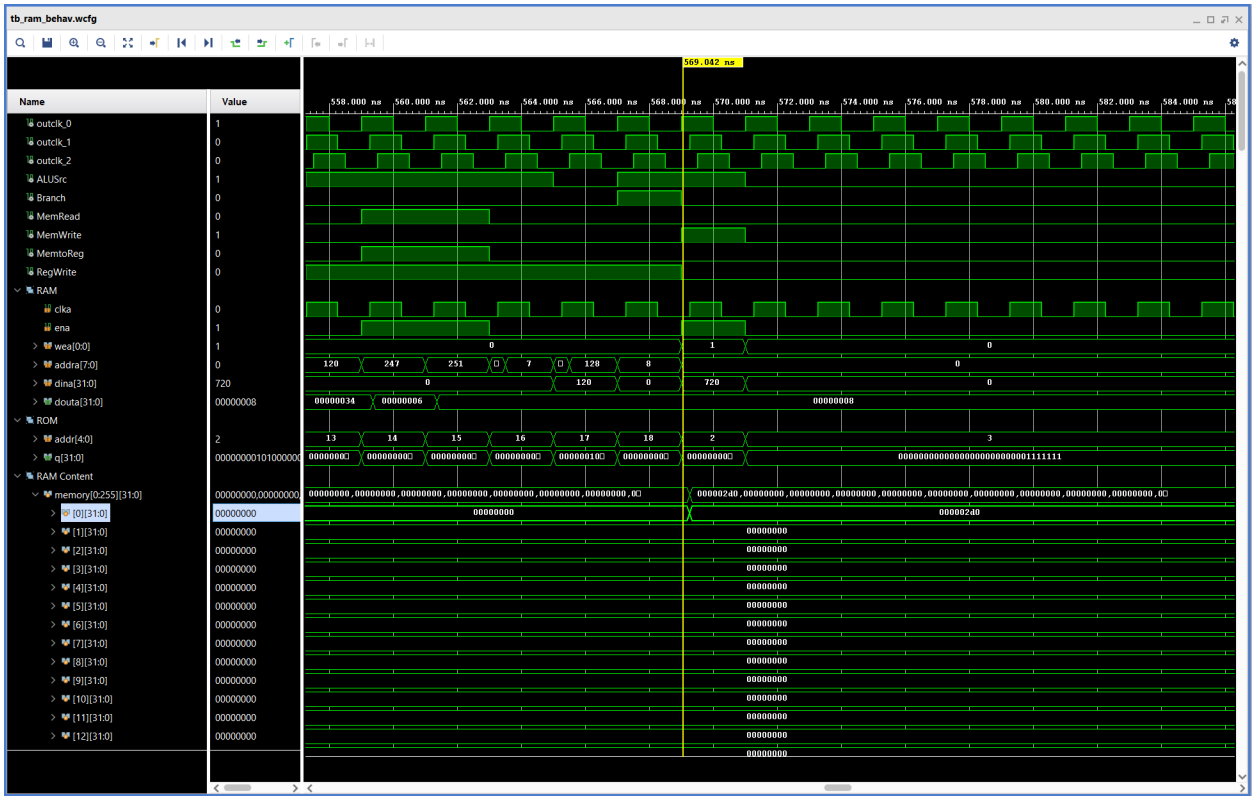

Figure 7: TA's Factorial 6 Output Memory

Memory Contents of My Program

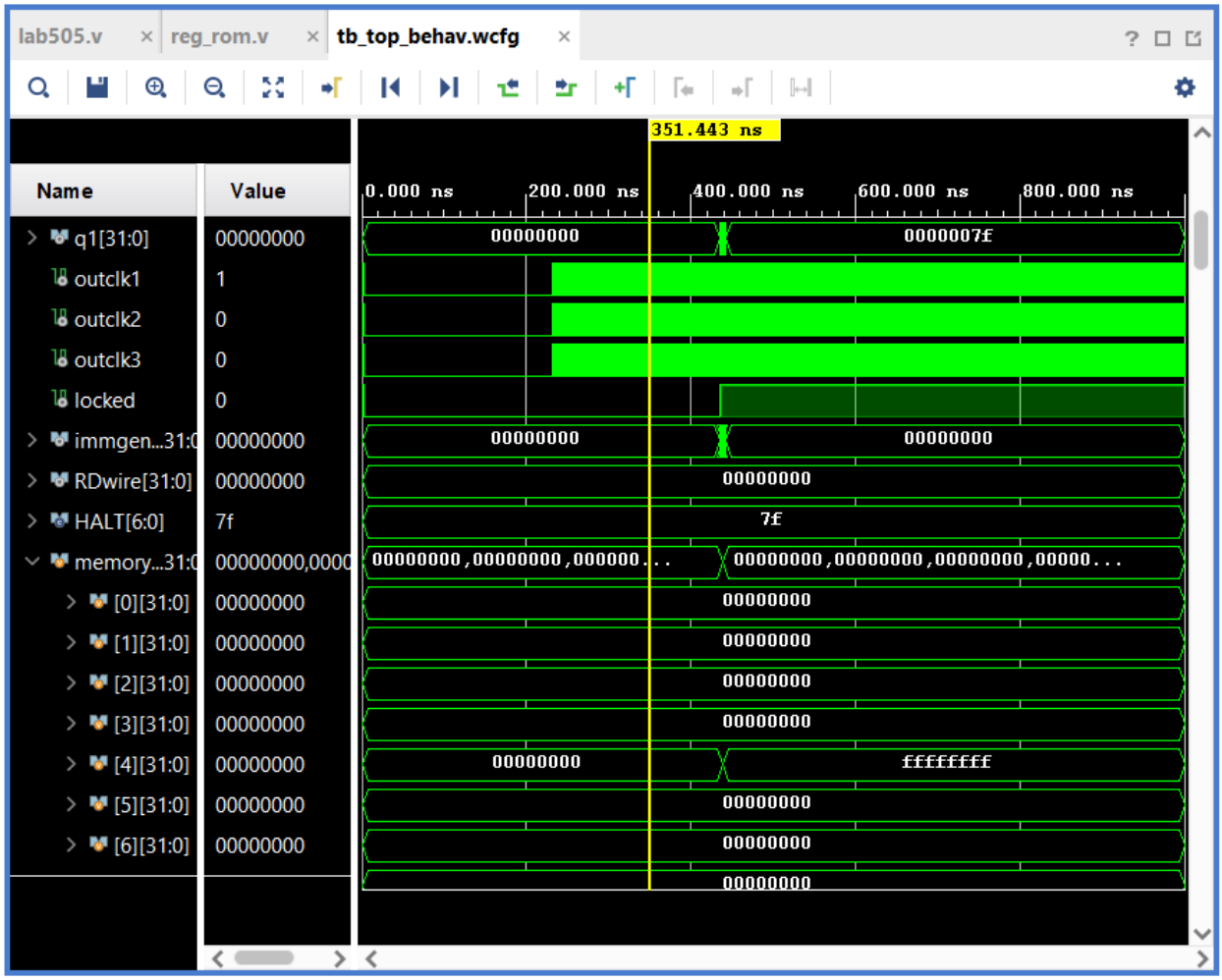

Figure 8: Memory Content of My Program

Modules and Testbench Output

ALU .v

```
module ALU
(
  output wire signed [31:0] Y,
  output zero,

  input signed [31:0] A,
  input signed [31:0] B,
  input [4:0] opcode
  );

  reg [31:0] YI;
  reg zeroI;
always @ (*) begin

case (opcode)
    5'b00111: begin
    YI = A & B;
    zeroI = 0;
     if (YI == 0)

        zeroI = 1;
     else
        zeroI = 0;
    end
    5'b00110: begin
    YI = A | B;
    zeroI = 0;
        if (YI == 0)

        zeroI = 1;
         else
        zeroI = 0;
    end
    5'b00000: begin
    YI = A + B;
        if (YI == 0)

        zeroI = 1;
             else
        zeroI = 0;
    end
    5'b10000: begin
    YI = A - B;
    if (YI == 0)
        zeroI = 1;
             else
        zeroI = 0;
    end
    5'b01000: begin //mult
    YI = A * B;
    zeroI = 0;
        if (YI == 0)
        zeroI = 1;
         else
        zeroI = 0;

    end
    5'b00001: begin
     YI = A << B;
     if (YI == 0)
     zeroI = 1;
          else
        zeroI = 0;
    end
    endcase
 end
assign Y = YI;
assign zero = zeroI;

endmodule
```

Figure 9: ALU.v

```
    initial begin
        $monitor("%t op=%b A=%d B=%d Y=%d zero=%b", $time, opcode, A, B, Y, zero);

        #0;
        opcode = 5'b00111;  // AND
        A = 32'h0005;
        B = 32'h0004;       // 32'h0004
        #1;
        opcode = 5'b00110;  // OR
        A = 32'h0007;
        B = 32'h0008;       // 32'h000f
        #1;
        opcode = 5'b00000;  // ADD
        A = 32'h0005;
        B = 32'h0004;       // 32'h0009
        #1;
        opcode = 5'b10000;  // SUB
        A = 32'h0009;
        B = 32'h0004;       // 32'h0005
        #1;
        opcode = 5'b01000;  // SUB w/ zero
        A = 32'h0007;
        B = 32'h0007;       // 32'h0000, zero = 1
        #1;
        opcode = 5'b00001;  // MUL
        A = 32'h0003;
        B = 32'h0004;       // 32'h000c
        #1;
        $stop;

    end
endmodule
```

Figure 10: ALU testbench

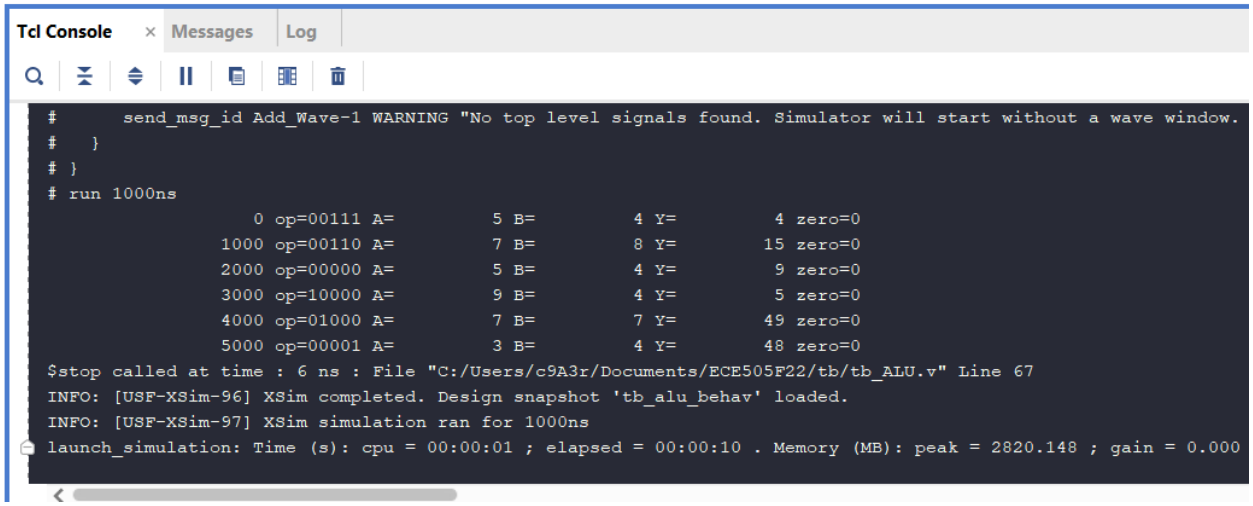

Figure 11: ALU Testbench Passed

ALU Control.v

```
module alu_control (
    input       [4:0] instr_split,  // {instr[30],instr[25], instr[14:12] (funct3)}
    input       [1:0] aluop,
    output wire     [4:0] aluopcode
);


assign aluopcode = (aluop == 2'b00) ? 5'b00000:
    (aluop == 2'b01) ? 5'b10000:
    (aluop == 2'b10) ? instr_split:
    (aluop == 2'b11) ? {2'b00, instr_split[2:0]}:
    //assign instr_splt = {(q1[30]), (q1[25]), (q1[14:12])};
    5'b00000; //default


endmodule
```

Figure 12: Module ALU_control

```
module tb_alu_control();

    reg         [4:0] instr_split;  // {instr[30],instr[25], instr[14:12] (funct3)}
    reg         [1:0] aluop;
    wire    [4:0] aluopcode;

    alu_control TEST(instr_split, aluop, aluopcode);

    initial begin
        $monitor("%t instr_split=%b aluop=%b aluopcode=%b", $time, instr_split, aluop, aluopcode);

        #0;
        instr_split = 5'b10101;
        aluop = 2'b00;  // Add
        #1;
        aluop = 2'b01;  // sub
        #1;
        aluop = 2'b10;  // r-type funct2
        #1;
        aluop = 2'b11;  // i-type funct3
        #1;
        $stop;

    end
endmodule
```

Figure 13: alu_control testbench

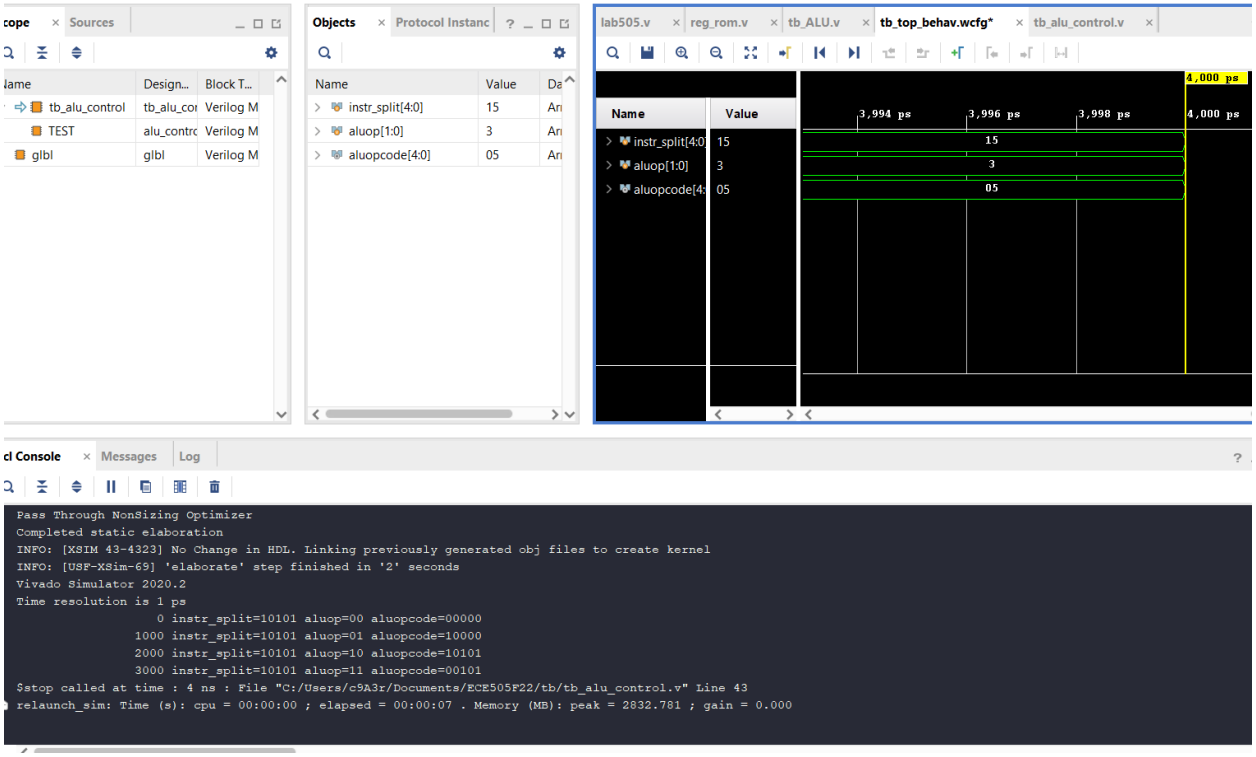

Figure 14: ALU_control Testbench Passed

Control_Unit.v

```
module control_unit(
    input   [6:0] instr,
    output [1:0] aluop,
    output  Branch,
    output  MemRead,
    output  MemtoReg,
    output  MemWrite,
    output  ALUSrc,
    output  RegWrite,
    output  Jump,
    output JALR,
    output BNE,
    output JAL
);

assign Branch = (instr == 7'b0110011) ? 1'b0: //R-type
                                (instr == 7'b0010011) ? 1'b0: // I-type
                                (instr == 7'b0100011) ? 1'b0: // S-type
                                (instr == 7'b0000011) ? 1'b0:    // L-type
                                (instr == 7'b1100011) ? 1'b1:    // B-type
                                (instr == 7'b1101111) ? 1'b1:    // J-Type

                                    1'b0; //default
assign aluop = (instr == 7'b0110011) ? 2'b10: //R-type
                                (instr == 7'b0010011) ? 2'b11: // I-type
                                (instr == 7'b0100011) ? 2'b00: // S-type
                                (instr == 7'b0000011) ? 2'b00:   // L-type
                                (instr == 7'b1100011) ? 2'b01:   // B-type
                                (instr == 7'b1101111) ? 2'b00:   // JAL

                                    2'b00;
```

```
assign MemRead = (instr == 7'b0110011) ? 1'b0: //R-type
                                 (instr == 7'b0010011) ? 1'b0: // I-type
                                 (instr == 7'b0100011) ? 1'b0: // S-type
                                 (instr == 7'b0000011) ? 1'b1:    // L-type
                                 (instr == 7'b1100011) ? 1'b0:    // B-type
                                 (instr == 7'b1101111) ? 1'b0:    // JAL

                                      1'b0; //default
assign MemtoReg = (instr == 7'b0110011) ? 1'b0: //R-type
                                 (instr == 7'b0010011) ? 1'b0: // I-type
                                 (instr == 7'b0100011) ? 1'b0: // S-type
                                 (instr == 7'b0000011) ? 1'b1:    // L-type
                                 (instr == 7'b1100011) ? 1'b0:    // B-type
                                 (instr == 7'b1101111) ? 1'b0:    // JAL

                                      1'b0; //default
assign MemWrite = (instr == 7'b0110011) ? 1'b0: //R-type
                                 (instr == 7'b0010011) ? 1'b0: // I-type
                                 (instr == 7'b0100011) ? 1'b1: // S-type
                                 (instr == 7'b0000011) ? 1'b0:    // L-type
                                 (instr == 7'b1100011) ? 1'b0:    // B-type
                                 (instr == 7'b1101111) ? 1'b0:    // JAL
                                 //(instr == 7'b0010011) ? 1'b0:  // JALR
                                      1'b0; //default
assign ALUSrc = (instr == 7'b0110011) ? 1'b0: //R-type
                                 (instr == 7'b0010011) ? 1'b1: // I-type
                                 (instr == 7'b0100011) ? 1'b1: // S-type
                                 (instr == 7'b0000011) ? 1'b1:    // L-type
                                 (instr == 7'b1100011) ? 1'b0:    // B-type
                                 (instr == 7'b1101111) ? 1'b0:    // JAL
                                      1'b0; //default
```

```
assign RegWrite = (instr == 7'b0110011) ? 1'b1: //R-type
                                  (instr == 7'b0010011) ? 1'b1: // I-type
                                  (instr == 7'b0100011) ? 1'b0: // S-type
                                  (instr == 7'b0000011) ? 1'b1:    // L-type
                                  (instr == 7'b1100011) ? 1'b0:    // B-type
                                  (instr == 7'b1101111) ? 1'b1:    // JAL

                                       1'b0; //default
assign Jump = (instr == 7'b0110011) ? 1'b0: //R-type
                                  (instr == 7'b0010011) ? 1'b0: // I-type
                                  (instr == 7'b0100011) ? 1'b0: // S-type
                                  (instr == 7'b0000011) ? 1'b0:    // L-type
                                  (instr == 7'b1100011) ? 1'b0:    // B-type
                                  (instr == 7'b1101111) ? 1'b1:    // JAL

                                       1'b0; //default

//Flags for BNE and JALR
assign BNE = (instr == 7'b1100011) ? 1'b1 : 1'b0;
assign JALR = (instr == 7'b1100111) ? 1'b1 : 1'b0;
assign JAL = (instr == 7'b1101111) ? 1'b1: 1'b0;

endmodule
```

Figure 15: Module Control Unit (Modified for BNE, JAL and JALR)

```
    initial begin
        $monitor("%t instr=%b aluop=%b Branch=%b MemRead=%b MemtoReg=%b MemWrite=%b ALUSrc=%b RegWrite=%b Jump=%b",
                 $time, instr,aluop,Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite, Jump);

        #0;
        instr = 7'b0110011; // R-type
        #1;
        instr = 7'b0010011; // I-type
        #1;
        instr = 7'b0100011; // S-type
        #1;
        instr = 7'b0000011; // L-type
        #1;
        instr = 7'b1100011; // B-type
        #1;
        instr = 7'b0000000; // default
        #1;
        instr = 7'b1101111; // JAL
        # 1
        $stop;

    end


endmodule
```

Figure 16: control unit testbench

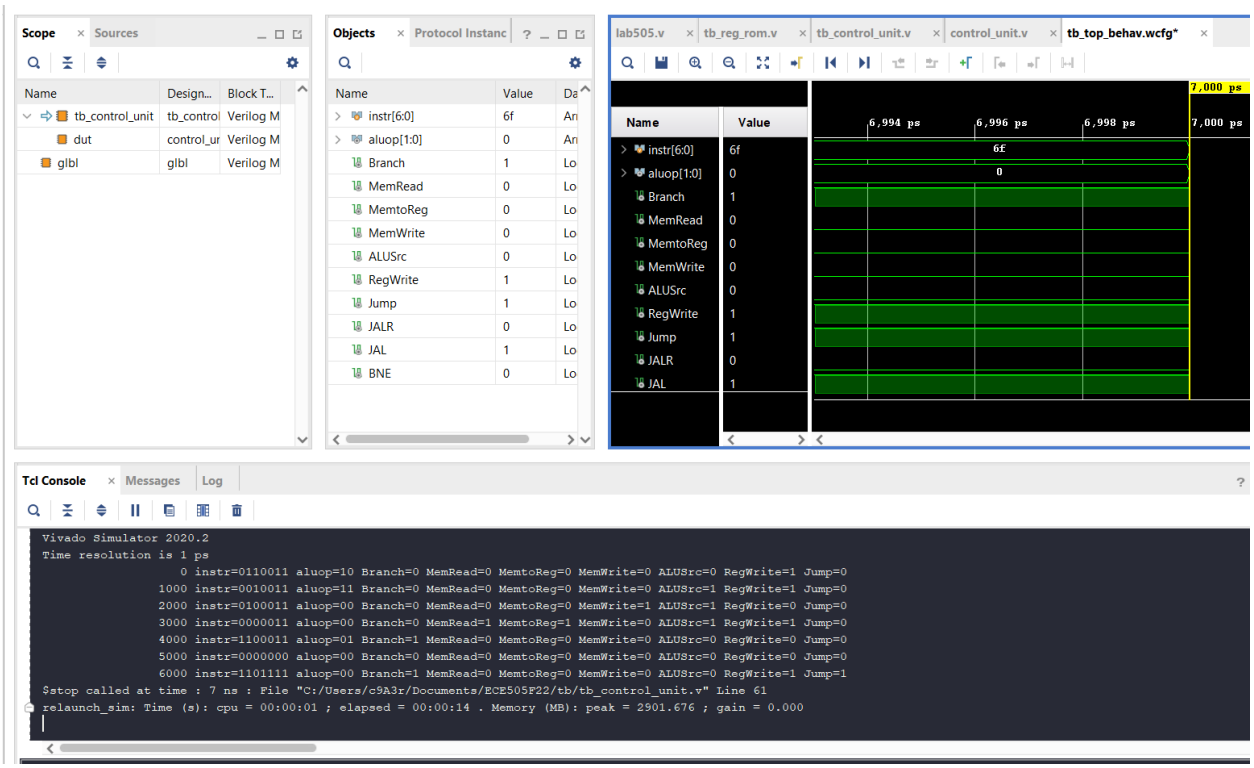

Figure 17: Control Unit Testbench Passed

Reg_file.v

```
module reg_file(clk, wren, wd, rr1, rr2, wr, rd1, rd2);
  input clk, wren; //clock and write enable ports
  input [4:0] rr1, rr2, wr; //5 bit register address inputs
  input [31:0] wd; //data to write
  output [31:0] rd1, rd2; //read data outputs

  reg [31:0] file [31:0]; //32 registers 32bits wide
   initial begin

    // init rest of your file
   file[0]  = 32'h0;
   file[1]  = 32'h0;
   file[2]  = 32'h0;
   file[3]  = 32'h0;
   file[4]  = 32'h0;
   file[5]  = 32'h0;
   file[6]  = 32'h0;
   file[7]  = 32'h0;
   file[8]  = 32'h0;
   file[9]  = 32'h0;
   file[10]  = 32'h0;
   file[11]  = 32'h0;
   file[12]  = 32'h0;
   file[13]  = 32'h0;
   file[14]  = 32'h0;
   file[15]  = 32'h0;
   file[16]  = 32'h0;
   file[17]  = 32'h0;
   file[18]  = 32'h0;
   file[19]  = 32'h0;
   file[20]  = 32'h0;
   file[21]  = 32'h0;
   file[22]  = 32'h0;
   file[23]  = 32'h0;
   file[24]  = 32'h0;
   file[25]  = 32'h0;
   file[26]  = 32'h0;
   file[27]  = 32'h0;
   file[28]  = 32'h0;
   file[29]  = 32'h0;
   file[30]  = 32'h0;
   file[31]  = 32'h0;
   end

  //should be combinational
  assign rd1 = file[rr1];  // finish your desigh
  assign rd2 = file[rr2]; // finish your desigh

  //write data Logic when the wren triggered
  always @(posedge clk) begin
    // finish your design
    if(wren) begin
          file[wr] <= wd;
     end
  end

endmodule
```

Figure 18: Module Reg_File

```
  reg_file TEST(clk, wren, wd, rr1, rr2, wr, rd1, rd2);

  initial begin
  $display("wren, wd, rr1, rr2, wr, rd1, rd2");
  clk = 1; // clock in test bench
  end

  always
  #0.50 clk = ~clk;

  always @(posedge clk) begin
  $display("RegWrite=%h, writeData=%h, address1=%h, address2=%h, writeReg=%h,  destR1=%h,  destR2=%h", wren, wd, rr1, rr2, wr, rd1, rd2);
  end

  initial begin
    #0 wren = 1;
    wr = 5'h0;
    wd = 32'h0; //write h0
    rr1 = 5'h0; //rd1 is 32'h0
    rr2 = 5'h0; //rd2 is 32'h0
    #2 wren = 1;
    wr = 5'h1;
    wd = 32'h1; //write h1
    rr1 = 5'h0; //rd1 is 32'h0
    rr2 = 5'h0; //rd2 is 32'h1
    #2 wren = 1;
    wr = 5'h2;
    wd = 32'h2; //write h2
    rr1 = 5'h0; //rd1 is 32'h0
    rr2 = 5'h1; //rd2 is 32'h1
    #2 wren = 1;
    wr = 5'h7;
    wd = 32'h12; //write h2
    rr1 = 5'h2; //rd1 is 32'h2
    rr2 = 5'h1; //rd2 is 32'h1
    #2 wren = 0;
    wr = 5'h7;
    wd = 32'h12; //don't write
    rr1 = 5'h2; //rd1 is 32'h2
    rr2 = 5'h7; //rd2 is 32'h12

    #2;
    #2;
    $stop;
  end
endmodule
```

Figure 19: reg_file testbench

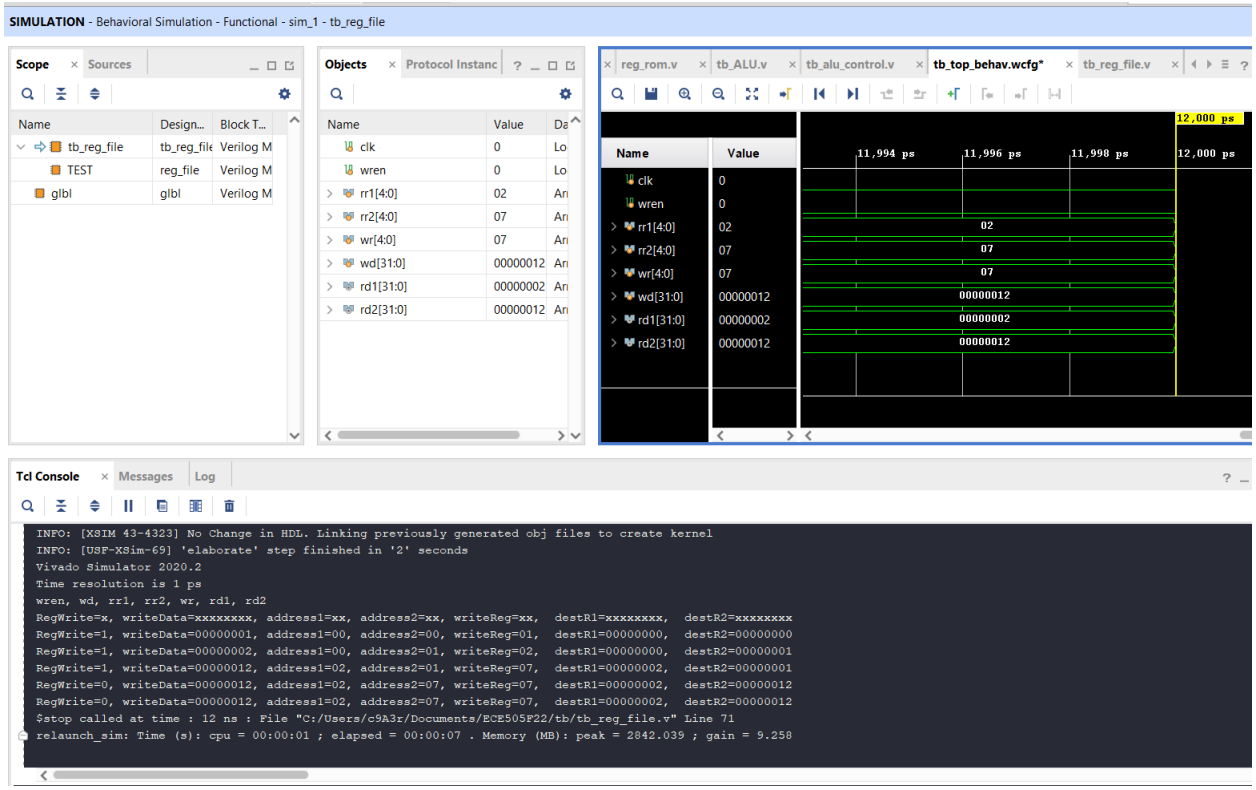

Figure 20: reg_file Testbench Passed (Factorial 6)

Imm_gen.v

```
//intermediate generator module
//takes in all 32 bits of instructions
//outputs the 12 bit immediate based on I-type or SB-type or S-type

module imm_gen(instr, out);
   input [31:0] instr; //32 bit instruction
   output [31:0] out; //  output reg [31:0] out;
   wire [6:0] opcode = instr[6:0];

    assign out  = (opcode == 7'b0010011) ? {{20{instr[31]}}, instr[31:20]} : (opcode == 7'b0000011) ? {{20{instr[31]}}, instr[31:20]} : //I-type
                  (opcode == 7'b0100011) ? {{20{instr[31]}}, instr[31:25], instr[11:7]} : // S-type
                  (opcode == 7'b1100011) ? {{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0} : // B type
                  (opcode == 7'b1101111) ? {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0} : // JAL type
                  (opcode == 7'b1110011) ? {{20{instr[31]}}, instr[31:20]} : //u-type
                        0; // default


endmodule
```

Figure 21: Module imm_gen

```
module tb_imm_gen;
  reg [31:0] instr; //32 bit instruction
  wire [31:0] out;

  imm_gen TEST(instr, out);

  initial begin
  $display("instr, out");
  end

  initial begin
  $monitor("%d, %d", instr, out);
    #0 instr = 32'b00000000011100000000000000010011; //I-type. d7
    #1 instr = 32'b11100000011100000000000000010011; //I-type. d3591
    #1 instr = 32'b11100000011100000000000000100011; //S-type. d3584
    #1 instr = 32'b00000000011100000000000110100011; //S-type. d3
    #1 instr = 32'b00000000000000000000000101100011; //SB-type. d1
    #1 instr = 32'b00000010000000000000000001100011; //SB-type. d16
    #1 instr = 32'b00000000000000000000000011100011; //SB-type. d1024
    #1 instr = 32'b10000000000000000000000001100011; //SB-type. d2048
    #1 instr = 32'b00000010000000000000000000110011; //R-type. d0
    #1;
    $stop;
  end
endmodule
```

Figure 22: imm_gen testbench

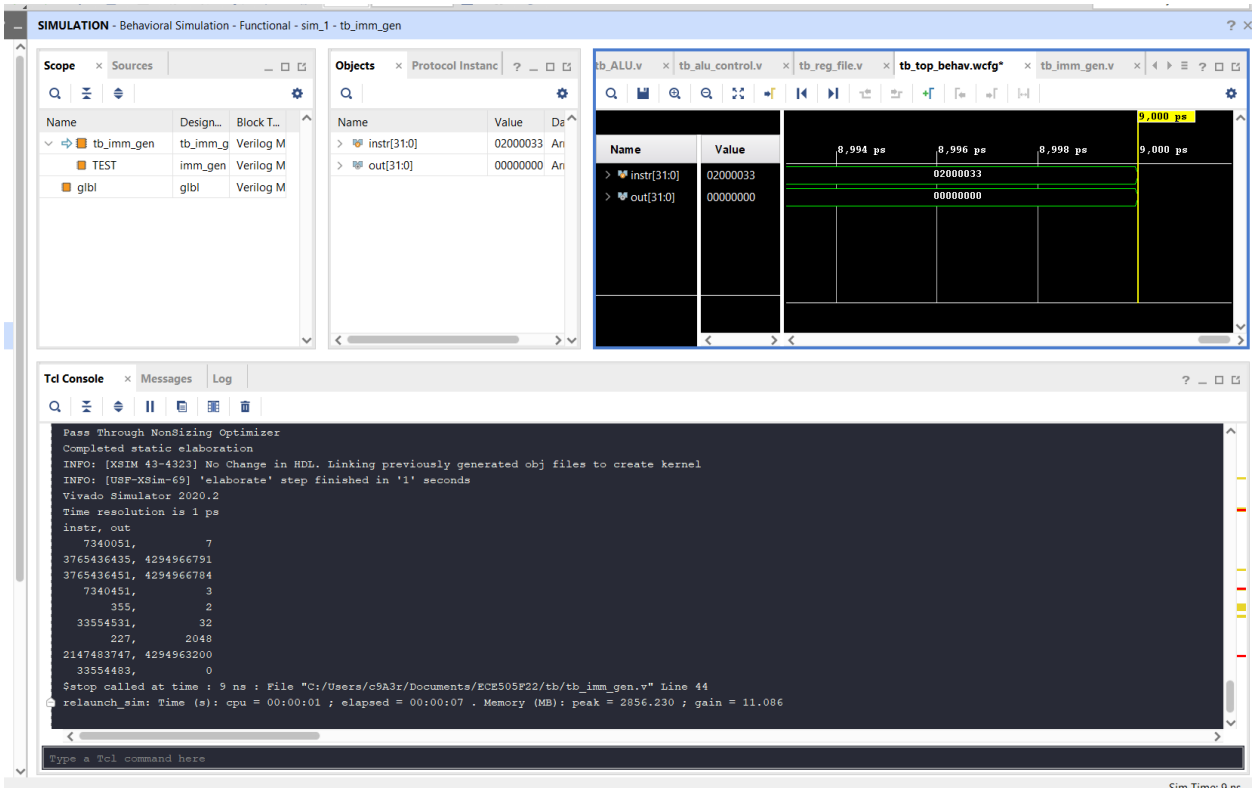

Figure 23: imm_gen Testbench Passed (Factorial 6)

IP Block Clock mmcm wiz_0

```
module tb_clkWizard(
    );
    reg clk=1;
    wire clk1, clk2, clk3;
    reg reset=0;
    wire lock;
    always
        #1 clk = ~clk;

    clk_wiz_0 clkWiz(
        .clk_out1(clk1),
        .clk_out2(clk2),
        .clk_out3(clk3),
        .reset(reset),
        .locked(lock),
        .clk_in1(clk)
    );

    initial begin
        #1
        reset = 1;
        #1
        reset = 0;
    end

endmodule
```

Figure 24: Clk_Wiz testbench

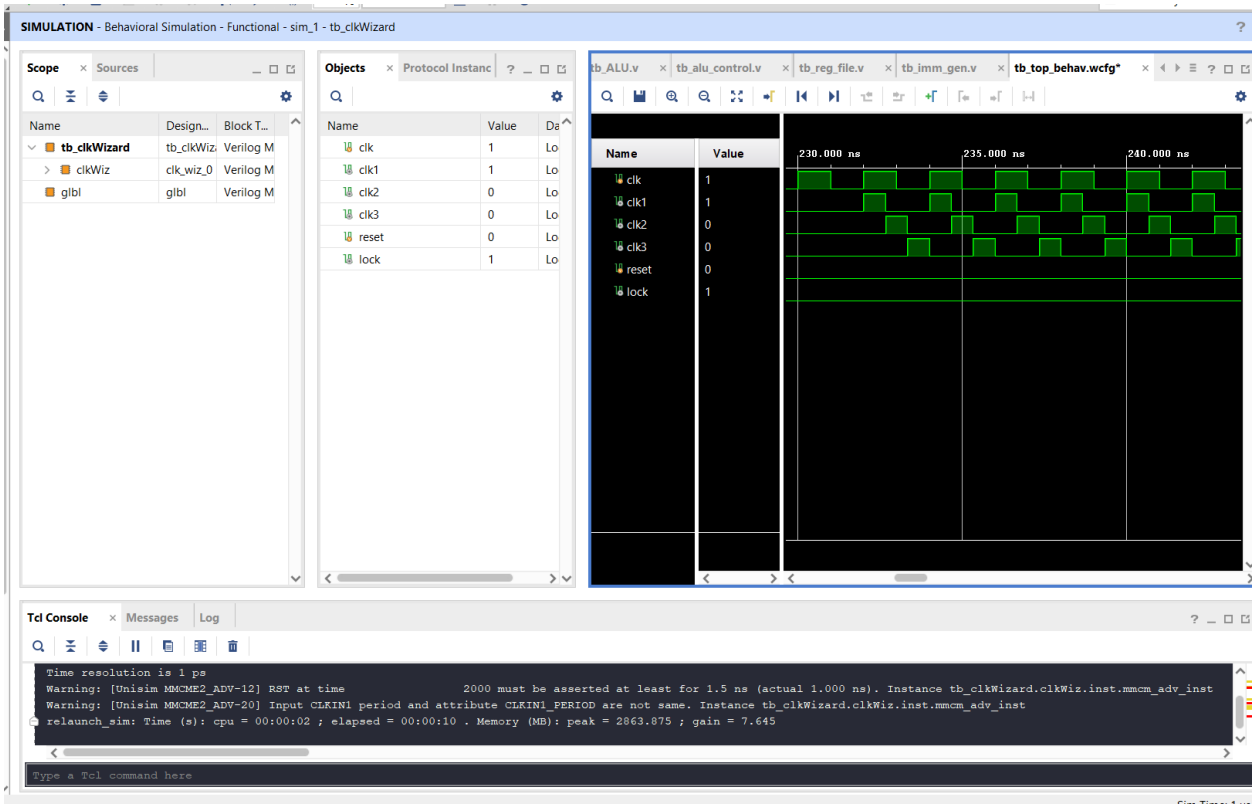

Figure 25: Clock Testbench with adjusted timing to meet the 20MHz requested by time.xdc

IP Block ram0 blk_mem_gen0

```
    // generate clk
    always
    #0.5 clk = ~clk;

    // try write and read
    initial begin
    #0
    en = 1;
    en_w = 0;
    addr = 8'b0;
    dataIn = 32'h6;

    // [0][:] = h6
    $display(ram0.clka);
    #1
    en = 1;
    en_w = 1;
    addr = 8'b0;
    dataIn = 32'h6;

    // read [0]
    #1
    en = 1;
    en_w = 0;
    addr = 8'b0;
    dataIn = 32'h7;

    // [0][:] = h6
    #1
    en = 1;
    en_w = 1;
    addr = 8'b0;
    dataIn = 32'h6;

    // read = [0]
    #1
    en = 1;
    en_w = 0;
    addr = 8'b0;
    dataIn = 32'h6;

    end


endmodule
```

Figure 26: block RAM testbench

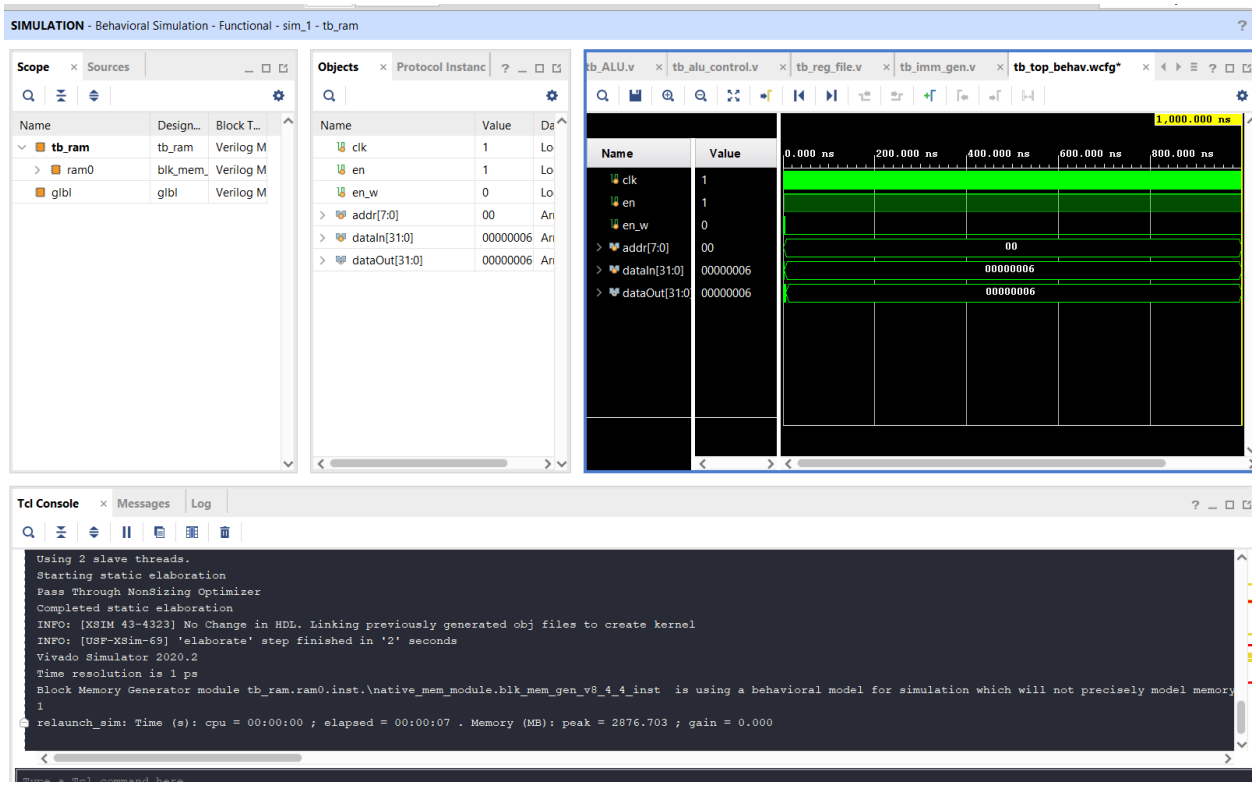

Figure 27: Block Ram Testbench Passed (Factorial 6)

Reg_rom.v

```
module reg_rom(addr, q);
  input [4:0] addr; //5 bit register address input
  output [31:0] q;  //read data outputs

  reg [31:0] file [31:0]; //32 registers 32bits wide
  integer i;

  initial begin
  // Init the instruction codes

  //factorial 6 //part b

  file[0] = 32'h00600513; //addi a0, x0, 6
  file[1] = 32'h00c000ef; //jal ra, fact (12)
  file[2] = 32'h00a02023; //sw a0, 0(x0)
  file[3] = 32'h0000007f; //HALT
  //fact:
  file[4] = 32'hff810113; //addi sp, sp, -8
  file[5] = 32'h00112223; //sw  ra, 4(sp)
  file[6] = 32'h00a12023; //sw x10 0(x2)
  file[7] = 32'hfff50513; //addi x10 x10 -1
  file[8] = 32'h00051863; //bne a0, x0, else (16)
  file[9] = 32'h00100513; //addi a0, x0, 1
  file[10] = 32'h00810113; //addi sp, sp, 8
  file[11] = 32'h00008067; //jalr x0, 0(ra)
  //else:
  file[12] = 32'hfe1ff0ef; //jal ra, fact(-32)
  file[13] = 32'h00050293; //addi t0, a0,0
  file[14] = 32'h00012503; //lw  a0, 0(sp)
  file[15] = 32'h00412083; //lw  ra, 4(sp)
  file[16] = 32'h00810113; //addi sp, sp, 8
  file[17] = 32'h02550533; //mul a0, a0, t0
  file[18] = 32'h00008067; //jalr x0, 0(ra)
```

```
  //prog2
//   file[0] = 32'h00800293;
//   file[1] = 32'h00f00313;
//   file[2] = 32'h0062a023;
//   file[3] = 32'h005303b3;
//   file[4] = 32'h40530e33;
//   file[5] = 32'h03c384b3;
//   file[6] = 32'h00428293;
//   file[7] = 32'hffc21903;
//   file[8] = 32'h41248933;
//   file[9] = 32'h00291913;
//   file[10] = 32'h0122a023;
//   file[11] = 32'h0000007f;//HALT
//   file[12] = 32'h00000000;
//   file[13] = 32'h00000000;
//   file[14] = 32'h00000000;
//   file[15] = 32'h00000000;
//   file[16] = 32'h00000000;
//   file[17] = 32'h00000000;
//   file[18] = 32'h00000000;

//   // prog1
//   file[0] = 32'h00000093;
//   file[1] = 32'h01000113;
//   file[2] = 32'h06400193;
//   file[3] = 32'h00800213;
//   file[4] = 32'h002082b3;
//   file[5] = 32'h00418333;
//   file[6] = 32'h0050a023;
//   file[7] = 32'h00612223;
//   file[8] = 32'h0000007f;//HALT
```

```
   //Fill the rest with 0's
  for (i = 19; i < 32; i = i + 1)begin
    file[i] = 32'h00000000;
  end
  end

  // output the Instruction code
  assign q = file[addr];

endmodule
```

Figure 28: reg_rom with Factorial 6 loaded as the program of choice

Program 1 and 2 are commented out

```
module rb_reg_rom;
reg [4:0] addr; //5 bit register address input
wire [31:0] q; //read data outputs
integer i;

  reg_rom TEST(addr, q);

  initial begin
  $display("t=%d, addr=%d, instr=%h", $time, addr, q);
  end

  initial begin
    for (i = 0; i < 32; i = i + 1)begin
      addr = i;
      #1;
    end
    $stop;
  end
endmodule
```

Figure 29: reg_rom testbench

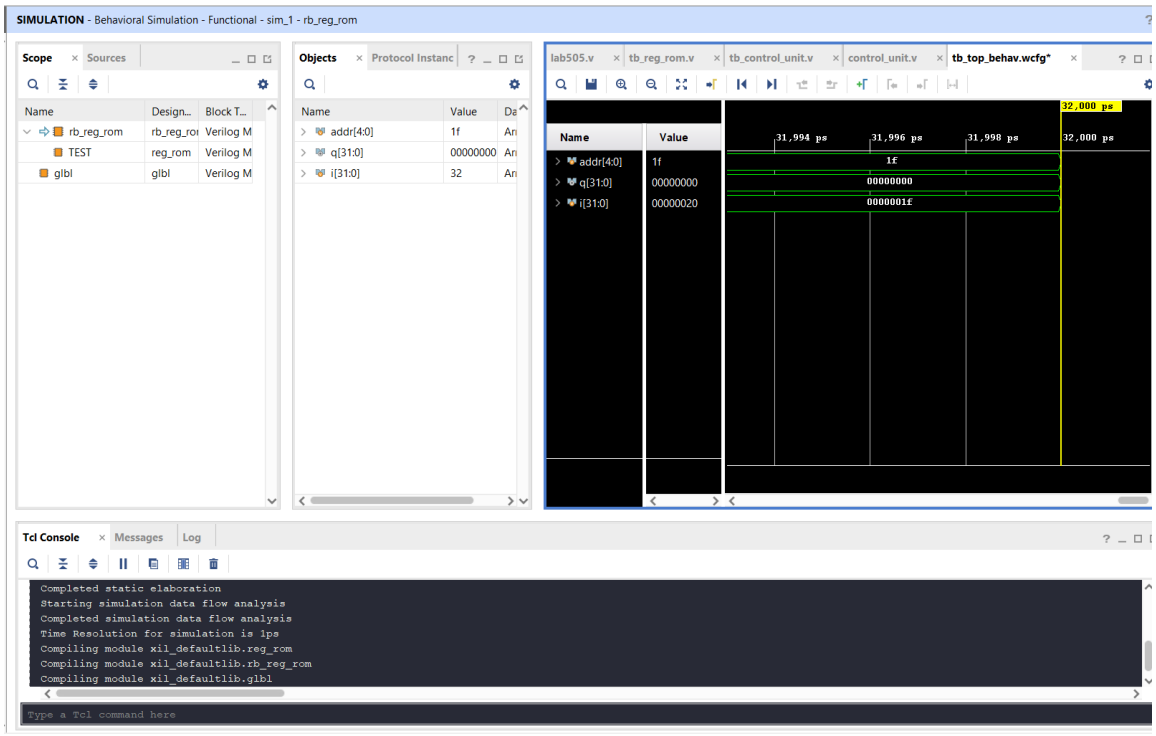

Figure 30: reg_rom Testbench Passed (Factorial 6)

Lab505.v

```
`timescale 1ns/1ns

module lab505(CLOCK_20);
   input CLOCK_20;

   // You need to set it as follow when implementation your design
   (* DONT_TOUCH = "TRUE" *)reg  [10:0]           PC;        // PC current
   (* DONT_TOUCH = "TRUE" *)wire [10:0]        PC_next;   // PC next to be latched
   (* DONT_TOUCH = "TRUE" *)wire [10:0]        PC_plus;   // PC + 4
   (* DONT_TOUCH = "TRUE" *)wire [10:0]        PC_offset; // PC offset for branching
   (* DONT_TOUCH = "TRUE" *)wire                   to_branch; // branch condition

   (* DONT_TOUCH = "TRUE" *)wire run;

   (* DONT_TOUCH = "TRUE" *)wire [31:0]        A;         // ALU input A
   (* DONT_TOUCH = "TRUE" *)wire [31:0]        B;         // ALU input B
   (* DONT_TOUCH = "TRUE" *)wire signed [31:0] Y;          // From a1 of ALU.v
   (* DONT_TOUCH = "TRUE" *)wire [1:0]      aluop;         // From ctrl of control_unit.v
   (* DONT_TOUCH = "TRUE" *)wire [4:0]      aluopcode;     // From aluctrl of alu_control.v
   (* DONT_TOUCH = "TRUE" *)wire            zero;          // From a1 of ALU.v


   //Control Unit
   (* DONT_TOUCH = "TRUE" *)wire Branch;
   (* DONT_TOUCH = "TRUE" *)wire MemRead;
   (* DONT_TOUCH = "TRUE" *)wire MemtoReg;
   (* DONT_TOUCH = "TRUE" *)wire MemWrite;
   (* DONT_TOUCH = "TRUE" *)wire ALUSrc;
   (* DONT_TOUCH = "TRUE" *)wire RegWrite;
   (* DONT_TOUCH = "TRUE" *)wire Jump;

    // CU Flags fof BNE and JALR
   (* DONT_TOUCH = "TRUE" *)wire BNE;
   (* DONT_TOUCH = "TRUE" *)wire JALR;
   (* DONT_TOUCH = "TRUE" *)wire JAL;


   //register file
   (* DONT_TOUCH = "TRUE" *)wire [31:0]        rd1_w;
   (* DONT_TOUCH = "TRUE" *)wire [31:0]        rd2_w;
   (* DONT_TOUCH = "TRUE" *)wire [31:0]        wd;
  // wire [31:0] wd;

   //reg_rom
   (* DONT_TOUCH = "TRUE" *)wire [31:0] q1;

   //Output clks
   (* DONT_TOUCH = "TRUE" *)wire outclk1;
   (* DONT_TOUCH = "TRUE" *)wire outclk2;
   (* DONT_TOUCH = "TRUE" *)wire outclk3;
   // remeber to use your locked signal
   (* DONT_TOUCH = "TRUE" *)wire locked;
   //(* DONT_TOUCH = "TRUE" *)wire clk_fb;
   //imm_gen
   (* DONT_TOUCH = "TRUE" *)wire [31:0]immgenout;

   //RAM
   (* DONT_TOUCH = "TRUE" *)wire [31:0] RDwire;
```

```
   // your PC intiial is better to be -4 to make sure the first clock will trigger your first instruction
   initial
    PC = -4;
    parameter HALT = 7'b1111111;

// design run signal which indicate run and halt
   assign run = (ql[6:0] != HALT) ? 1 : 0;


    // =============== PC =======================
    always @(posedge outclkl & locked) begin
      if (run & locked) begin
         PC <= PC_next;
      end
    end

   // finish your PC design
      assign PC_plus = PC + 4;
      assign PC_offset = PC + immgenout;
      assign PC_next = (Jump  & !BNE & !JALR) ? PC_offset :
                       (Branch & !zero & BNE) ? PC_offset :
                       (Branch &  zero  & BNE)  ? PC_plus   :
                       (JALR) ? (rdl_w + immgenout)  :
                       PC_plus;
      assign to_branch = ql[12] ^ zero; //DAFQ IS THIS
      //assign PC_next = ;



   // ================ ALU =================
   // prepare for A and B
   assign A = rdl_w;
   assign B = (ALUSrc) ? immgenout : rd2_w;  // 1 = immediate, 0 = rd2

    ALU alu(
       .Y (Y[31:0]),
       .zero (zero),
       .A (A[31:0]),
       .B (B[31:0]),
       .opcode (aluopcode[4:0]));

    // ================ controller =================

    control_unit control_unit(
        .instr (ql[6:0]),
        .aluop (aluop[1:0]),
        .Branch (Branch),
        .MemRead (MemRead),
        .MemtoReg (MemtoReg),
        .MemWrite (MemWrite),
        .ALUSrc (ALUSrc),
        .RegWrite (RegWrite),
        .Jump (Jump),
        .JALR (JALR),
        .BNE (BNE),
        .JAL (JAL));
```

```
    // ================== ALU controller ============


     alu_control alu_control(
         .instr_split ({ql[30], ql[25], ql[14:12]}),
         .aluop       (aluop[1:0]),
         .aluopcode   (aluopcode[4:0]));


    // ================ register file =================


    assign wd[31:0] = (JALR||JAL) ? PC_plus : (MemtoReg ?  RDwire[31:0]: Y[31:0]);

     reg_file reg_file(
         .clk (outclk3 & locked),
         .wren (RegWrite),
         .rrl (ql[19:15]),
         .rr2 (ql[24:20]),
         .wr (ql[11:7]),
         .wd (wd[31:0]),
         .rd1 (rd1_w[31:0]),
         .rd2 (rd2_w[31:0]));

     // ================ immediate generator =======

     imm_gen imm_gen(
         .instr (ql[31:0]),
         .out (immgenout[31:0]));

     // ================== RAM ========================
     blk_mem_gen_0 ram0 (
         .clka(outclk2 & locked),
         .ena(MemRead | MemWrite),
         .wea(MemWrite),
         .addra(Y[31:0]),
         .dina(rd2_w[31:0]),
         .douta(RDwire[31:0])
     );

     // ================== Register ROM ===============================
     reg_rom reg_rom(
         .addr (PC >> 2), //shift to use PC +4 or else it would be PC +1
         .q (ql[31:0]));

     // ================== MMCM ==============================
     //in fetch/PC clk1
     //reg_file is clk 3
     //RAM/Data mem is clk2

     clk_wiz_0 clkWiz(
         .clk_out1(outclk1),
         .clk_out2(outclk2),
         .clk_out3(outclk3),
         .reset(0),
         .locked(locked),
         .clk_in1(CLOCK_20));

endmodule
```

Figure 31: lab505.v The Top File

```
module tb_top(

    );
    reg clk=0;

    always
        #1 clk = ~clk;

    lab505 uut (.CLOCK_20(clk));

endmodule
```

Figure 32: top file testbench when simulating the whole CPU

Time.xdc

Constraint file

```
create_clock -add -name CLOCK_20 -period 50.00 [get_ports CLOCK_20]
```

Timing Constraints

After adjusting the clocks to meet the 20MHz requirement. Below is the implemented design passing the timing constraints

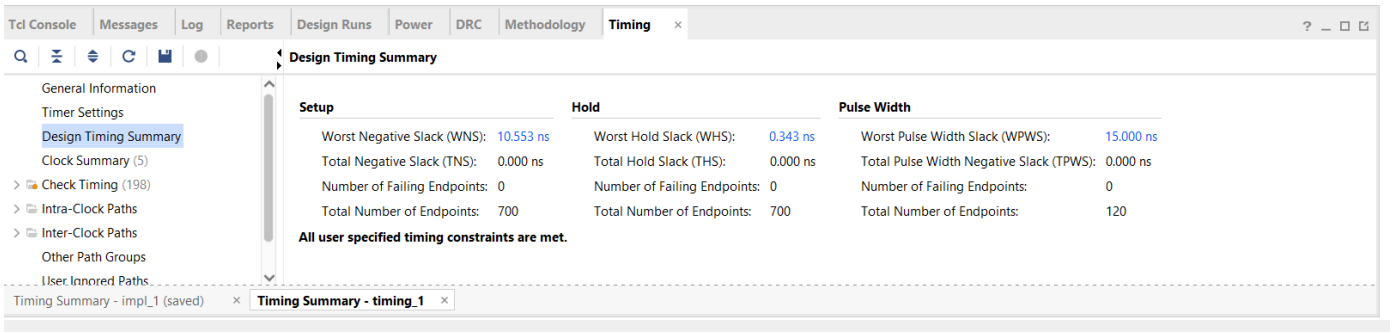

Figure 33: Timing Summary

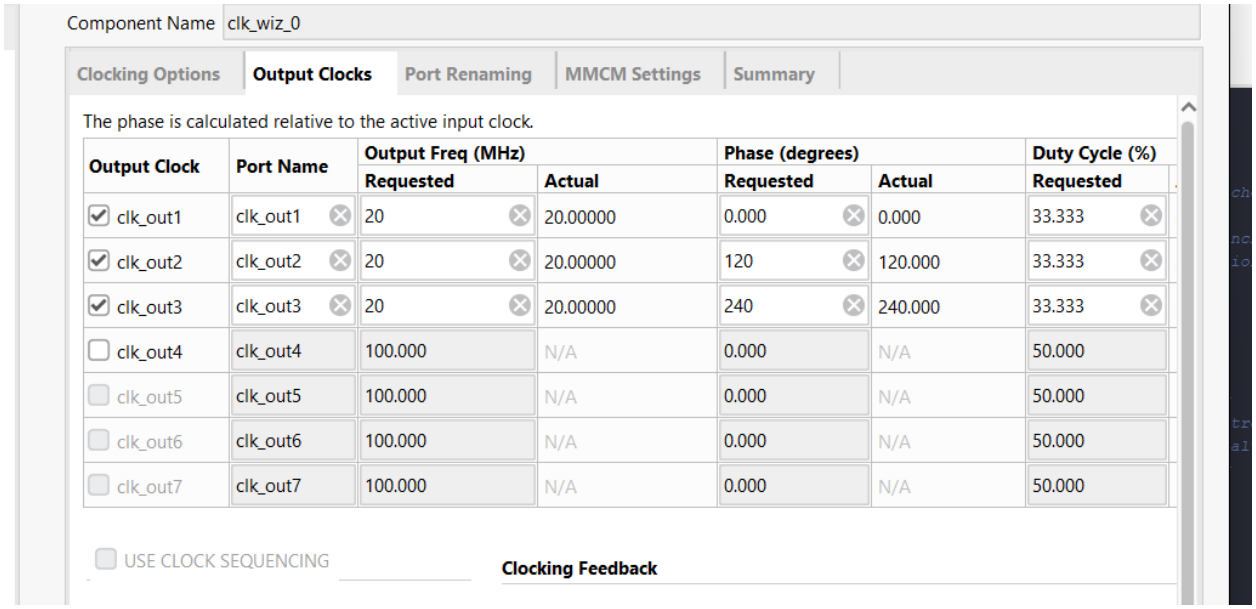

Figure 34: Output Clocks at 20MHz

FPGA Resources

From the Implementation Report here is the activity

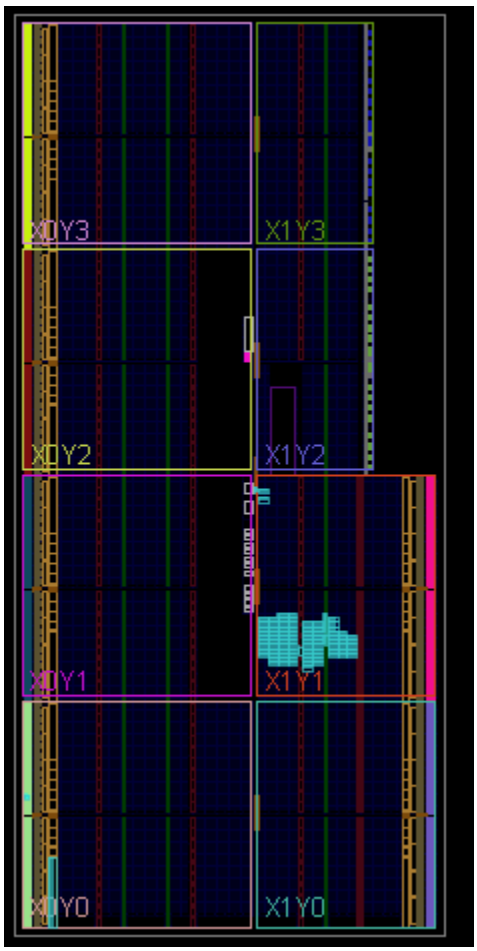

Figure 35: Resource Map; CLBs in X1Y1

Reports | Design Runs | Power | DRC | Timing | **Utilization**

Hierarchy

Name	Slice LUTs (41000)	Slice Registers (82000)	F7 Muxes (20500)	Slice (10250)	LUT as Logic (41000)	LUT as Memory (13400)	Block RAM Tile (135)	DSPs (240)	Bonded IOB (300)	BUFGCTRL (32)	MMCME2_ADV (6)
N lab505	570	44	1	159	522	48	0.5	3	1	6	1
alu (ALU)	355	33	1	106	355	0	0	3	0	0	0
clkWiz (clk_wiz_0)	0	0	0	0	0	0	0	0	0	4	1
ram0 (blk_mem_gen_0)	0	0	0	0	0	0	0.5	0	0	0	0
reg_file (reg_file)	49	0	0	13	1	48	0	0	0	1	0

Figure 36: Utilization Report

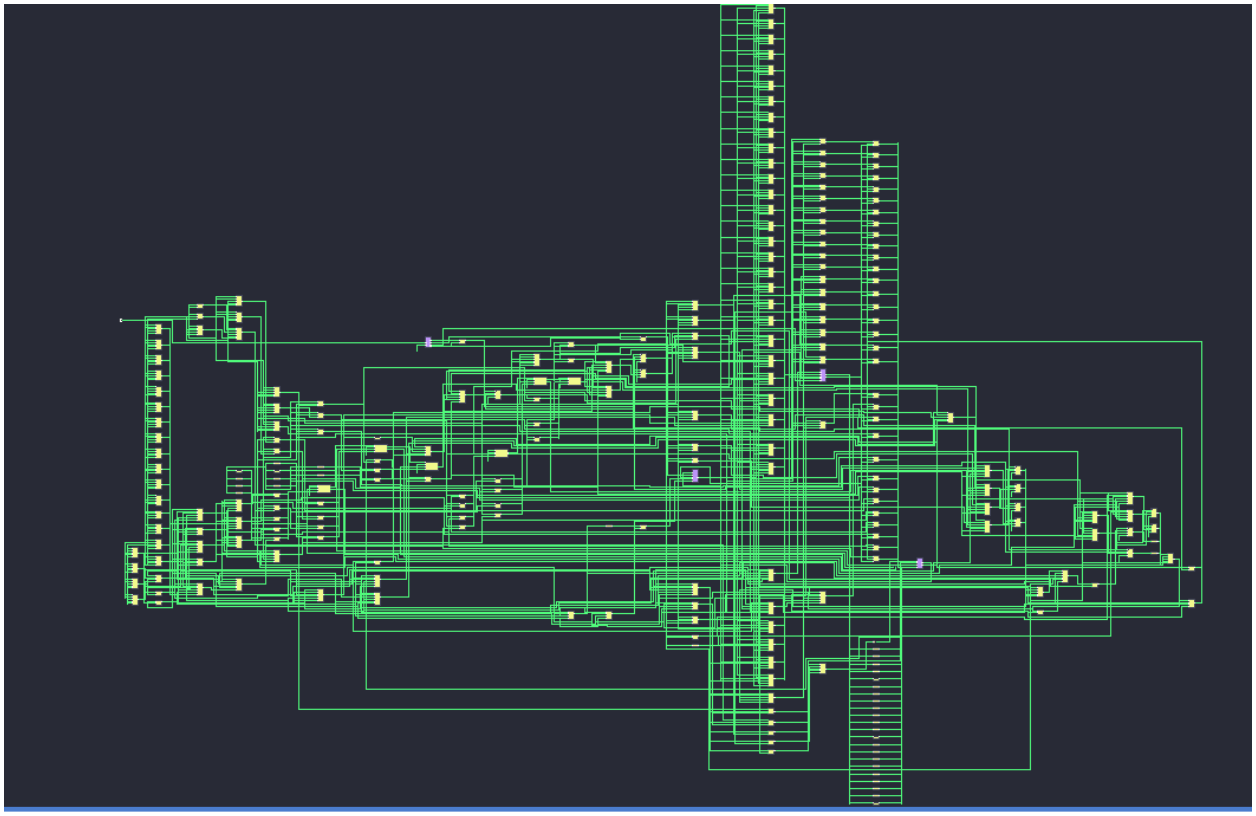

Figure 37: Schematic

Appendix

After a few tries, the fastest the Clock would go with still meeting the time requirements was 50MHz. The base clock speed was 20MHz.

50MHz/20MHz = 2.5 Faster

Here is the Timing Report:

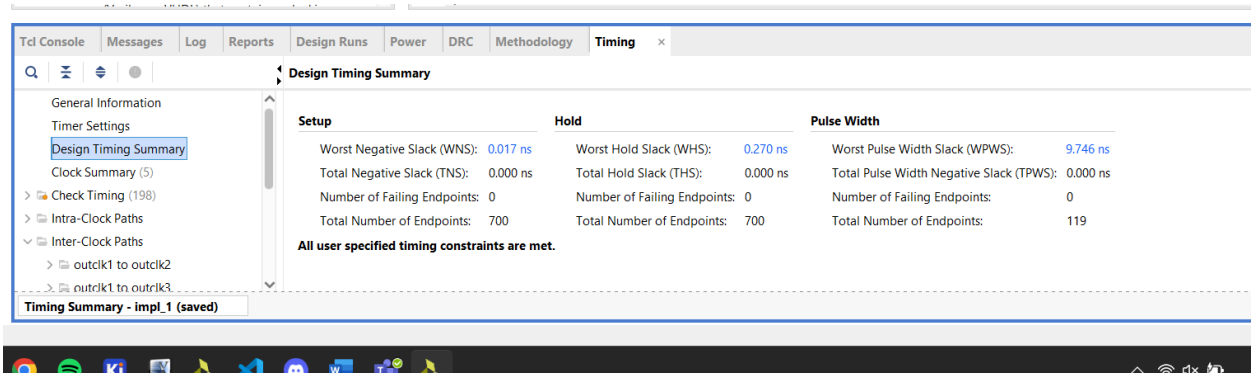

Figure 38: 50MHz timing report

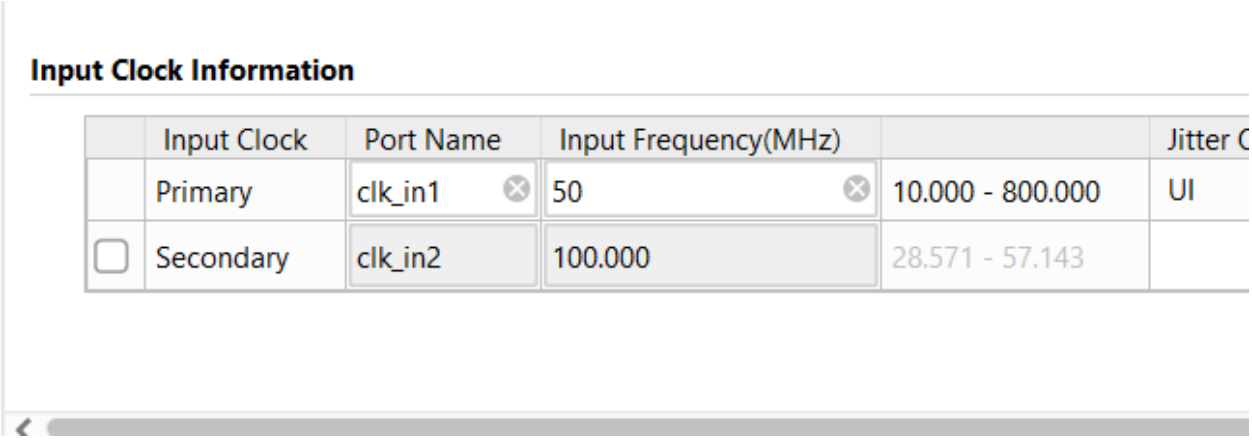

Figure 39: 50MHz input CLK

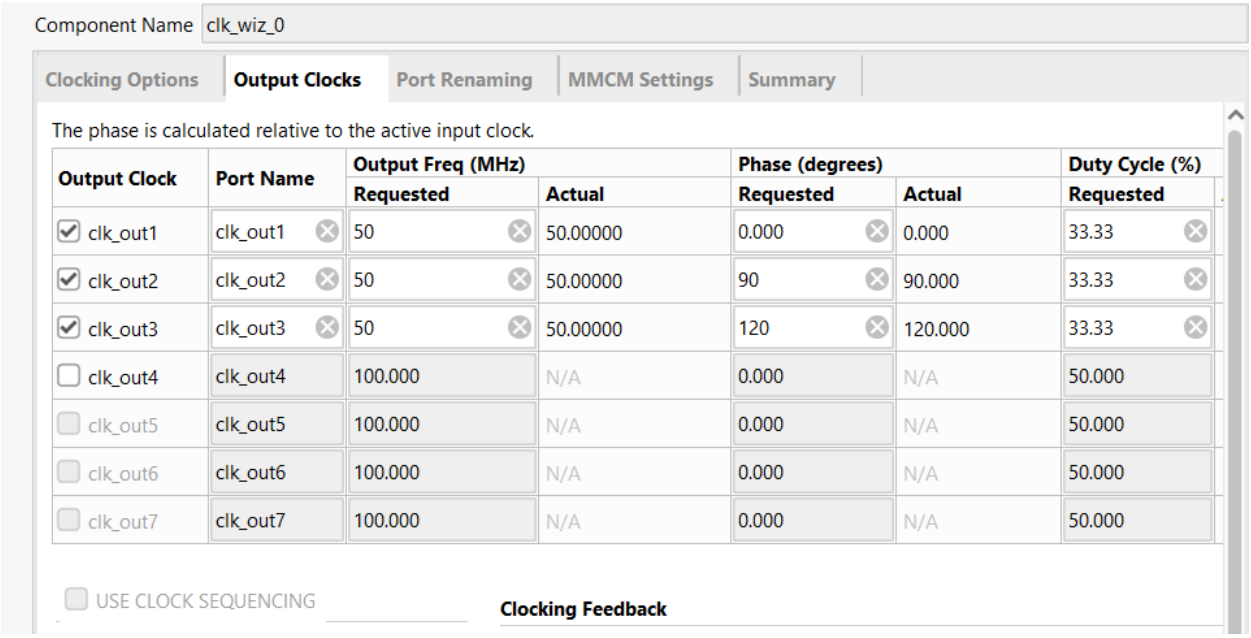

Figure 40: Output CLK frequencies

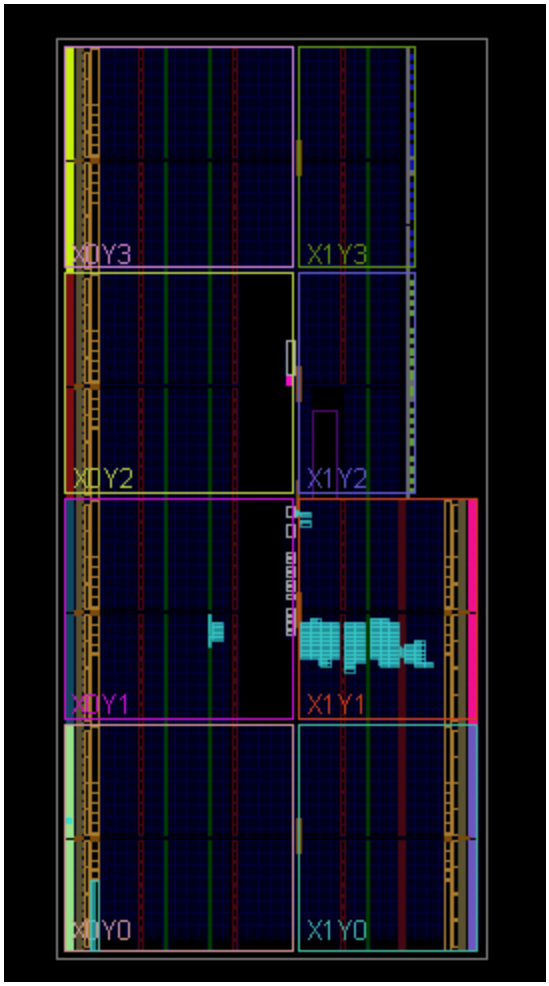

Figure 41: 50MHz Resource Ma