Lab Report #3: Simple Processor

Linh Ngo

Brian Dawson

ELC 363 - Computer Engineering Laboratory I

The College of New Jersey

Department of Electrical and Computer Engineering (ECE)

Introduction including Problem Description:

In this lab the students were required to use the Xilinx design package for FPGAs to construct and implement a simple processor. The students needed to study the architecture of the processor in order to process a set a given instruction word formats with 16-bit word-length. The students used an architecture diagram and a controller flow diagram in order to create the processor. The students also used the Von Neuman approach to design the processor as well as implementing machine language code to test the processor.

Results:

a. RTL for each instruction and addressing mode combination

NOT

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S2 : AC <= ~AC

INCA

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S3 : AC <= AC + 1

JPA (If AC <= 0)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1

JPA (If AC > 0 AND AM = 0) (DIRECT)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S7 : PC <= IR (Address Portion)

JPA (If AC > 0 AND AM =1) (INDIRECT)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S4 : MA <= IR (Address Portion)
4. S5 : MD <= M[MA] (Data in memory address)
5. S6 : PC <= MD

STA (AM = 0) (DIRECT)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S8 : MA <= IR (Address Portion)
4. S11 : M[MA] <= AC and AC <= 0

STA (AM = 1) (INDIRECT)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S8 : MA <= IR (Address Portion)
4. S9 : MD <= M[MA] (Data in memory address)
5. S10 : MA <= MD
6. S11 : M[MA] <= AC and AC <= 0

LDA (AM = 0) (Direct)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S8 : MA <= IR (Address Portion)
4. S12 : MD <= M[MA]
5. S16 : AC <= MD

LDA (AM = 1) (Indirect)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S8 : MA <= IR (Address Portion)
4. S12 : MD <= M[MA]
5. S13 : MA <= M[MD]
6. S14 : MD <= M[MA]
7. S16 : AC <= MD

ADC (AM = 0) (Direct)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1

3. S8 : MA <= IR (Address Portion)
4. S12 : MD <= M[MA]
5. S15 : AC <= AC + C + MD

ADC(AM = 1) (Indirect)

1. S0 : IR<=M(PC)
2. S1 : PC <= PC + 1
3. S8 : MA <= IR (Address Portion)
4. S12 : MD <= M[MA]
5. S13 : MA <= M[MD]
6. S14 : MD <= M[MA]
7. S15 : AC <= AC + C + MD

b. Number of clock cycles for each instruction and addressing mode combination

Instruction with addressing mode	Clock cycles
NOT	3
ADC Indirect	7
ADC Direct	5
JPA Direct	3
JPA Indirect	5
JPA if ACC <= 0	2
INCA	3
STA Direct	4
STA Indirect	6
LDA Direct	5
LDA Indirect	7

c. Verilog design code

```
`timescale 1ns / 1ps
module alu(
    input wire [15:0]a,b,
    input wire [2:0]control,
    input wire ci,
    output reg [15:0]y,
    output reg co
    );


    always @(*) begin
        case(control)
            3'b000: y =~a;  //NOTa = Not the Accumulator(Invert it)
            3'b001: {co,y} <= a+b+ci;//Add to input with carry
            3'b010: y = (a + 1); //Increment the Accumulator
            3'b011: y = 0; //Zeroing the Accumulator (Clearing accumulator)
            3'b100: y = b; //Passing B (Used in loading accumulator)
        endcase
    end
endmodule


/**********************************************************************************************
**/
module controller(
    input [3:0]opcode,
    input pos, //most significant bit
    input clk, reset,
    //input am,

    output reg IR_enable, MD_enable, MA_enable, PC_enable, AC_enable, C_enable,
    output reg MUX1, MUX3,
    output reg [2:0] ALU_control,
    output reg [1:0] MUX2,
    output reg RWn
);

reg [4:0] present_state, next_state;

    parameter S00 = 5'h00, S01 = 5'h01, S02 = 5'h02, S03 = 5'h03,S04 = 5'h04,
    S05 = 5'h05, S06 = 5'h06, S07 = 5'h07, S08 = 5'h08,S09 = 5'h09, S10 = 5'h0A,
    S11 = 5'h0B, S12 = 5'h0C, S13 = 5'h0D,S14 = 5'h0E, S15 = 5'h0F, S16 = 5'h10;

    initial begin
        present_state=S00;
    end

    always@(negedge clk or posedge reset) begin
        if (reset == 1'b1) begin
            present_state=S00;
        end
        else begin
            present_state = next_state;
        end
    end
```

```
always @(present_state) begin
    case(present_state)

        S00: begin
            IR_enable = 1;
            MD_enable = 0;
            MA_enable = 0;
            PC_enable = 0;
            AC_enable = 0;
            C_enable= 0;
            MUX1 = 0;
            MUX2 = 0;
            MUX3 = 0;
            RWn=1;
            next_state = S01;
        end


        S01: begin
            IR_enable = 0;
            MD_enable = 0;
            MA_enable = 0;
            PC_enable = 1;
            AC_enable = 0;
            C_enable= 0;
            MUX1 = 0;
            MUX2 = 0;
            MUX3 = 0;
            RWn=1;

            if(opcode[3:1] == 3'b000) begin
                next_state = S02;
            end

            else if (opcode[3:1] == 3'b011) begin
                next_state = S03;
            end

            else if (opcode[3:1] == 3'b010) begin
                if (pos == 0) begin
                    if (opcode [0] == 1) begin //look at lsb of opcode
                      next_state = S04;
                   end
               else
                    next_state = S07;
            end
            else
                    next_state = S00;
           end

           else
            next_state = S08;
           end

        S02: begin
```

```
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 0;
    PC_enable = 0;
    AC_enable = 1;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 0;
    RWn=1;
    next_state = S00;
    ALU_control[2:0]= 0;
end

S03: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 0;
    PC_enable = 0;
    AC_enable = 1;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 0;
    RWn=1;
    next_state = S00;
    ALU_control[2:0]= 2;

end

S04: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 1;
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 0;
    RWn=1;
    next_state = S05;
end

S05: begin
    IR_enable = 0;
    MD_enable = 1;
    MA_enable = 0;
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 1;
    RWn=1;
    next_state = S06;
end
```

```
S06: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 0;
    PC_enable = 1;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 2;
    MUX3 = 0;
    RWn=1;
    next_state = S00;
end

S07: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 0;
    PC_enable = 1;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 1;
    MUX3 = 0;
    RWn=1;
    next_state = S00;
end

S08: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 1;
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 0;
    RWn=1;

   if (opcode[3:1] == 3'b100) begin
       if (opcode[0] ==1) begin
            next_state = S09;
       end
       else if (opcode[0] ==0) begin
            next_state = S11;
       end
   end
   else
        next_state = S12;
   end

S09: begin
    IR_enable = 0;
    MD_enable = 1;
    MA_enable = 0;
```

```
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 1;
    RWn=1;
    next_state = S10;
end

S10: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 1;
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 1;
    MUX2 = 0;
    MUX3 = 0;
    RWn=1;
    next_state = S11;
end

S11: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 0;
    PC_enable = 0;
    AC_enable = 1;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 1;
     ALU_control[2:0]=3;
    #3 RWn=0;
    next_state = S00;
end

S12: begin
    IR_enable = 0;
    MD_enable = 1;
    MA_enable = 0;
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 1;
    RWn=1;

   if (opcode[0] == 1) begin
       next_state = S13;
   end
   else begin
       if (opcode[3:1] ==3'b001)
            next_state = S15;
```

```
       else
        next_state = S16;
   end
   end


S13: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 1;
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 1;
    MUX2 = 0;
    MUX3 = 0;
    RWn=1;
    next_state = S14;
end

S14: begin
    IR_enable = 0;
    MD_enable = 1;
    MA_enable = 0;
    PC_enable = 0;
    AC_enable = 0;
    C_enable= 0;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 1;
    RWn=1;

   if (opcode[3:1] ==3'b001) begin
       next_state = S15;
   end
   else
       next_state = S16;
   end

S15: begin
    IR_enable = 0;
    MD_enable = 0;
    MA_enable = 0;
    PC_enable = 0;
    AC_enable = 1;
    C_enable= 1;
    MUX1 = 0;
    MUX2 = 0;
    MUX3 = 0;
    RWn=1;
    ALU_control[2:0]=1;
    next_state = S00;
end

S16: begin
    IR_enable = 0;
```

```
                MD_enable = 0;
                MA_enable = 0;
                PC_enable = 0;
                AC_enable = 1;
                C_enable= 0;
                MUX1 = 0;
                MUX2 = 0;
                MUX3 = 0;
                RWn=1;
                ALU_control[2:0]=4;
                next_state = S00;
            end

    endcase
    end

endmodule


/**************************************************************************************************
*/
module data_path(
    input CLK,
    input reset,

    input MD_enable,
    input IR_enable,
    input AC_enable,
    input PC_enable,
    input MA_enable,
    input C_enable,

    input [2:0]alu_control,
    input mux1,
    input mux3,
    input [1:0]mux2,

    input [15:0] RD,

    output reg [15:0] WD,
    output reg [11:0] A,
    output reg [15:0] IR

);

     reg [15:0] AC;
     reg [15:0] MD;
     reg [11:0] PC;
     reg [11:0] MA;
     reg C;

     wire [15:0] y;
     wire co;

   initial begin
        if( reset == 1'b1) begin
            PC[11:0] = 12'h000;
```

```
        MA[11:0] = 12'h000;
        MD[15:0] = 16'h0000;
        AC[15:0] = 16'h0000;
        IR[15:0] = 16'h0000;
        C = 1'b0;
        A[11:0] = 12'h000;
     end
end


always @ (posedge CLK ) begin
    if ( MD_enable == 1'b1 ) begin
        #1 MD = RD;
    end
    else if ( IR_enable == 1'b1 ) begin
        #1 IR = RD;
    end
end

//Call ALU... And reassigning AC when needed
alu op3 ( AC, MD, alu_control, C, y, co);
always @ (posedge CLK ) begin
    if ( AC_enable == 1'b1) begin
        #1 AC = y;
    end
    if (C_enable == 1'b1 ) begin
         C = co;
    end
end

//Mux 1
always @ (posedge CLK) begin
    if ( mux1 == 1'b0 && MA_enable == 1) begin
        MA = IR[11:0];
    end
    else if ( mux1 == 1'b1 && MA_enable == 1 ) begin
        MA = MD[11:0];
    end
end

//Mux 2
always @ (posedge CLK) begin
    if ( mux2 == 2'b00 && PC_enable == 1) begin
        PC = PC + 1;
    end
    else if ( mux2 == 2'b01 && PC_enable == 1 ) begin
        PC = IR[11:0];
    end
    else if ( mux2 == 2'b10 && PC_enable == 1 ) begin
        PC = MD[11:0];
    end
end

//Mux 3
always @ (posedge CLK) begin
    if ( mux3 == 1'b0 ) begin
        A = PC;
```

```
        end
        else if ( mux3 == 1'b1 ) begin
            A = MA;
        end
    end

     always @ (posedge CLK) begin
       WD = AC;
     end


endmodule

/**********************************************************************************************
**/
module simpleCPU(
    input CLK, reset,
    input [15:0] RD,
    output [15:0] WD,
    output RWn,
    output [11:0] A
);


    wire mux1,mux3,IR_enable,MD_enable,AC_enable,MA_enable,PC_enable, C_enable;
    wire [1:0] mux2;
    wire[2:0]alu_control;
    wire [15:0] IR;



    data_path m2(  CLK, reset, MD_enable, IR_enable, AC_enable, PC_enable, MA_enable,
C_enable,
    alu_control[2:0], mux1, mux3, mux2[1:0], RD[15:0], WD[15:0], A[11:0], IR[15:0]);

    controller m3( IR[15:12], WD[15], CLK, reset, IR_enable, MD_enable, MA_enable, PC_enable,
AC_enable, C_enable,
    mux1, mux3,alu_control[2:0],mux2[1:0], RWn );

endmodule
```

d. Verilog test bench code

```
`timescale 1ns / 1ps
module procssor_tb;
// Inputs
    reg CLK=0;
    reg reset=1;
    reg [15:0] RD;
    wire [11:0]A;
    wire [15:0] WD;
    wire RWn;
    reg [15:0] memory[0:18];

    initial begin
        $readmemh("C:/Users/ngol1.LIONS.006/Desktop/memory.dat", memory);
    end


    // Instantiate the Unit Under Test (UUT)
    simpleCPU uut (
    .CLK(CLK),
    .reset(reset),
    .RD(RD),
    .WD(WD),
    .A(A),
    .RWn(RWn)
    );


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

    always begin
        #2 CLK = ~CLK;
    end

    always @* begin
        if (RWn == 1'b0) begin
            memory[A] = WD;
        end
        else begin
            if (RWn == 1'b1)
                RD = memory[A];
        end
    end

    initial #250 $finish;

endmodule
```

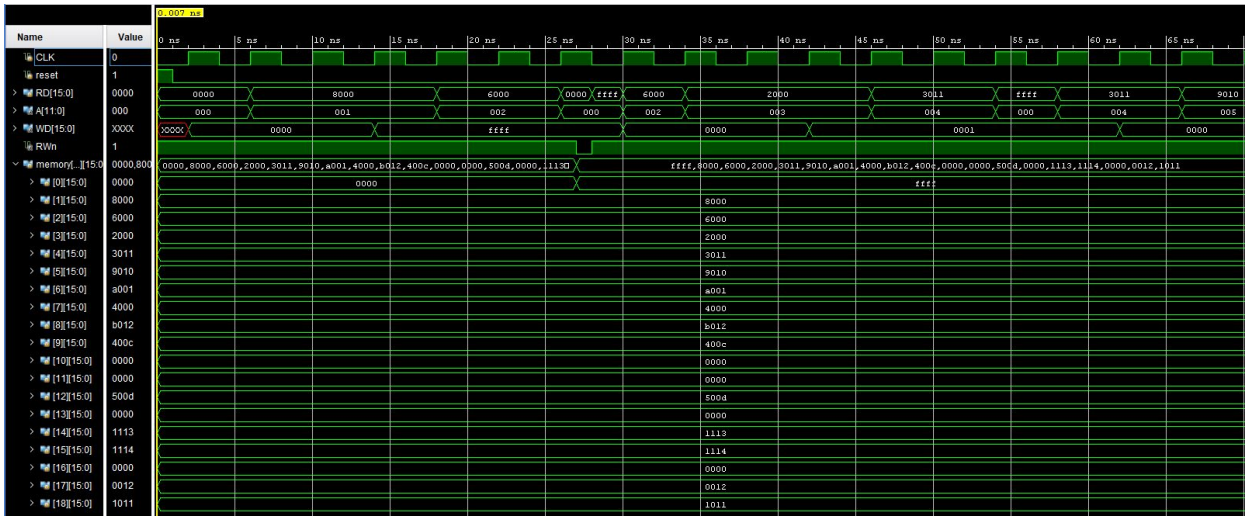
e. Assembly language test program

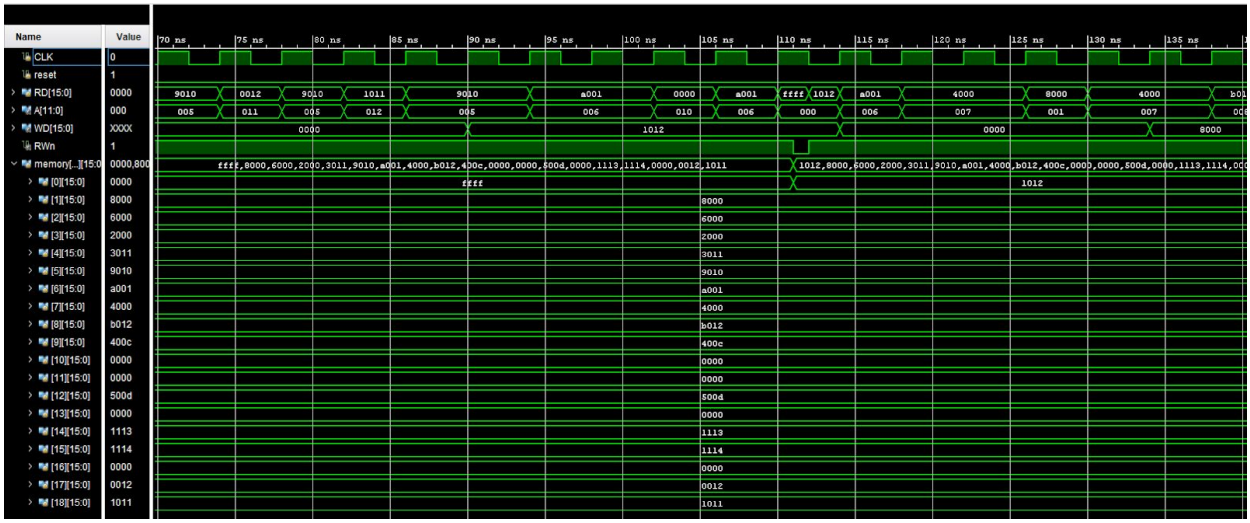
```
not
sta &0
inca
adc &0
adc * &17
sta * &16
lda &1
jpa $0
lda * &18
jpa $13
not
not
jpa * $14
not
(14) 4371
(15) 4372
not
(17) 18
(18) 4113
```

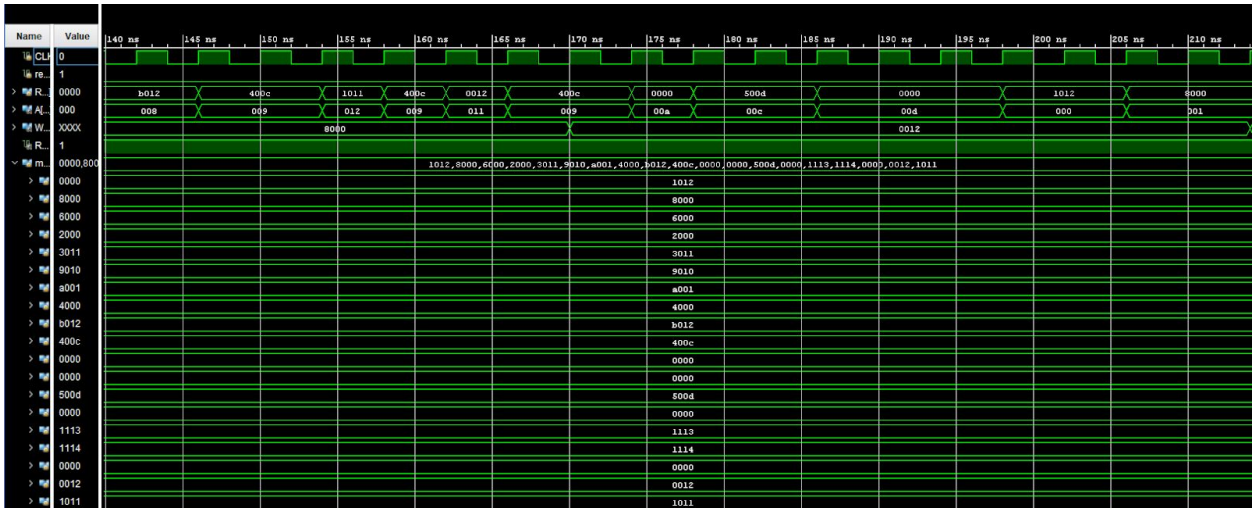
f. Machine language test program

```
@000   0000
@001   8000
@002   6000
@003   2000
@004   3011
@005   9010
@006   A001
@007   4000
@008   B012
@009   400C
@00A   0000
@00B   0000
@00C   500D
@00D   0000
@00E   1113
@00F   1114
@010   0000
@011   0012
@012   1011
```

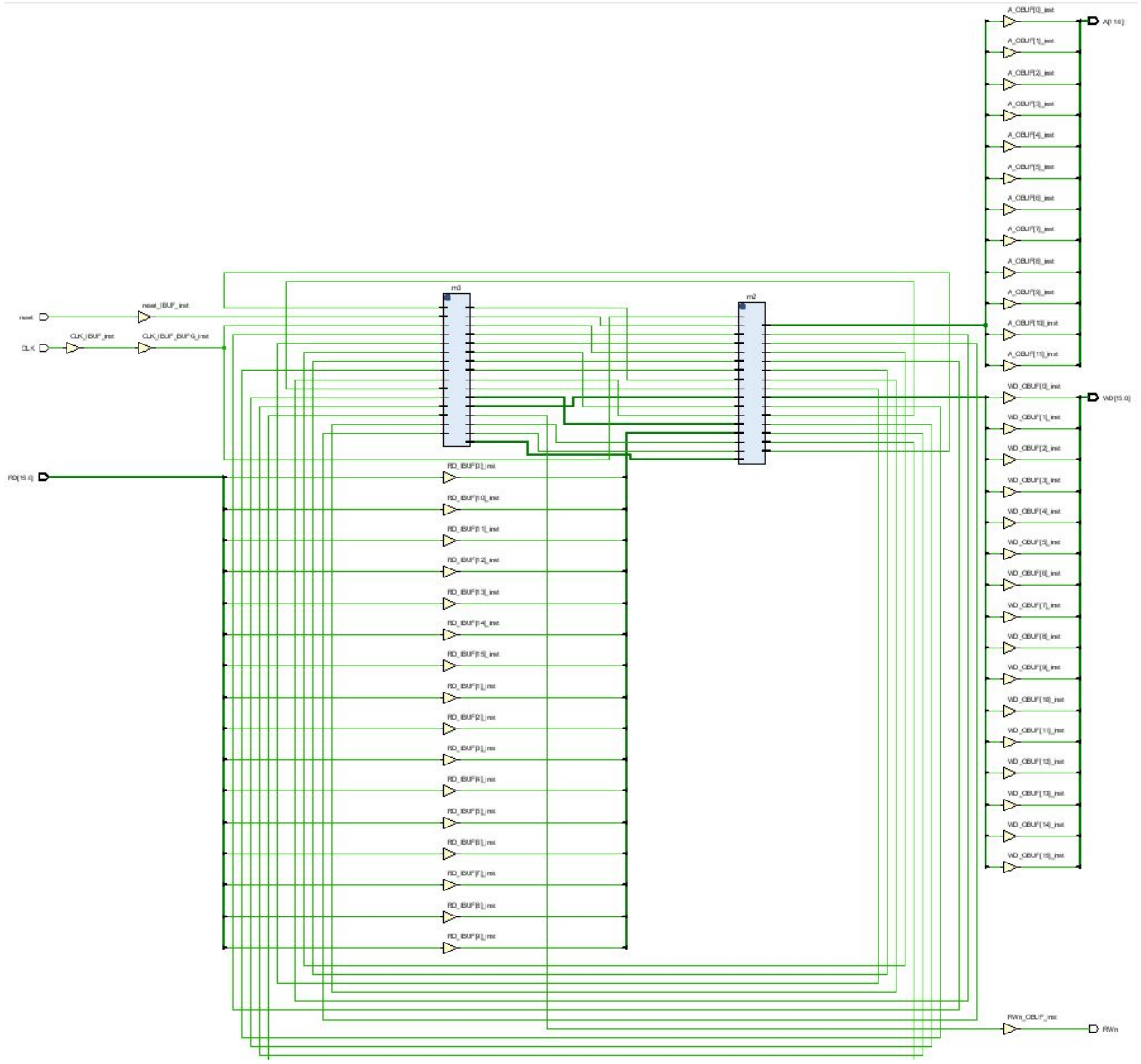
g. Waveforms from verification of design



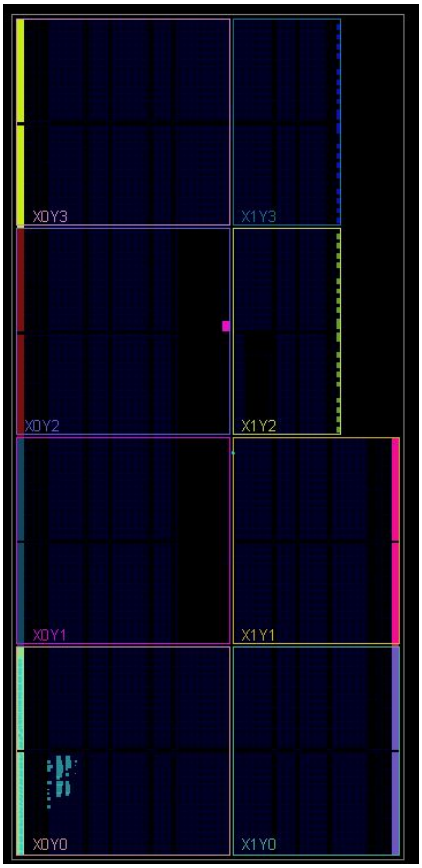


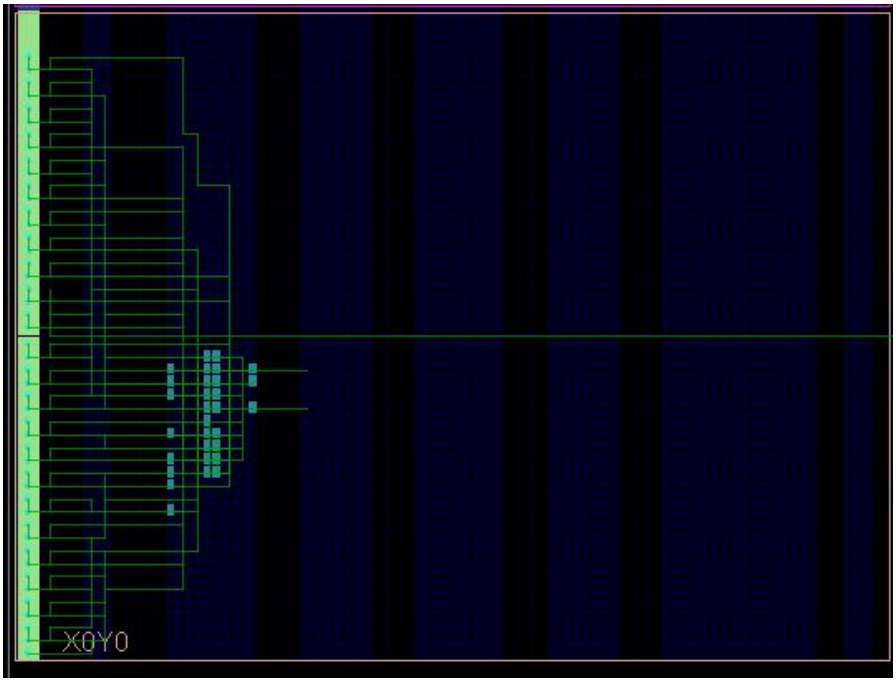


h. Design schematic from Xilinx synthesis of design with no area constraint and clock period of 1 µS as the timing constraint



i. Routed Design





j. Post place and route timing report

```
Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------
------------------------------------------------------------
| Tool Version : Vivado v.2017.4 (win64) Build 2086221 Fri Dec 15 20:55:39 MST 2017
| Date         : Sun Nov  3 13:12:13 2019
| Host         : STEM241-07 running 64-bit major release  (build 9200)
| Command      : report_timing_summary -max_paths 10 -file simpleCPU_timing_summary_routed.rpt
-rpx simpleCPU_timing_summary_routed.rpx -warn_on_violation
| Design       : simpleCPU
| Device       : 7k70t-fbv676
| Speed File   : -1  PRODUCTION 1.12 2017-02-17
------------------------------------------------------------------------------------------------
------------------------------------------------------------

Timing Summary Report

------------------------------------------------------------------------------------------------
--
| Timer Settings
| --------------
------------------------------------------------------------------------------------------------
--

  Enable Multi Corner Analysis               :  Yes
  Enable Pessimism Removal                   :  Yes
  Pessimism Removal Resolution               :  Nearest Common Node
  Enable Input Delay Default Clock           :  No
  Enable Preset / Clear Arcs                 :  No
  Disable Flight Delays                      :  No
  Ignore I/O Paths                           :  No
  Timing Early Launch at Borrowing Latches   :  false

  Corner  Analyze    Analyze
  Name    Max Paths  Min Paths
  ------  ---------  ---------
  Slow    Yes        Yes
  Fast    Yes        Yes



check_timing report

Table of Contents
-----------------
1. checking no_clock
2. checking constant_clock
3. checking pulse_width_clock
4. checking unconstrained_internal_endpoints
5. checking no_input_delay
6. checking no_output_delay
7. checking multiple_clock
8. checking generated_clocks
9. checking loops
10. checking partial_input_delay
11. checking partial_output_delay
12. checking latch_loops

1. checking no_clock
--------------------
 There are 107 register/latch pins with no clock driven by root clock pin: CLK (HIGH)
```

There are 17 register/latch pins with no clock driven by root clock pin: m3/ALU_control_reg[0]/Q (HIGH)

There are 17 register/latch pins with no clock driven by root clock pin: m3/ALU_control_reg[1]/Q (HIGH)

There are 17 register/latch pins with no clock driven by root clock pin: m3/ALU_control_reg[2]/Q (HIGH)

There are 19 register/latch pins with no clock driven by root clock pin: m3/FSM_sequential_present_state_reg[0]/Q (HIGH)

There are 19 register/latch pins with no clock driven by root clock pin: m3/FSM_sequential_present_state_reg[1]/Q (HIGH)

There are 19 register/latch pins with no clock driven by root clock pin: m3/FSM_sequential_present_state_reg[2]/Q (HIGH)

There are 19 register/latch pins with no clock driven by root clock pin: m3/FSM_sequential_present_state_reg[3]/Q (HIGH)

There are 19 register/latch pins with no clock driven by root clock pin: m3/FSM_sequential_present_state_reg[4]/Q (HIGH)

2. checking constant_clock

There are 0 register/latch pins with constant_clock.

3. checking pulse_width_clock

There are 0 register/latch pins which need pulse_width check

4. checking unconstrained_internal_endpoints

There are 220 pins that are not constrained for maximum delay. (HIGH)

There are 0 pins that are not constrained for maximum delay due to constant clock.

5. checking no_input_delay

There are 17 input ports with no input delay specified. (HIGH)

There are 0 input ports with no input delay but user has a false path constraint.

6. checking no_output_delay

There are 29 ports with no output delay specified. (HIGH)

There are 0 ports with no output delay but user has a false path constraint

There are 0 ports with no output delay but with a timing clock defined on it or propagating through it

7. checking multiple_clock

There are 0 register/latch pins with multiple clocks.

8. checking generated_clocks

There are 0 generated clocks that are not connected to a clock source.

9. checking loops

There are 0 combinational loops in the design.

10. checking partial_input_delay

There are 0 input ports with partial input delay specified.

11. checking partial_output_delay

There are 0 ports with partial output delay specified.

12. checking latch_loops

There are 0 combinational latch loops in the design through latch input

---
| Design Timing Summary
| ---

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

All user specified timing constraints are met.

| Clock Summary
| ---

Clock	Waveform(ns)	Period(ns)	Frequency(MHz)
CLK	{0.000 500.000}	1000.000	1.000

| Intra Clock Table
| ---

```
Clock                    WNS(ns)         TNS(ns)   TNS Failing Endpoints   TNS Total Endpoints
WHS(ns)        THS(ns)   THS Failing Endpoints  THS Total Endpoints      WPWS(ns)      TPWS(ns)
TPWS Failing Endpoints  TPWS Total Endpoints
-----                      -------         -------   --------------------   --------------------
-------        -------   --------------------  -------------------      --------      --------
----------------------  --------------------


------------------------------------------------------------------------------------------------
--
| Inter Clock Table
| -----------------
------------------------------------------------------------------------------------------------
--

From Clock     To Clock             WNS(ns)        TNS(ns)   TNS Failing Endpoints  TNS Total
Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
----------        --------                  -------           -------    --------------------
-------------------      -------      -------  --------------------  -------------------


------------------------------------------------------------------------------------------------
--
| Other Path Groups Table
| -----------------------
------------------------------------------------------------------------------------------------
--

Path Group     From Clock    To Clock          WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS
Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
----------     ----------     --------           -------       -------  --------------------
-------------------      -------      -------  --------------------  -------------------


------------------------------------------------------------------------------------------------
--
| Timing Details
| --------------
------------------------------------------------------------------------------------------------
--
```

Discussion:

In this lab we were able to correctly implement all of the components of the lab. The various components of the design code, the ALU, controller, and datapath all worked correctly. They produced the correct waveforms. The testbench also was correct. The fully implemented processor's waveforms were correct. The RTL's were correct as well as the number of clock cycles for each instruction and addressing type. The assembly language program created the

correct machine language program. The design schematic from the synthesis with no area constraint was right as well as the routed design and timing report.

Conclusions:

All of the components of the lab worked as expected. The ALU, control, and datapath all worked correctly individually. They produced the correct outputs and waveforms. They also then worked together once paired with the testbench. This shows that the processor works as intended. All of the diagrams were correct and showed the right information.