Fall 2019

California State University, Northridge

Department of Electrical and Computer Engineering

ECE 526L

Experiment # 9
Modeling a Sequence Controller

December 10, 2019

Authors: Ridge Tejuco
Professor: Ronald Mehler Ph.D

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, nor have I allowed or will I allow anyone to copy my work.

Name (printed) _____

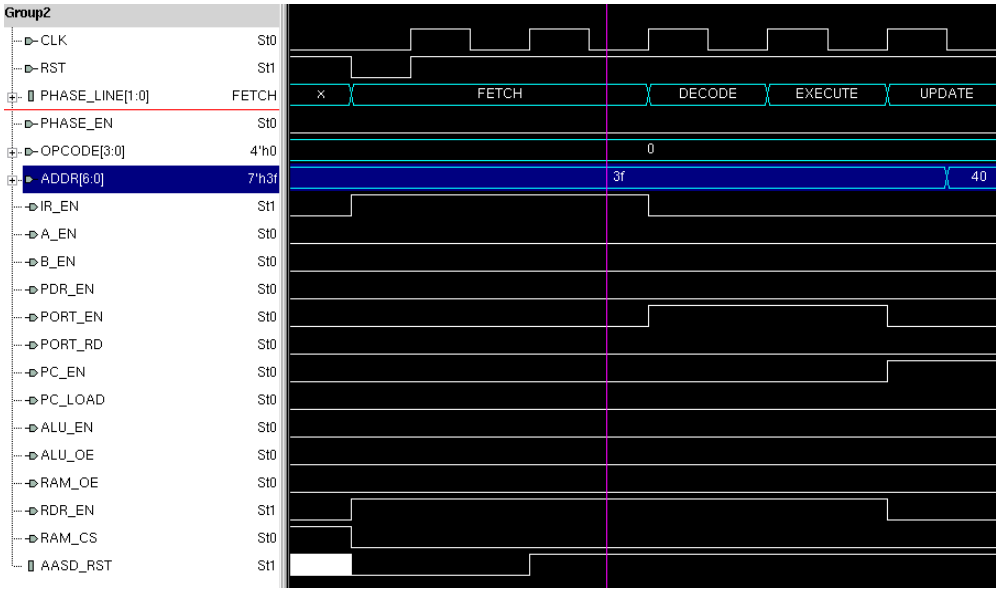
Name (signed) _____

Date _____

Analysis of Sequence Controller

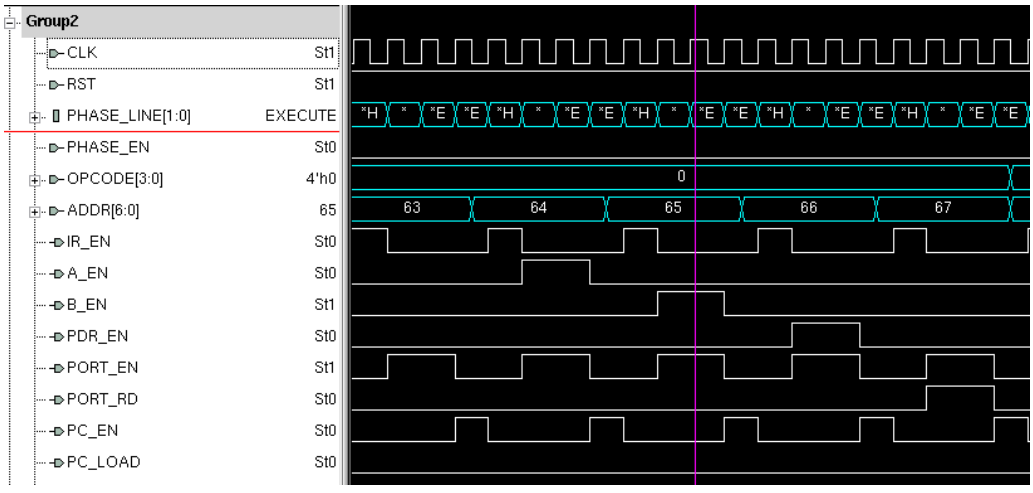
To test the sequence controller, the opcodes were cycled through to test each operation. For opcode 4'b0000 and 4'b0001 the address line was cycled through to test the output for the memory mapping. Figure 1 shows the result for OP 4'b0000 and ADDR $(63)_{decimal}$.

Figure 1. Output of OPCODE = 4b'0000 at address 63.



Looking closer the flags are changed at FETCH and DECODE. The flags stay the same at EXECUTE and change again at update. In the Fetch cycle the IR_EN, the RAM_CS, and the RDR_EN are enabled.

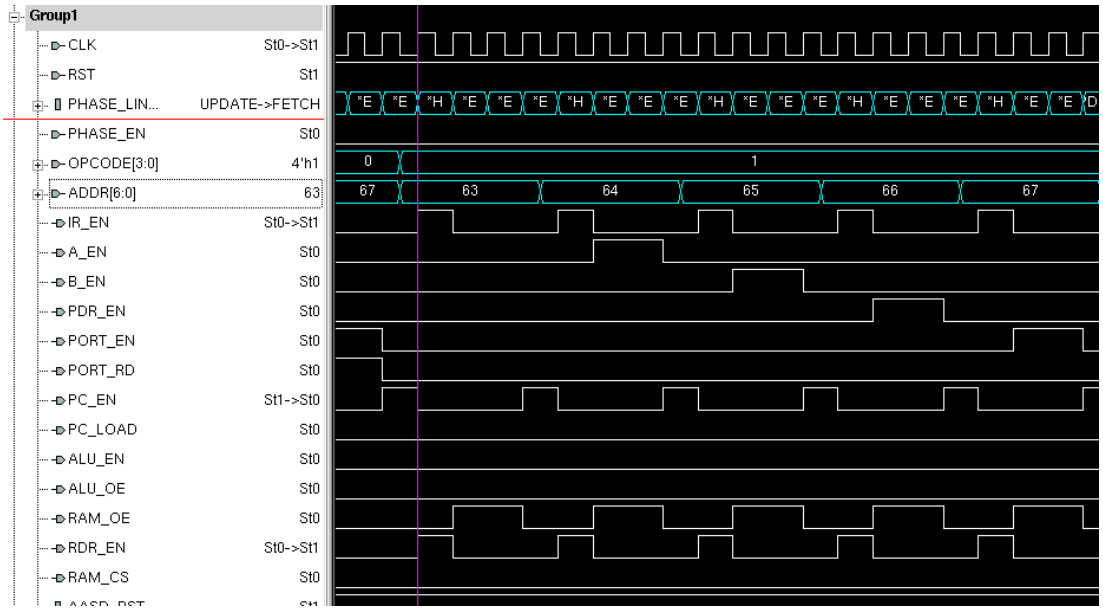
Figure 2. Cycling through ADDR



In figure 2, ADDR line is cycled through all the way to 67. In the DECODE cycle the the flags cycle through A_EN, B_EN, PDR_EN and PORT_RD, which is expected.

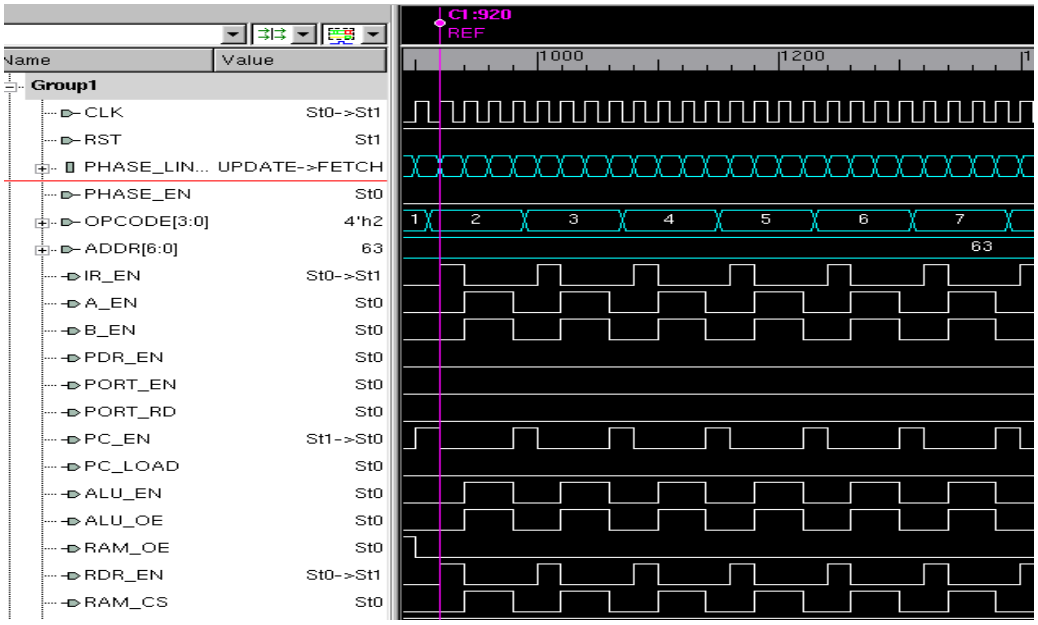
In Figure 3, at 440 ps, the opcode 4’b0001 is tested. Similar to 4’b0000, the ADDR line is cycled through as the memory mapping produces different output flags. It again cycles through flags A_EN, B_EN, PDR_EN and PORT_RD but instead of enabling PORT_EN it enables RDR_EN to write to RAM.

Figure 3. OPCODE = 4’b0001



In figure 4, at 920 ps, the opcodes 4’b0010 to 4’b0111 were tested. These are the ALU operations.

Figure 4. ALU opcodes



In figure 5, at 1400 ps, the Branch operations were tested. During these operations, the PC_EN and PC_LOAD are enabled.

Also in figure 5, the PHASE_EN is disabled by setting it high. The Phase is prevented from moving to the next cycle.

Figure 5. Branch opcode and PHASE_EN = 1'b0

```
//TB_CONTROLLER.sv
//Ridge Tejuco
//DEC 10,2019
`timescale 1 ns / 100 ps
module TB_CONTROLLER();
  parameter DELAY = 8;
  reg CLK,RST,IF,PHASE_EN;
  reg [3:0] FLAGS,OPCODE;
  reg [6:0] ADDR;
  wire IR_EN,A_EN;
  wire B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,PC_LOAD;
  wire ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS;
TOP_CONTROLLER DUT(CLK,RST,PHASE_EN,FLAGS,OPCODE,ADDR,FLAGS,IF,IR_EN,A_EN,
     B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,PC_LOAD,
     ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS);
  initial begin
     #1 CLK = 1'b1;
     #1 forever #(DELAY/8) CLK = ~CLK;
  end
  initial begin
     $vcdpluson;
     RST = 1'b1;
     PHASE_EN = 1'b0;
     FLAGS = 4'b0; IF = 1'b0;
     #1 RST = 1'b0;
     #1 RST = 1'b1;
     OPCODE = 4'b0000; //LOAD
     ADDR = 7'b0111111;
     #1 repeat (4) #DELAY ADDR = ADDR + 1;
     #DELAY
     OPCODE = 4'b0001; //STORE
     ADDR = 7'b0111111;
     repeat (4) #DELAY ADDR = ADDR + 1;

     #DELAY ADDR = 7'b0111111;
     repeat (11) #DELAY OPCODE = OPCODE + 1;

     #DELAY $finish;
  end
Endmodule
```

```
/*=============================================
CSUN - 526 - Lab 9 and 10
Ridge Tejuco
November 7, 2019
=============================================
Sequence Controller
=============================================*/
`timescale 1ns / 100ps
//typedef enum reg [1:0] {FETCH,DECODE,EXECUTE,UPDATE} PHASE;
module SEQUENCE_CONTROLLER(ADDR,OPCODE,PHASE_IN,FLAGS,IF,
     IR_EN,A_EN,B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,
     PC_LOAD,ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS);
    input [6:0] ADDR;
    input [3:0] OPCODE;
    input PHASE PHASE_IN;
    input [3:0] FLAGS;
    input IF;
  reg [12:0] OUT;
    output reg IR_EN,A_EN,B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN;
  output reg PC_LOAD,ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS;

  always@(*) begin

{IR_EN,A_EN,B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,PC_LOAD,ALU_EN,ALU_OE,RAM_OE
,RDR_EN,RAM_CS} <= OUT;
  end

    always_ff@(PHASE_IN,OPCODE,ADDR) begin
      case (PHASE_IN) inside
     FETCH:
     OUT <= 13'b1000000000010;
     DECODE:
     case (OPCODE) inside
          4'b0000:// LOAD
          case (ADDR) inside
               7'b01?????: OUT <= 13'b0000100000010; //ADDR = 32 - 63
               7'b1000000: OUT <= 13'b0100100000000; //ADDR = 64
               7'b1000001: OUT <= 13'b0010100000000; //ADDR = 65
               7'b1000010: OUT <= 13'b0001100000000; //ADDR = 66
               7'b1000011: OUT <= 13'b0000110000000; //ADDR = 67
```

```
          endcase
          4'b0001:  //STORE
          case (ADDR) inside
               7'b01?????: OUT <= 13'b0000000000100; //ADDR = 32 - 63
               7'b1000000: OUT <= 13'b0100000000100; //ADDR = 64
               7'b1000011: OUT <= 13'b0010000000100; //ADDR = 65
               7'b1000010: OUT <= 13'b0001000000100; //ADDR = 66
               7'b1000011: OUT <= 13'b0000100000100; //ADDR = 67
               default:  OUT <= 13'b0000000000001;
          endcase
          4'b001?,4'b01??:
          OUT <= 13'b0110000011001; //ALU Operations
          4'b10??, 4'b1100:
          OUT <= 13'b0000001100001;  //Branch operations
          default:
          OUT <= 13'b0000000000001; //illegal
      endcase
      EXECUTE: OUT <= OUT;
      UPDATE:
      OUT <= (OUT & 13'b0000001100000) | 13'b0000001000000;
      default: OUT <= 13'b0000000000001;
      endcase
    end
endmodule
```

```
//TOP_CONTROLLER.sv
typedef enum reg [1:0] {FETCH,DECODE,EXECUTE,UPDATE} PHASE;
`timescale 1 ns / 100 ps
module
TOP_CONTROLLER(CLK,RST,PHASE_EN,FLAGS,OPCODE,ADDR,FLAGS,IF,IR_EN,A_EN,
     B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,PC_LOAD,
     ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS);
  input CLK,RST,IF,PHASE_EN;
  input [3:0] FLAGS,OPCODE;
  input [6:0] ADDR;
  output IR_EN,A_EN;
  output B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,PC_LOAD;
  output ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS;
  PHASE PHASE_LINE;
  reg AASD_RST;

  /*module SEQUENCE_CONTROLLER(ADDR,OPCODE,PHASE_IN,FLAGS,IF,
     IR_EN,A_EN,B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,
     PC_LOAD,ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS);*/
  SEQUENCE_CONTROLLER DUT(ADDR,OPCODE,PHASE_LINE,FLAGS,IF,
     IR_EN,A_EN,B_EN,PDR_EN,PORT_EN,PORT_RD,PC_EN,
     PC_LOAD,ALU_EN,ALU_OE,RAM_OE,RDR_EN,RAM_CS);

  //module AASD (OUT,CLK,RST);
  AASD AASD_MOD(AASD_RST,CLK,RST);

  //module PHASE_GEN(CLK,RST,EN,PHASE);
  PHASE_GEN PHASE_MOD(CLK,AASD_RST,PHASE_EN,PHASE_LINE);

endmodule
```

```
/*==================================
CSUN - ECE 526L
Ridge Tejuco
==================================
AASD.sv
GOALS
DFF1 <= RST,DFF2 <= DFF1 @ posedge clk when RST = 1
DFF 1&2 <= 0 @ negedge RST
==================================*/
`timescale 1 ns / 100 ps
module AASD (OUT,CLK,RST);
    output reg OUT;
    input wire CLK,RST;
    reg DFF0_OUT;
    always @(posedge(CLK), negedge(RST)) begin
      if(RST == 1'b0) begin
           DFF0_OUT <= 1'b0;
           OUT <= 1'b0;
      end else begin
           OUT <= DFF0_OUT;
           DFF0_OUT <= RST;
      end
    end
endmodule
```

```
/*=============================================
CSUN - ECE526 - LAB9/10
Ridge Tejuco
November 7, 2019
=============================================
PHASE_GEN.sv
-output Phase 1 and 2 with reset and clock inputs
-use AASD reset
=============================================*/
//typedef enum reg [1:0] {FETCH,DECODE,EXECUTE,UPDATE} PHASE;
`timescale 1 ns / 100 ps
module PHASE_GEN(CLK,RST,EN,PHASE_OUT);
    input CLK, RST,EN;
    output PHASE PHASE_OUT;
    always_ff@(posedge CLK, negedge RST) begin
      if(EN == 1'b0) begin
     if(RST == 1'b0)
                PHASE_OUT <= PHASE_OUT.first();
           else
                PHASE_OUT <= PHASE_OUT.next();
     end else
     PHASE_OUT <= PHASE_OUT;
    end
endmodule
```