



CAL POLY

CPE 133 Final Lab

***Robotic Arm
with IR input and Servo Output***

Designed by:

Ethan Vosburg (evosburg@calpoly.edu)

Wyatt Tack (wtack@calpoly.edu)

December 8, 2023

Table of Contents

1 Project Description	3
2 Design.....	3
2.1 High Level Box Diagram.....	3
3 Simulation and Debugging	7
3.1 Simulation Code	8
4 Implemented Code	11
4.1 Slow Clock Module 1	11
4.2 Slow Clock Module 2	11
4.3 Modified Accumulator	11
4.4 Top Module	12
4.5 FSM Module 1	15
4.6 FSM Module 2	17
4.7 Decoder Module	23
4.8 PWM Module	25

1 Project Description

Our design focused on using a Field Programmable Gate Array to construct an Infrared Remote-Controlled Motorized Arm. Our arm would be controlled by 3 servo motors, allowing for 3 degrees of freedom, which we chose to simulate the Open/-Close of the hand, the up and down swivel of the elbow, and the up and down motion of the shoulder/base of the arm. The servo motors would need to be controlled by a module that was capable of taking in the total count of an accumulator and using PWM¹ to control the angle that the accumulator total corresponded to. The accumulator would be connected to a series of FSMs², that would take the single changing bit from the Infrared sensor and convert it into logic, which would be deciphered by another FSM to determine which button on the remote was pressed and provide either a +1 or a -1³ to the accumulator storing the total value, corresponding to the angle the specific joint was at.

¹Pulse Width Modulation

²Finite State Machines

³through an 8-bit sequence of 2's complement

2 Design

2.1 High Level Box Diagram

Our design revolved around one input and three identical outputs. Our main input was the IR Receiver module, which would take data from our remote, running at NEC Protocol. Our three outputs were the PWM lines connected to our 3 servo motors, representing the individual joints on our arm.⁴

⁴1 time-dependent input and 3 time-dependent outputs

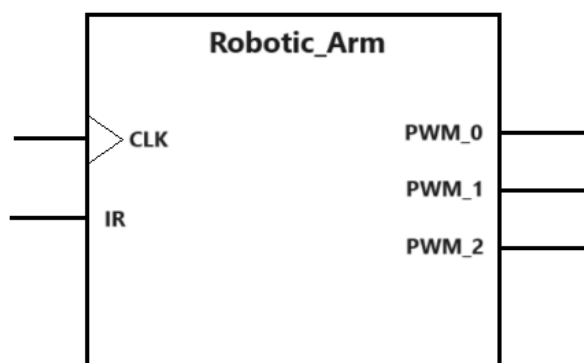


Figure 1: High-Level Black Box Diagram

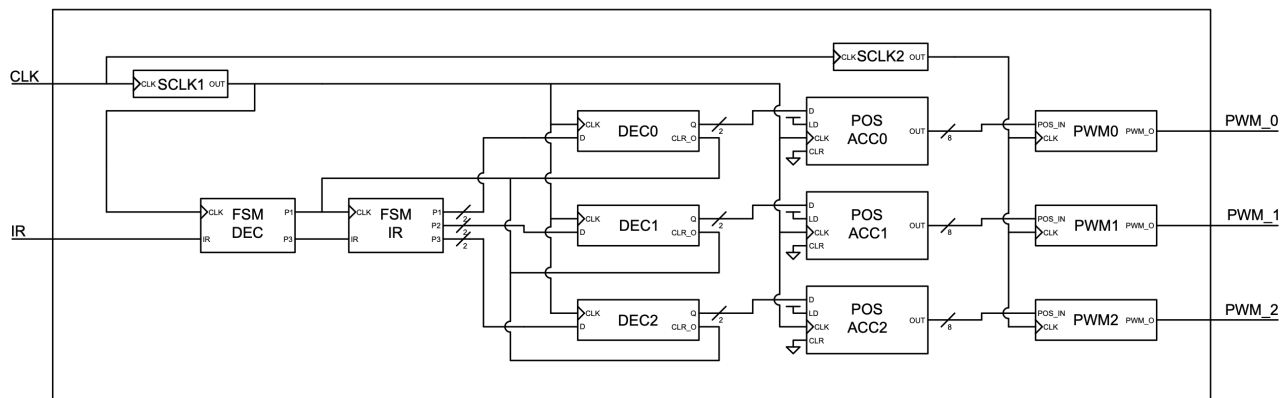


Figure 2: Low Level Structural Diagram

Our lower-level design consisted of 13 individual modules. Our design comprised of a linear flow of data, with the IR decoding section being made of 2 FSMs and a slow clock, the angle storage line, being comprised of 3 sets of a modified accumulator and a decoder module, and the PWM line comprised of another slow clock and 3 sets of PWM outputting modules for the servo motors.

NEC Protocol 0 VS 1

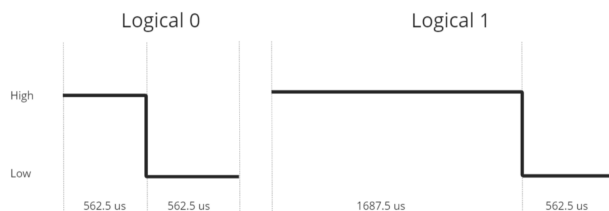


Figure 3: Low Level Structural Diagram

Along with each bitstream being a long low pulse and a shorter high pulse before 8 bits of 0 then 8 bits of 1 as a header, then the actual stream of bits is received. For instance, below is shown the oscilloscope output of the IR receiver module when the VOL+ button is pressed on the remote, which would output a long header of 8 logical 0s, 8 logical 1s, and then the hexadecimal code of 0x629D would be Displayed.

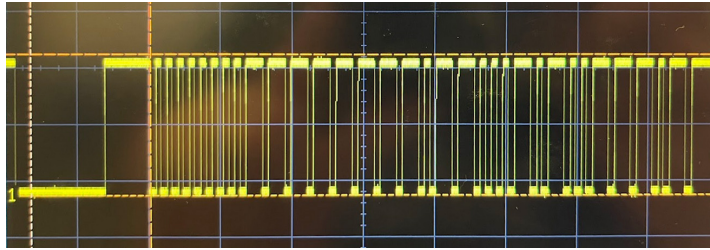


Figure 4: Oscilloscope Output of VOL+ button on IR Receiver Module

Our first FSM would take in the stream of bits, and output a 0 and a clock pulse when the stream for 0 was detected, or a 1 and a clock pulse when the stream for 1 is detected. The second FSM ran off the clock and output bit from the first FSM, and would take in each bit to determine which 16 bit long code the stream output to with each button press, and if the stream aligned with one of the buttons we chose, the state machine would output either a logical +1 or -1 (in 2's complement for 2 bits) at one of the 3 ports, corresponding to which servo the button pressed was meant to control. Both FSM state diagrams are shown below.

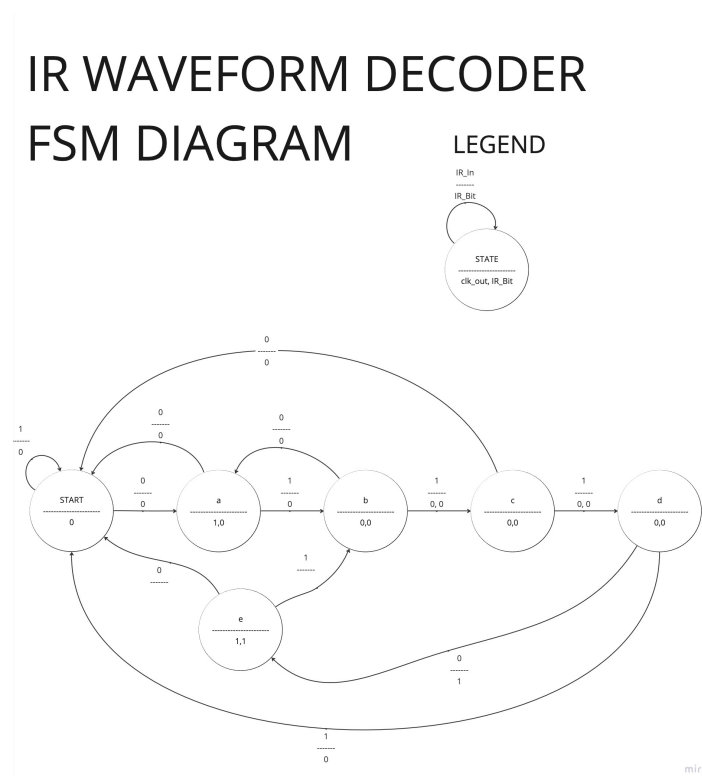


Figure 5: State Diagram of first “waveform decoder” FSM



Figure 6: State Diagram of second “bitstream decoder” FSM

To time the bit processing of the IR decoding portion (The 2 FSMs), a modified version of Dr. Mealy's "Slow Clock" `clock_div2.vhd` module was included, changing the clock to output 1,778 Hz frequency.

The next portion was the data storage portion of the module, which would use an accumulator to keep a running total of each value to keep the angle of the servo motor at. A modified 8-bit accumulator was used, that added an upper and lower cap to its maximum and minimum values that could be stored, so that data overflow wouldn't happen (for instance if a -1 was entered but the value stored was 0). A decoder was added in line with the accumulator to clock pulse the second FSM to set it's output back to 0 (since the FSM's clock depends on the first FSM's output, when the receiver isn't getting data the value of the first FSM output stays constant until its clock is pulsed). The FSM after detecting the right code would output its +/- 1 into the decoder, which would be in sync with the timing of the accumulator, and would output it only once on one clock pulse to not add additional unintended data, and would also pulse the clock of the second FSM, to set it's outputs back to low.

The last portion was the PWM output section, which consisted of a second slow clock (The modified Dr. Mealy's `clock_div2.vhd`) running at a flat 1000 Hz frequency, along with a new PWM module, which would take in the stored total (representing the angle) from the accumulator, and output a PWM signal to the servo motor, corresponding to the angle the servo should be

set to. The data storage and PWM sections were duplicated to a total of 3 of each line, each connecting to an output from the second FSM.

3 Simulation and Debugging

Our main simulation source was of the top module, which in the Vivado simulation we displayed the inner connections, showing. The 5 main linear modules worked, being the first “waveform decoder” FSM, the second “IR bitstream” FSM, and one line of the decoder, the modified accumulator, and the PWM output.

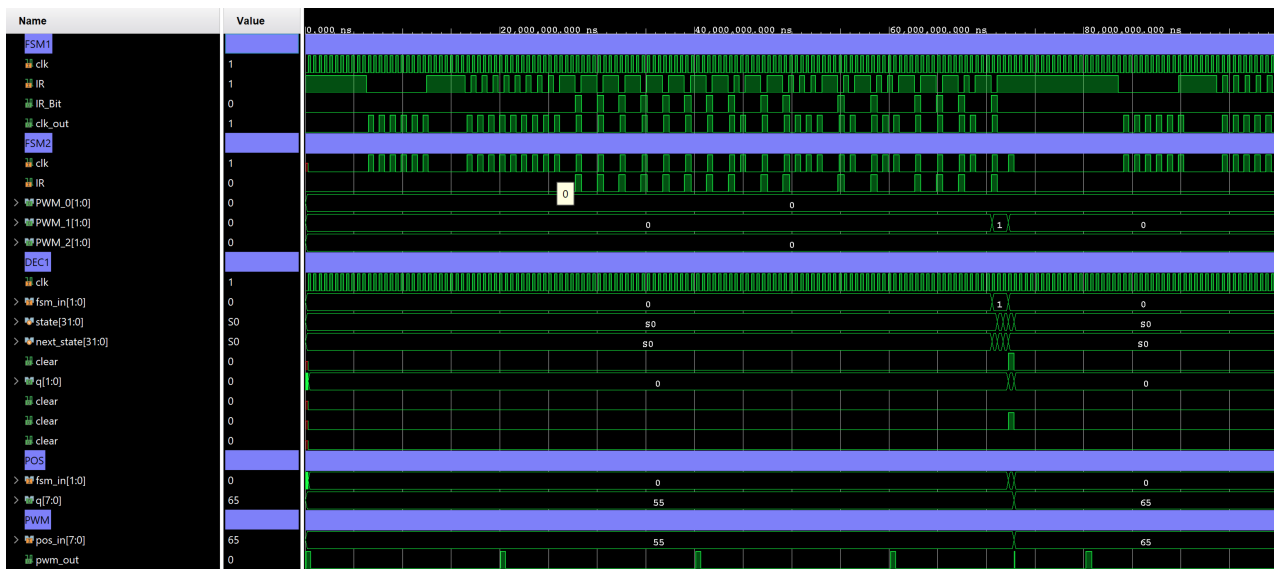


Figure 7: Simulation of Robotic Arm Top Module

Our code for simulation was made by re-creating the oscilloscope output of the “VOL+” Button, making it determine the timing diagram for our only input, the IR input. We then simulated the output of the corresponding PWM_1 line, showing that its Pulse Width duty cycle changed with each button press, and therefore each correct bitstream of data. Our simulation output is shown above, as well as our simulation code is shown below.

3.1 Simulation Code

```
1      'timescale 1ns / 1ps
2      //////////////////////////////////////
3      // Company: Cal Poly SLO
4      // Engineer: Wyatt Tack
5      //
6      // Create Date: 11/29/2023 01:59:28 PM
7      // Design Name: Robot Arm Test Bench
8      // Module Name: FSM_TB
9      // Project Name: Robotic Arm
10     // Target Devices: Basys 3 Development Board
11     // Tool Versions:
12     // Description:
13     //
14     // Dependencies:
15     //
16     // Revision:
17     // Revision 0.01 - File Created
18     // Additional Comments:
19     //
20     //////////////////////////////////////
21
22
23     module FSM_TB();
24     logic clk;
25     logic IR;
26     logic PWM_0, PWM_1, PWM_2;
27
28     Robot_Arm UUT (.clk(clk), .IR(IR), .PWM_0(PWM_0), .PWM_1(PWM_1), .
        PWM_2(PWM_2));
29
30     always begin//100 MHz, 10ns Period
31     clk = 1;
32     #5;
33     clk = 0;
34     #5;
35     end
36
37     always begin
38     IR = 1;
39     #6287500;
40
41     IR = 0;
42     #6187500;
43
44     IR = 1;
45     #3937500;
46
47     IR = 0;
48     #6000000;
49
50     IR = 1; //0
51     #5360000;
52     IR = 0;
53     #6000000;
54
55     IR = 1; //0
56     #5360000;
57     IR = 0;
58     #6000000;
59
60     IR = 1; //0
61     #5360000;
62     IR = 0;
63     #6000000;
64
65     IR = 1; //0
```



```
66 | #536000;
67 | IR = 0;
68 | #600000;
69 |
70 | IR = 1; //0
71 | #536000;
72 | IR = 0;
73 | #600000;
74 |
75 | IR = 1; //0
76 | #536000;
77 | IR = 0;
78 | #600000;
79 |
80 | IR = 1; //0
81 | #536000;
82 | IR = 0;
83 | #600000;
84 |
85 | IR = 1; //0
86 | #536000;
87 | IR = 0;
88 | #600000;
89 |
90 | IR = 1; //1
91 | #1644000;
92 | IR = 0;
93 | #600000;
94 |
95 | IR = 1; //1
96 | #1644000;
97 | IR = 0;
98 | #600000;
99 |
100 | IR = 1; //1
101 | #1644000;
102 | IR = 0;
103 | #600000;
104 |
105 | IR = 1; //1
106 | #1644000;
107 | IR = 0;
108 | #600000;
109 |
110 | IR = 1; //1
111 | #1644000;
112 | IR = 0;
113 | #600000;
114 |
115 | IR = 1; //1
116 | #1644000;
117 | IR = 0;
118 | #600000;
119 |
120 | IR = 1; //1
121 | #1644000;
122 | IR = 0;
123 | #600000;
124 |
125 | IR = 1; //1
126 | #1644000;
127 | IR = 0;
128 | #600000;
129 |
130 | //initializer ^^
131 | IR = 1; //0
132 | #536000;
133 | IR = 0;
134 | #600000;
135 |
```

```
136 IR = 1; //1
137 #1644000;
138 IR = 0;
139 #600000;
140
141 IR = 1; //1
142 #1644000;
143 IR = 0;
144 #600000;
145
146 IR = 1; //0
147 #536000;
148 IR = 0;
149 #600000;
150
151 IR = 1; //0
152 #536000;
153 IR = 0;
154 #600000;
155
156 IR = 1; //0
157 #536000;
158 IR = 0;
159 #600000;
160
161 IR = 1; //1
162 #1644000;
163 IR = 0;
164 #600000;
165
166 IR = 1; //0
167 #536000;
168 IR = 0;
169 #600000;
170
171 IR = 1; //1
172 #1644000;
173 IR = 0;
174 #600000;
175
176 IR = 1; //0
177 #536000;
178 IR = 0;
179 #600000;
180
181 IR = 1; //0
182 #536000;
183 IR = 0;
184 #600000;
185
186 IR = 1; //1
187 #1644000;
188 IR = 0;
189 #600000;
190
191 IR = 1; //1
192 #1644000;
193 IR = 0;
194 #600000;
195
196 IR = 1; //1
197 #1644000;
198 IR = 0;
199 #600000;
200
201 IR = 1; //0
202 #536000;
203 IR = 0;
204 #600000;
205
```

```
206 IR = 1; //1
207 #1644000;
208 IR = 0;
209 #600000;
210
211 //end code^^^
212 IR=1;
213 #6187500;
214 end
215 endmodule
```

4 Implemented Code

Our final code consisted of one top module, 2 FSMs, 2 Modified Dr. Mealy modules, A Modified Accumulator, a new Decoder, and a new PWM Module, resulting in 3 modified Modules and 5 brand new Low Level Modules. The code is shown below, modified modules first, then everything else in order of hierarchy and flow of data. All code can be additionally found at the following Github link:
<https://github.com/EthanV1920/CPE-133-Final-Lab>

4.1 Slow Clock Module 1

Only changed line 38 to change the frequency of the clock.

```
1 -- Only line(38) changed is the MAX_COUNT constant
2 constant max_count : integer := (28124);
```

4.2 Slow Clock Module 2

Only changed line 38 to change the frequency of the clock.

```
1 -- Only line(38) changed is the MAX_COUNT constant
2 constant max_count : integer := (500);
```

4.3 Modified Accumulator

Ethan Vosburg modified the accumulator to add an upper and lower cap to its maximum and minimum values that could be stored, so that data overflow wouldn't happen (for instance if a -1 was entered but the value stored was 0). The code is shown below.

```

1 module POS_ACC(
2     // Inputs
3     input [1:0] fsm_in, // Signed 2bit input
4     input clk, ld, clear, // Clock, load, and clear inputs
5
6     // Outputs
7     output logic [7:0] q = 8'd55 // Accumulator output
8 );
9
10 // Accumulator Logic
11 always_ff @ (posedge clk)
12 begin
13     if (clear)
14         begin
15             // Reset the accumulator
16             q <= 8'd55;
17         end
18     else if (ld)
19         // Increment the accumulator when load is +1
20         // $display("q = %d", q); // Debug code
21         if ((q >= 8'd55) && (q <= 8'd245))
22             begin
23                 if (fsm_in == 2'b01)
24                     begin
25                         q <= q + 8'd10;
26                     end
27                 end
28             // Decerement the accumulator when load is -1
29             if ((q >= 8'd65) && (q <= 8'd255))
30                 if (fsm_in == 2'b11)
31                     q <= q - 8'd10;
32             end
33     end
34 endmodule
35

```

4.4 Top Module

```

1 'timescale 1ns / 1ps
2 ///////////////////////////////////////////////////////////////////
3 // Company: Cal Poly SLO
4 // Engineer: Ethan Vosburg and Wyatt Tack
5 //
6 // Create Date: 11/29/2023 02:09:10 PM
7 // Design Name: Robot Arm Top Module
8 // Module Name: Robot_Arm
9 // Project Name: Robotic Arm
10 // Target Devices: Baysys 3 Development Board
11 // Tool Versions:
12 // Description: Top module for the robotic arm linking all modules
13 // together
14 // Dependencies:
15 //
16 // Revision: 1.0
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22 module Robot_Arm(
23     // Input Ports
24     input clk,
25     input IR,
26

```

```
27
28 // Output Ports
29 output PWM_0, PWM_1, PWM_2,
30 output PWR, GND,
31
32 // Debug Ports
33 output debug_0, debug_1, debug_2, debug_3, debug_4, debug_5
34 );
35 logic SCLK_0, IR_BIT, FSM_CLK_IN, FSM_CLK_OUT;
36 logic DEC_CLR_0, DEC_CLR_1, DEC_CLR_2;
37 logic [1:0] FSM_DEC_0, FSM_DEC_1, FSM_DEC_2;
38 logic [1:0] ACC_0, ACC_1, ACC_2;
39 logic [7:0] ACC_0_PWM_0, ACC_1_PWM_1, ACC_2_PWM_2;
40 logic debug_led_0, debug_led_1, debug_led_2;
41
42
43 assign PWR = 1;
44 assign GND = 0;
45
46 assign FSM_CLK_IN = FSM_CLK_OUT | DEC_CLR_0 | DEC_CLR_1 | DEC_CLR_2;
47
48 clk_div2_0 SCLK_D_0 (
49     .clk(clk),
50     .sclk(SCLK_0)
51 );
52
53 clk_div2_1 SCLK_D_1 (
54     .clk(clk),
55     .sclk(SCLK_1)
56 );
57
58 FSM_Decoder FSM1 (
59     .clk(SCLK_0),
60     .IR(IR),
61     .IR_Bit(IR_BIT),
62     .clk_out(FSM_CLK_OUT)
63 );
64
65
66 FSM_IR FSM2 (
67     .clk(FSM_CLK_IN),
68     .IR(IR_BIT),
69     .PWM_0(FSM_DEC_0),
70     .PWM_1(FSM_DEC_1),
71     .PWM_2(FSM_DEC_2)
72 );
73
74 assign debug_0 = FSM_CLK_OUT;
75 assign debug_1 = FSM_CLK_IN;
76 assign debug_2 = DEC_CLR_2;
77 assign debug_3 = FSM_DEC_2[0];
78 assign debug_4 = DEC_CLR_2;
79 assign debug_5 = FSM_DEC_2[0];
80
81
82
83 DEC DEC_0 (
84     .fsm_in(FSM_DEC_0),
85     .clk(SCLK_0),
86     .q(ACC_0),
87     .clear(DEC_CLR_0)
88 );
89
90 // assign debug_0 = ACC_0[0];
91 // assign debug_1 = ACC_0[1];
92
93 DEC DEC_1 (
94     .fsm_in(FSM_DEC_1),
95     .clk(SCLK_0),
96     .q(ACC_1),
```

```
97     .clear(DEC_CLR_1)
98   );
99
100 // assign debug_2 = ACC_1[0];
101 // assign debug_3 = ACC_1[1];
102
103 DEC DEC_2 (
104   .fsm_in(FSM_DEC_2),
105   .clk(SCLK_0),
106   .q(ACC_2),
107   .clear(DEC_CLR_2)
108 );
109
110 // assign debug_4 = ACC_2[0];
111 // assign debug_5 = ACC_2[1];
112
113 POS_ACC POS_ACC_0 (
114   .fsm_in(ACC_0),
115   .clk(SCLK_0),
116   .clear(1'b0),
117   .ld(1'b1),
118   .q(ACC_0_PWM_0)
119 );
120
121 POS_ACC POS_ACC_1 (
122   .fsm_in(ACC_1),
123   .clk(SCLK_0),
124   .clear(1'b0),
125   .ld(1'b1),
126   .q(ACC_1_PWM_1)
127 );
128
129 POS_ACC POS_ACC_2 (
130   .fsm_in(ACC_2),
131   .clk(SCLK_0),
132   .clear(1'b0),
133   .ld(1'b1),
134   .q(ACC_2_PWM_2)
135 );
136
137 PWM PWM_D_0 (
138   .clk(SCLK_1),
139   .pos_in(ACC_0_PWM_0),
140   .pwm_out(PWM_0)
141 );
142
143 PWM PWM_D_1 (
144   .clk(SCLK_1),
145   .pos_in(ACC_1_PWM_1),
146   .pwm_out(PWM_1)
147 );
148
149 PWM PWM_D_2 (
150   .clk(SCLK_1),
151   .pos_in(ACC_2_PWM_2),
152   .pwm_out(PWM_2)
153 );
154
155 endmodule
```

4.5 FSM Module 1

```
1  'timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Wyatt Tack
5  //
6  // Create Date: 12/01/2023 02:14:47 PM
7  // Design Name: FSM IR NEC decoder
8  // Module Name: FSM_Decoder
9  // Project Name: IR controlled robotic arm
10 // Target Devices: Elegoo IR Remote and reciever module
11 // Tool Versions:
12 // Description: Uses an FSM to decode the output of an NEC protocol
    IR module to
13 //          Determine if the output is a 1 or a 0 bit
14 //
15 // Dependencies: Slow Clock 1 (1778 HZ)
16 //
17 // Revision:
18 // Revision 0.01 - File Created
19 // Additional Comments:
20 //
21 ///////////////////////////////////////////////////////////////////
22
23
24 module FSM_Decoder(
25     // Inputs
26     input clk, // Clock input
27     input IR, // IR input
28
29     // Outputs
30     output logic IR_Bit, // IR output logic
31     output logic clk_out // Clock output logic
32 );
33
34 // Defined states for delay
35 typedef enum {Start, a, b, c, d, e} STATES;
36
37 // Defines the current and next state
38 STATES PS, NS;
39
40 // Clock logic
41 always_ff@(posedge clk)
42 begin
43     PS <= NS;
44 end
45
46 // State logic
47 always_comb
48 begin
49     // Set IR_Bit and clk_out to 0
50     IR_Bit = 0;
51     clk_out = 0;
52 case (PS)
53     Start:
54         begin
55             clk_out = 0;
56             if (IR)
57                 begin
58                     NS = Start;
59                     IR_Bit = 0;
60                 end
61             else
62                 begin
63                     NS = a;
64                     IR_Bit = 0;
65                 end
```

```
66     end
67     a:
68     begin
69     clk_out = 1;
70     if (IR)
71     begin
72     NS = b;
73     IR_Bit = 0;
74     end
75     else
76     begin
77     NS = Start;
78     IR_Bit = 0;
79     end
80     end
81     b:
82     begin
83     clk_out = 0;
84     if (IR)
85     begin
86     NS = c;
87     IR_Bit = 0;
88     end
89     else
90     begin
91     NS = a;
92     IR_Bit = 0;
93     end
94     end
95     c:
96     begin
97     clk_out = 0;
98     if (IR)
99     begin
100     NS = d;
101     IR_Bit = 0;
102     end
103     else
104     begin
105     NS = Start;
106     IR_Bit = 0;
107     end
108     end
109     d:
110     begin
111     clk_out = 0;
112     if (IR)
113     begin
114     NS = Start;
115     IR_Bit = 0;
116     end
117     else
118     begin
119     NS = e;
120     IR_Bit = 1;
121     end
122     end
123     e:
124     begin
125     clk_out = 1;
126     IR_Bit = 1;
127     if (IR)
128     begin
129     NS = b;
130     //IR_Bit = 0;
131     end
132     else
133     begin
134     NS = Start;
135     //IR_Bit = 0;
```



```
136         end
137     end
138
139
140     default:
141         NS = Start;
142     endcase
143     end
144 endmodule
```

4.6 FSM Module 2

```
1
2 'timescale 1ns / 1ps
3 ///////////////////////////////////////////////////////////////////
4 // Company: Cal Poly SLO
5 // Engineer: Wyatt Tack
6 //
7 // Create Date: 11/28/2023 10:26:21 AM
8 // Design Name: IR FSM Decoder
9 // Module Name: FSM_IR
10 // Project Name: Remote Control Servo Arm
11 // Target Devices: Elegoo IR remote and receiver module
12 // Tool Versions:
13 // Description: Uses an FSM to decode a 38kHz IR signal from a remote
14 //
15 // to add +/- 1 to a variable controlling the position
16 // of a servo motor
17 //
18 // Dependencies: FSM IR Bit Decoder
19 //
20 // Revision: 1.0
21 // Revision 0.01 - File Created
22 // Additional Comments:
23 //
24 ///////////////////////////////////////////////////////////////////
25 module FSM_IR(
26     // Inputs
27     input clk, // Clock input
28     input IR, // IR input
29
30     // Outputs
31     output logic [1:0] PWM_0, PWM_1, PWM_2 // PWM outputs
32 );
33
34 // Define states
35 typedef enum {Start, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11,
36             a12, a13, a14, a15, a16,
37             b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13, b14, b15,
38             c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16,
39             d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15,
40             e5, e6, e7, e8, e9, e10, e11, e12, e13, e14,
41             f7, f8, f9, f10, f11, f12, f13} STATES;
42
43 // Define current and next state
44 STATES PS, NS;
45
46 // Clock logic
47 always_ff@(posedge clk)
48 begin
49     PS <= NS;
50 end
51
```

```
52 // FSM logic
53 always_comb
54 begin
55   PWM_0 = 0;
56   PWM_1 = 0;
57   PWM_2 = 0;
58   case (PS)
59     Start://11111...
60     begin
61       PWM_0 = 0;
62       PWM_1 = 0;
63       PWM_2 = 0;
64       if (IR) NS = Start;
65       else NS = a1;
66     end
67     a1://0
68     begin
69       if (IR) NS = a2;
70       else NS = b2;
71     end
72     a2://01
73     begin
74       if (IR) NS = a3;
75       else NS = b3;
76     end
77     a3://011
78     begin
79       if (!IR) NS = a4;
80       else NS = Start;
81     end
82     a4://0110
83     begin
84       if (!IR) NS = a5;
85       else NS = Start;
86     end
87     a5://01100
88     begin
89       if (!IR) NS = a6;
90       else NS = Start;
91     end
92     a6://011000
93     begin
94       if (IR) NS = a7;
95       else NS = Start;
96     end
97     a7://0110001
98     begin
99       if (!IR) NS = a8;
100      else NS = Start;
101    end
102    a8://01100010
103    begin
104      if (IR) NS = a9;
105      else NS = Start;
106    end
107    a9://011000101
108    begin
109      if (!IR) NS = a10;
110      else NS = Start;
111    end
112    a10://0110001010
113    begin
114      if (!IR) NS = a11;
115      else NS = Start;
116    end
117    a11://01100010100
118    begin
119      if (IR) NS = a12;
120      else NS = Start;
121    end
```

```
122     a12://011000101001
123     begin
124     if (IR) NS = a13;
125     else NS = Start;
126     end
127     a13://0110001010011
128     begin
129     if (IR) NS = a14;
130     else NS = Start;
131     end
132     a14://01100010100111
133     begin
134     if (!IR) NS = a15;
135     else NS = Start;
136     end
137     a15://011000101001110
138     begin
139     if (IR) NS = a16;
140     else NS = Start;
141     end
142     a16://0110001010011101
143     begin
144     PWM_1 = 2'b01;
145     NS = Start;
146     end
147     b2://00
148     begin
149     if (IR) NS = c3;
150     else NS = d3;
151     end
152     b3://010
153     begin
154     if (IR) NS = b4;
155     else NS = Start;
156     end
157     b4://0101
158     begin
159     if (!IR) NS = b5;
160     else NS = Start;
161     end
162     b5://01010
163     begin
164     if (!IR) NS = b6;
165     else NS = Start;
166     end
167     b6://010100
168     begin
169     if (!IR) NS = b7;
170     else NS = Start;
171     end
172     b7://0101000
173     begin
174     if (!IR) NS = b8;
175     else NS = Start;
176     end
177     b8://01010000
178     begin
179     if (IR) NS = b9;
180     else NS = Start;
181     end
182     b9://010100001
183     begin
184     if (!IR) NS = b10;
185     else NS = Start;
186     end
187     b10://0101000010
188     begin
189     if (IR) NS = b11;
190     else NS = Start;
191     end
```

```
192     b11://01010000101
193     begin
194     if (!IR) NS = b12;
195     else NS = Start;
196     end
197     b12://010100001010
198     begin
199     if (IR) NS = b13;
200     else NS = Start;
201     end
202     b13://0101000010101
203     begin
204     if (IR) NS = b14;
205     else NS = Start;
206     end
207     b14://01010000101011
208     begin
209     if (IR) NS = b15;
210     else NS = Start;
211     end
212     b15://010100001010111
213     begin
214     PWM_1 = 2'b11;
215     NS = Start;
216     end
217     c3://001
218     begin
219     if (!IR) NS = c4;
220     else NS = Start;
221     end
222     c4://0010
223     begin
224     if (!IR) NS = c5;
225     else NS = Start;
226     end
227     c5://00100
228     begin
229     if (!IR) NS = c6;
230     else NS = Start;
231     end
232     c6://001000
233     begin
234     if (IR) NS = c7;
235     else NS = d7;
236     end
237     c7://0010001
238     begin
239     if (!IR) NS = c8;
240     else NS = Start;
241     end
242     c8://00100010
243     begin
244     if (IR) NS = c9;
245     else NS = Start;
246     end
247     c9://001000101
248     begin
249     if (IR) NS = c10;
250     else NS = Start;
251     end
252     c10://0010001011
253     begin
254     if (!IR) NS = c11;
255     else NS = Start;
256     end
257     c11://00100010110
258     begin
259     if (IR) NS = c12;
260     else NS = Start;
261     end
```

```
262     c12://001000101101
263     begin
264     if (IR) NS = c13;
265     else NS = Start;
266     end
267     c13://0010001011011
268     begin
269     if (IR) NS = c14;
270     else NS = Start;
271     end
272     c14://00100010110111
273     begin
274     if (!IR) NS = c15;
275     else NS = Start;
276     end
277     c15://001000101101110
278     begin
279     if (IR) NS = c16;
280     else NS = Start;
281     end
282     c16://0010001011011101
283     begin
284     PWM_0 = 2'b11;
285     NS = Start;
286     end
287     d3://000
288     begin
289     if (!IR) NS = d4;
290     else NS = Start;
291     end
292     d4://0000
293     begin
294     if (IR) NS = d5;
295     else NS = e5;
296     end
297     d5://00001
298     begin
299     if (!IR) NS = d6;
300     else NS = Start;
301     end
302     d6://000010
303     begin
304     if (!IR) NS = e7;
305     else NS = Start;
306     end
307     d7://0010000
308     begin
309     if (!IR) NS = d8;
310     else NS = Start;
311     end
312     d8://00100000
313     begin
314     if (IR) NS = d9;
315     else NS = Start;
316     end
317     d9://001000001
318     begin
319     if (IR) NS = d10;
320     else NS = Start;
321     end
322     d10://0010000011
323     begin
324     if (!IR) NS = d11;
325     else NS = Start;
326     end
327     d11://00100000110
328     begin
329     if (IR) NS = d12;
330     else NS = Start;
331     end
```

```
332     d12://001000001101
333     begin
334     if (IR) NS = d13;
335     else NS = Start;
336     end
337     d13://0010000011011
338     begin
339     if (IR) NS = d14;
340     else NS = Start;
341     end
342     d14://00100000110111
343     begin
344     if (IR) NS = d15;
345     else NS = Start;
346     end
347     d15://001000001101111
348     begin
349     PWM_2 = 2'b01;
350     NS = Start;
351     end
352     e5://00000
353     begin
354     if (!IR) NS = e6;
355     else NS = Start;
356     end
357     e6://000000
358     begin
359     if (!IR) NS = f7;
360     else NS = Start;
361     end
362     e7://0000100
363     begin
364     if (!IR) NS = e8;
365     else NS = Start;
366     end
367     e8://00001000
368     begin
369     if (IR) NS = e9;
370     else NS = Start;
371     end
372     e9://000010001
373     begin
374     if (IR) NS = e10;
375     else NS = Start;
376     end
377     e10://0000100011
378     begin
379     if (IR) NS = e11;
380     else NS = Start;
381     end
382     e11://00001000111
383     begin
384     if (IR) NS = e12;
385     else NS = Start;
386     end
387     e12://000010001111
388     begin
389     if (!IR) NS = e13;
390     else NS = Start;
391     end
392     e13://0000100011110
393     begin
394     if (IR) NS = e14;
395     else NS = Start;
396     end
397     e14://00001000111101
398     begin
399     PWM_0 = 2'b01;
400     NS = Start;
401     end
```

```

402         f7://00000000
403         begin
404         if (!IR) NS = f8;
405         else NS = Start;
406         end
407         f8://00000000
408         begin
409         if (IR) NS = f9;
410         else NS = Start;
411         end
412         f9://000000001
413         begin
414         if (IR) NS = f10;
415         else NS = Start;
416         end
417         f10://0000000011
418         begin
419         if (IR) NS = f11;
420         else NS = Start;
421         end
422         f11://00000000111
423         begin
424         if (IR) NS = f12;
425         else NS = Start;
426         end
427         f12://000000001111
428         begin
429         if (IR) NS = f13;
430         else NS = Start;
431         end
432         f13://0000000011111
433         begin
434         PWM_2 = 2'b11;
435         NS = Start;
436         end
437     default:
438     NS = Start;
439     endcase
440     end
441 endmodule

```

4.7 Decoder Module

```

1  'timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Ethan Vosburg
5  //
6  // Create Date: 12/05/2023 10:14:10 PM
7  // Design Name: Decoder
8  // Module Name: DEC
9  // Project Name: Robotic Arm with IR input and Servo Output
10 // Target Devices: Basys 3 Development Board
11 // Tool Versions:
12 // Description: Recognizes the input from the IR FSM and passes on
    cycle to accumulator
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22

```

```
23 module DEC(  
24     // Inputs  
25     input [1:0] fsm_in, // 2-bit signed input logic  
26     input clk, // Clock input  
27  
28     // Outputs  
29     output logic [1:0] q, // 2-bit signed output logic  
30     output logic clear // Clear output logic  
31 );  
32  
33 // Defined states for delay  
34 typedef enum {S0, S1, S2, S3} STATES;  
35  
36 // Defines the current and next state  
37 STATES state, next_state = S0;  
38  
39 // Clock logic  
40 always_ff@(negedge clk) begin  
41     state <= next_state;  
42     q = 2'b00;  
43     clear = 1'b0;  
44  
45     // State logic  
46     case (state)  
47         S2: begin  
48             // Set output and clear when in state 2  
49             if (fsm_in == 2'b01) begin  
50                 q = 2'b01;  
51                 clear = 1'b1;  
52             end else if (fsm_in == 2'b11) begin  
53                 q = 2'b11;  
54                 clear = 1'b1;  
55             end  
56         end  
57         S3: begin  
58             // Set output and clear when in state 3  
59             q = 2'b00;  
60             clear = 1'b0;  
61         end  
62     endcase  
63  
64 end  
65  
66 // Delay using state logic  
67 always_comb begin  
68     next_state = state; // Default to stay in current state  
69     case (fsm_in)  
70         2'b01:  
71             case (state)  
72                 S0: next_state = S1;  
73                 S1: next_state = S2;  
74                 S2: next_state = S3;  
75                 S3: next_state = S0;  
76             endcase  
77         2'b11:  
78             case (state)  
79                 S0: next_state = S1;  
80                 S1: next_state = S2;  
81                 S2: next_state = S3;  
82                 S3: next_state = S0;  
83             endcase  
84         default: next_state = S0;  
85     endcase  
86 end  
87 endmodule
```


4.8 PWM Module

```
1  'timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Ethan Vosburg
5  //
6  // Create Date: 12/06/2023 11:32:09 PM
7  // Design Name: PWM Module
8  // Module Name: PWM
9  // Project Name: Robotic Arm with IR input and Servo Output
10 // Target Devices: Basys 3 Development Board
11 // Tool Versions:
12 // Description: Recive the position from the accumulator and output a
    PWM signal
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22
23 module PWM(
24     // Inputs
25     input [7:0] pos_in,
26     input clk,
27     // Outputs
28     output logic pwm_out
29 );
30
31 // Registers to store the period and duty cycle
32 reg [10:0] period = 0;
33 reg [7:0] duty = 0;
34
35 // PWM logic
36 always @(posedge clk ) begin
37     if (period < 11'd2000)
38         begin
39             // Set pwm output high for a portion of the
                period
40             if (duty < pos_in)
41                 begin
42                     duty <= duty + 1;
43                     pwm_out <= 1;
44                 end
45             else
46                 begin
47                     // Set pwm output low for the rest of the
                        period
48                     pwm_out <= 0;
49                 end
50             period <= period + 1;
51         end
52     else
53         begin
54             // Reset period and duty cycle
55             period <= 0;
56             duty <= 0;
57         end
58     end
59 end
60
61 endmodule
```