



CAL POLY

CPE 233

Hardware

Assignment 7

Assembling the Otter Wrapper

Report by:

Ethan Vosburg (evosburg@calpoly.edu)

March 2, 2024

Table of Contents

1 Project Description	3
1.1 Otter Wrapper	3
2 Structural Design	3
2.1 OTTER Wrapper	3
3 Synthesis Warnings	3
3.1 Otter Wrapper Synthesis Warnings	3
4 Source Code	4
5 Conclusion	9

1 Project Description

In this project, the MCU is brought to a fully functional state with the addition of the wrapper that is used to interface with the MCU and allow it to control different aspects of the Basys3 board. This assignment marks a landmark in the development of the Otter. Upon completion of this project, a functional product will be made that can be used in the real world, the only thing that will be missing is the interrupt logic.

1.1 Otter Wrapper

The Otter Wrapper is the bridge between the MCU and the hardware of the Basys3 board. In the wrapper, several demuxes are used to control the flow of data in and out of the MCU. The wrapper handles the control of the LEDs, the 7-segment display, the buttons, and the switches. There is not much logic in the wrapper but it is necessary to create the bridge between the MCU and the hardware.

2 Structural Design

2.1 OTTER Wrapper

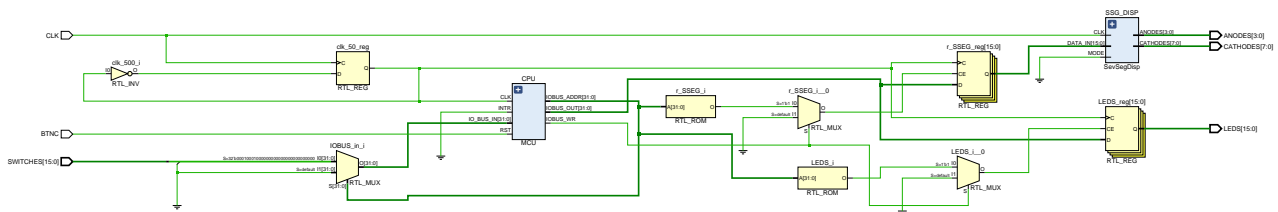


Figure 1: OTTER MCU Elaborated Design

3 Synthesis Warnings

3.1 Otter Wrapper Synthesis Warnings

- ▼ Synthesis (62 warnings)
 - > [Synth 8-3848] Net csr_rd in module/entity MCU does not have driver. [MCU.sv:33] (2 more like this)
 - > [Synth 8-7129] Port INTR in module ControlUnitFSM is either unconnected or has no load (20 more like this)
 - [Synth 8-7080] Parallel synthesis criteria is not met

Figure 2: Synthesis Warnings for MCU

The warnings referenced in Figure 2 are not important as there is hardware that will still need to be implemented in future assignments. This means there are various wires that do not have a driver or that do not lead to a defined location. This is fine and in this case does not prevent the design from working. Vivado is simply letting you know that these lines will not be implemented in to the design at this point.

4 Source Code

Listing 1: Control Unit FSM

```

1  'timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Ethan Vosburg
5  //
6  // Create Date: 02/21/2024 03:42:56 PM
7  // Module Name: ControlUnitFSM
8  // Project Name: Risc-V MCU
9  // Target Devices: Basys3
10 // Description: Control Unit FSM for the Risc-V MCU
11 //
12 // Revision:
13 // Revision 0.01 - File Created
14 //
15 ///////////////////////////////////////////////////////////////////
16
17
18 module ControlUnitFSM(
19     // Inputs
20     input CLK,                // Clock
21     input RST,                // Reset
22     input INTR,               // Interrupt
23     input [6:0] opcode,       // Opcode
24     input [2:0] funct3,       // Function 3
25
26     // Outputs
27     output logic PC_WE,       // Program Counter Write Enable
28     output logic RF_WE,       // Register File Write Enable
29     output logic memWE2,      // Memory Write Enable 2
30     output logic memRDEN1,    // Memory Read Enable 1
31     output logic memRDEN2,    // Memory Read Enable 2
32     output logic reset,       // Reset
33     output logic csr_WE,      // Control and Status Register Write Enable
34     output logic int_taken,    // Interrupt
35     output logic mret_exec    // MRET Execution
36 );
37
38 // Define the state type
39 typedef enum { ST_INIT, ST_FETCH, ST_EXEC, ST_WB } state_type;
40
41 // Define state and next state
42 state_type state, next_state;
43
44 // Transition State
45 always_ff @(posedge CLK) begin
46     if (RST) begin
47         state <= ST_INIT;
48     end else begin
49         state <= next_state;
50     end
51 end
52
53 // Define the state logic

```

```
54 always_comb begin
55     // Initialize all the outputs to zeros
56     PC_WE = 1'b0;
57     RF_WE = 1'b0;
58     memWE2 = 1'b0;
59     memRDEN1 = 1'b0;
60     memRDEN2 = 1'b0;
61     reset = 1'b0;
62     csr_WE = 1'b0;
63     int_taken = 1'b0;
64     mret_exec = 1'b0;
65
66     case (state)
67     ST_INIT: begin
68         reset = 1'b1;
69         next_state = ST_FETCH;
70     end
71     ST_FETCH: begin
72         memRDEN1 = 1'b1;
73         next_state = ST_EXEC;
74     end
75     ST_EXEC: begin
76         case (opcode)
77         // J-Type Instruction
78         7'b1101111: begin
79             PC_WE = 1'b1;
80             RF_WE = 1'b1;
81             next_state = ST_FETCH;
82         end
83         // R-Type Instruction
84         7'b0110011: begin
85             PC_WE = 1'b1;
86             RF_WE = 1'b1;
87             next_state = ST_FETCH;
88         end
89         // I-Type Instruction
90         7'b0010011: begin
91             PC_WE = 1'b1;
92             memRDEN1 = 1'b1;
93             RF_WE = 1'b1;
94             next_state = ST_FETCH;
95         end
96         // JALR Instruction
97         7'b1100111: begin
98             PC_WE = 1'b1;
99             memRDEN1 = 1'b1;
100             RF_WE = 1'b1;
101             next_state = ST_FETCH;
102         end
103         // Loading Instructions
104         7'b0000011: begin
105             PC_WE = 1;
106             memRDEN1 = 1'b1;
107             memRDEN2 = 1;
108             RF_WE = 1'b1;
109             next_state = ST_WB;
110         end
111         // B-Type Instruction
112         7'b1100011: begin
113             PC_WE = 1'b1;
114             next_state = ST_FETCH;
115         end
116         // U-Type Instruction
117         7'b0110111: begin
118             PC_WE = 1'b1;
119             RF_WE = 1'b1;
120             memRDEN1 = 1'b1;
121             next_state = ST_FETCH;
122         end
123         // AUIPC Instruction
```

```
124         7'b0010111: begin
125             PC_WE = 1'b1;
126             RF_WE = 1'b1;
127             memRDEN1 = 1'b1;
128             next_state = ST_FETCH;
129         end
130         // S-Type Instructions
131         7'b0100011: begin
132             PC_WE = 1'b1;
133             memRDEN1 = 1'b1;
134             memWE2 = 1;
135             next_state = ST_FETCH;
136         end
137
138         default: next_state = ST_INIT; // Should never happen
139     endcase
140 end
141 ST_WB: begin
142     RF_WE = 1'b1;
143     PC_WE = 0;
144     next_state = ST_FETCH;
145 end
146 default: next_state = ST_INIT; // Should never happen
147 endcase
148 end
149 endmodule
```

Listing 2: Control Unit Decoder

```

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Ethan Vosburg
5  //
6  // Create Date: 02/21/2024 03:47:15 PM
7  // Module Name: ControlUnitDecoder
8  // Project Name: Otter MCU
9  // Target Devices: Basys3
10 // Description: Control Unit Decoder for controlling the ALU
11 //
12 // Revision:
13 // Revision 0.01 - File Created
14 //
15 ///////////////////////////////////////////////////////////////////
16
17 module ControlUnitDecoder(
18     // Inputs
19     input br_eq,           // Branch Equal
20     input br_lt,           // Branch Less Than
21     input br_ltu,          // Branch Less Than Unsigned
22     input [6:0] funct7,    // Function 7
23     input [6:0] opcode,    // Opcode
24     input [2:0] funct3,    // Function 3
25
26     // Outputs
27     output logic [3:0] ALU_FUN, // ALU Function
28     output logic [1:0] srcA_SEL, // Source A Select
29     output logic [2:0] srcB_SEL, // Source B Select
30     output logic [2:0] PC_SEL,   // Program Counter Select
31     output logic [1:0] RF_SEL,   // Register File Select
32 );
33
34 always_comb begin
35     // Initialize all the outputs to zeros
36     ALU_FUN = 4'b0000;
37     srcA_SEL = 2'b0;
38     srcB_SEL = 3'b0;
39     PC_SEL = 3'b0;
40     RF_SEL = 2'b00;
41
42     case (opcode)
43         // R-Type Instruction
44         7'b0110011: begin
45             ALU_FUN = {funct7[5], funct3};
46             srcA_SEL = 2'b0;
47             srcB_SEL = 0;
48             PC_SEL = 3'b0;
49             RF_SEL = 2'b11;
50             // Logic for ALU Select B
51             // if ({funct7[5], funct3} == 4'b0010) srcB_SEL = 3'b1;
52             // if ({funct7[5], funct3} == 4'b0100) srcB_SEL = 3'b0;
53         end
54         // I-Type Instruction
55         7'b0010011: begin
56             // Take care of special case with SRAI
57             if (funct3 == 3'b101)
58                 ALU_FUN = {funct7[5], funct3};
59             else
60                 ALU_FUN = {1'b0, funct3};
61
62             // Configure selects
63             srcA_SEL = 2'b0;
64             srcB_SEL = 3'b1;
65             RF_SEL = 2'b11;
66             PC_SEL = 3'b0;
67         end
68     end
69     // JALR Instructions

```

```
70      7'b1100111: begin
71          PC_SEL = 3'b1;
72          // ALU_FUN = {1'b0, funct3};
73          // srcA_SEL = 2'b0;
74          // srcB_SEL = 3'b1;
75          RF_SEL = 2'b0;
76      end
77      // Loading Instructions
78      7'b0000011: begin
79          PC_SEL = 3'b0;
80          RF_SEL = 2;
81          srcA_SEL = 0;
82          srcB_SEL = 1;
83          ALU_FUN = 4'b0000;
84      end
85      // J-Type Instruction
86      7'b1101111: begin
87          PC_SEL = 3'b11;
88          RF_SEL = 2'b0;
89      end
90      // B-Type Instruction
91      7'b1100011: begin
92          case(funct3)
93              3'b000: begin
94                  if (br_eq)
95                      PC_SEL = 3'b10;
96                  else
97                      PC_SEL = 3'b0;
98              end
99
100             3'b101: begin
101                 if (!br_lt)
102                     PC_SEL = 3'b10;
103                 else
104                     PC_SEL = 3'b0;
105             end
106
107             3'b111: begin
108                 if (!br_ltu)
109                     PC_SEL = 3'b10;
110                 else
111                     PC_SEL = 3'b0;
112             end
113
114             3'b100: begin
115                 if (br_lt)
116                     PC_SEL = 3'b10;
117                 else
118                     PC_SEL = 3'b0;
119             end
120
121             3'b110: begin
122                 if (br_ltu)
123                     PC_SEL = 3'b10;
124                 else
125                     PC_SEL = 3'b0;
126             end
127
128             3'b001: begin
129                 if (!br_eq)
130                     PC_SEL = 3'b10;
131                 else
132                     PC_SEL = 3'b0;
133             end
134
135             default: PC_SEL = 3'b111;
136          endcase
137      end
138      // U-Type Instruction
139      7'b0110111: begin
```



```
140         ALU_FUN = 4'b1001;
141         srcA_SEL = 2'b1;
142         PC_SEL = 3'b0;
143         RF_SEL = 2'b11;
144     end
145
146     // AUIPC Command
147     7'b0010111: begin
148         ALU_FUN = 4'b0000;
149         srcA_SEL = 2'b1;
150         srcB_SEL = 3'b11;
151         PC_SEL = 3'b0;
152         RF_SEL = 2'b11;
153     end
154     // S-Type Instruction
155     7'b0100011: begin
156         ALU_FUN = 0;
157         srcA_SEL = 0;
158         srcB_SEL = 2;
159         PC_SEL = 0;
160     end
161
162     default: begin
163         // Should not be used
164         ALU_FUN = 4'b0000;
165         srcA_SEL = 2'b1;
166         srcB_SEL = 3'b11;
167         PC_SEL = 3'b0;
168         RF_SEL = 2'b11;
169     end
170 endcase
171
172 end
173
174
175 endmodule
```

5 Conclusion

In this project, the Otter wrapper was successfully constructed and tested using the provided file `Test_All.mem`. The hardware was found to be functional and it is now ready to be used in projects. While the hardware is not finished, it can be used. In future hardware assignments, interrupts will be added to complete the Otter. All code for this assignment can be found [here](#).