CPE 233 Hardware Assignment 7

Assembeling the Otter Wrapper

Report by:
Ethan Vosburg (evosburg@calpoly.edu)

March 2, 2024

Table of Contents

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

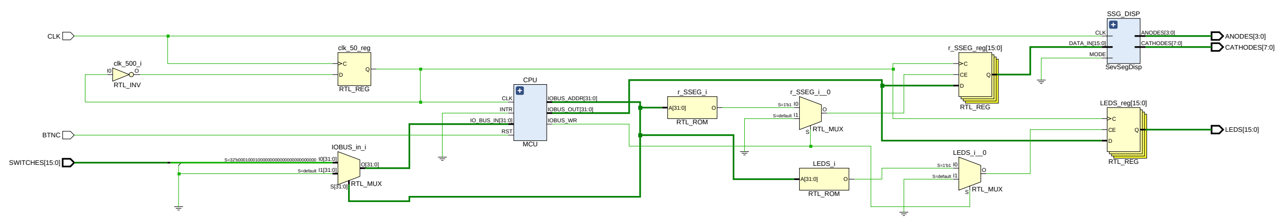
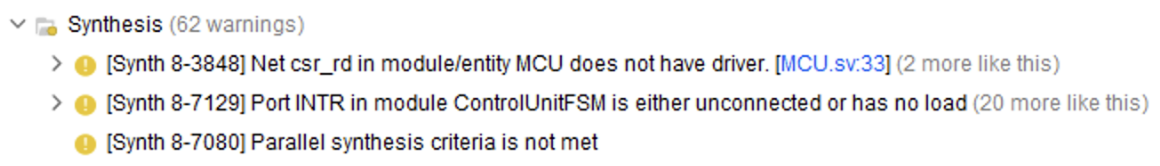


Figure 1: OTTER MCU Eleborated Design

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

```
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/21/2024 03:42:56 PM
// Module Name: ControlUnitFSM
// Project Name: Risc-V MCU
// Target Devices: Basys3
// Description: Control Unit FSM for the Risc-V MCU
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ControlUnitFSM(
    // Inputs
    input CLK,                           // Clock
    input RST,                           // Reset
    input INTR,                          // Interrupt
    input [6:0] opcode,                  // Opcode
    input [2:0] funct3,                  // Function 3

    // Outputs
    output logic PC_WE,                  // Program Counter Write Enable
    output logic RF_WE,                  // Register File Write Enable
    output logic memWE2,                 // Memory Write Enable 2
    output logic memRDEN1,               // Memory Read Enable 1
    output logic memRDEN2,              // Memory Read Enable 2
    output logic reset,                  // Reset
    output logic csr_WE,                 // Control and Status Register Write Enable
    output logic int_taken,              // Interrupt
    output logic mret_exec               // MRET Execution
);
    // Define the state type
    typedef enum { ST_INIT, ST_FETCH, ST_EXEC, ST_WB } state_type;

    //  Define state and next state
    state_type state, next_state;

    // Transition State
    always_ff @(posedge CLK) begin
        if (RST) begin
            state <= ST_INIT;
        end else begin
            state <= next_state;
        end
    end


    // Define the state logic
    always_comb begin
        // Initialize all the outputs to zeros
        PC_WE = 1'b0;
        RF_WE = 1'b0;
        memWE2 = 1'b0;
        memRDEN1 = 1'b0;
        memRDEN2 = 1'b0;
        reset = 1'b0;
        csr_WE = 1'b0;
        int_taken = 1'b0;
        mret_exec = 1'b0;

        case (state)
            ST_INIT: begin
                reset = 1'b1;
                next_state = ST_FETCH;
            end
            ST_FETCH: begin
                memRDEN1 = 1'b1;
                next_state = ST_EXEC;
            end
            ST_EXEC: begin
                case (opcode)
                // J-Type Instruction
                7'b1101111: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // R-Type Instruction
                7'b0110011: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // I-Type Instruction
                7'b0010011: begin
                    PC_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // JALR Instruction
                7'b1100111: begin
                    PC_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // Loading Instructions
                7'b0000011: begin
                    PC_WE = 1;
                    memRDEN1 = 1'b1;
                    memRDEN2 = 1;
                    RF_WE = 1'b1;
                    next_state = ST_WB;
                end
                // B-Type Instruction
                7'b1100011: begin
                    PC_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // U-Type Instruction
                7'b0110111: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    next_state = ST_FETCH;
                end
                // AUIPC Instruction
                7'b0010111: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    next_state = ST_FETCH;
                end
                // S-Type Instructions
                7'b0100011: begin
                    PC_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    memWE2 = 1;
                    next_state = ST_FETCH;
                end

                default: next_state = ST_INIT; // Should never happen
                endcase
            end
            ST_WB: begin
                RF_WE = 1'b1;
                PC_WE = 0;
                next_state = ST_FETCH;
            end
            default: next_state = ST_INIT; // Should never happen
        endcase
    end
endmodule
```

Listing 2: Control Unit Decoder

```
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/21/2024 03:47:15 PM
// Module Name: ControlUnitDecoder
// Project Name: Otter MCU
// Target Devices: Basys3
// Description: Control Unit Decoder for controlling the ALU
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ControlUnitDecoder(
    // Inputs
    input br_eq,                          // Branch Equal
    input br_lt,                          // Branch Less Than
    input br_ltu,                         // Branch Less Than Unsigned
    input [6:0] funct7,                   // Function 7
    input [6:0] opcode,                   // Opcode
    input [2:0] funct3,                   // Function 3

    // Outputs
    output logic [3:0] ALU_FUN,           // ALU Function
    output logic [1:0] srcA_SEL,          // Source A Select
    output logic [2:0] srcB_SEL,          // Source B Select
    output logic [2:0] PC_SEL,            // Program Counter Select
    output logic [1:0] RF_SEL             // Register File Select
    );

    always_comb begin
        // Initialize all the outputs to zeros
        ALU_FUN = 4'b0000;
        srcA_SEL = 2'b0;
        srcB_SEL = 3'b0;
        PC_SEL = 3'b0;
        RF_SEL = 2'b00;

        case (opcode)
        // R-Type Instruction
        7'b0110011: begin
            ALU_FUN = {funct7[5], funct3};
            srcA_SEL = 2'b0;
            srcB_SEL = 0;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
            // Logic for ALU Select B
            // if ({funct7[5], funct3} == 4'b0010) srcB_SEL = 3'b1;
            // if ({funct7[5], funct3} == 4'b0100) srcB_SEL = 3'b0;
        end
        // I-Type Instruction
        7'b0010011: begin
            // Take care of special case with SRAI
            if (funct3 == 3'b101)
                ALU_FUN = {funct7[5], funct3};
            else
                ALU_FUN = {1'b0, funct3};

            // Configure selects
            srcA_SEL = 2'b0;
            srcB_SEL = 3'b1;
            RF_SEL = 2'b11;
            PC_SEL = 3'b0;
        end
        // JALR Instructions
        7'b1100111: begin
            PC_SEL = 3'b1;
            // ALU_FUN = {1'b0, funct3};
            // srcA_SEL = 2'b0;
            // srcB_SEL = 3'b1;
            RF_SEL = 2'b0;
        end
        // Loading Instructions
        7'b0000011: begin
            PC_SEL = 3'b0;
            RF_SEL = 2;
            srcA_SEL = 0;
            srcB_SEL = 1;
            ALU_FUN = 4'b0000;
        end
        // J-Type Instruction
        7'b1101111: begin
            PC_SEL = 3'b11;
            RF_SEL = 2'b0;
        end
        // B-Type Instruction
        7'b1100011: begin
            case(funct3)
                3'b000: begin
                    if (br_eq)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b101: begin
                    if (!br_lt)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b111: begin
                    if (!br_ltu)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b100: begin
                    if (br_lt)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b110: begin
                    if (br_ltu)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b001: begin
                    if (!br_eq)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                default: PC_SEL = 3'b111;
            endcase
        end
        // U-Type Instruction
        7'b0110111: begin
```

```
            ALU_FUN = 4'b1001;
            srcA_SEL = 2'b1;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
        end

        // AUIPC Command
        7'b0010111: begin
            ALU_FUN = 4'b0000;
            srcA_SEL = 2'b1;
            srcB_SEL = 3'b11;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
        end
        // S-Type Instruction
        7'b0100011: begin
            ALU_FUN = 0;
            srcA_SEL = 0;
            srcB_SEL = 2;
            PC_SEL = 0;
        end

        default: begin
            // Should not be used
            ALU_FUN = 4'b0000;
            srcA_SEL = 2'b1;
            srcB_SEL = 3'b11;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
        end
        endcase

    end


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.