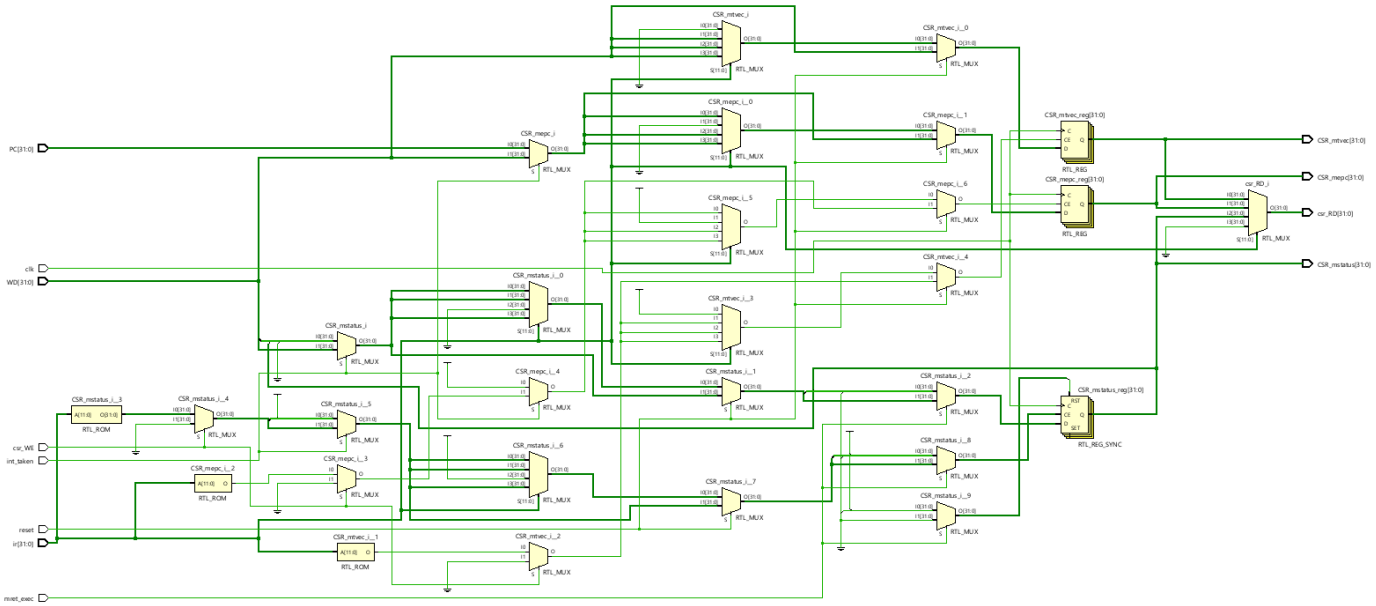
CPE 233 HW 8

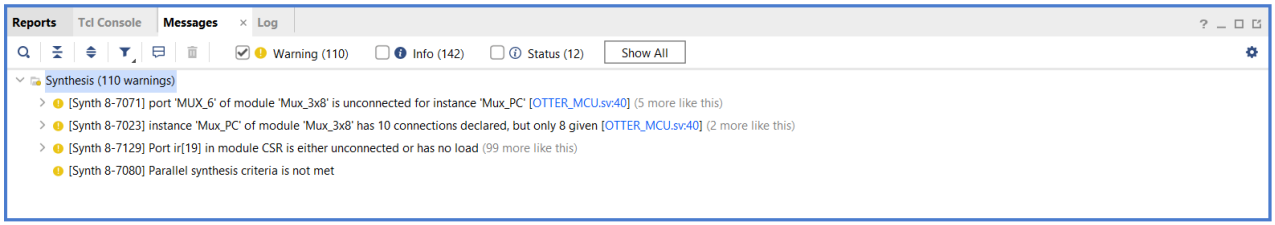
Wyatt Tack

1. Behavior Description: The CSR functions as a separate register file, relating specifically to the interrupt service routine. Three main registers are stored in the CSR, being the program address for the service routing (MTVEC), the interrupt enable status (MTSTATUS), and the stored current program count when an interrupt is pressed (MEPC). These addresses can be written to and read from, and are output asynchronously to the control unit FSM to determine what to do when an interrupt signal occurs.

2. Structural Design for CSR:

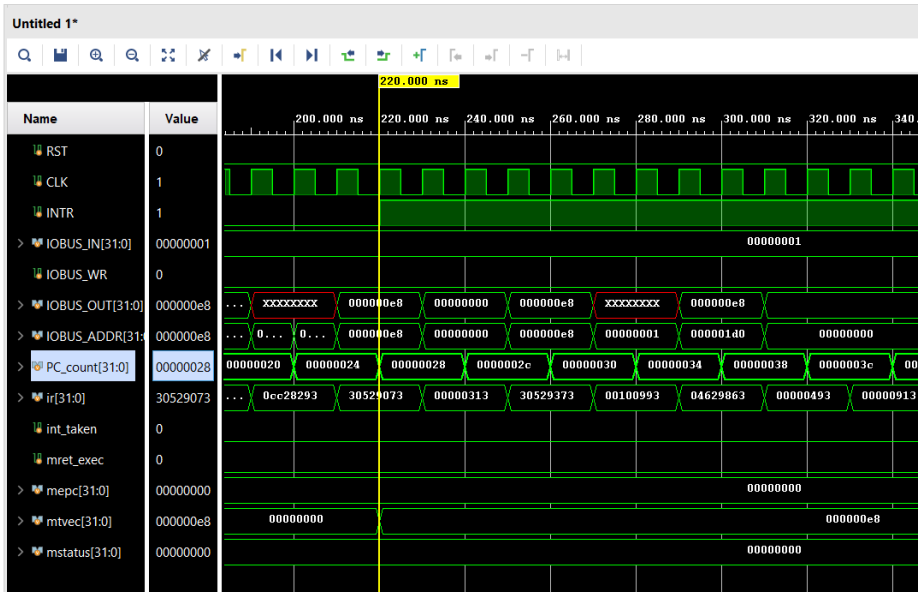


3. Synthesis Warnings Listing:

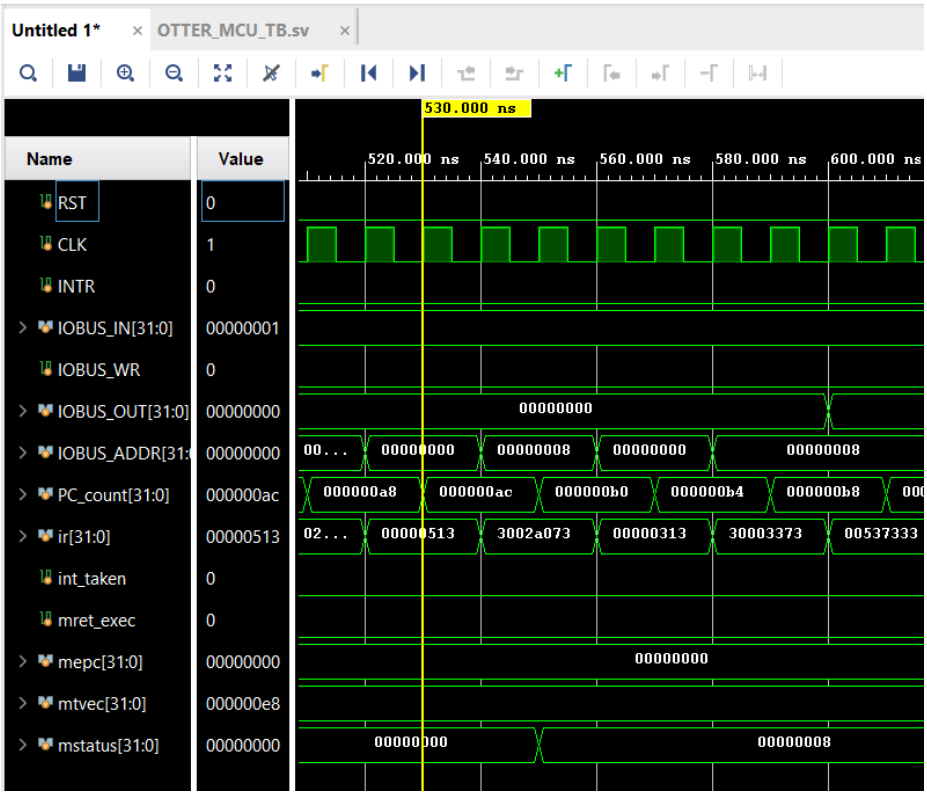


4. Verification (specified instructions executed after time marker):

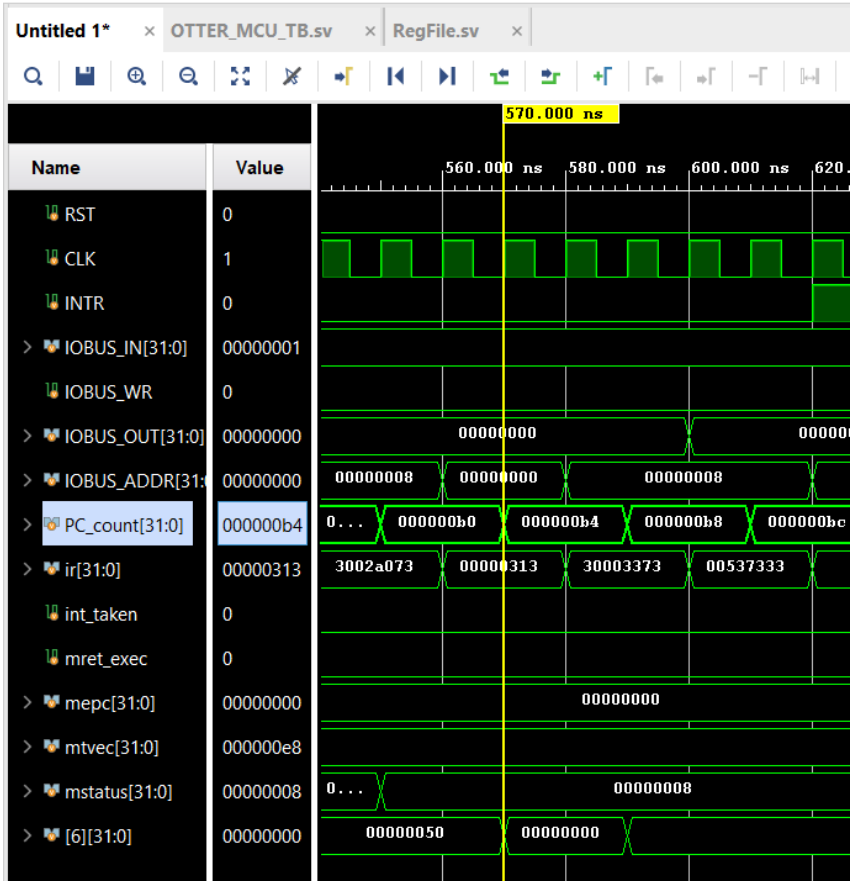
CSRRW:



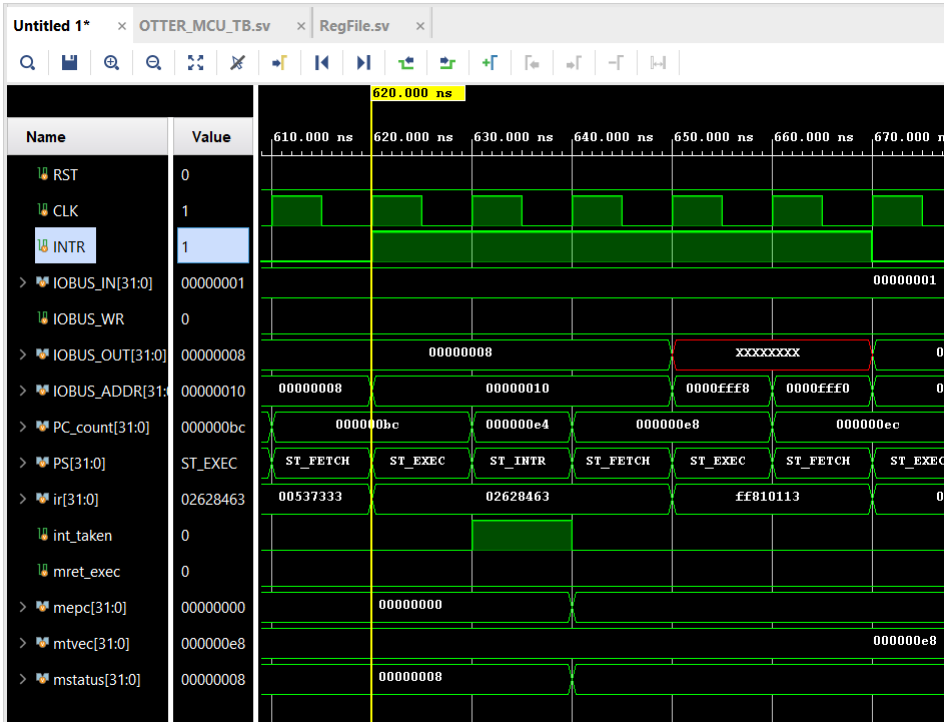
CSRRS:



CSRRC:


Untitled 1*
OTTER_MCU_TB.sv
RegFile.sv
570.000 ns
560.000 ns
580.000 ns
600.000 ns
Name
Value
RST
0
CLK
1
INTR
0
IOBUS_IN[31:0]
00000001
IOBUS_WR
0
IOBUS_OUT[31:0]
00000000
IOBUS_ADDR[31:
00000000
PC_count[31:0]
000000b4
ir[31:0]
00000313
int_taken
0
mret_exec
0
mepc[31:0]
00000000
mtvec[31:0]
000000e8
mstatus[31:0]
00000008
[6][31:0]
00000000
00000000
00000008
00000000
00000008
0...
000000b0
000000b4
000000b8
000000bc
3002a073
00000313
30003373
00537333
00000000
0...
00000008
00000050
00000000

INTR:


Untitled 1*
OTTER_MCU_TB.sv
RegFile.sv
620.000 ns
610.000 ns
620.000 ns
630.000 ns
640.000 ns
650.000 ns
660.000 ns
Name
Value
RST
0
CLK
1
INTR
1
IOBUS_IN[31:0]
00000001
IOBUS_WR
0
IOBUS_OUT[31:0]
00000008
IOBUS_ADDR[31:
00000010
PC_count[31:0]
000000bc
PS[31:0]
ST_EXEC
ir[31:0]
02628463
int_taken
0
mret_exec
0
mepc[31:0]
00000000
mtvec[31:0]
000000e8
mstatus[31:0]
00000008
00000001
00000008
xxxxxxxx
00000008
00000010
0000fff8
0000fff0
000000bc
000000e4
000000e8
000000ec
ST_FETCH
ST_EXEC
ST_INTR
ST_FETCH
ST_EXEC
ST_FETCH
00537333
02628463
ff810113
00000000
000000e8
00000008

MRET:

5. System Verilog Source Code:

For Control System Registers:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 02/29/2024
// Design Name: Control Status Registers
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Registers that hold data for interupts
//
//////////////////////////////////////////////////////////////////
module CSR(
input clk,
input reset, mret_exec, int_taken, csr_WE,
input [31:0] ir, PC, WD,
output logic[31:0] CSR_mstatus, CSR_mepc, CSR_mtvec,
output logic[31:0] csr_RD
    );
always_ff@(posedge clk)
begin
    if(csr_WE == 1)      //write data synchronously
        begin
        case(ir[31:20])
        12'h305: CSR_mtvec <= WD;
        12'h341: CSR_mepc <= WD;
        12'h300: CSR_mstatus <= WD;
        endcase
        end
    if(int_taken == 1)  //when interupt
        begin
        CSR_mepc <= PC;
        CSR_mstatus[7] <= CSR_mstatus[3];
        CSR_mstatus[3] <= 1'b0;
        end
    if(reset == 1)      //if reset
        begin
        CSR_mtvec <= 0;
        CSR_mepc <= 0;
        CSR_mstatus <= 0;
        end
    if(mret_exec == 1)  //when intr end, reset status enable
        begin
        CSR_mstatus[3] <= CSR_mstatus[7];
        CSR_mstatus[7] <= 1'b0;
        end
end
always_comb begin
    case(ir[31:20])     //output asynch read
        12'h305: csr_RD = CSR_mtvec;
        12'h341: csr_RD = CSR_mepc;
        12'h300: csr_RD = CSR_mstatus;
        default: csr_RD = 0;
    endcase
end
endmodule
```

For Control Unit Decoder:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 02/01/2024
// Design Name: Control Unit Decoder
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Sends non timing dependent selection signals
//              for data manipulation in otter MCU
//
//////////////////////////////////////////////////////////////

module CU_DCDR(
input [31:0] ir,
input br_eq, br_lt, br_ltu,
input int_taken,
output logic [3:0] ALU_FUN,
output logic [1:0] srcA_SEL,
output logic [2:0] srcB_SEL,
output logic [2:0] PC_SEL,
output logic [1:0] RF_SEL
    );
always_comb begin
    ALU_FUN = 0;
    srcA_SEL = 0;
    srcB_SEL = 0;
    PC_SEL = 0;
    RF_SEL = 0;
    case (ir[6:0]) //op-codes
    7'b0110011: //All R-Type
    begin
        ALU_FUN = {ir[30], ir[14:12]};
        srcA_SEL = 0;
        srcB_SEL = 0;
        PC_SEL = 0;
        RF_SEL = 3;
    end
    7'b0010011: //1st set of I-Type Instructions
    begin
        if (ir[14:12] == 3'b101) ALU_FUN = {ir[30], ir[14:12]};
        else ALU_FUN = {1'b0, ir[14:12]};
        srcA_SEL = 0;
        srcB_SEL = 1;
        PC_SEL = 0;
        RF_SEL = 3;
    end
    7'b0000011: //2nd set of I-Type Instructions
    begin
        ALU_FUN = 4'b0000;
        srcA_SEL = 0;
        srcB_SEL = 1;
        PC_SEL = 0;
        RF_SEL = 2;
    end
    7'b1100111: //Last set of I-Type (for jalr)
    begin
        PC_SEL = 1;
        RF_SEL = 0;
    end
    7'b0100011: //All S-Type Instructions
    begin
        ALU_FUN = 4'b0000;
        srcA_SEL = 0;
        srcB_SEL = 2;
        PC_SEL = 0;
    end
    7'b1100011: //All B-Type Instructions (include conditions)
    begin
        case(ir[14:13])
        2'b00: //equal
        begin
        PC_SEL = {1'b0,(ir[12]^br_eq),1'b0};
        end
        2'b10: //less than
        begin
        PC_SEL = {1'b0,(ir[12]^br_lt),1'b0};
        end
        2'b11: //less than unsigned
        begin
        PC_SEL = {1'b0,(ir[12]^br_ltu),1'b0};
        end
        default:
        begin
        PC_SEL = 0;
        end
        endcase
    end
    7'b0110111: //1st set of U-Type (lui)
    begin
        ALU_FUN = 4'b1001;
        srcA_SEL = 1;
        PC_SEL = 0;
        RF_SEL = 3;
    end
    7'b0010111: //2nd set of U-Type (auipc)
    begin     //auipc*********************************auipc
        ALU_FUN = 4'b0000;
        srcA_SEL = 1;
        srcB_SEL = 3;
        PC_SEL = 0;
        RF_SEL = 3;
    end
    7'b1101111: //All J-Type instructions (jal)
    begin
        PC_SEL = 3;
        RF_SEL = 0;
    end
    7'b1110011: //All CSR instructions
    begin
        if(ir[14:12] == 3'b000)PC_SEL = 5;//if mret, branch to mepc

        else begin
        RF_SEL = 2'b01;
        srcB_SEL = 3'b100;
        case(ir[14:12])
        3'b011://c
        begin
        ALU_FUN = 4'b0111;
        srcA_SEL = 2'b10;
        end
        3'b010://s
        begin
        ALU_FUN = 4'b0110;
        srcA_SEL = 2'b00;
        end
        3'b001://w
        begin
        ALU_FUN = 4'b1001;
        srcA_SEL = 2'b00;
        end
        endcase
        end
    end
    endcase
    if(int_taken == 1)PC_SEL = 3'b100;//if interupt branch to ISR

end
endmodule
```

For Control Unit FSM:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 02/01/2024
// Design Name: Control Unit FSM
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Sends selection signals for timing specific
//              operations in memory of OTTER MCU
//
//////////////////////////////////////////////////////////////////


module CU_FSM(
input RST, clk,
input INTR,
input [31:0] ir,
output logic PC_WE, RF_WE, memWE2, memRDEN1, memRDEN2, reset,
output logic csr_WE, int_taken, mret_exec
    );
typedef enum {ST_INIT, ST_FETCH, ST_EXEC, ST_WRITE, ST_INTR} state_type;
state_type NS, PS;
always_ff@(posedge clk) begin //state register
    if(RST == 1) PS<=ST_INIT;
    else PS<=NS;
end
always_comb begin //input/output logic
    PC_WE = 0;
    RF_WE = 0;
    memWE2 = 0;
    memRDEN1 = 0;
    memRDEN2 = 0;
    reset = 0;
    csr_WE = 0;
    int_taken = 0;
    mret_exec = 0;
case(PS)
    ST_INIT: begin
    reset = 1'b1;
    NS = ST_FETCH;
    end
    ST_FETCH: begin
    memRDEN1 = 1'b1;
    NS = ST_EXEC;
    end
    ST_EXEC: begin
        case (ir[6:0]) //op-codes
        7'b0110011: //All R-Type
        begin
            PC_WE = 1;
            RF_WE = 1;
        end
        7'b0010011: //1st set of I-Type Instructions
        begin
            PC_WE = 1;
            RF_WE = 1;
        end
        7'b0000011: //2nd set of I-Type Instructions
        begin
            memRDEN2 = 1;
        end
        7'b1100111: //Last set of I-Type (for jalr)
        begin
            PC_WE = 1;
            RF_WE = 1;
        end
        7'b0100011: //All S-Type Instructions
        begin
            PC_WE = 1;
            memWE2 = 1;
        end
        7'b1100011: //All B-Type Instructions (include conditions)
        begin
            PC_WE = 1;
        end
        7'b0110111: //1st set of U-Type (lui)
        begin
            PC_WE = 1;
            RF_WE = 1;
        end
        7'b0010111: //2nd set of U-Type (auipc)
        begin
            PC_WE = 1;
            RF_WE = 1;
        end
        7'b1101111: //All J-Type instructions (jal)
        begin
            PC_WE = 1;
            RF_WE = 1;
        end
        7'b1110011: //All CSR instructions
        begin
            PC_WE = 1;
            if(ir[14:12] == 3'b000)mret_exec = 1'b1;
            else begin
            csr_WE = 1;
            RF_WE = 1;
            end
        end
        endcase

        //state selecter
        if (INTR == 1 && ir[6:0] != 7'b0000011) NS = ST_INTR;//!load+INTR->INTR
        else if(ir[6:0] == 7'b0000011) NS = ST_WRITE;        //load->WRITEBACK
        else NS = ST_FETCH;                                  //!load+!INTR->FETCH
    end
    ST_WRITE: begin
        //state calculator
        if(INTR == 1)NS = ST_INTR;
        else NS = ST_FETCH;
        PC_WE = 1;
        RF_WE = 1;
    end
    ST_INTR: begin
    int_taken = 1;
    PC_WE = 1;
    NS = ST_FETCH;
    end
    default: begin
    NS = ST_INIT;
    end
endcase
end
endmodule
```