HW4: It's Alive

Summer 2021

Thiha Myint

Demo

https://www.youtube.com/watch?v=XW_ZmWWtXjE

Description

The purpose of this lab to write a system Verilog files that do calculation to use on Basys board. The value is to be read from switches and adds 1 to it, then the output on the board shows the expected value as LEDS. The finite state machine is created to write and read value. The otter state is implemented according to the diagram found from lab manual. Then, the branch condition generator, immediate generator and target generator is created from the schematic. After creating required files from the schematic, wrapper files is created to put everything into single file and connected each other. Expected outcome was seen and it was works on the board when put in generate bitstream

Simulation Waveform

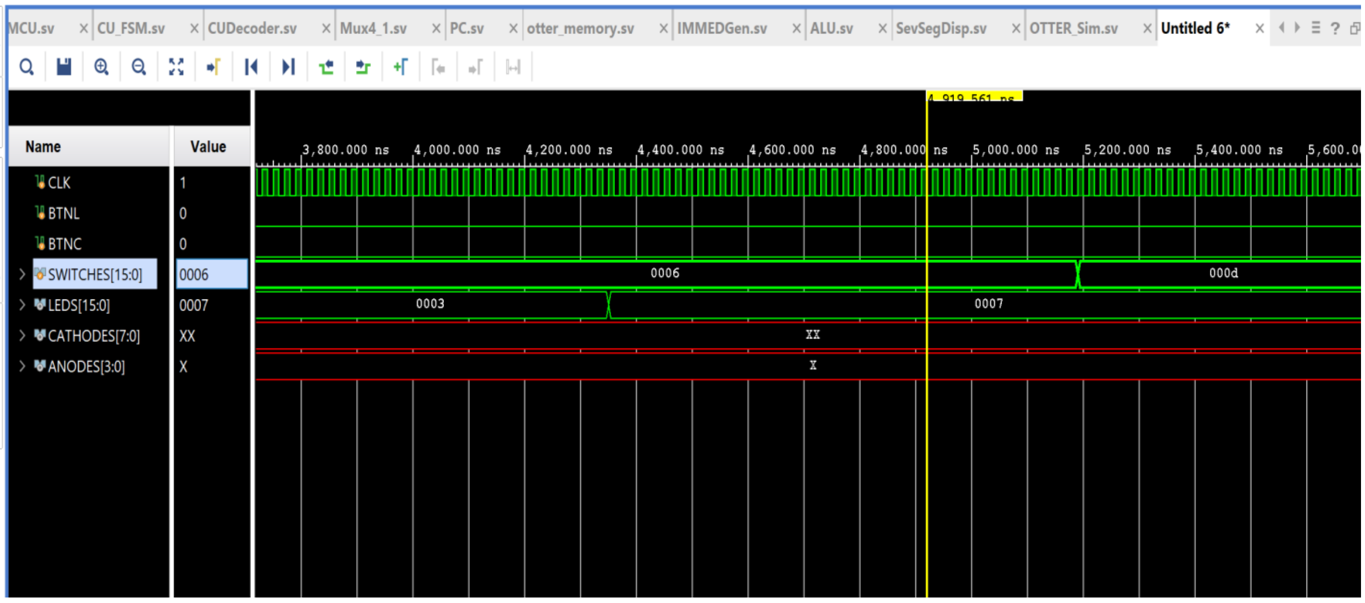

Fig: Waveform of LEDS showing SWITCHS +1 value when turn on

Code

```
`timescale 1ns / 1ps

module CU_FSM(
    input CLK, INT1, RST,
    input [6:0] CU_OPCODE,
    output logic PC_WRITE, REG_WRITE, MEM_WRITE, MEM_READ1, MEM_READ2
    );

    typedef enum {FETCH, DECODE, WB} STATES;
     STATES NS, PS;

     typedef enum logic [6:0]     {
             LUI      = 7'b0110111,
             AUIPC    = 7'b0010111,
             JAL      = 7'b1101111,
             JALR     = 7'b1100111,
             BRANCH   = 7'b1100011,
             LOAD     = 7'b0000011,
             STORE    = 7'b0100011,
             OP_IMM   = 7'b0010011,
             OP       = 7'b0110011
    } opcode_t;

    opcode_t  OPCODE;
    assign OPCODE = opcode_t' (CU_OPCODE);

    always_ff @ (posedge CLK)
    begin
       if (RST)
           PS <= FETCH;
       else
           PS <= NS;
    end

    always_comb
    begin
        PC_WRITE = 0; REG_WRITE = 0; MEM_WRITE = 0; MEM_READ1 = 0; MEM_READ2
= 0;
        case(PS)

            FETCH:
            begin
                PC_WRITE = 0; REG_WRITE = 0; MEM_WRITE = 0; MEM_READ1 = 1;
MEM_READ2 = 0;
                NS = DECODE;
            end

            DECODE:
            begin
                case(CU_OPCODE)
                    LUI:
                    begin
                        PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    AUIPC:
                    begin
                        PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    JAL:
                    begin
                        PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    JALR:
                    begin
                        PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    BRANCH:
                    begin
                        PC_WRITE = 1; REG_WRITE = 0; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    LOAD:
                    begin
                        PC_WRITE = 0; REG_WRITE = 0; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 1; NS = WB;
                    end
                    STORE:
                    begin
                        PC_WRITE = 1; REG_WRITE = 0; MEM_WRITE = 1; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    OP_IMM:
                    begin
                        PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    OP:
                    begin
                        PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1
= 0;
                        MEM_READ2 = 0; NS = FETCH;
                    end
                    default:
                    begin
                        PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1
= 0; MEM_READ2 = 0; NS = FETCH;
                    end
                endcase
            end
            WB:
```

```
            begin
                PC_WRITE = 1; REG_WRITE = 1; MEM_WRITE = 0; MEM_READ1 = 0;
MEM_READ2 = 0;
                NS = FETCH;
            end
            default:
                NS = FETCH;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps

module OTTER_MCU(
    input CLK, RST,INTR,
    input [31:0] IOBUS_IN,
    output logic IOBUS_WR,
    output logic [31:0] IOBUS_OUT, IOBUS_ADDR
    );
//------INTERNAL WIRES&REGS-------
    wire pcWrite, regWrite, memWrite, memRead1, memRead2;
    wire alu_srcA;
    wire [3:0] alu_fun;
    wire [1:0] alu_srcB, pcSource, rf_wr_sel;
    wire [31:0] pc_4, pc_in, pc_out;
    wire [31:0] ir;
    wire [4:0] rd_addr, rs1_addr, rs2_addr;
    wire [31:0] rd, rs1, rs2;
    wire [6:0] opcode;
    wire [2:0] func3;
    wire [6:0] func7;
    wire [31:0] Itype, Utype, Stype, Btype, Jtype;
    wire [31:0] jalr, branch, jal;
    wire br_eq, br_lt, br_ltu;
    wire [31:0] alu_arg1, alu_arg2, alu_out;
    wire [31:0] mem_out,CSRreg;
//--------------------------------
//----------CONTROL PATH----------
    CU_FSM OTTER_FSM(
    .INT1(INTR),
    .RST(RST),
    .CU_OPCODE(opcode),
    .PC_WRITE(pcWrite),
    .REG_WRITE(regWrite),
    .MEM_WRITE(memWrite),
    .MEM_READ2(memRead2),
    .MEM_READ1(memRead1),
    .CLK(CLK) );

    CUDecoder OTTER_Decoder(
    .BR_EQ(br_eq),
    .BR_LT(br_lt),
    .BR_LTU(br_ltu),
    .CU_OPCODE(opcode),
    .FUNC3(func3),
    .FUNC7(func7),
    .ALU_FUN(alu_fun),
    .ALU_SRCA(alu_srcA),
    .ALU_SRCB(alu_srcB),
    .PC_SOURCE(pcSource),
    .RF_WR_SEL(rf_wr_sel) );

//---------------------------------
//--------DATA PATH---------------
    Mux4_1 PC_MUX(
    .ZERO(pc_4),
    .ONE(jalr),                        //---
    .TWO(branch),                        //---
    .THREE(jal),                      //---
    .SEL(pcSource),
    .MUX_OUT(pc_in));

    PC myPC(
    .CLK(CLK),
    .RESET(RST),
    .PC_WRITE(pcWrite),
    .DIN(pc_in),
    .DOUT(pc_out));

    OTTER_mem_byte MEM(
    .MEM_ADDR1(pc_out),
    .MEM_ADDR2(alu_out),
    .MEM_READ1(memRead1),
    .MEM_READ2(memRead2),
    .MEM_WRITE2(memWrite),
    .MEM_DOUT1(ir),
    .MEM_DOUT2(mem_out),
    .MEM_CLK(CLK),
    .MEM_DIN2(rs2),
    .MEM_SIZE(ir[13:12]),
    .IO_WR(IOBUS_WR),
    .IO_IN(IOBUS_IN),
    .MEM_SIGN(ir[14]));

    IMMEDGen IMM_GEN(
    .ir(ir),
    .Itype(Itype),
    .Utype(Utype),
    .Stype(Stype),
    .Btype(Btype),
    .Jtype(Jtype));

    Mux4_1 REG_RD(
    .ZERO(pc_4),
    .ONE(CSRreg),                        //---
    .TWO(mem_out),                        //---
    .THREE(alu_out),                      //---
    .SEL(rf_wr_sel),
    .MUX_OUT(rd));

    RegisterFile REG_FILE(
    .ADR1(rs1_addr),
    .ADR2(rs2_addr),
    .WA(rd_addr),
    .EN(regWrite),
    .WD(rd),
    .RS1(rs1),
    .RS2(rs2),
    .CLK(CLK));

    Mux4_1 ALU_ARG1(
    .ZERO(rs1),
    .ONE(Utype),                           //---
    .TWO(32'b0),                           //---
    .THREE(32'b0),                         //---
    .SEL({1'b0,alu_srcA}),
    .MUX_OUT(alu_arg1));

    Mux4_1 ALU_ARG2(
    .ZERO(rs2),
    .ONE(Itype),                           //---
    .TWO(Stype),                           //---
    .THREE(pc_out),                         //---
    .SEL(alu_srcB),
    .MUX_OUT(alu_arg2));

   ALU OTTER_ALU(
   .A(alu_arg1),
   .B(alu_arg2),
   .ALU_FUN(alu_fun),
   .ALU_OUT(alu_out));
//--------------------------------
//------COMBINATIONAL LOGIC-------
    assign pc_4 = pc_out + 4;
    assign rd_addr = ir[11:7];
    assign rs1_addr = ir[19:15];
    assign rs2_addr = ir[24:20];
    assign opcode = ir[6:0];
    assign func3 = ir[14:12];
    assign func7 = ir[31:25];
    assign jalr = rs1 + Itype;
    assign jal = pc_out + Jtype;
    assign branch = pc_out + Btype;
    assign br_eq = (rs1 == rs2) ? 1:0;
    assign br_lt = ($signed(rs1) < $signed(rs2)) ? 1:0;
    assign br_ltu = (rs1 < rs2) ? 1:0;
    assign IOBUS_OUT = rs2;
    assign CSRreg = 32'b0;
    assign IOBUS_ADDR = alu_out;
//--------------------------------
Endmodule
```

```
`timescale 1ns / 1ps
```

```
module IMMEDGen(
    input [31:0] ir,
    output logic [31:0] Itype, Utype, Stype, Jtype, Btype
    );
    always_comb
    begin
    Utype = { ir[31:12], 12'b000000000000 };
    Itype = { {21{ir[31]}}, ir[30:20] };
    Stype = { {21{ir[31]}}, ir[30:25], ir[11:8], ir[7]};
    Btype = { {19{ir[31]}}, ir[7], ir[30:25], ir[11:8], 1'b0};
    Jtype = { {12{ir[31]}}, ir[19:12], ir[20], ir[30:25], ir[24:21], 1'b0};
    end

endmodule
```

Sim File

```
`timescale 1ns / 1ps

module OTTER_TB;
    reg CLK;
    reg BTNL;
    reg BTNC;
    reg [15:0] SWITCHES;
    wire [15:0] LEDS;
    wire [7:0] CATHODES;
    wire [3:0] ANODES;

    OTTER_Wrapper DUT(.*);

    initial begin CLK = 1'b0; forever #10 CLK=~CLK; end

    initial begin
        BTNC = 1'b1;
        BTNL = 1'b0;
        SWITCHES = 16'd9;
            repeat(20)@(posedge CLK);
            BTNC = 1'b0;
            repeat(80)@(posedge CLK);
        SWITCHES = 16'd2;
        repeat(80)@(posedge CLK);
        SWITCHES = 16'd6;
        repeat(80)@(posedge CLK);
        SWITCHES = 16'd13;
        repeat(80)@(posedge CLK);
            $finish;
    end
endmodule
```