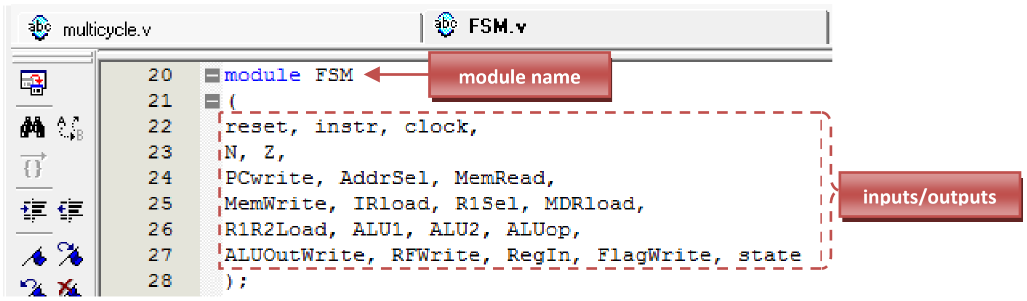
Multi-cycle Processor Modification

Introduction

Modification of the multi-cycle processor deals with changing the datapath and control. In this design, the datapath is implemented in '*multicycle.v*' (*Verilog implementation),* and in '*multicycle.bdf*' (*schematic implementation*), and the control is implemented in '*FSM.v*'. When implementing the design, modifying the control first is suggested.

Review of Verilog

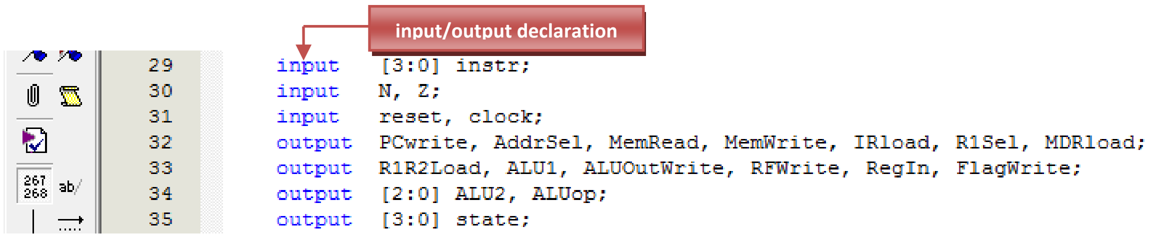
Verilog code consists of three major parts: the header, the module declaration, and the body. Module declaration consists of the keyword '*module*' and its module name, and all inputs and outputs appear in brackets. See Figure 1.1.



```
module FSM
(
reset, instr, clock,
N, Z,
PCwrite, AddrSel, MemRead,
MemWrite, IRload, R1Sel, MDRload,
R1R2Load, ALU1, ALU2, ALUop,
ALUOutWrite, RFWrite, RegIn, FlagWrite, state
);
```

Figure 1.1 – module declaration with the module name and its inputs and outputs

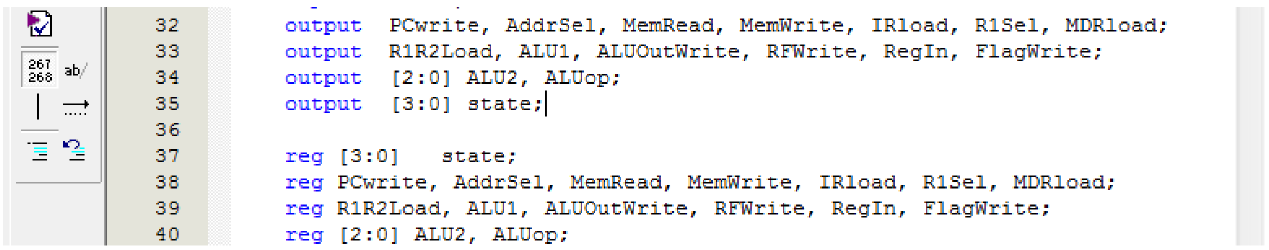
To add an input or output in Verilog code, simply add the name inside the brackets of the module declaration. Then, add a line indicating whether it is an input or output, together with its size (if the signal is more than one bit in size).



```
input   [3:0] instr;
input   N, Z;
input   reset, clock;
output  PCwrite, AddrSel, MemRead, MemWrite, IRload, R1Sel, MDRload;
output  R1R2Load, ALU1, ALUOutWrite, RFWrite, RegIn, FlagWrite;
output  [2:0] ALU2, ALUop;
output  [3:0] state;
```

Figure 1.2 – declaration of inputs and outputs

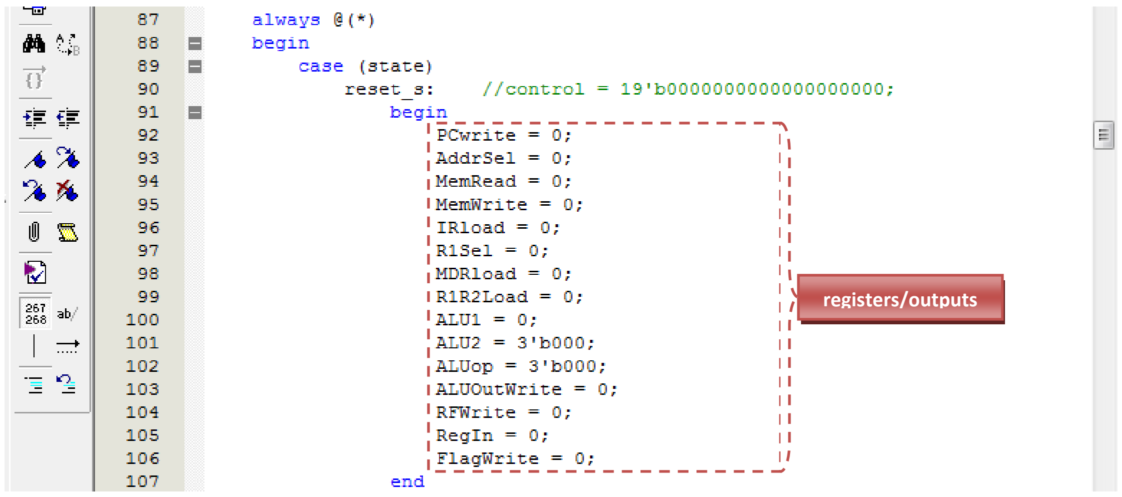
If any output requires its value to be stored, or requires to be used in an *always* block, then a '*reg'* declaration is required. See Figure 1.3 and 1.4.



```
output  PCwrite, AddrSel, MemRead, MemWrite, IRload, R1Sel, MDRload;
output  R1R2Load, ALU1, ALUOutWrite, RFWrite, RegIn, FlagWrite;
output  [2:0] ALU2, ALUop;
output  [3:0] state;

reg [3:0]   state;
reg PCwrite, AddrSel, MemRead, MemWrite, IRload, R1Sel, MDRload;
reg R1R2Load, ALU1, ALUOutWrite, RFWrite, RegIn, FlagWrite;
reg [2:0] ALU2, ALUop;
```

Figure 1.3 – outputs and registers declarations

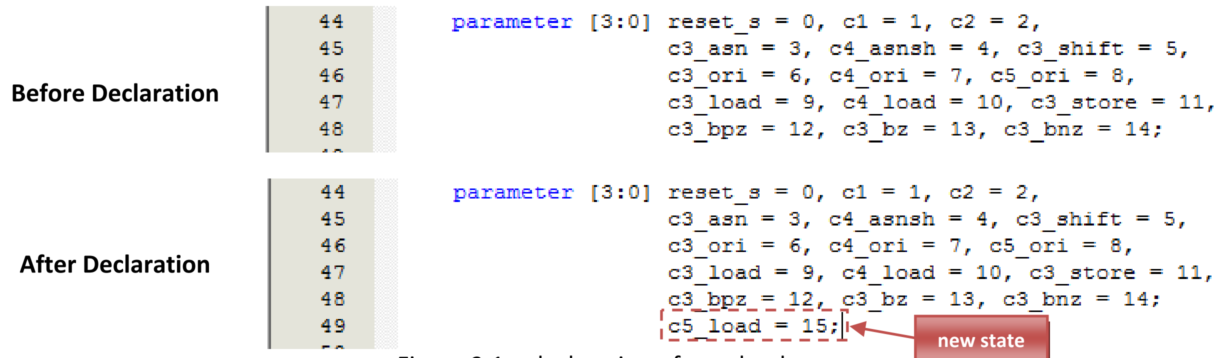


```
always @(*)
begin
    case (state)
        reset_s:    //control = 19'b0000000000000000000;
            begin
                PCwrite = 0;
                AddrSel = 0;
                MemRead = 0;
                MemWrite = 0;
                IRload = 0;
                R1Sel = 0;
                MDRload = 0;
                R1R2Load = 0;
                ALU1 = 0;
                ALU2 = 3'b000;
                ALUop = 3'b000;
                ALUOutWrite = 0;
                RFWrite = 0;
                RegIn = 0;
                FlagWrite = 0;
            end
```

Figure 1.4 – outputs which require their values to be stored

Modifying the Control

The Verilog implementation of the control is called '*FSM.v*'. If new signals are to be introduced, refer to "Review of Verilog". If new states are required, add the new states to the parameter declarations. Each state parameter contains the state name and a numerical value. Any numerical value can be assigned to a state, as long as it is unique. The width of the parameter values may have to be increased from the default size of 4 bits.



Before Declaration

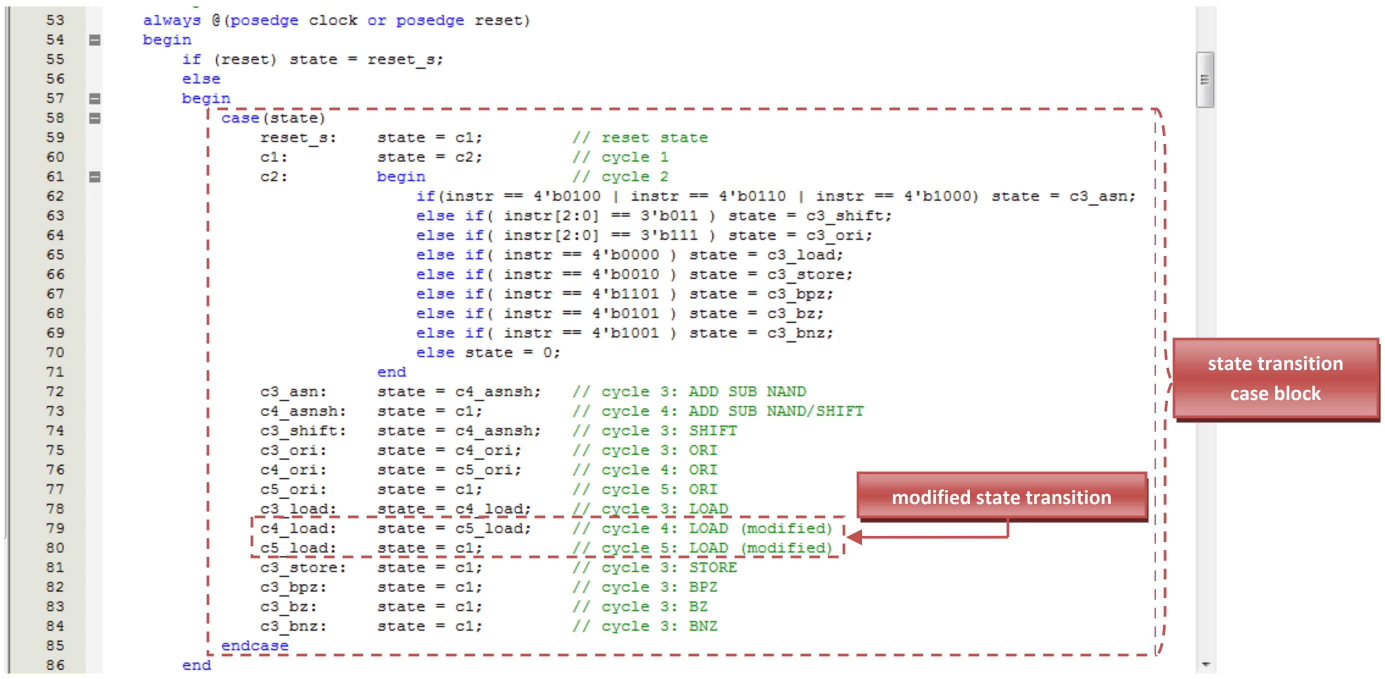
```
parameter [3:0] reset_s = 0, c1 = 1, c2 = 2,
                c3_asn = 3, c4_asnsh = 4, c3_shift = 5,
                c3_ori = 6, c4_ori = 7, c5_ori = 8,
                c3_load = 9, c4_load = 10, c3_store = 11,
                c3_bpz = 12, c3_bz = 13, c3_bnz = 14;
```

After Declaration

```
parameter [3:0] reset_s = 0, c1 = 1, c2 = 2,
                c3_asn = 3, c4_asnsh = 4, c3_shift = 5,
                c3_ori = 6, c4_ori = 7, c5_ori = 8,
                c3_load = 9, c4_load = 10, c3_store = 11,
                c3_bpz = 12, c3_bz = 13, c3_bnz = 14;
                c5_load = 15;
```

Figure 2.1 – declaration of new load state

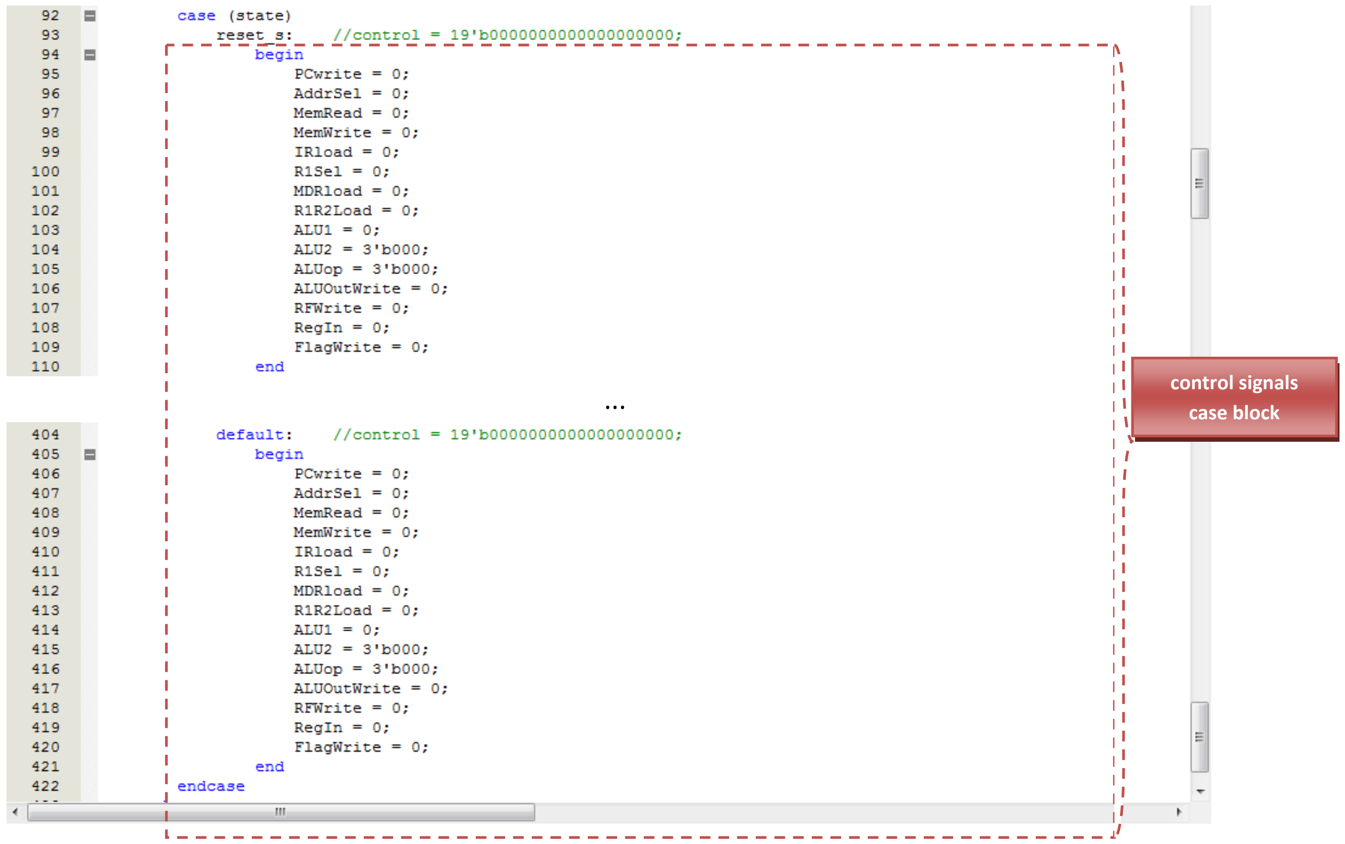
Once a new state has been declared, the state transition *case* block must be modified. The name of each new state needs to be included as a new case of the state transition *case* block. See Figure 2.2.



```
always @(posedge clock or posedge reset)
begin
    if (reset) state = reset_s;
    else
    begin
        case(state)
            reset_s:    state = c1;         // reset state
            c1:         state = c2;         // cycle 1
            c2:         begin               // cycle 2
                            if(instr == 4'b0100 | instr == 4'b0110 | instr == 4'b1000) state = c3_asn;
                            else if( instr[2:0] == 3'b011 ) state = c3_shift;
                            else if( instr[2:0] == 3'b111 ) state = c3_ori;
                            else if( instr == 4'b0000 ) state = c3_load;
                            else if( instr == 4'b0010 ) state = c3_store;
                            else if( instr == 4'b1101 ) state = c3_bpz;
                            else if( instr == 4'b0101 ) state = c3_bz;
                            else if( instr == 4'b1001 ) state = c3_bnz;
                            else state = 0;
                        end
            c3_asn:     state = c4_asnsh;   // cycle 3: ADD SUB NAND
            c4_asnsh:   state = c1;         // cycle 4: ADD SUB NAND/SHIFT
            c3_shift:   state = c4_asnsh;   // cycle 3: SHIFT
            c3_ori:     state = c4_ori;     // cycle 3: ORI
            c4_ori:     state = c5_ori;     // cycle 4: ORI
            c5_ori:     state = c1;         // cycle 5: ORI
            c3_load:    state = c4_load;    // cycle 3: LOAD
            c4_load:    state = c5_load;    // cycle 4: LOAD (modified)
            c5_load:    state = c1;         // cycle 5: LOAD (modified)
            c3_store:   state = c1;         // cycle 3: STORE
            c3_bpz:     state = c1;         // cycle 3: BPZ
            c3_bz:      state = c1;         // cycle 3: BZ
            c3_bnz:     state = c1;         // cycle 3: BNZ
        endcase
    end
```

Figure 2.2 – modified state transition case block with new load state introduced

After the new state has been added and the state transition case block has been modified, the control signals *case* block (in the level-sensitive *always* block) can now be modified. See Figure 2.3.



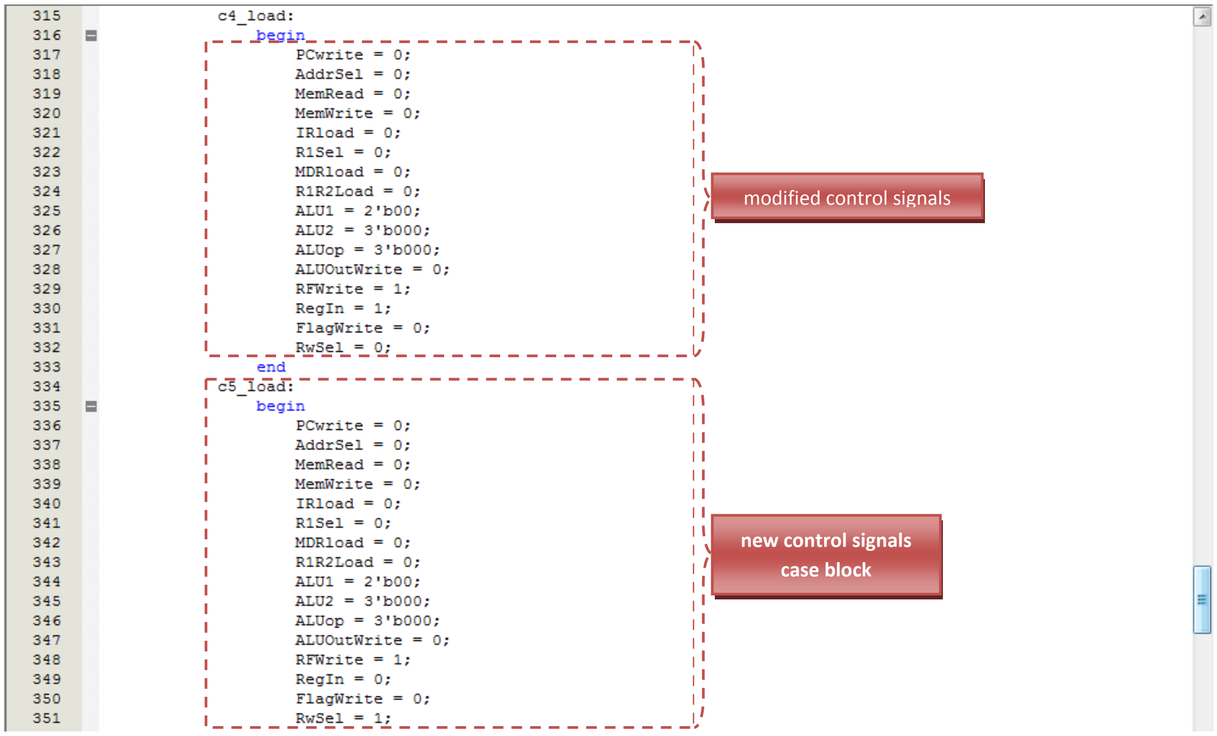
```
case (state)
    reset_s:    //control = 19'b0000000000000000000;
        begin
            PCwrite = 0;
            AddrSel = 0;
            MemRead = 0;
            MemWrite = 0;
            IRload = 0;
            R1Sel = 0;
            MDRload = 0;
            R1R2Load = 0;
            ALU1 = 0;
            ALU2 = 3'b000;
            ALUop = 3'b000;
            ALUOutWrite = 0;
            RFWrite = 0;
            RegIn = 0;
            FlagWrite = 0;
        end

                                    ...

    default:    //control = 19'b0000000000000000000;
        begin
            PCwrite = 0;
            AddrSel = 0;
            MemRead = 0;
            MemWrite = 0;
            IRload = 0;
            R1Sel = 0;
            MDRload = 0;
            R1R2Load = 0;
            ALU1 = 0;
            ALU2 = 3'b000;
            ALUop = 3'b000;
            ALUOutWrite = 0;
            RFWrite = 0;
            RegIn = 0;
            FlagWrite = 0;
        end
endcase
```

Figure 2.3 – control signals case block with output control signals

When a new state is added, the control signals for that state need to be added in the control signals case block. When making modifications to existing states, changes can be made directly to each control signal. See Figure 2.4.



```
            c4_load:
                begin
                    PCwrite = 0;
                    AddrSel = 0;
                    MemRead = 0;
                    MemWrite = 0;
                    IRload = 0;
                    R1Sel = 0;
                    MDRload = 0;
                    R1R2Load = 0;
                    ALU1 = 2'b00;
                    ALU2 = 3'b000;
                    ALUop = 3'b000;
                    ALUOutWrite = 0;
                    RFWrite = 1;
                    RegIn = 1;
                    FlagWrite = 0;
                    RwSel = 0;
                end
            c5_load:
                begin
                    PCwrite = 0;
                    AddrSel = 0;
                    MemRead = 0;
                    MemWrite = 0;
                    IRload = 0;
                    R1Sel = 0;
                    MDRload = 0;
                    R1R2Load = 0;
                    ALU1 = 2'b00;
                    ALU2 = 3'b000;
                    ALUop = 3'b000;
                    ALUOutWrite = 0;
                    RFWrite = 1;
                    RegIn = 0;
                    FlagWrite = 0;
                    RwSel = 1;
```

Figure 2.4 – modification of control signals