# System Synthesis and Modeling (CSCE 3953)

# Final Project (Step 3) Report

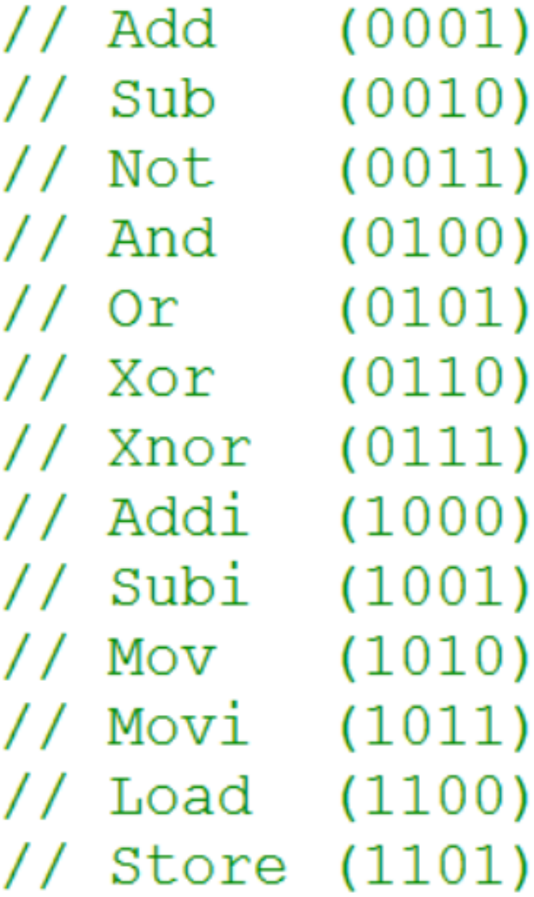
Zack Fravel

11/9/16

zpfravel@uark.edu

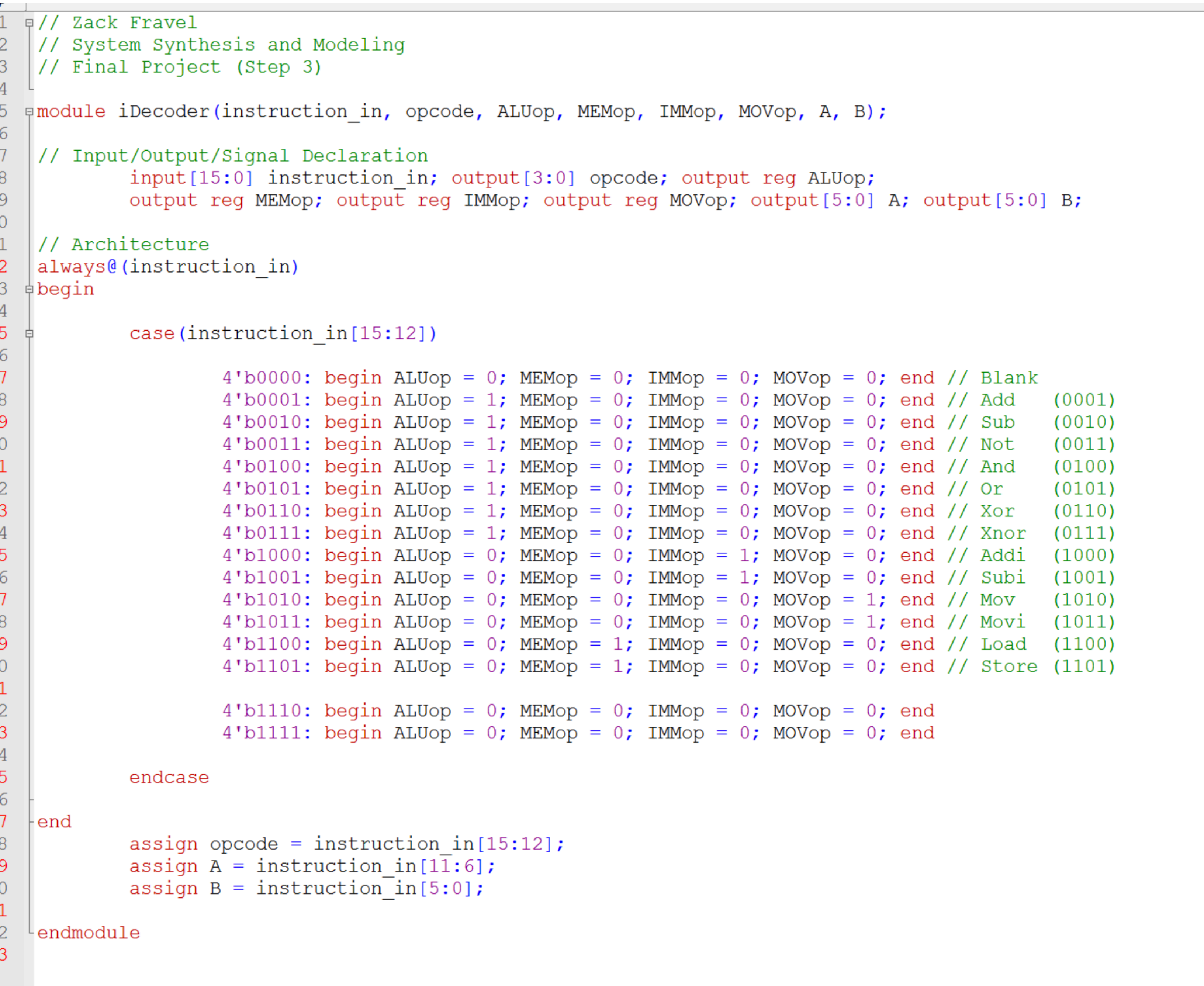
**Report**

The third step of the final project was to design an instruction decoder that takes in a 16 bit instruction from the instruction register (IR) and outputs signals necessary for controlling the FSM's that control the signal flow throughout the system depending on the instruction sent. For our system, we have thirteen instructions. The decoder, for each instruction, needs to send out the signals for the FSM's to activate as well as the two parameters for each instruction (final 12 bits). This leaves 4 bits for the opcode, which I encoded as seen in the image on this page. I decided to have my module output the instruction's opcode to the FSM module, along with an associated signal to tell the module which FSM to active, along with the two 6 bit parameters. I am planning on breaking up the operations into four different FSMs for signal path (ALUop, MEMop, IMMop, MOVop) and the signals outputting from my instruction decoder reflect the nature of the four different FSM's purpose. I also synthesized the decoder and included all the verilog, testbench, and waveform images in the report.

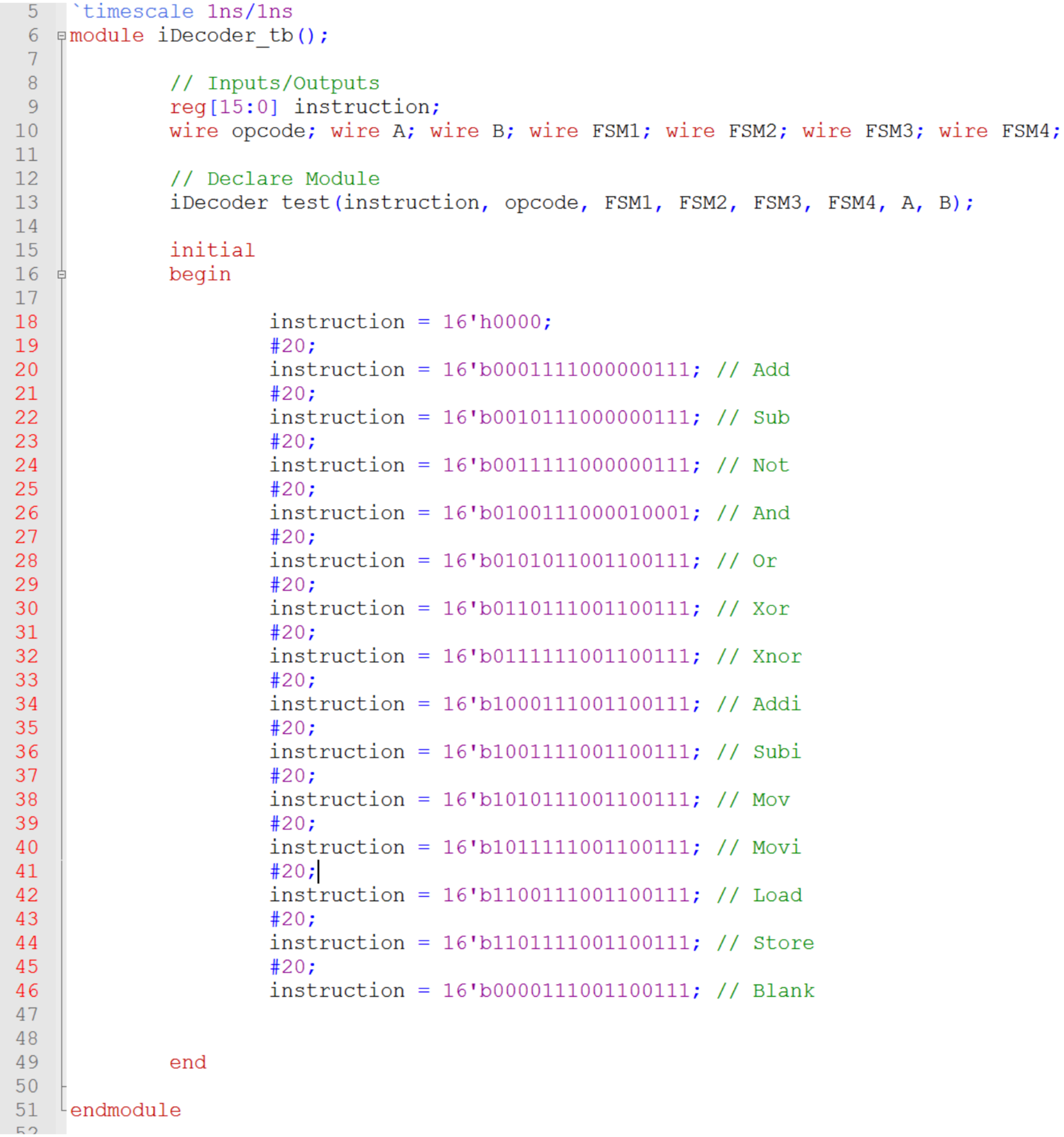


```
// Add   (0001)
// Sub   (0010)
// Not   (0011)
// And   (0100)
// Or    (0101)
// Xor   (0110)
// Xnor  (0111)
// Addi  (1000)
// Subi  (1001)
// Mov   (1010)
// Movi  (1011)
// Load  (1100)
// Store (1101)
```

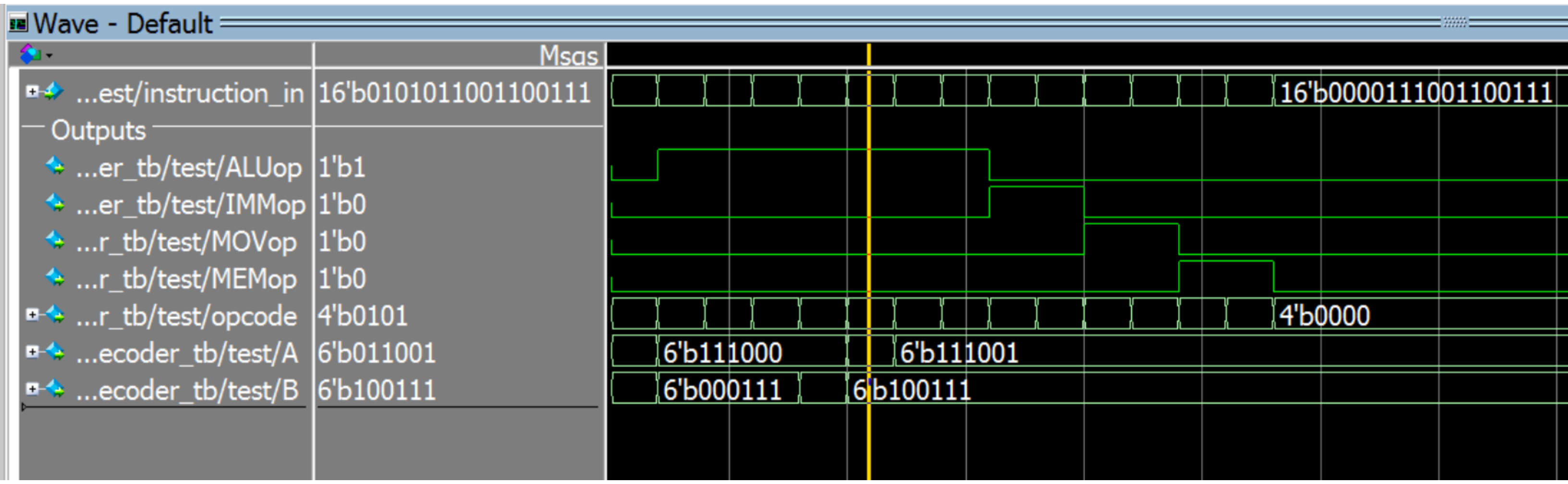
## Verilog/Testbench/Waveforms



```verilog
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 3)

module iDecoder(instruction_in, opcode, ALUop, MEMop, IMMop, MOVop, A, B);

// Input/Output/Signal Declaration
        input[15:0] instruction_in; output[3:0] opcode; output reg ALUop;
        output reg MEMop; output reg IMMop; output reg MOVop; output[5:0] A; output[5:0] B;

// Architecture
always@(instruction_in)
begin

        case(instruction_in[15:12])

                4'b0000: begin ALUop = 0; MEMop = 0; IMMop = 0; MOVop = 0; end // Blank
                4'b0001: begin ALUop = 1; MEMop = 0; IMMop = 0; MOVop = 0; end // Add   (0001)
                4'b0010: begin ALUop = 1; MEMop = 0; IMMop = 0; MOVop = 0; end // Sub   (0010)
                4'b0011: begin ALUop = 1; MEMop = 0; IMMop = 0; MOVop = 0; end // Not   (0011)
                4'b0100: begin ALUop = 1; MEMop = 0; IMMop = 0; MOVop = 0; end // And   (0100)
                4'b0101: begin ALUop = 1; MEMop = 0; IMMop = 0; MOVop = 0; end // Or    (0101)
                4'b0110: begin ALUop = 1; MEMop = 0; IMMop = 0; MOVop = 0; end // Xor   (0110)
                4'b0111: begin ALUop = 1; MEMop = 0; IMMop = 0; MOVop = 0; end // Xnor  (0111)
                4'b1000: begin ALUop = 0; MEMop = 0; IMMop = 1; MOVop = 0; end // Addi  (1000)
                4'b1001: begin ALUop = 0; MEMop = 0; IMMop = 1; MOVop = 0; end // Subi  (1001)
                4'b1010: begin ALUop = 0; MEMop = 0; IMMop = 0; MOVop = 1; end // Mov   (1010)
                4'b1011: begin ALUop = 0; MEMop = 0; IMMop = 0; MOVop = 1; end // Movi  (1011)
                4'b1100: begin ALUop = 0; MEMop = 1; IMMop = 0; MOVop = 0; end // Load  (1100)
                4'b1101: begin ALUop = 0; MEMop = 1; IMMop = 0; MOVop = 0; end // Store (1101)

                4'b1110: begin ALUop = 0; MEMop = 0; IMMop = 0; MOVop = 0; end
                4'b1111: begin ALUop = 0; MEMop = 0; IMMop = 0; MOVop = 0; end

        endcase

end
        assign opcode = instruction_in[15:12];
        assign A = instruction_in[11:6];
        assign B = instruction_in[5:0];

endmodule
```

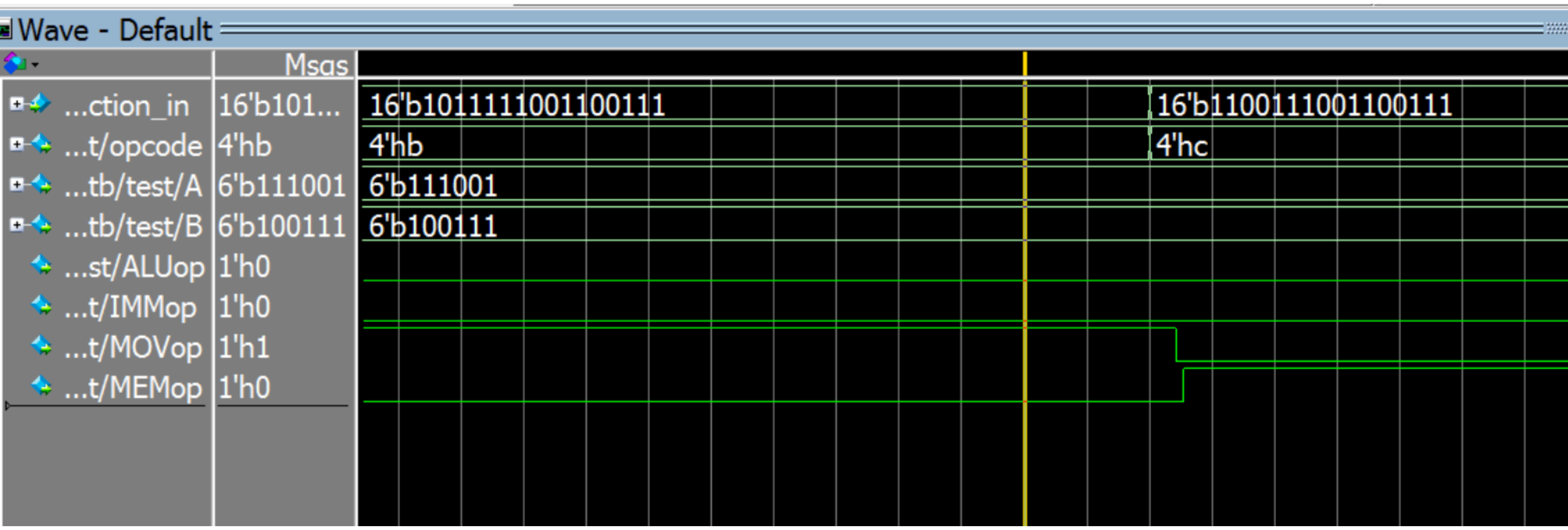


```verilog
`timescale 1ns/1ns
module iDecoder_tb();

        // Inputs/Outputs
        reg[15:0] instruction;
        wire opcode; wire A; wire B; wire FSM1; wire FSM2; wire FSM3; wire FSM4;

        // Declare Module
        iDecoder test(instruction, opcode, FSM1, FSM2, FSM3, FSM4, A, B);

        initial
        begin

                instruction = 16'h0000;
                #20;
                instruction = 16'b0001111000000111; // Add
                #20;
                instruction = 16'b0010111000000111; // Sub
                #20;
                instruction = 16'b0011111000000111; // Not
                #20;
                instruction = 16'b0100111000010001; // And
                #20;
                instruction = 16'b0101011001100111; // Or
                #20;
                instruction = 16'b0110111001100111; // Xor
                #20;
                instruction = 16'b0111111001100111; // Xnor
                #20;
                instruction = 16'b1000111001100111; // Addi
                #20;
                instruction = 16'b1001111001100111; // Subi
                #20;
                instruction = 16'b1010111001100111; // Mov
                #20;
                instruction = 16'b1011111001100111; // Movi
                #20;
                instruction = 16'b1100111001100111; // Load
                #20;
                instruction = 16'b1101111001100111; // Store
                #20;
                instruction = 16'b0000111001100111; // Blank


        end

endmodule
```



Wave - Default

```
                              Msgs
...est/instruction_in         16'b0101011001100111
Outputs
  ...er_tb/test/ALUop         1'b1
  ...er_tb/test/IMMop         1'b0
  ...r_tb/test/MOVop          1'b0
  ...r_tb/test/MEMop          1'b0
...r_tb/test/opcode           4'b0101
...ecoder_tb/test/A           6'b011001
...ecoder_tb/test/B           6'b100111
```



Wave - Default

```
                   Msgs
...ction_in        16'b101...
...t/opcode        4'hb
...tb/test/A       6'b111001
...tb/test/B       6'b100111
...st/ALUop        1'h0
...t/IMMop         1'h0
...t/MOVop         1'h1
...t/MEMop         1'h0
```