Mark Mitri 02/19/2022
ECE201 Project 1

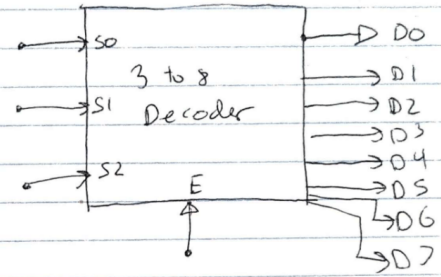
Part 1: Gate-level & Behavioral 3-to-8 decoder

Design Diagram:

Mark Mitri 2/10/22
ECE 201 Project 1

1. 1) Behavior level of 3-8 decoder (one verilog module) (one-hot)
2) Gate level of 3-8 decoder (using basic modules)

3-8 decoder



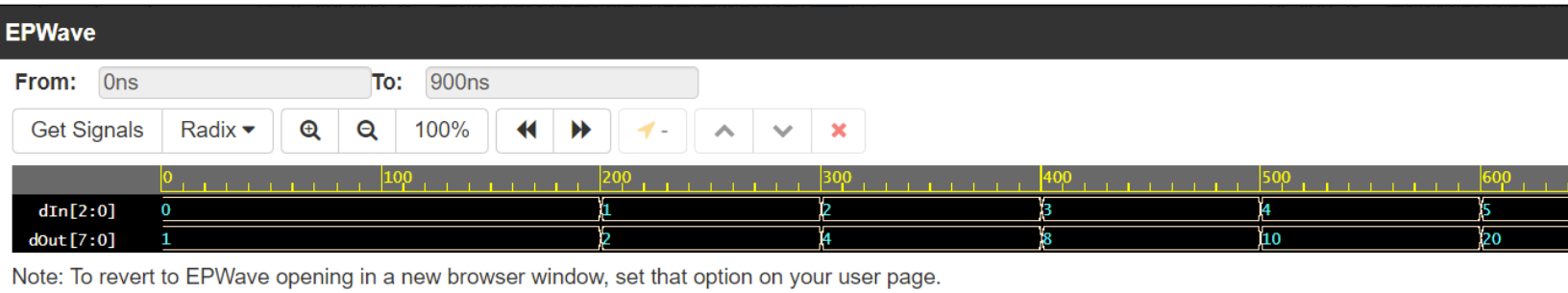
S0	S1	S2	E	D0	D1	D2	D3	D4	D5	D6	D7
x	x	x	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

$D0 = S0\,S1\,S2$ $D4 = \bar{S0}\,S1\,S2$
$D1 = S0\,S1\,\bar{S2}$ $D5 = \bar{S0}\,S1\,S1\,\bar{S2}$
$D2 = S0\,\bar{S1}\,S2$ $D6 = \bar{S0}\,\bar{S1}\,S2$
$D3 = S0\,\bar{S1}\,\bar{S2}$ $D7 = \bar{S0}\,\bar{S1}\,\bar{S2}$

3-to-8 Decoder Gate level

```
// Mark Mitri
// ECE201
// Project 1
// 3 to 8 Decoder gate level

module threeto8decoder_gatelevel(input s0,input s1,input s2, output [7:0] dOut);

  assign dOut[0]=and(~s0,~s1,~s2);
  assign dOut[1]=and(~s0,~s1,s2);
  assign dOut[2]=and(~s0,s1,~s2);
  assign dOut[3]=and(~s0,s1,s2);
  assign dOut[4]=and(s0,~s1,s2);
  assign dOut[5]=and(s0,~s1,s2);
  assign dOut[6]=and(s0,s1,~s2);
  assign dOut[7]=and(s0,s1,s2);

endmodule


// Mark Mitri
// ECE201 Project 1
// 3 to 8 decoder gate level

module threeto8decoder_gatelevel_testbench;
  reg s0;
  reg s1;
  reg s2;
  wire [7:0] dOut;

  threeto8decoder_gatelevel uut ( .s0(s0),.s1(s1),.s2(s2),.dOut(dOut) );

  initial begin
    s0=1'b0;s1=1'b0;s2=1'b0;
    #100 s0=1’b0;s1=1’b0;s2=1’b0;
    #100 s0=1’b0;s1=1’b0;s2=1’b1;
    #100 s0=1’b0;s1=1’b1;s2=1’b0;
    #100 s0=1’b0;s1=1’b1;s2=1’b1;
    #100 s0=1’b1;s1=1’b0;s2=1’b0;
    #100 s0=1’b1;s1=1’b0;s2=1’b1;
    #100 s0=1’b1;s1=1’b1;s2=1’b0;
    #100 s0=1’b1;s1=1’b1;s2=1’b1;
```

```
 end

 initial begin
   $monitor ("%t | s0 = %d| s1 = %d| s2 = %d| dOut = %d", $time, s0, s1, s2, dOut);
   $dumpfile("dump.vcd");
   $dumpvars(1);
         end
endmodule
```

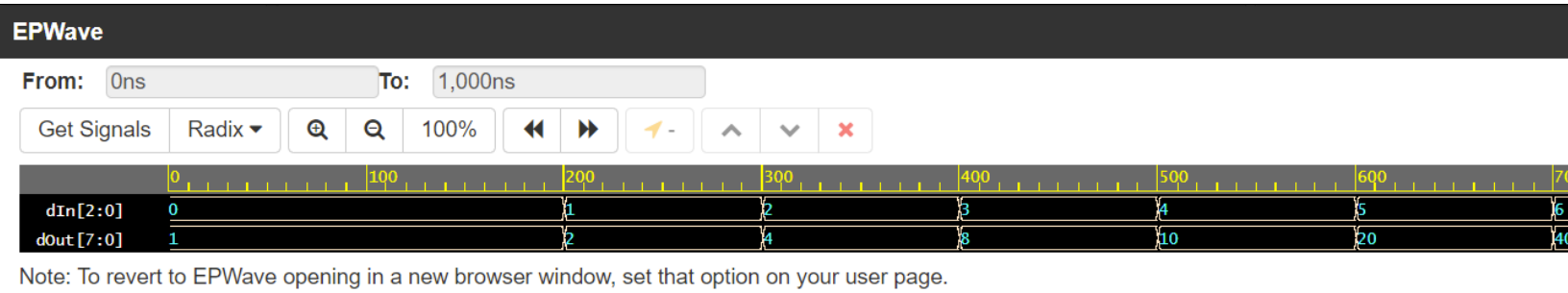


Note: To revert to EPWave opening in a new browser window, set that option on your user page.

3-to-8 Decoder Behavioral

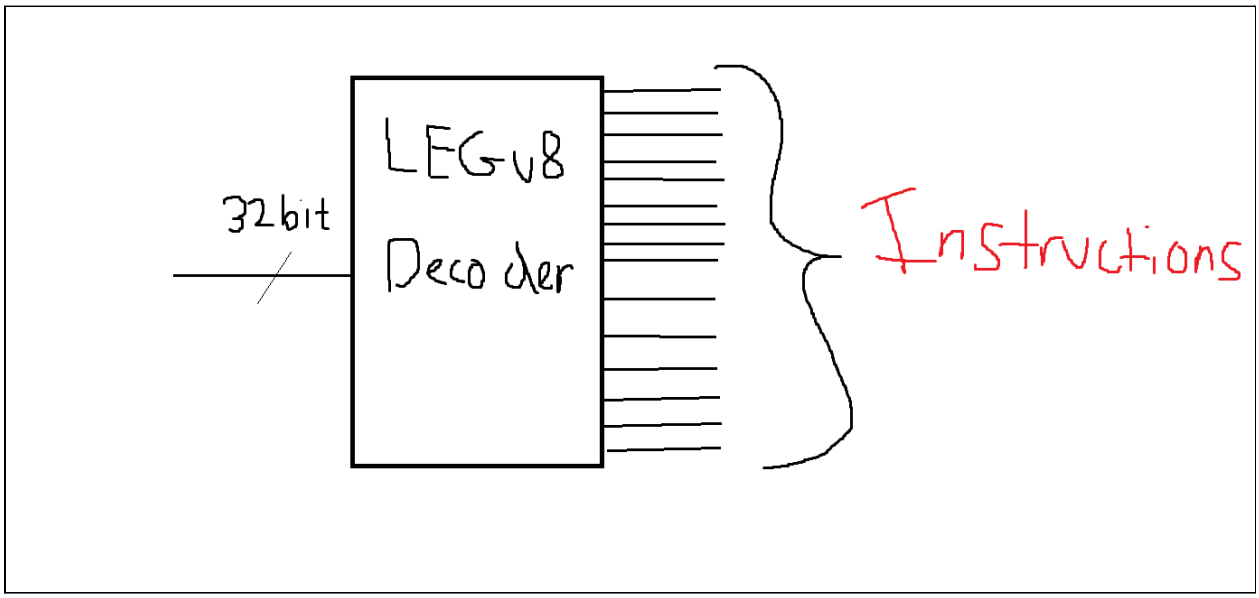
```
// Mark Mitri
// ECE201 Project 1
// Part 1
// 3-8 decoder behavioral

module threeTo8decoder_behavioral(dIn,dOut);
         input [2:0] dIn;
   output reg [7:0] dOut;

always @(dIn)
   case (dIn)
      3'b000 : dOut = 8'b00000001;
      3'b001 : dOut = 8'b00000010;
      3'b010 : dOut = 8'b00000100;
      3'b011 : dOut = 8'b00001000;
      3'b100 : dOut = 8'b00010000;
      3'b101 : dOut = 8'b00100000;
      3'b110 : dOut = 8'b01000000;
```

```
      3'b111 : dOut = 8'b10000000;
      default : dOut = 8'b00000000;
   endcase
endmodule


// Mark Mitri
// ECE201 Project 1
// Part 1
// 3-8 decorder behavioral
// testbench

module threeTo8decoder_behavioral_tb;
          reg [2:0] dIn;
          wire [7:0] dOut;

          threeTo8decoder_behavioral uut (.dIn(dIn), .dOut(dOut));

   initial begin
     $dumpfile("dump.vcd");
     $dumpvars(1);
          dIn = 0; #100;
       dIn = 3'b000; #100;
       dIn = 3'b001; #100;
       dIn = 3'b010; #100;
       dIn = 3'b011; #100;
       dIn = 3'b100; #100;
       dIn = 3'b101; #100;
       dIn = 3'b110; #100;
       dIn = 3'b111; #100;
       #100 $stop;
   end

          initial begin
     $monitor ("%t | dIn = %d| dOut = %d", $time, dIn, dOut);
     $dumpfile("dump.vcd");
     $dumpvars(1);
          end
endmodule
```



Part 2: LEGv8 Decoder

Design Diagram



LEGv8 Decoder Code

```
// Mark Mitri
// ECE 201 Project 1
// Part 2
// 32bit LEGv8 Decoder
```

```
/*
Outputs: AND, ADD, ORR, ADDS, EOR, SUB, LSR,
                  LSL, BR, ANDS, SUBS, ORRI, EORI, ADDI
       ANDI, ADDIS, SUBI, SUBIS, ANDIS, B, BL

* From Computer Organization and Design Textbook: Fig. 2.20
Instruction    Opcode          Opcode Size   Instruction Format
AND            10001010000   11              R-Format
ADD            10001011000   11            R-Format
ORR            10101010000          11                    R-Format
ADDS           10101011000          11                    R-Format
EOR            11101010000          11                    R-Format
SUB            11001011000          11                    R-Format
LSR            11010011010          11                    R-Format
LSL            11010011011          11                    R-Format
BR             11010110000          11                    R-Format
ANDS           11101010000          11                    R-Format
SUBS           11101011000   11                  R-Format
ORRI           1011001000   10                   I-Format
EORI           1101001000     10                 I-Format
ADDI           1001000100   10                   I-Format
ANDI           1001001000   10                   I-Format
ADDIS 1011000100     10                  I-Format
SUBI           1101000100   10                   I-Format
SUBIS          1111000100     10                 I-Format
ANDIS 1111001000   10                  I-Format
B              000101         06                   B-Format
BL             100101         06                   B-Format
*/

module legV8decoder(bus, instruction);
  input [31:0] bus;
  input opcodeSix = [31:25] bus;
  input opcodeEleven = [31:20] bus;
  output reg [7:0] instruction;

  if(//opcode length == 6)
    // always block for opcode length 6 instructions
    always @(bus)
      begin
                  case(bus)
        6'b000101:instruction = 6'b000001; // Instruction = B
        6'b100101:instruction = 6'b000010; // Instruction = BL
```

```
        default:instruction = 6'b000000;
      endcase
   else if(//opcode length == 10)
    begin
    // always block for opcode length 10 instructions
    always @(bus) begin
      case(bus)
        10'b1011001000:instruction = 6'b000011; // Instruction = ORRI
        10'b1101001000:instruction = 6'b000100; // Instruction = EORI
        10'b1001000100:instruction = 6'b000101; // Instruction = ADDI
        10'b1001001000:instruction = 6'b000110; // Instruction = ANDI
        10'b1011000100:instruction = 6'b000111; // Instruction = ADDIS
        10'b1101000100:instruction = 6'b001000; // Instruction = SUBI
        10'b1111000100:instruction = 6'b001001; // Instruction = SUBIS
        10'b1111001000:instruction = 6'b001010; // Instruction = ANDIS
        default:instruction = 6'b000000;
      endcase
   else if(//opcode length == 11)
    begin
    // always block for opcode length 11 instructions
    always @(bus) begin
      case(bus)
        11'b10001010000:instruction = 6'b001011; // Instruction = AND
        11'b10001011000:instruction = 6'b001100; // Instruction = ADD
        11'b10101010000:instruction = 6'b001101; // Instruction = ORR
        11'b10101011000:instruction = 6'b001110; // Instruction = ADDS
        11'b11101010000:instruction = 6'b001111; // Instruction = EOR
        11'b11001011000:instruction = 6'b010000; // Instruction = SUB
        11'b11010011010:instruction = 6'b010001; // Instruction = LSR
        11'b11010011011:instruction = 6'b010010; // Instruction = LSL
        11'b11010110000:instruction = 6'b010011; // Instruction = BR
        11'b11101010000:instruction = 6'b010100; // Instruction = ANDS
        11'b11101011000:instruction = 6'b010101; // Instruction = SUBS
        default: instruction = 6'b000000;
      endcase
    end else
endmodule


// Mark Mitri
// ECE 201 Project 1
// Part 2
// 32bit LEGv8 Decoder
// Testbench
```

```
module legV8decoder;
  reg [31:0] bus;
  reg [31:25] opcodeSix;
  reg [31:20] opcodeEleven;
  wire instruction;

  legV8decoder uut(.bus(bus), .opcodeSix(opcodeSix), .opcodeEleven(opcodeEleven),
.instruction(instruction));

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);



  end
endmodule
```

The LEGv8 Decoder was the hardest part for me. I left the parts of the code that I didn't understand how to accomplish and I also was not sure of how to do the testbench.