Combinational-Circuit Building Blocks

- Objectives

  This chapter introduces several logical networks that are useful as building blocks for larger systems. The objectives of this section are to:
  - Discuss naming conventions for digital signals.
  - Define and demonstrate the operation of decoders, encoders and multiplexers, including their use as universal gate networks.
  - Demonstrate how Verilog can be used to model the behavior of these networks.

- Reading Assignment
  - Sections 2.8-10 of the text.

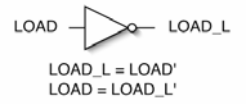
Signal Names

- Choose signal names to:
  - Indicate an action that is controlled (RESET, LOAD)
  - A condition that is detected (READY, ERROR)
  - The type of data carried on a bus (DATA, ADDRESS)
- The Active Level for a signal is the level (high or low) that causes the indicated action to occur. It is the level that causes the signal to be asserted.
  - Active Level notation:

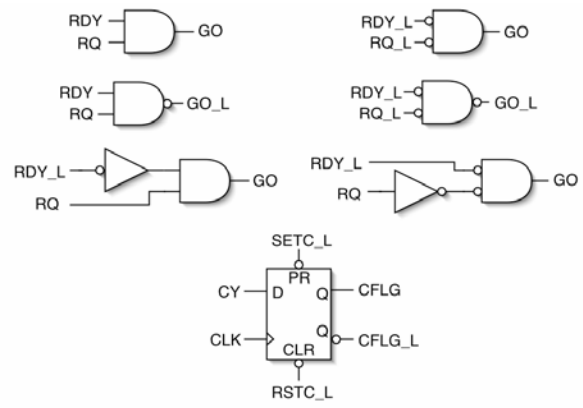
High	Low
RESET+	RESET-
RESET	RESET*
RESET	RESET/
RESET	/RESET
RESET	RESET_L

The last one is used in the text.

- Signal Names and Equations
  - The active level symbols (/, * or -) are just other symbols in the name, not negation operators.
  - Only signal names should appear on the left side of an equation
- Signal names can be combined with logical operators to form the right side of an equation.

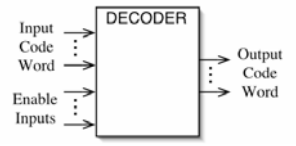


- Active Levels for Pins



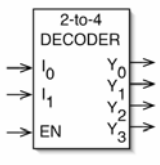
Decoders

- Decoders are used to map code words from one code into code words from another code.
- Decoders usually have enable inputs in addition to the code word inputs.
  - When one or more of the enable inputs are deasserted, the outputs all take on a default value.
    - The default is usually 0 if the outputs are asserted high and 1 if they are asserted low.
    - The default could also be the high impedance state for tri-state outputs.
  - When the enable inputs are all asserted, the decoder translates an input code into an output code.



Binary Decoders

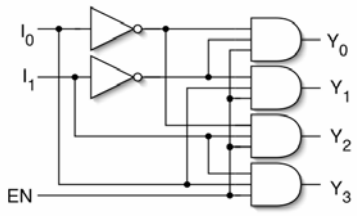
- The most common decoders are binary decoders that translate the binary number code into a one-hot or 1-out-of-n code.
- If there are *n* input terminals, then a complete binary decoder has 2^n output terminals.
- There may be less than a complete decoding (e.g., decimal numbers)
- Example: 2-to-4 binary decoder.



SYMBOL

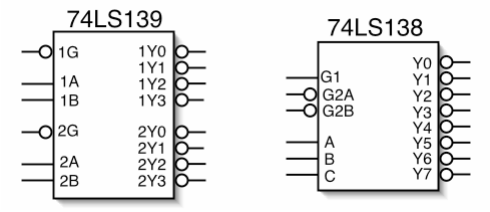
EN	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

TRUTH TABLE

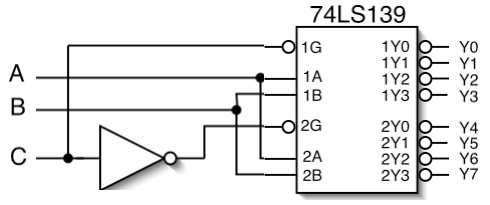


LOGIC DIAGRAM

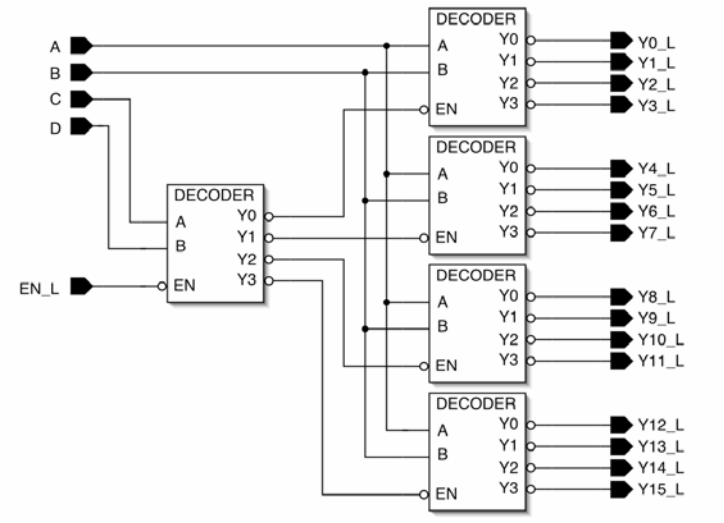
- Examples of decoder chips



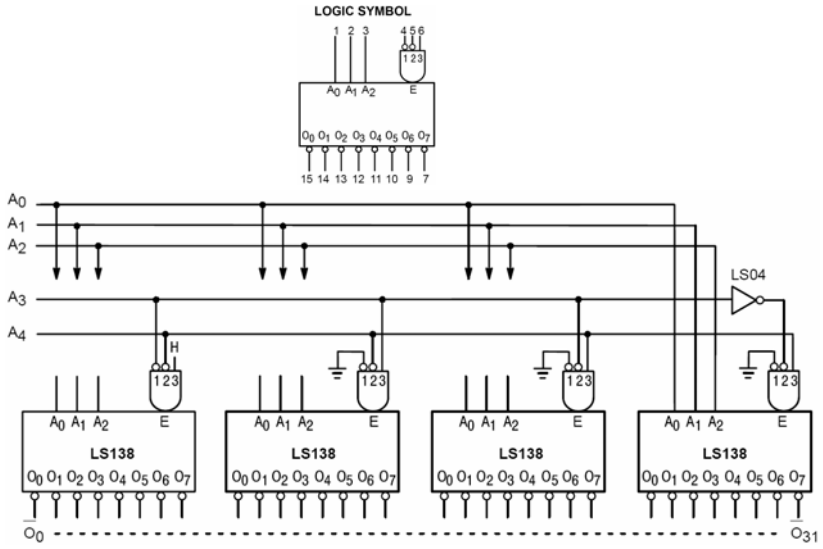
- Exercise: How could you use the 74LS139 to implement a 3-to-8 decoder?



- Cascading Binary Decoders



☐ 74LS138



Verilog Descriptions of Decoders

☐ The Verilog Case Statement

- This statement can be used within always blocks to select one of several alternatives.
- General form:

```
Case (expression)
      alternative 1: statement;
      alternative 2: statement;
            .
            .
            .
      alternative 3: statement;
      [default: statement];
endcase
```

◆ The statement associated with the first alternative to match the value of the expression is executed and the rest skipped.

◆ If the expression does not match any alternative and there is a default statement, it is executed.

- If an output signal has the same value for several alternatives, the alternative conditions can be combined (separated by commas) to give one alternative that is selected when any one of the alternative conditions is true.
  - For example the two alternatives

    3'b000: F = 1;

    3'b001: F = 1;

    are equivalent to:

    3b'000, 3'b001: F = 1;

- Verilog descriptions of a 2-to-4 decoder:

```
module dec2to4(W, Y, En);
    input [1:0] W;
    input En;
    output [0:3] Y;
    reg [0:3] Y;

    always @(*)
        case ({en,W})
            3'b100: Y = 4'b1000;
            3'b101: Y = 4'b0100;
            3'b110: Y = 4'b0010;
            3'b111: Y = 4'b0001;
            default: Y = 4'b0000;
        endcase

endmodule
```

Verilog if-then-else statements:

- This is a conventional branch statement with the following syntax:

```
if (conditional_expression)
    statement;
```

```
if (conditional_expression)
    statement;
else
    statement;
```

- A chain of sequential tests can be constructed as follows:

```
if (conditional_expression)
    statement;
else if
    statement;
else if
    statement
else
    statement;
```


Alternate description of dec2to4:

```
module dec2to4(W, Y, En);
    input [1:0] W;
    input En;
    output [0:3] Y;
    reg [0:3] Y;

    always @(*)
    begin
    if (En==0)
        Y = 4'b0000;
    else
        case (W)
            0: Y = 4'b1000;
            1: Y = 4'b0100;
            2: Y = 4'b0010;
            3: Y = 4'b0001;
        endcase
    end

endmodule
```

- Verilog description of a 4-to-16 decoder constructed as a tree of 2-to-4 decoders:

```
module dec4to16 (W, Y, En);
    input [3:0] W;
    input En;
    output [0:15] Y;
    wire [0:3] M;

    dec2to4 Dec1 (W[3:2], M[0:3], En);
    dec2to4 Dec2 (W[1:0], Y[0:3], M[0]);
    dec2to4 Dec3 (W[1:0], Y[4:7], M[1]);
    dec2to4 Dec4 (W[1:0], Y[8:11], M[2]);
    dec2to4 Dec5 (W[1:0], Y[12:15], M[3]);

endmodule
```


- The Verilog "for loop"
  - Syntax:

    for (initial_index; terminal_index; increment) statement;
  - Example: A 2-to-4 binary decoder specified using the for loop.

```
module dec2to4 (W, Y, En);
    input [1:0] W;
    input En;
    output [0:3] Y;
    reg [0:3] Y;
    integer k;

    always @(*)
        for (k = 0; k <= 3; k = k+1)
            if ((W==k) && (En==1))
                Y[k] = 1;
            else
                Y[k] = 0;
endmodule
```

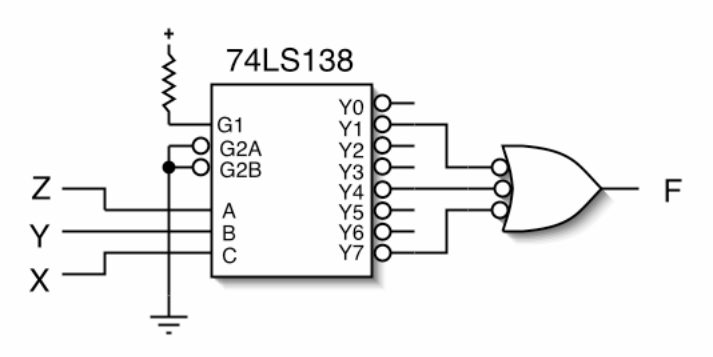
- Conditions for combinational behavior in case statements
  - Case statements can violate the condition that all outputs are assigned in every control path.
  - This can easily happen if a default condition is not specified.
    - In this case, synthesis of the case statement will lead to a sequential circuit.
  - To be safe, always use a default statement unless all possible conditions are included as alternatives.
- Specifying don't care conditions
  - The signal value x can be used to specify a don't care.
    - If an output signal is assigned the value x, then the synthesis tools will treat this as a don't care condition. That is, the synthesis tool if free to assign either 0 or 1 to the signal.
  - This is useful in situations where you can collect all the input conditions that are know to never occur into the default alternative of a case statement.

- Example: BCD decoder
  - This circuit has four binary inputs and ten binary outputs. The *i*th output is asserted if the binary inputs are the binary number *i*, where $0 \leq i \leq 9$. The inputs will never be a number greater than 9 (or if they are, we don't care what the output is).

```
module (X, Y);
        input [3:0] X; output [0:9] Y; reg [0:9] Y;
        always @(*) begin
                Y = 0;
                case (X)
                        0: Y[0] = 1;
                        1: Y[1] = 1;
                        2: Y[2] = 1;
                        3: Y[3] = 1;
                        4: Y[4] = 1;
                        5: Y[5] = 1;
                        6: Y[6] = 1;
                        7: Y[7] = 1;
                        8: Y[8] = 1;
                        9: Y[9] = 1;
                        default: Y = 'bx;
                endcase
        end
endmodule
```

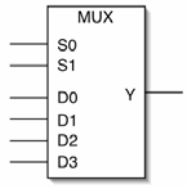
- Binary Decoders as Minterm Generators

Realize $F = \Sigma_{X,Y,Z}(1, 4, 7)$ with a decoder:



Multiplexers (MUX)

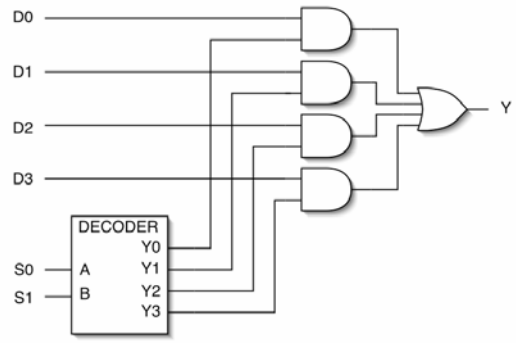
- Also called Data Selectors
- Example: 4-to-1 MUX



Symbol

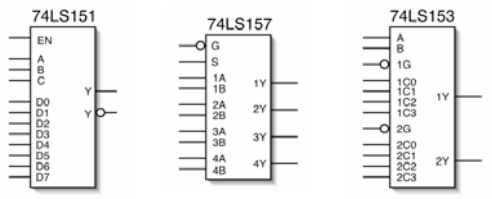
S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Truth Table

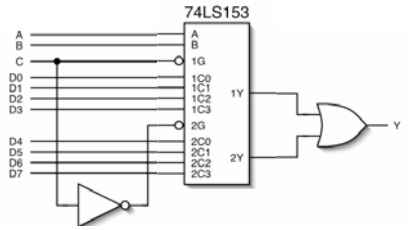


Logic Diagram

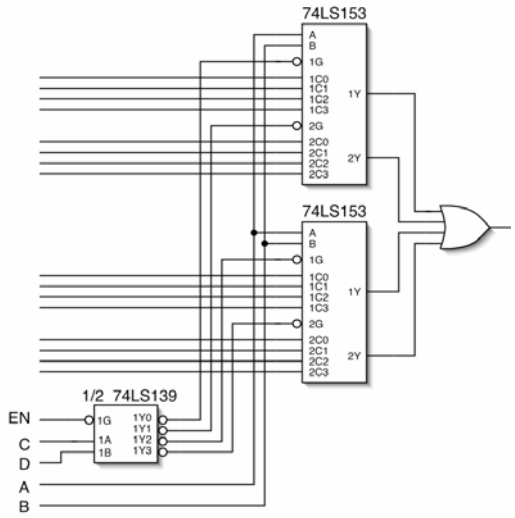
- Examples Of Multiplexer Chips



- Exercise: How could we use the 74LS153 to implement an 8-to-1 multiplexer?



- Cascading multiplexers



Verilog Descriptions of Multiplexers

- The Verilog Conditional Operator:
  - This is essentially the C conditional operator

    conditional_expression ? true_expression : false_expression
  - Example:
    - A = (B<C) ? (D+5) : (D+2);
- Description of a 2-to 1 MUX using a conditional operator

```
module mux2to1 (w0, w1, s, f);
    input w0, w1, s;
    output f;

    assign f = s ? w1: w0;

endmodule
```


- Description of a 4-to-1 MUX using conditional operators:

```
module mux4to1 (w0, w1, w2, w3, S, f);
    input w0, w1, w2, w3;
    input [1:0] S;
    output f;

    assign f = S[1] ? (S[0] ? w3 : w2) : (S[0] ? w1 : w0);

endmodule
```

- Description of a 4-to-1 MUX using if-then-else statements:

```
module mux4to1 (W, S, f);
    input [0:3] W;
    input [1:0] S;
    output f;
    reg f;

    always @(*)
        if (S == 0)
            f = W[0];
        else if (S == 1)
            f = W[1];
        else if (S == 2)
            f = W[2];
        else if (S == 3)
            f = W[3];

endmodule
```


- Description of a 4-to-1 MUX using a case statement:

```
module mux4to1 (W, S, f);
    input [0:3] W;
    input [1:0] S;
    output f;
    reg f;

    always @(*)
        case (S)
            0: f = W[0];
            1: f = W[1];
            2: f = W[2];
            3: f = W[3];
        endcase

endmodule
```

- Verilog description of a 16-to-1 MUX constructed as a tree of 4-to-1 decoders:

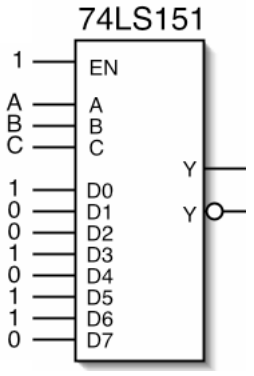
```
module mux16to1 (W, S, f, M);
    input [0:15] W;
    input [3:0] S;
    output f;
    output [3:0] M;
    wire [0:3] M;

    mux4to1 Mux1 (W[0:3], S[1:0], M[0]);
    mux4to1 Mux2 (W[4:7], S[1:0], M[1]);
    mux4to1 Mux3 (W[8:11], S[1:0], M[2]);
    mux4to1 Mux4 (W[12:15], S[1:0], M[3]);
    mux4to1 Mux5 (M[0:3], S[3:2], f);

endmodule
```

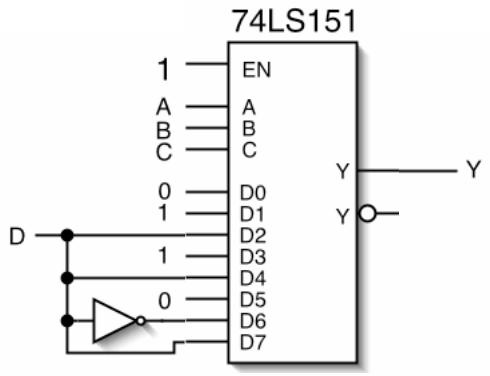

- Multiplexers as Function Generators

C	B	A	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



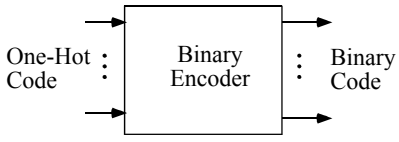
- Realizing a 4-variable function with the 74LS151

C	B	A	Y D=0	Y D=1
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	0	1



Encoders

- Encoders are code translators that perform a transformation that is the inverse of a decoder transformation.
- Binary encoders
  - Binary decoders translate from the binary code to the one-hot code.
  - Binary encoders translate from the one-hot code to the binary code.



- Implementation of a 8-to-3 binary encoder

One-Hot Input								Binary Output		
w7	w6	w5	w4	w3	w2	w1	w0	y2	y1	y0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$Y2 = w7 + w6 + w5 + w4$$
$$Y1 = w7 + w6 + w3 + w2$$
$$Y0 = w7 + w5 + w3 + w1$$

Priority Encoders

- A priority encoder has n inputs and $\lceil \log_2 n \rceil$ outputs.
- The output signals are a binary number such that its value is the highest index value of all the inputs that are 1.
- Example: 4-to-2 priority encoder:

w3	w2	w1	w0	y1	y0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

$$y1 = w3' \cdot w2 + w3$$
$$y2 = w3' \cdot w2' \cdot w1 + w3$$
$$z = w0 + w1 + w2 + w3$$

- What is the purpose of the signal z?

The Verilog casex and casez statements

- These statements allow the user to specify don't care conditions that the synthesizer can utilize to simplify the circuit.
- For a casez statement, the z value may appear in an alternative value and is treated as a don't care that matches any value in the controlling condition.
- The casex statement is just like a casez statement except that either the x value or the z value may be used.
- In both statements, a ? symbol my be used in place of x or z.
- The alternatives do not have to be mutully exclusive in the casex and casez statements.
  - The first matching alternative has priority and is selected.

- Verilog description of a priority encoder:

```
module priority (W, Y, z);
    input [3:0] W;
    output [1:0] Y;
    output z;
    reg [1:0] Y;
    reg z;

    always @(*)
        begin
            z = 1;
            casex (W)
                4'b1xxx: Y = 3;
                4'b01xx: Y = 2;
                4'b001x: Y = 1;
                4'b0001: Y = 0;
                default:  begin
                              z = 0 ;
                              Y = 2'bxx;
                          end
            endcase
        end
endmodule
```

Tips & Tricks

- Use multiplexers and decoders to implement combinational logic functions.
- Match negative assertion of signals with bubbles

Pitfalls

- Multiplexer and decoder outputs are usually asserted low.
- Getting the wrong assignment of variables to the select inputs of a multiplexer or decoder when using it to realize truth tables.
- Getting sequential behavior due to incomplete sensitivity lists or not specifying all output values in all control paths of procedural code.

Review

- Active signal levels and signal assertions.
  - Enable signals
- The relationship between binary decoders and minterms.
- The relationship between multiplexers minterms.
- Verilog procedural statements and “always” blocks
  - if-then-else
  - case, casez and casex
  - for loop
- Conditions for combinational behavior of always blocks