The material covered in this class will be as follows:

⇒ Multiplexers.
⇒ Boolean function implementation.
⇒ Three-state gates.
⇒ HDL for combinational circuits.
⇒ Gate level modelling.

Multiplexers

A multiplexer is a combinational circuit that selects one of many input lines (normally $2^n$ lines) and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines (normally n selection lines).

2-to-1 Line Multiplexer

A 2-to-1 line multiplexer has two inputs, one selection line and one output. This is shown in the following logic circuit.
4-to-1 Line Multiplexer

A 4-to-1 line multiplexer consists of four AND gates. Each input is connected to one AND gate. Selection lines S1 and S0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate that provides the output of the multiplexer Y.

The output of the multiplexer is then given by:

\[ Y = S_1 \cdot S_0 \cdot I_0 + S_1 \cdot S_0 \cdot I_1 + S_1 \cdot S_0 \cdot I_2 + S_1 \cdot S_0 \cdot I_3 \]

The function table of the multiplexer is shown next.

<table>
<thead>
<tr>
<th>( S_1 )</th>
<th>( S_0 )</th>
<th>( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( I_0 )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( I_1 )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( I_2 )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( I_3 )</td>
</tr>
</tbody>
</table>
Multiplexers may have an enable input, similar to decoders, to control the operation of the unit. A quadruple 2-to-1 multiplexer with enable input is shown next.

![Diagram of a quadruple 2-to-1 multiplexer with enable input]

The function table of the quadruple 2-to-1 multiplexer with the enable input will be as follows:

<table>
<thead>
<tr>
<th>E</th>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>All 0's</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Select A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Select B</td>
</tr>
</tbody>
</table>

**Boolean Function Implementation**

A multiplexer is a decoder and an OR gate that provides the output. The multiplexer can be used to implement Boolean
functions of $n$ variables. This can be achieved using either $2^n$-to-1 multiplexer or $2^{(n-1)}$-to-1 multiplexer.

1. Using $2^n$-to-1 multiplexer

The $n$ variables are connected to the $n$ selection lines. Each input of the multiplexer is set to 0 or 1, depending on which minterm of the function is present.

Example: Implement $F(x,y,z) = \sum (1,2,6,7)$ using 8-to-1 multiplexer.

Solution: Connect the variables $x$, $y$, $z$ to the selection inputs $S_2$, $S_1$, and $S_0$. Then set $I_0 = I_3 = I_4 = I_5 = 0$ and $I_1 = I_2 = I_6 = I_7 = 1$.

2. Using $2^{(n-1)}$-to-1 multiplexer

We connect $(n-1)$ variables to the selection lines. The multiplexer inputs are going to be either 0 or 1 or the remaining variable or the complement of the remaining variable.

Example: Implement $F(x,y,z) = \sum (1,2,6,7)$ using 4-to-1 multiplexer.

Solution: Connect the variables $x$ and $y$ to the selection inputs $S_1$, and $S_0$. The inputs of the multiplexers can be obtained from the truth table as shown below.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Three State Gates

A three state gate is a digital circuit that exhibits three states. Two states are equivalent to logic 0 and 1. The third state is a high impedance state which is controlled by a control input $C$. The most commonly used 3-state gate is the buffer.

$$C \uparrow \quad \text{ENB} \quad \text{Input A} \quad \text{Output Y}$$

The output $Y = A$ when $C = 1$ and is high impedance state when $C = 0$.

It is possible to implement multiplexers using 3-state buffers as shown.

$$S \quad \text{ENB} \quad \text{A} \quad \text{ENB} \quad \text{B} \quad \text{ENB} \quad \text{Y}$$

A 4-to-1 multiplexer may be constructed using four 3-state buffers and a 2-to-4 decoder.
HARDWARE DESCRIPTION LANGUAGE (HDL)

Gate level modelling

Verilog recognizes 12 basic gates as predefined primitives. Four are of the three state type. The other eight are and, nand, or, nor, xor, xnor, not, and buf.

The four three state type are bufif1, bufif0, notif1, and notif0.

Vectors can be specified with square brackets and two numbers indicating the length of the vector.

Examples:

Output [0:3] D;

Wire [7:0] SUM;

D[2] specifies bit 2 of D. SUM[2:0] specifies the three least significant bits of vector SUM.

HDL example 4-1

// Gate-level description of a 2-to-4-line decoder
// Figure 4-19
module decoder_g1([A,B,E,D);
    input A,B,E;
    output [0:3]D;
    wire Anot, Bnot, Enot;
not
n1 (A\neg, A),
n2 (B\neg, B),
n3 (E\neg, E);
and
n4 (D[0], A\neg, B\neg, E\neg),
n5 (D[1], A\neg, B, E\neg),
n6 (D[2], A, B\neg, E\neg),
n7 (D[3], A, B, E\neg),
endmodule

Two or more modules can be combined to build a hierarchical description of a design.

There are two types of design methodologies:

1. Top-down design
2. Bottom-up design

HDL example 4-2

// Gate-level hierarchical description of a 4-bit adder
// Description of half-adder (Fig. 4-5b)
module halfadder(S,C,x,y);
  input x,y;
  output S,C;
// Instantiate primitive gates
  xor (S,x,y);
  and (C,x,y);
endmodule
// Description of full adder (Fig. 4-8)
module fulladder(S,C,x,y,z);
    input x,y,z;
    output S,C;
    wire S1,D1,D2;    //outputs of first xor and two and gates
Instantiate the half adder
    Halfadder HA1 (S1,D1,x,y),
        HD2 (S,D2,S1,z);
    Or g1(C,D2,D1);
endmodule

// Description of 4-bit adder (Fig. 4-9)
module _4bit_adder (S,C4,A,B,C0);
    input [3:0] A,B;
    input C0;
    output [3:0] S;
    output C4;
    wire C1,C2,C3;    //Intermediate carries
// Instantiate the full adder
    fulladder    FA0 (S[0], C1, A[0], B[0], C0),
    fulladder    FA1 (S[1], C2, A[1], B[1], C1),
    fulladder    FA2 (S[2], C3, A[2], B[2], C2),
    fulladder    FA3 (S[3], C4, A[3], B[3], C3);
endmodule