Chapter 6: MSI Logic

This chapter introduces a group of combinational logic building blocks that are commonly used in digital design. As we move into systems that are larger than individual gates, there are naming conventions that are used to describe the size of the logic. Table 6.1 gives these naming conventions. In this chapter, we will look at *medium- scale integrated circuit* (MSI) logic. Each of these building blocks can be implemented using the combinational logic design steps covered in Chaps. 4 and 5. The goal of this chapter is to provide an understanding of the basic principles of MSI logic.

Commonly Used Names to Describe The Size of Digital Logic

Name	Example	# of Transistors
SSI - Small Scale Integrated Circuits	Individual Gates (NAND, INV)	10's
MSI - Medium Scale Integrated Circuits	Decoders, Multiplexers	100's
LSI – Large Scale Integrated Circuits	Arithmetic Circuits, RAM	1k – 10k
VLSI – Very Large Scale Integrated Circuits	Microprocessors	100k – 1M

While there are names for logic sizes above 1M transistor such as ULSI (Ultra), the term "VLSI" is now used to describe all integrated circuits that are so large they require CAD tools for their design, synthesis and implementation.

Table 6.1
Naming convention for the size of digital systems

Learning Outcomes—After completing this chapter, you will be able to:

6.1 Design a decoder circuit using both the classical digital design approach and the modern HDL-based approach.
6.2 Design an encoder circuit using both the classical digital design approach and the modern HDL-based approach.
6.3 Design a multiplexer circuit using both the classical digital design approach and the modern HDL-based approach.
6.4 Design a demultiplexer circuit using both the classical digital design approach and the modern HDL-based approach.

6.1 Decoders

A decoder is a circuit that takes in a binary *code* and has outputs that are asserted for specific values of that code. The code can be of any type or size (e.g., unsigned, two's complement, etc.). Each output will assert for only specific input codes. Since combinational logic circuits only produce a single output, this means that within a decoder, there will be a separate combinational logic circuit for each output.

6.1.1 Example: One-Hot Decoder

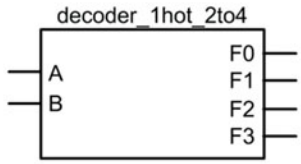
A one-hot decoder is a circuit that has n inputs and 2^n outputs. Each output will assert for one and only one input code. Since there are 2^n outputs, there will always be one and only one output asserted at

B. J. LaMeres, *Introduction to Logic Circuits & Logic Design with Verilog*,
https://doi.org/10.1007/978-3-030-13605-5_6

any given time. Example 6.1 shows the process of designing a 2-to-4 one-hot decoder by hand (i.e., using the classical digital design approach).

Example: 2-to-4 One-Hot Decoder - Logic Synthesis by Hand

The block diagram and truth table for this system are as follows:



A	B	F3	F2	F1	F0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

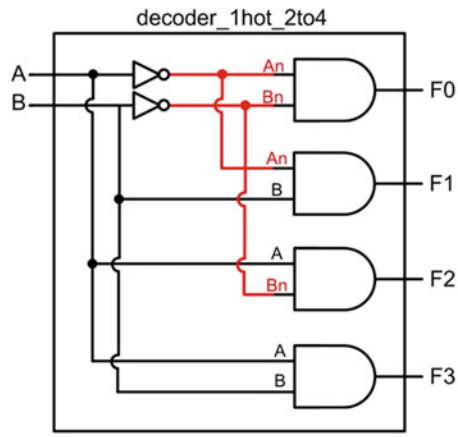
Each output asserts for a specific input code. This is where the term "one-hot" comes from. Each output is only "hot" for one input code.

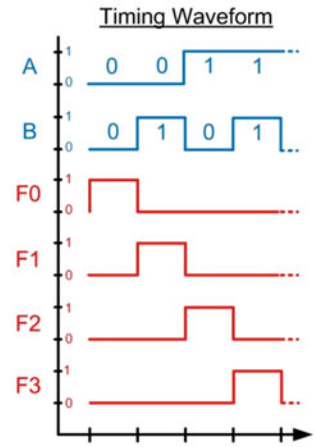
When designing this circuit, each output needs to have its own separate combinational logic circuit. This is the same as if there were four separate truth tables. This design could be implemented using 4x, 2-input K-maps to form the logic expressions for these outputs; however, by inspection a minterm list for each output will be the most minimal circuit.

$F0 = \sum_{A,B}(0) = A' \cdot B'$ $\quad F2 = \sum_{A,B}(2) = A \cdot B'$

$F1 = \sum_{A,B}(1) = A' \cdot B$ $\quad F3 = \sum_{A,B}(3) = A \cdot B$

When implementing the final decoder, the input inversions for A and B can be shared across all of the AND gates.



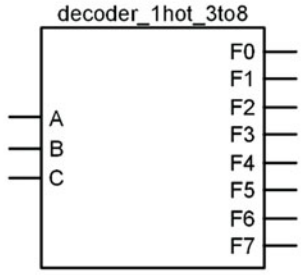


Example 6.1
2-to-4 one-hot decoder: logic synthesis by hand

As decoders get larger, it is necessary to use hardware description languages to model their behavior. Example 6.2 shows how to model a 3-to-8 one-hot decoder in Verilog with continuous assignment and logic operators.

Example: 3-to-8 One-Hot Decoder – Verilog Modeling using Logical Operators

The block diagram and truth table for this system are as follows:



A	B	C	F7	F6	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

To implement this in Verilog using logical operators, we must first determine the logic that will be used in the continuous assignment. Again, since each logic function only has one input code corresponding to an output of '1', the minterm can be used to implement the logic.

$$F0 = \sum_{A,B,C}(0) = A'\cdot B'\cdot C' \qquad F4 = \sum_{A,B,C}(4) = A\cdot B'\cdot C'$$
$$F1 = \sum_{A,B,C}(1) = A'\cdot B'\cdot C \qquad F5 = \sum_{A,B,C}(5) = A\cdot B'\cdot C$$
$$F2 = \sum_{A,B,C}(2) = A'\cdot B\cdot C' \qquad F6 = \sum_{A,B,C}(6) = A\cdot B\cdot C'$$
$$F3 = \sum_{A,B,C}(3) = A'\cdot B\cdot C \qquad F7 = \sum_{A,B,C}(7) = A\cdot B\cdot C$$

In Verilog, each of the outputs requires a separate continuous assignment.

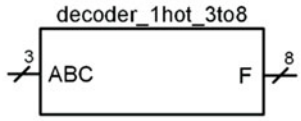
```
module decoder_1hot_3to8
  (output wire F0, F1, F2, F3, F4, F5, F6, F7,
   input  wire A, B, C);

  assign F0 = ~A & ~B & ~C;
  assign F1 = ~A & ~B &  C;
  assign F2 = ~A &  B & ~C;
  assign F3 = ~A &  B &  C;
  assign F4 =  A & ~B & ~C;
  assign F5 =  A & ~B &  C;
  assign F6 =  A &  B & ~C;
  assign F7 =  A &  B &  C;

endmodule
```

Example 6.2
3-to-8 one-hot decoder: Verilog modeling using logical operators

This description can be further simplified by using vector notation for the ports and describing the functionality using conditional operators. Example 6.3 shows how to model the 3-to-8 one-hot decoder in Verilog using continuous assignment with conditional operators.

Example: 3-to-8 One-Hot Decoder – Verilog Modeling using Conditional Operators

The block diagram and truth table for this system are as follows. Notice that the input and output ports now use vectors in order to create a more compact description.



ABC	F(7)	F(6)	F(5)	F(4)	F(3)	F(2)	F(1)	F(0)
"000"	0	0	0	0	0	0	0	1
"001"	0	0	0	0	0	0	1	0
"010"	0	0	0	0	0	1	0	0
"011"	0	0	0	0	1	0	0	0
"100"	0	0	0	1	0	0	0	0
"101"	0	0	1	0	0	0	0	0
"110"	0	1	0	0	0	0	0	0
"111"	1	0	0	0	0	0	0	0

The following shows a technique to model the decoder using continuous assignment with conditional operators. Note that the output will be "unknown" (X) if the input code is not one of the eight possible binary input values.

```
module decoder_1hot_3to8 (output wire [7:0] F,
                          input  wire [2:0] ABC);

   assign F = (ABC == 3'b000) ? 8'b0000_0001 :
              (ABC == 3'b001) ? 8'b0000_0010 :
              (ABC == 3'b010) ? 8'b0000_0100 :
              (ABC == 3'b011) ? 8'b0000_1000 :
              (ABC == 3'b100) ? 8'b0001_0000 :
              (ABC == 3'b101) ? 8'b0010_0000 :
              (ABC == 3'b110) ? 8'b0100_0000 :
              (ABC == 3'b111) ? 8'b1000_0000 :
              8'bXXXX_XXXX;

endmodule
```

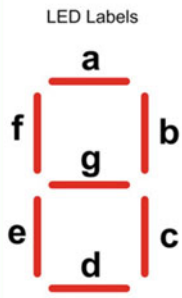
Example 6.3
3-to-8 one-hot decoder: Verilog modeling using conditional operators

6.1.2 Example: 7-Segment Display Decoder

A 7-segment display decoder is a circuit used to drive character displays that are commonly found in applications such as digital clocks and household appliances. A character display is made up of seven individual LEDs, typically labeled a–g. The input to the decoder is the binary equivalent of the decimal or Hex character that is to be displayed. The output of the decoder is the arrangement of LEDs that will form the character. Decoders with 2-inputs can drive characters "0" to "3." Decoders with 3-inputs can drive characters "0" to "7." Decoders with 4-inputs can drive characters "0" to "F" with the case of the Hex characters being "A, b, c or C, d, E and F."

Let's look at an example of how to design a 3-input, 7-segment decoder by hand. The first step in the process is to create the truth table for the outputs that will drive the LEDs in the display. We'll call these outputs F_a, F_b, . . ., F_g. Example 6.4 shows how to construct the truth table for the 7-segment display decoder. In this table, a logic 1 corresponds to the LED being ON.

Example: 7-Segment Display Decoder - Truth Table



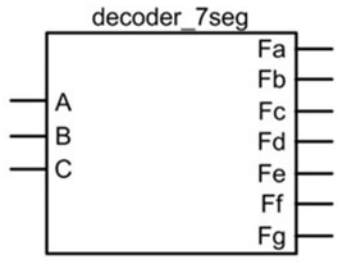
A	B	C		F_a	F_b	F_c	F_d	F_e	F_f	F_g
0	0	0		1	1	1	1	1	1	0
0	0	1		0	1	1	0	0	0	0
0	1	0		1	1	0	1	1	0	1
0	1	1		1	1	1	1	0	0	1
1	0	0		0	1	1	0	0	1	1
1	0	1		1	0	1	1	0	1	1
1	1	0		1	0	1	1	1	1	1
1	1	1		1	1	1	0	0	0	0

Example 6.4
7-Segment display decoder: truth table

If we wish to design this decoder by hand, we need to create seven separate combinational logic circuits. Each of the outputs (F_a – F_g) can be put into a 3-input K-map to find the minimized logic expression. Example 6.5 shows the design of the decoder from the truth table in Example 6.4 by hand.

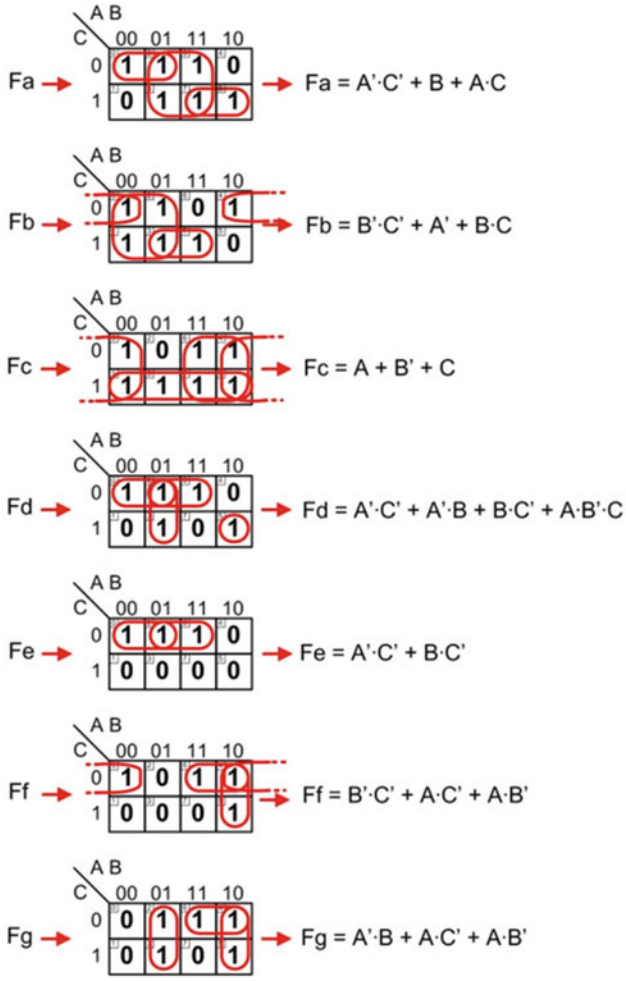
Example: 7-Segment Display Decoder – Logic Synthesis by Hand

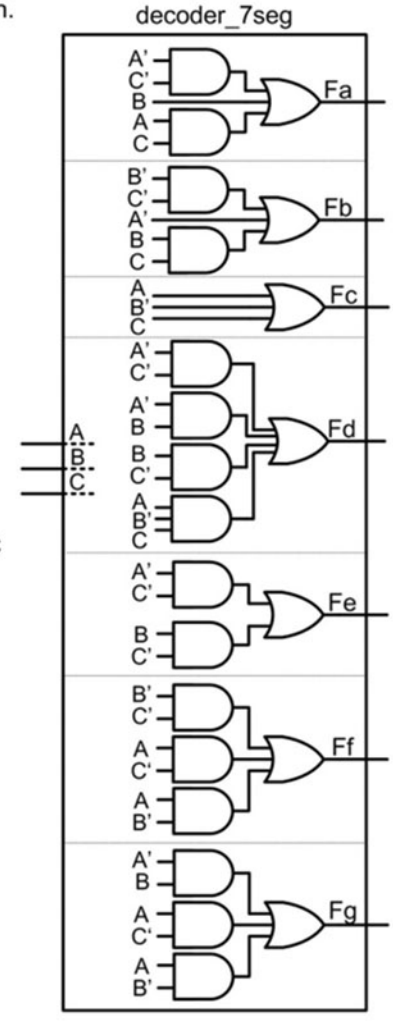
The block diagram and truth table for this system are as follows:



A	B	C	Fa	Fb	Fc	Fd	Fe	Ff	Fg
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

Each output of the decoder needs its own logic expression.



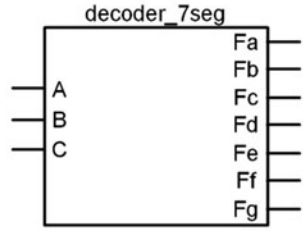


Example 6.5
7-Segment display decoder: logic synthesis by hand

This same functionality can be implemented in Verilog using concurrent modeling techniques. Example 6.6 shows how to model the 7-segment decoder in Verilog using continuous assignment with logic operators.

Example: 7-Segment Display Decoder – Verilog Modeling using Logical Operators

The block diagram and truth table for this system are as follows:



A	B	C	Fa	Fb	Fc	Fd	Fe	Ff	Fg
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

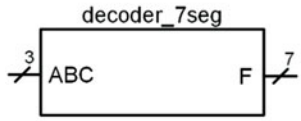
```
module decoder_7seg (output wire Fa, Fb, Fc, Fd, Fe, Ff, Fg,
                     input  wire A, B, C);

   assign Fa = (~A & ~C) | (B) | (A & C);
   assign Fb = (~B & ~C) | (~A) | (B & C);
   assign Fc = (A) | (~B) | (C);
   assign Fd = (~A & ~C) | (~A & B) | (B & ~C) | (A & ~B & C);
   assign Fe = (~A & ~C) | (B & ~C);
   assign Ff = (~B & ~C) | (A & ~C) | (A & ~B);
   assign Fg = (~A & B) | (A & ~C) | (A & ~B);

endmodule
```

Example 6.6
7-Segment display decoder: Verilog modeling using logical operators

Again, a more compact description of the decoder can be accomplished if the ports are described as vectors and a conditional operator is used. Example 6.7 shows how to model the 7-segment decoder in Verilog using continuous assignment with conditional operators.

Example: 7-Segment Decoder – Verilog Modeling using Conditional Operators

The block diagram and truth table for this system are as follows:



	a	b	c	d	e	f	g
ABC	F(6)	F(5)	F(4)	F(3)	F(2)	F(1)	F(0)
"000"	1	1	1	1	1	1	0
"001"	0	1	1	0	0	0	0
"010"	1	1	0	1	1	0	1
"011"	1	1	1	1	0	0	1
"100"	0	1	1	0	0	1	1
"101"	1	0	1	1	0	1	1
"110"	1	0	1	1	1	1	1
"111"	1	1	1	0	0	0	0

The following shows a technique to model the decoder using continuous assignment with conditional operators.

```
module decoder_7seg (output wire [6:0] F,
                     input  wire [2:0] ABC);

   assign F = (ABC == 3'b000) ? 7'b111_1110 :
              (ABC == 3'b001) ? 7'b011_0000 :
              (ABC == 3'b010) ? 7'b110_1101 :
              (ABC == 3'b011) ? 7'b111_1001 :
              (ABC == 3'b100) ? 7'b011_0011 :
              (ABC == 3'b101) ? 7'b101_1011 :
              (ABC == 3'b110) ? 7'b101_1111 :
              (ABC == 3'b111) ? 7'b111_0000 :
             8'bXXXX_XXXX;

endmodule
```

Example 6.7
7-Segment display decoder: Verilog modeling using conditional operators

Concept Check

CC6.1 In a decoder, a logic expression is created for each output. Once all of the output logic expressions are found, how can the decoder logic be further minimized?

A) By using K-maps to find the output logic expressions.

B) By buffering the inputs so that they can drive a large number of other gates.

C) By identifying any logic terms that are used in multiple locations (inversions, product terms, and sum terms) and sharing the interim results among multiple circuits in the decoder.

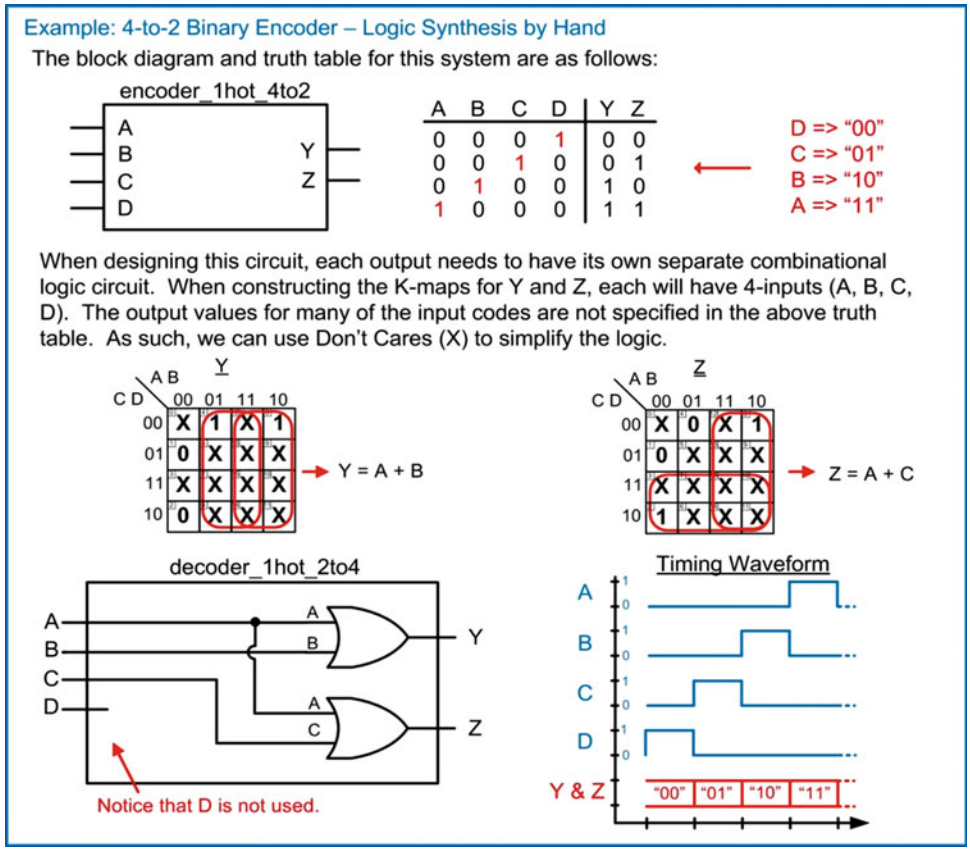
D) By ignoring fan-out.

6.2 Encoders

An encoder works in the opposite manner as a decoder. An assertion on a specific input port corresponds to a unique code on the output port.

6.2.1 Example: One-Hot Binary Encoder

A one-hot binary encoder has n outputs and 2^n inputs. The output will be an n-bit, binary code which corresponds to an assertion on one and only one of the inputs. Example 6.8 shows the process of designing a 4-to-2 binary encoder by hand (i.e., using the classical digital design approach).

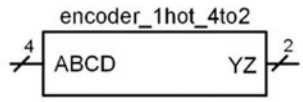


Example 6.8
4-to-2 binary encoder: logic synthesis by hand

In Verilog, an encoder can be implemented using continuous assignment with either logical or conditional operators. Example 6.9 shows how to model the encoder in Verilog using these techniques.

Example: 4-to-2 Binary Encoder – Verilog Modeling using Continuous Assignment

The block diagram and truth table for this system are as follows:



ABCD	YZ
"0 0 0 1"	"0 0"
"0 0 1 0"	"0 1"
"0 1 0 0"	"1 0"
"1 0 0 0"	"1 1"

The following are two different ways to implement the behavior of the encoder with continuous assignment: (1) with logical operators; and (2) with conditional operators.

(1)

```
module encoder_1hot_4to2 (output wire [1:0] YZ,
                          input  wire [3:0] ABCD);

   assign YZ[1] = ABCD[3] | ABCD[2];
   assign YZ[0] = ABCD[3] | ABCD[1];

endmodule
```

(2)

```
module encoder_1hot_4to2 (output wire [1:0] YZ,
                          input  wire [3:0] ABCD);

   assign YZ = (ABCD == 4'b0001) ? 2'b00 :
               (ABCD == 4'b0010) ? 2'b01 :
               (ABCD == 4'b0100) ? 2'b10 :
               (ABCD == 4'b1000) ? 2'b11 :
               2'bXX;
endmodule
```

Example 6.9
4-to-2 binary encoder: Verilog modeling using logical and conditional operators

Concept Check

CC6.2 If it is desired to have the outputs of an encoder produce 0's for all input codes not defined in the truth table, can "don't cares" be used when deriving the minimized logic expressions? Why?

A) No. Don't cares aren't used in encoders.

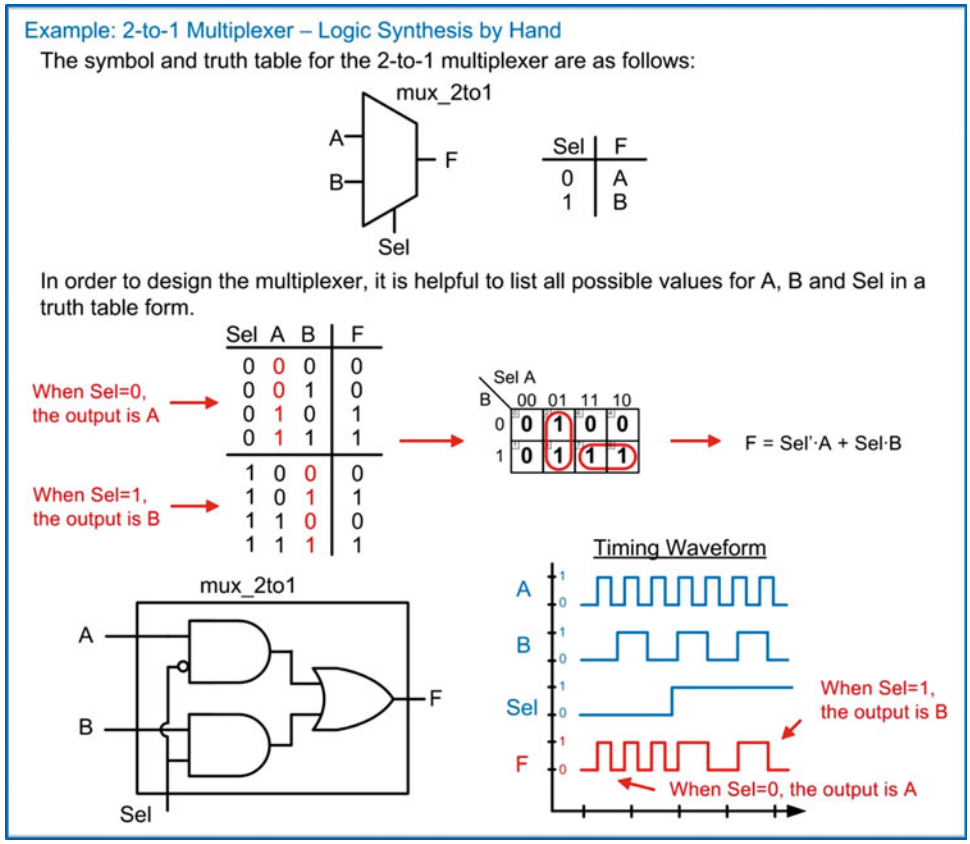
B) Yes. Don't cares can always be used in K-maps.

C) Yes. All that needs to be done is to treat each X as a 0 when forming the most minimal prime implicant.

D) No. Each cell in the K-map corresponding to an undefined input code needs to contain a 0 so don't cares are not applicable.

6.3 Multiplexers

A multiplexer is a circuit that passes one of its multiple inputs to a single output based on a select input. This can be thought of as a digital switch. The multiplexer has n select lines, 2^n inputs, and one output. Example 6.10 shows the process of designing a 2-to-1 multiplexer by hand (i.e., using the classical digital design approach).

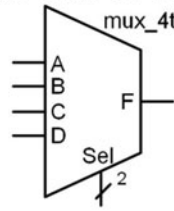


Example 6.10
2-to-1 multiplexer: logic synthesis by hand

In Verilog, a multiplexer can be implemented using continuous assignment with either logical or conditional operators. Example 6.11 shows how to model the multiplexer in Verilog using these techniques.

Example: 4-to-1 Multiplexer – Verilog Modeling using Continuous Assignment

The symbol and truth table for the 4-to-1 multiplexer are as follows:



Sel	F
"00"	A
"01"	B
"10"	C
"11"	D

The following are two different ways to implement the behavior of the multiplexer with continuous assignment: (1) with logical operators; and (2) with conditional operators.

(1)

```
module mux_4to1 (output wire F,
                 input  wire A, B, C, D,
                 input  wire [1:0] Sel);

   assign F = (A & ~Sel[1] & ~Sel[0]) |
              (B & ~Sel[1] &  Sel[0]) |
              (C &  Sel[1] & ~Sel[0]) |
              (D &  Sel[1] &  Sel[0]);

endmodule
```

(2)

```
module mux_4to1 (output wire F,
                 input  wire A, B, C, D,
                 input  wire [1:0] Sel);

   assign F = (Sel == 2'b00) ? A :
              (Sel == 2'b01) ? B :
              (Sel == 2'b10) ? C :
              (Sel == 2'b11) ? D :
              1'bX;

endmodule
```

Example 6.11
4-to-1 multiplexer: Verilog modeling using logical and conditional operators

Concept Check

CC6.3 How are the product terms in a multiplexer based on the identity theorem?

A) Only the select product term will pass its input to the final sum term. Since all of the unselected product terms output 0, the input will be passed through the sum term because anything OR'd with a 0 is itself.

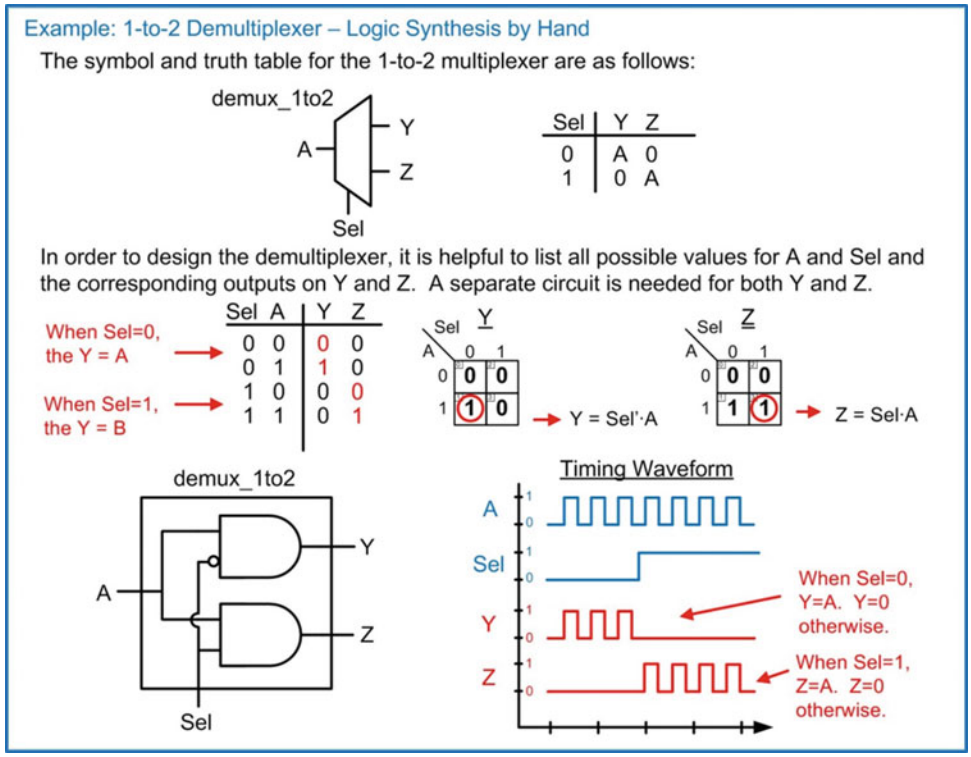
B) The select lines are complemented such that they activate only one OR gate.

C) The select line inputs will produce 1's on the inputs of the selected product term. This allows the input signal to pass through the selected AND gate because anything AND'd with a 1 is itself.

D) The select line inputs will produce 0's on the inputs of the selected sum term. This allows the input signal to pass through the selected OR gate because anything OR'd with a 0 is itself.

6.4 Demultiplexers

A demultiplexer works in a complementary fashion to a multiplexer. A demultiplexer has one input that is routed to one of its multiple outputs. The output that is active is dictated by a select input. A demux has n select lines that chooses to route the input to one of its 2^n outputs. When an output is not selected, it outputs a logic 0. Example 6.12 shows the process of designing a 1-to-2 demultiplexer by hand (i.e., using the classical digital design approach).

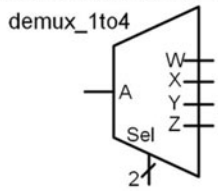


Example 6.12
1-to-2 demultiplexer: logic synthesis by hand

In Verilog, a demultiplexer can be implemented using continuous assignment with either logical or conditional operators. Example 6.13 shows how to model the demultiplexer in Verilog using these techniques.

Example: 1-to-4 Demultiplexer – Verilog Modeling using Continuous Assignment

The symbol and truth table for the 1-to-4 demultiplexer are as follows:



Sel	W	X	Y	Z
"00"	A	0	0	0
"01"	0	A	0	0
"10"	0	0	A	0
"11"	0	0	0	A

The following are two different ways to implement the behavior of the demultiplexer with continuous assignment: (1) with logical operators; and (2) with conditional operators.

(1)

```
module demux_1to4 (output wire W, X, Y, Z,
                   input  wire A,
                   input  wire [1:0] Sel);

   assign W = (A & ~Sel[1] & ~Sel[0]);
   assign X = (A & ~Sel[1] &  Sel[0]);
   assign Y = (A &  Sel[1] & ~Sel[0]);
   assign Z = (A &  Sel[1] &  Sel[0]);

endmodule
```

(2)

```
module demux_1to4 (output wire W, X, Y, Z,
                   input  wire A,
                   input  wire [1:0] Sel);

   assign W = (Sel == 2'b00) ? A : 1'b0;
   assign X = (Sel == 2'b01) ? A : 1'b0;
   assign Y = (Sel == 2'b10) ? A : 1'b0;
   assign Z = (Sel == 2'b11) ? A : 1'b0;

endmodule
```

Example 6.13
1-to-4 demultiplexer: Verilog modeling using logical and conditional operators

Concept Check

CC6.4 How many select lines are needed in a 1-to-64 demultiplexer?

A) 1 B) 4 C) 6 D) 64

Summary

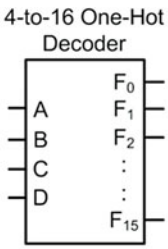
- The term medium-scale integrated circuit (MSI) logic refers to a set of basic combinational logic circuits that implement simple, commonly used functions such as decoders, encoders, multiplexers, and demultiplexers. MSI logic can also include operations such as comparators and simple arithmetic circuits.
- While an MSI logic circuit may have multiple outputs, each output requires its own unique logic expression that is based on the system inputs.
- A decoder is a system that has a greater number of outputs than inputs. The behavior of each output is based on each unique input code.

- An encoder is a system that has a greater number of inputs than outputs. A compressed output code is produced based on which input(s) lines are asserted.
- A multiplexer is a system that has one output and multiple inputs. At any given time, one and only one input is routed to the output based on the value on a set of *select lines*. For *n* select lines, a multiplexer can support 2^n inputs.
- A demultiplexer is a system that has one input and multiple outputs. The input is routed to one of the outputs depending on the value on a set of select lines. For n select lines, a demultiplexer can support 2^n outputs.
- HDLs are particularly useful for describing MSI logic due to their abstract modeling capability. Through the use of Boolean conditions and vector assignments, the behavior of MSI logic can be modeled in a compact and intuitive manner.

Exercise Problems

Section 6.1: Decoders

6.1.1 Design a 4-to-16 one-hot decoder by hand. The block diagram and truth table for the decoder are given in Fig. 6.1. Give the minimized logic expressions for each output (i.e., F_0, F_1, . . ., F_{15}) and the full logic diagram for the system.



A B C D	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0 0 1 0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0 0 1 1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0 1 0 0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0 1 0 1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0 1 1 0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0 1 1 1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1 0 0 0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1 0 0 1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1 0 1 0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1 0 1 1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1 1 0 0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1 1 0 1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 1 0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 1 1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 6.1
4-to-16 one-hot decoder functionality

6.1.2 Design a Verilog model for a 4-to-16 one-hot decoder using continuous assignment and gate-level primitives. Use the module port definition given in Fig. 6.2.

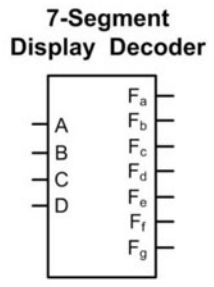
```
module decoder_1hot_4to16
  (output wire [15:0] F,
   input  wire [3:0]  ABCD);
                      :
```

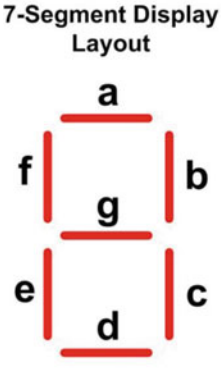
Fig. 6.2
4-to-16 one-hot decoder module definition

6.1.3 Design a Verilog model for a 4-to-16 one-hot decoder using continuous assignment and logical operators. Use the module port definition given in Fig. 6.2.

6.1.4 Design a Verilog model for a 4-to-16 one-hot decoder using continuous assignment and conditional operators. Use the module port definition given in Fig. 6.2.

6.1.5 Design a 4-input, 7-segment HEX character decoder by hand. The system has four inputs called A, B, C, and D. The system has seven outputs called F_a, F_b, F_c, F_d, F_e, F_f, and F_g. These outputs drive the individual LEDs within the display. A logic 1 on an output corresponds to the LED being ON. The display will show the HEX characters 0–9, A, b, c, d, E, and F corresponding to the 4-bit input code on A. A template for creating the truth tables for this system is provided in Fig. 6.3. Provide the minimized logic expressions for each of the seven outputs and the overall logic diagram for the decoder.





A	B	C	D		F_a	F_b	F_c	F_d	F_e	F_f	F_g
0	0	0	0								
0	0	0	1								
0	0	1	0								
0	0	1	1								
0	1	0	0								
0	1	0	1								
0	1	1	0								
0	1	1	1								
1	0	0	0								
1	0	0	1								
1	0	1	0								
1	0	1	1								
1	1	0	0								
1	1	0	1								
1	1	1	0								
1	1	1	1								

Fig. 6.3
7-segment display decoder truth table

6.1.6 Design a Verilog model for a 4-input, 7-segment HEX character decoder using continuous assignment and logical operators. Use the module port definition given in Fig. 6.4 for your design. The system has a 4-bit input vector called ABCD and a 7-bit output vector called F. The individual scalars within the output vector (i.e., F[6:0]) correspond to the character display segments a, b, c, d, e, f, and g, respectively. A logic 1 on an output corresponds to the LED being ON. The display will show the HEX characters 0–9, A, b, c, d, E, and F corresponding to the 4-bit input code on A. A template for creating the truth table is provided in. The signals in this table correspond to the ports in this problem as follows: $F_a = F(6)$, $F_b = F(5)$, $F_c = F(4)$, $F_d = F(3)$, $F_e = F(2)$, $F_f = F(1)$, and $F_g = F(0)$.

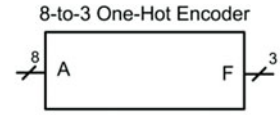
```
module decoder_7seg_4in
  (output wire [6:0] F,
   input  wire [3:0] ABCD);
                    :
```

Fig. 6.4
7-segment display decoder module definition

6.1.7 Design a Verilog model for a 4-input, 7-segment HEX character decoder using continuous assignment and conditional operators. Use the module port definition given in Fig. 6.4 for your design. The system has a 4-bit input vector called ABCD and a 7-bit output vector called F. The individual scalars within the output vector (i.e., F[6:0]) correspond to the character display segments a, b, c, d, e, f, and g, respectively. A logic 1 on an output corresponds to the LED being ON. The display will show the HEX characters 0–9, A, b, c, d, E, and F corresponding to the 4-bit input code on A. A template for creating the truth table is provided in. The signals in this table correspond to the ports in this problem as follows: $F_a = F(6)$, $F_b = F(5)$, $F_c = F(4)$, $F_d = F(3)$, $F_e = F(2)$, $F_f = F(1)$, and $F_g = F(0)$.

Section 6.2: Encoders

6.2.1 Design an 8-to-3 binary encoder by hand. The block diagram and truth table for the encoder are given in Fig. 6.5. Give the logic expressions for each output and the full logic diagram for the system.



A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)	F(2)	F(1)	F(0)
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

Fig. 6.5
8-to-3 one-hot encoder functionality

6.2.2 Design a Verilog model for an 8-to-3 binary encoder using continuous assignment and gate-level primitives. Use the module port definition given in Fig. 6.6.

```
module encoder_8to3_binary
  (output wire [2:0] F,
   input  wire [7:0] A);
                    :
```

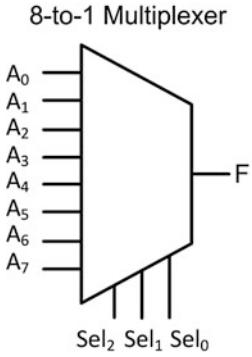
Fig. 6.6
8-to-3 one-hot encoder module functionality

6.2.3 Design a Verilog model for an 8-to-3 binary encoder using continuous assignment and logical operators. Use the module port definition given in Fig. 6.6.

6.2.4 Design a Verilog model for an 8-to-3 binary encoder using continuous assignment and conditional operators. Use the module port definition given in Fig. 6.6.

Section 6.3: Multiplexers

6.3.1 Design an 8-to-1 multiplexer by hand. The block diagram and truth table for the multiplexer are given in Fig. 6.7. Give the minimized logic expressions for the output and the full logic diagram for the system.



Sel_2	Sel_1	Sel_0	F
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

Fig. 6.7
8-to-1 multiplexer functionality

6.3.2 Design a Verilog model for an 8-to-1 multiplexer using continuous assignment and gate-level primitives. Use the module port definition given in Fig. 6.8.

```
module mux_8to1
  (output wire F,
   input  wire [7:0] A,
   input  wire [2:0] Sel);
                     :
```

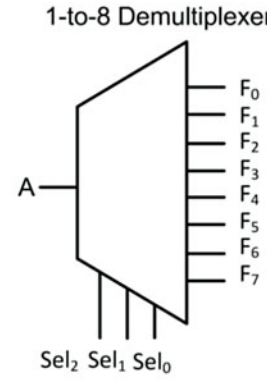
Fig. 6.8
8-to-1 multiplexer module definition

6.3.3 Design a Verilog model for an 8-to-1 multiplexer using continuous assignment and logical operators. Use the module port definition given in Fig. 6.8.

6.3.4 Design a Verilog model for an 8-to-1 multiplexer using continuous assignment and conditional operators. Use the module port definition given in Fig. 6.8.

Section 6.4: Demultiplexers

6.4.1 Design a 1-to-8 demultiplexer by hand. The block diagram and truth table for the demultiplexer are given in Fig. 6.9. Give the minimized logic expressions for each output and the full logic diagram for the system.



Sel_2	Sel_1	Sel_0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

Fig. 6.9
1-to-8 demultiplexer functionality

6.4.2 Design a Verilog model for a 1-to-8 demultiplexer using continuous assignment and gate-level primitives. Use the module port definition given in Fig. 6.10 for your design.

```
module demux_1to8
  (output wire [7:0] F,
   input  wire A,
   input  wire [2:0] Sel);
                     :
```

Fig. 6.10
1-to-8 demultiplexer module definition

6.4.3 Design a Verilog model for a 1-to-8 demultiplexer using continuous assignment and logical operators. Use the module port definition given in Fig. 6.10 for your design.

6.4.4 Design a Verilog model for a 1-to-8 demultiplexer using continuous assignment and conditional operators. Use the module port definition given in Fig. 6.10 for your design.