Chapter 8: Combinational Logic Modules

Prof. Ming-Bo Lin

Department of Electronic Engineering
National Taiwan University of Science and Technology
Syllabus

- Objectives
- Fundamentals of combinational logic modules
- Decoders
- Encoders
- Multiplexers
- Demultiplexers
- Comparators
Objectives

After completing this chapter, you will be able to:

- Understand the features of decoders
- Understand the features of encoders
- Understand the features of priority encoders
- Understand the features of multiplexers
- Understand the features of demultiplexers
- Describe how to design comparators and magnitude comparators
- Describe how to design a parameterized module


Syllabus

- Objectives
- Fundamentals of combinational logic modules
- Decoders
- Encoders
- Multiplexers
- Demultiplexers
- Comparators
Basic Combinational Logic Modules

- Decoder
- Encoder
- Multiplexer
- Demultiplexer
- Comparator
- Adder (CLA)
- Subtractor (subtractor)
- Multiplier
- PLA
- Parity Generator
Options for Modeling Combinational Logic

- Verilog HDL primitives
- Continuous assignment
- Behavioral statement
- Functions
- Task without delay or event control
- Combinational UDP
- Interconnected combinational logic modules
Syllabus

- Objectives
- Fundamentals of combinational logic modules
- Decoders
- Encoders
- Multiplexers
- Demultiplexers
- Comparators
Decoder Block Diagrams

- $n$-to-$m$ decoders

![Diagram of $n$-to-$m$ decoders](image)

(a) Noninverted output

(b) Inverted output
A 2-to-4 Decoder Example

(a) Logic symbol

<table>
<thead>
<tr>
<th>$E$</th>
<th>$x_1$</th>
<th>$x_0$</th>
<th>$Y_3$</th>
<th>$Y_2$</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) Function table

(c) Logic circuit
A 2-to-4 Decoder Example

// a 2-to-4 decoder with active low output
always @(x or enable_n)
  if (enable_n) y = 4'b1111; else
  case (x)
    2'b00 : y = 4'b1110;
    2'b01 : y = 4'b1101;
    2'b10 : y = 4'b1011;
    2'b11 : y = 4'b0111;
  endcase
endcase
A 2-to-4 Decoder with Enable Control

// a 2-to-4 decoder with active-high output
always @(x or enable)
    if (!enable) y = 4'b0000; else
    case (x)
        2'b00 : y = 4'b0001;
        2'b01 : y = 4'b0010;
        2'b10 : y = 4'b0100;
        2'b11 : y = 4'b1000;
    endcase
Syllabus

- Objectives
- Fundamentals of combinational logic modules
- Decoders
- Encoders
- Multiplexers
- Demultiplexers
- Comparators
Encoder Block Diagrams

- $m$-to-$n$ encoders

(a) Noninverted output

(b) Inverted output
A 4-to-2 Encoder Example

Q: What is the problem of this encoder?

<table>
<thead>
<tr>
<th>$I_3$</th>
<th>$I_2$</th>
<th>$I_1$</th>
<th>$I_0$</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Function table

(b) Logic circuit
A 4-to-2 Encoder Example

// a 4-to-2 encoder using if ... else structure
always @(in) begin
    if (in == 4'b0001) y = 0; else
    if (in == 4'b0010) y = 1; else
    if (in == 4'b0100) y = 2; else
    if (in == 4'b1000) y = 3; else
        y = 2'bx;
end
Another 4-to-2 Encoder Example

// a 4-to-2 encoder using case structure
always @(in)
    case (in)
        4'b0001 : y = 0;
        4'b0010 : y = 1;
        4'b0100 : y = 2;
        4'b1000 : y = 3;
        default : y = 2'bX;
    endcase
A 4-to-2 Priority Encoder

(a) Block diagram

(b) Function table

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_3$</td>
<td>$A_1$</td>
</tr>
<tr>
<td>$I_2$</td>
<td>$A_0$</td>
</tr>
<tr>
<td>$I_1$</td>
<td></td>
</tr>
<tr>
<td>$I_0$</td>
<td></td>
</tr>
</tbody>
</table>

- 0 0 0 1 0 0
- 0 0 1 $\phi$ 0 1
- 0 1 $\phi$ $\phi$ 1 0
- 1 $\phi$ $\phi$ $\phi$ 1 1
A 4-to-2 Priority Encoder Example

// using if ... else structure
assign valid_in = |in;
always @(in) begin
    if (in[3]) y = 3; else
    if (in[2]) y = 2; else
    if (in[1]) y = 1; else
    if (in[0]) y = 0; else
        y = 2'bx;
end
Another 4-to-2 Priority Encoder Example

// using casex structure
assign valid_in = |in;
always @(in) casex (in)
  4'b1xxx: y = 3;
  4'b01xx: y = 2;
  4'b001x: y = 1;
  4'b0001: y = 0;
  default: y = 2'bx;
endcase
Syllabus

- Objectives
- Fundamentals of combinational logic modules
- Decoders
- Encoders
- **Multiplexers**
- Demultiplexers
- Comparators
Multiplexer Block Diagrams

- $m$-to-1 ($m = 2^n$) multiplexers

(a) Without enable control
(b) With enable control
A 4-to-1 Multiplexer Example

- Gate-based 4-to-1 multiplexers

(a) Logic symbol
(b) Function table
(c) Logic circuit
An \( n \)-bit 4-to-1 Multiplexer Example

// an N-bit 4-to-1 multiplexer using conditional operator

```verilog
class parameter N = 4; //
input [1:0] select;
input [N-1:0] in3, in2, in1, in0;
output [N-1:0] y;
assign y = select[1] ?
    (select[0] ? in3 : in2) :
    (select[0] ? in1 : in0);
```

![Multiplexer Circuit Diagram](image)
The Second $n$-bit 4-to-1 Multiplexer Example

// an N-bit 4-to-1 multiplexer with enable control
parameter N = 4;
input [1:0] select;
input enable;
input [N-1:0] in3, in2, in1, in0;
output reg [N-1:0] y;

always @(select or enable or in0 or in1 or in2 or in3)
    if (!enable) y = {N{1'b0}};
    else y = select[1] ?
        (select[0] ? in3 : in2) :
        (select[0] ? in1 : in0) ;
The Third $n$-bit 4-to-1 Multiplexer Example

// an N-bit 4-to-1 multiplexer using case structure
parameter N = 8;
input [1:0] select;
input [N-1:0] in3, in2, in1, in0;
output reg [N-1:0] y;
always @(*)
    case (select)
        2'b11: y = in3;
        2'b10: y = in2;
        2'b01: y = in1;
        2'b00: y = in0;
    endcase
Syllabus

- Objectives
- Fundamentals of combinational logic modules
- Decoders
- Encoders
- Multiplexers
- Demultiplexers
- Comparators
DeMultiplexer Block Diagrams

- 1-to-\(m\) (\(m = 2^n\)) demultiplexers

(a) Without enable control

(b) With enable control
A 1-to-4 DeMultiplexer Example

- Gate-based 1-to-4 demultiplexers

### (a) Logic symbol

### (b) Function table

<table>
<thead>
<tr>
<th>$E$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$Y_3$</th>
<th>$Y_2$</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$D$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$D$</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$D$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$D$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### (c) Logic circuit
An $n$-bit 1-to-4 DeMultiplexer Example

// an N-bit 1-to-4 demultiplexer using if ... else structure
parameter N = 4;  // default width
input  [1:0] select;
input  [N-1:0] in;
output reg [N-1:0] y3, y2, y1, y0;
always @(select or in) begin
  if (select == 3) y3 = in; else y3 = {N{1'b0}};
  if (select == 2) y2 = in; else y2 = {N{1'b0}};
  if (select == 1) y1 = in; else y1 = {N{1'b0}};
  if (select == 0) y0 = in; else y0 = {N{1'b0}};
end
The Second $n$-bit 1-to-4 DeMultiplexer Example

// an N-bit 1-to-4 demultiplexer with enable control
parameter N = 4;    // Default width
...
output reg [N-1:0] y3, y2, y1, y0;
always @(select or in or enable) begin
  if (enable) begin
    if (select == 3) y3 = in; else y3 = {N{1'b0}};
    if (select == 2) y2 = in; else y2 = {N{1'b0}};
    if (select == 1) y1 = in; else y1 = {N{1'b0}};
    if (select == 0) y0 = in; else y0 = {N{1'b0}};
  end else begin
    y3 = {N{1'b0}}; y2 = {N{1'b0}}; y1 = {N{1'b0}}; y0 = {N{1'b0}}; end
end
Syllabus

- Objectives
- Fundamentals of combinational logic modules
- Decoders
- Encoders
- Multiplexers
- Demultiplexers
- Comparators
Comparators

- A 4-bit comparator

A 4-bit cascadable comparator block diagram
Types of Comparators

- Comparator
- Cascadable comparator
Comparators

An 8-bit comparator

Q: What will happen if you set the input value (010) at the rightmost end to other values?
A Simple Comparator Example

// an N-bit comparator module example

parameter N = 4;  // default size

input [N-1:0] a, b;

output cgt, clt, ceq;

assign cgt = (a > b);
assign clt = (a < b);
assign ceq = (a == b);
A Cascadable Comparator Example

parameter N = 4;
// I/O port declarations
input     Iagtb, Iaeqb, Ialtb;
input     [N-1:0] a, b;
output   Oagtb, Oaeqb, Oaltb;

// dataflow modeling using relation operators
assign Oaeqb = (a == b) && (Iaeqb == 1);  // =
assign Oagtb = (a > b) || ((a == b)&& (Iagtb == 1));  // >
assign Oaltb =  (a < b) ||  ((a == b)&& (Ialtb == 1));  // <