

# **Combinational Logic and Verilog**

# **Programmable Array Logic**

**PAL**

# Example of PAL. GAL16V8C

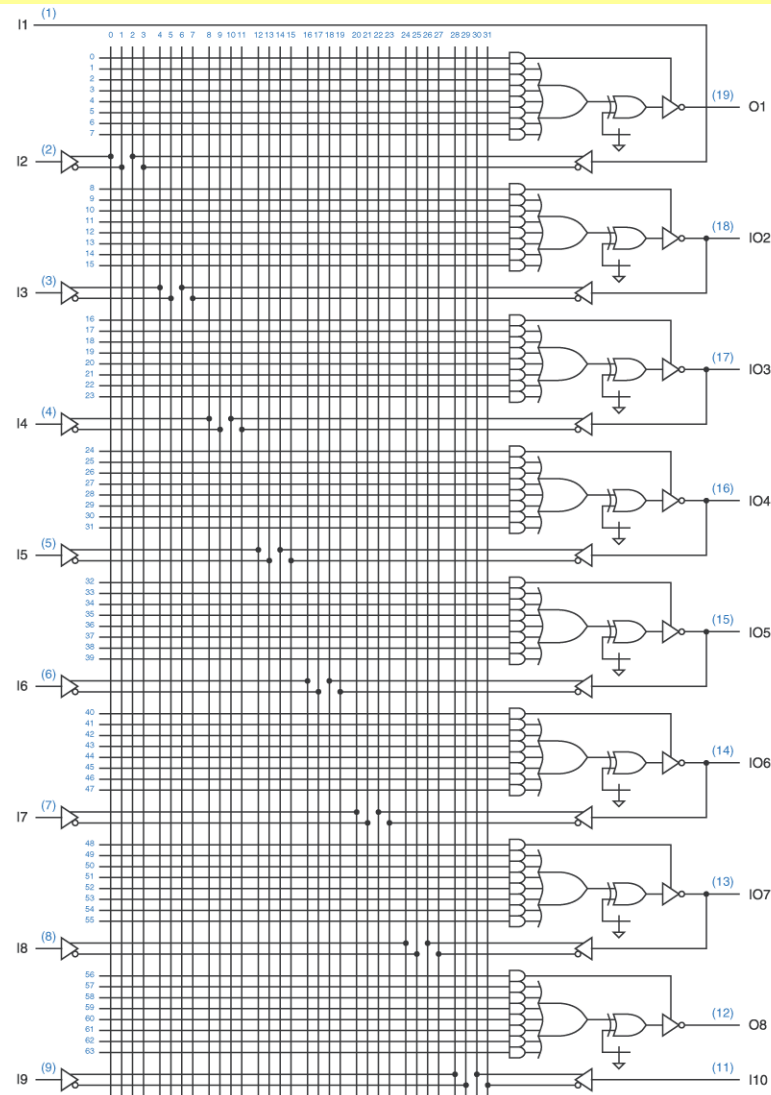
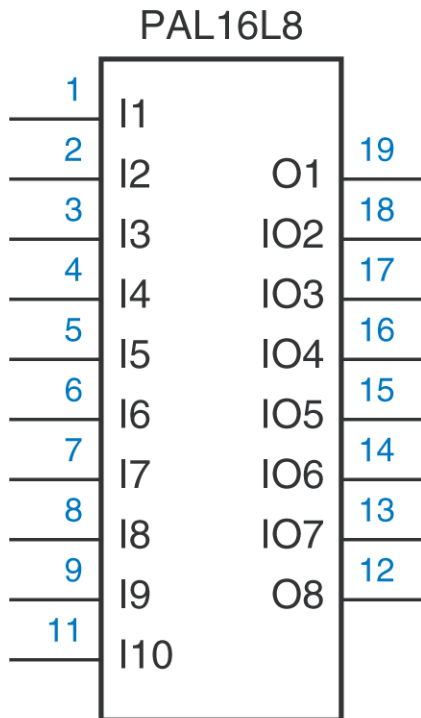


Figure 6-27

Logic diagram of the GAL16V8C.

# PALs

1. SOP
2. Multi-level
3. Flip-flops
4. 10 Inputs
5. 8 Inputs/output



## Figure 6-26. PAL16L8

# Figure 6-28. General CPLD architecture

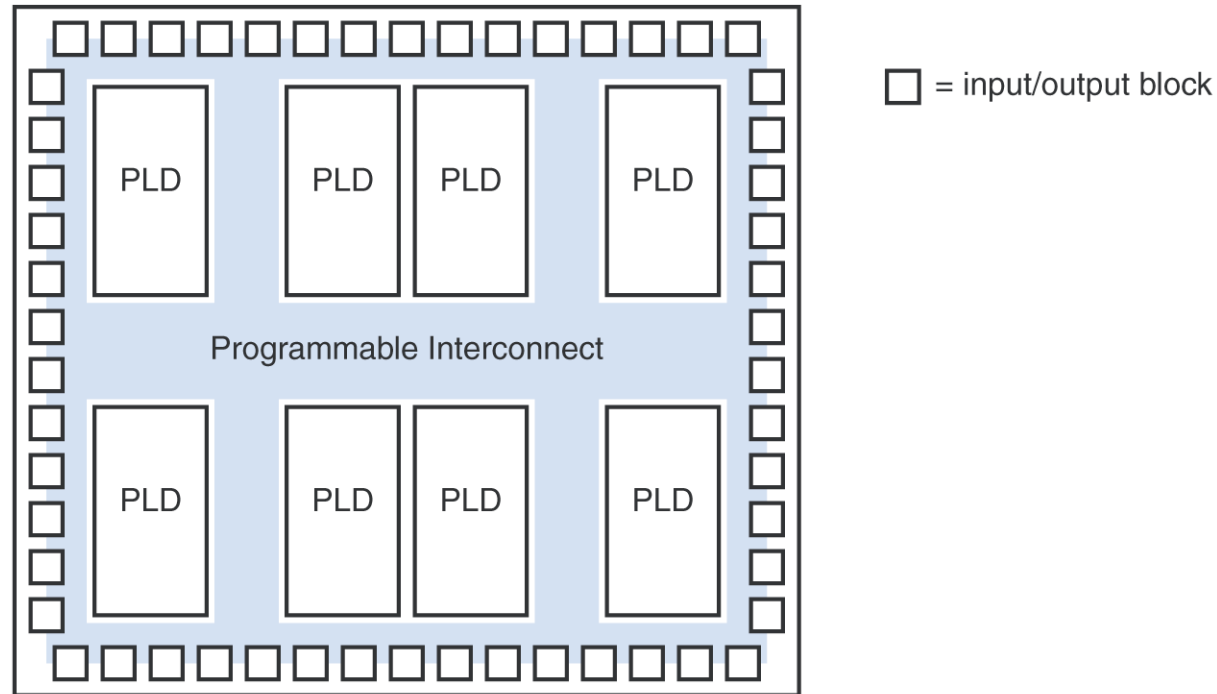


Figure 6-28  
General CPLD architecture.

**Complex PLD**

# 4\*3 PLA in CMOS logic

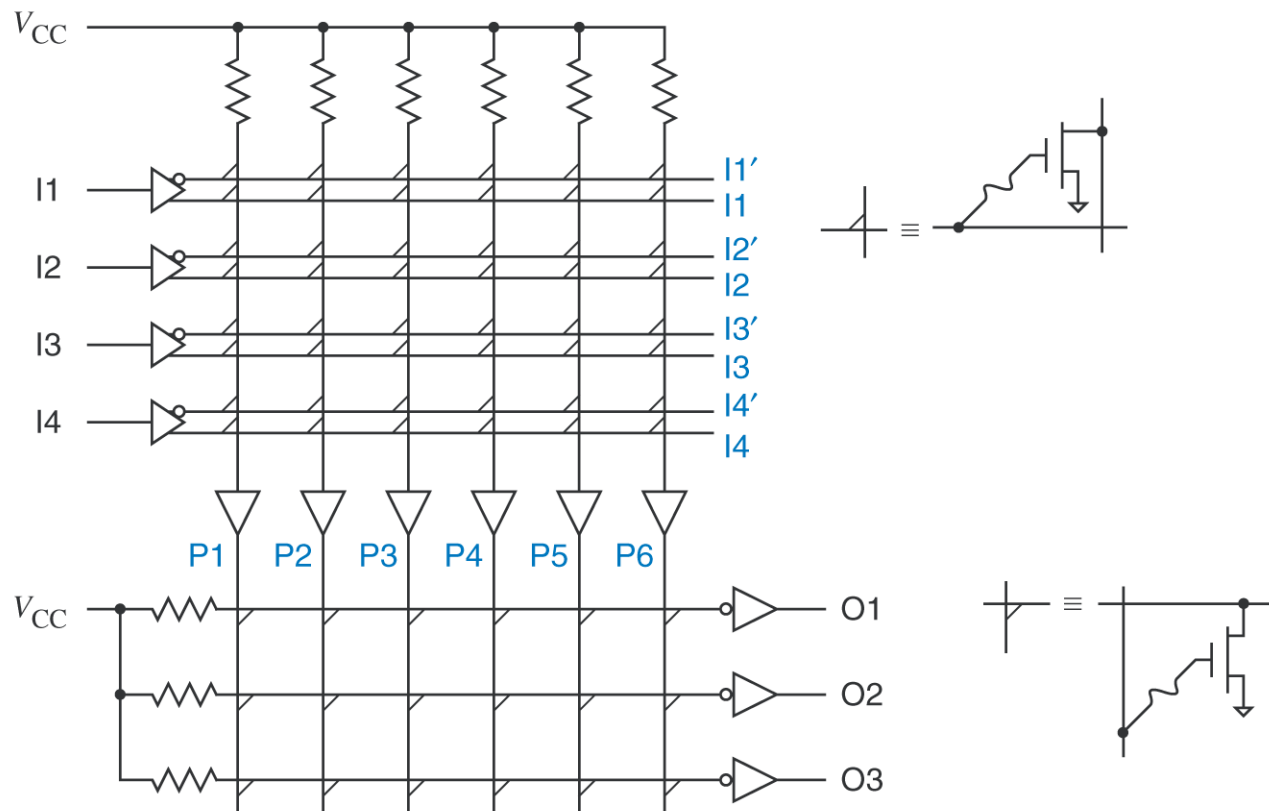
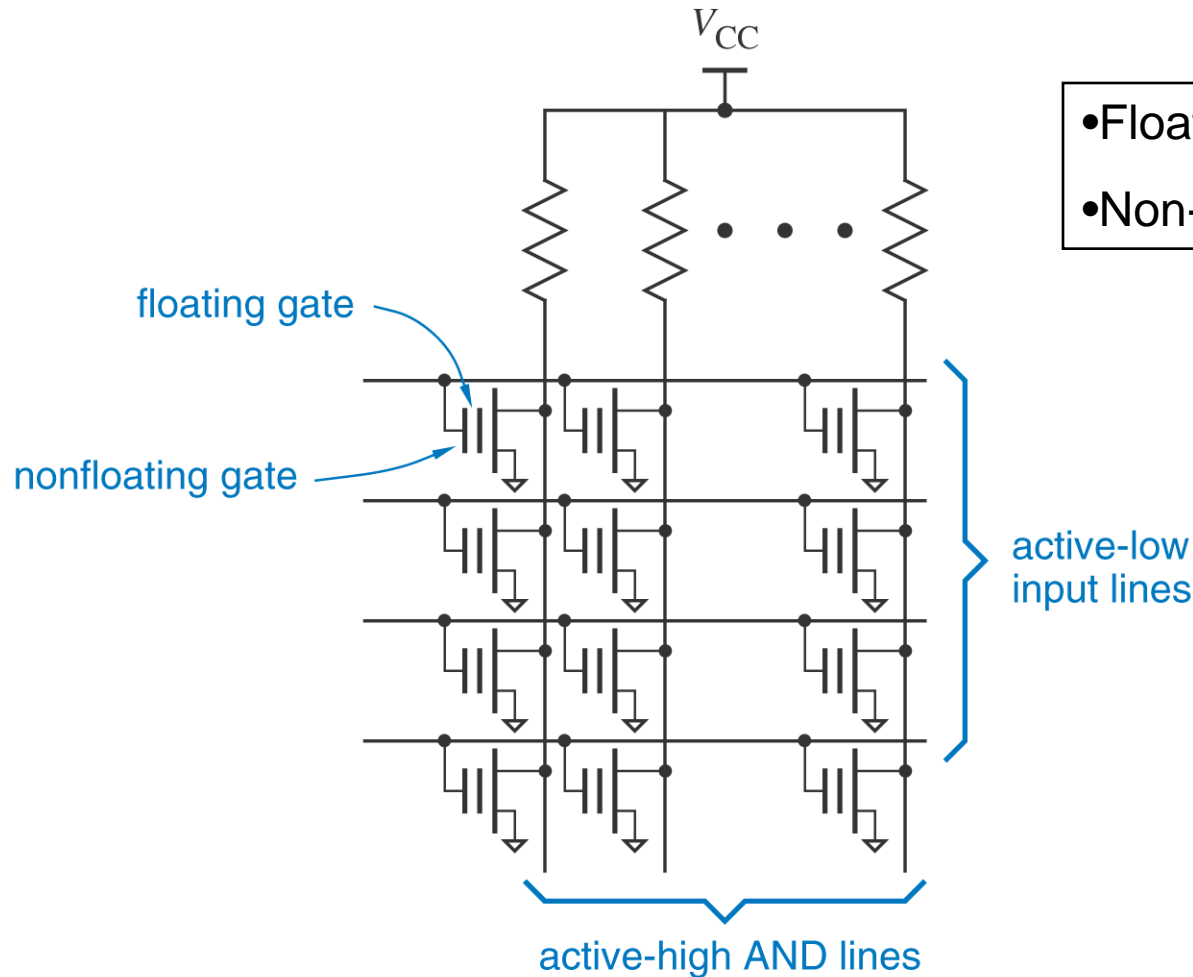


Figure 6-29

A  $4 \times 3$  PLA built using CMOS logic.

# EEPLDs



**AND plane of EEPLD using floating-gate MOS transistors**

# Decoders



# General Structure of a Decoder circuit

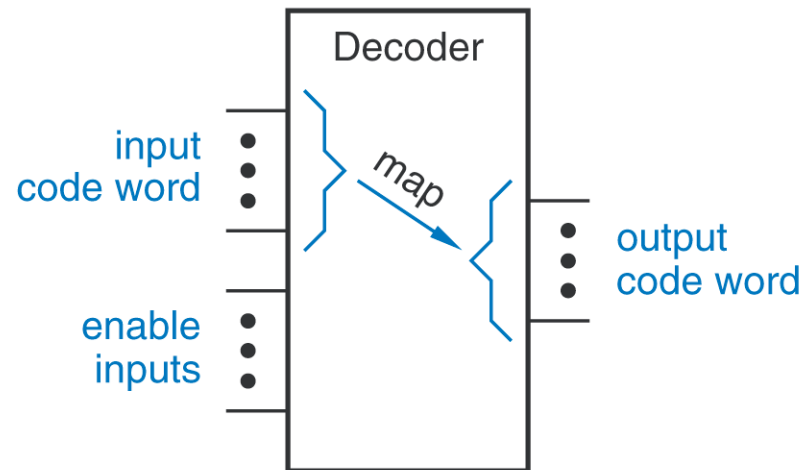



Figure 6-31

Decoder circuit structure.

# Example 1: Truth table for a 2-to-4 binary decoder.

*Example of a decoder circuit*

enable

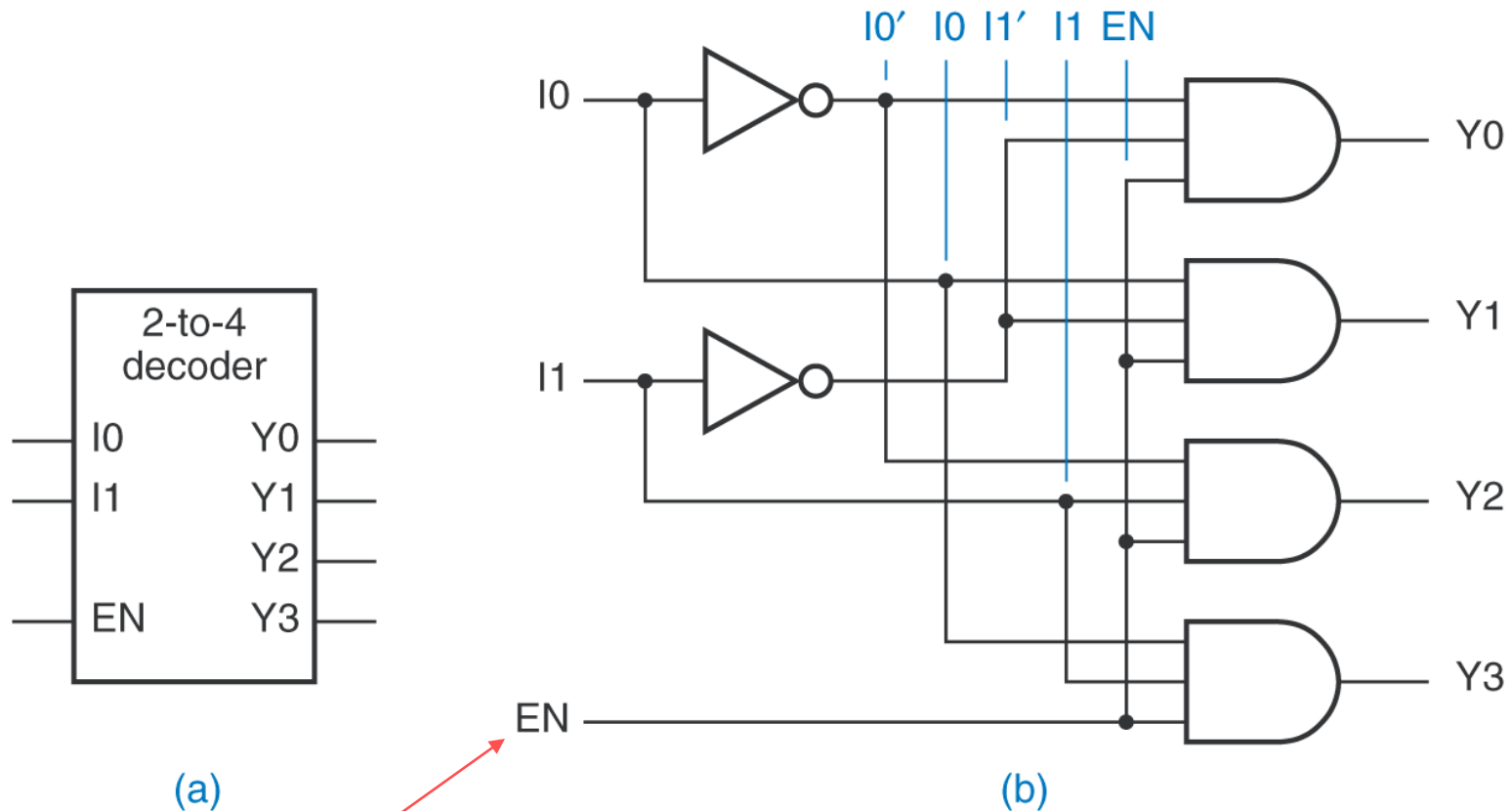


| <i>Inputs</i> |    |    | <i>Outputs</i> |    |    |    |
|---------------|----|----|----------------|----|----|----|
| EN            | I1 | I0 | Y3             | Y2 | Y1 | Y0 |
| 0             | x  | x  | 0              | 0  | 0  | 0  |
| 1             | 0  | 0  | 0              | 0  | 0  | 1  |
| 1             | 0  | 1  | 0              | 0  | 1  | 0  |
| 1             | 1  | 0  | 0              | 1  | 0  | 0  |
| 1             | 1  | 1  | 1              | 0  | 0  | 0  |

Table 6-4

Truth table for a 2-to-4 binary decoder.

# 2-to-4 decoder inside



**enable**

Figure 6-32

A 2-to-4 decoder: (a) inputs and outputs; (b) logic diagram.

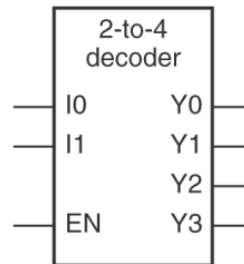
# Verilog for 2-to-4 decoder

```
module Vr2to4dec(I0, I1, EN, Y0, Y1, Y2, Y3);  
  input I0, I1, EN;  
  output Y0, Y1, Y2, Y3;  
  wire NOTI0, NOTI1;  
  
  INV U1 (NOTI0, I0);  
  INV U2 (NOTI1, I1);  
  AND3 U3 (Y0, NOTI0, NOTI1, EN);  
  AND3 U4 (Y1, I0, NOTI1, EN);  
  AND3 U5 (Y2, NOTI0, I1, EN);  
  AND3 U6 (Y3, I0, I1, EN);  
endmodule
```

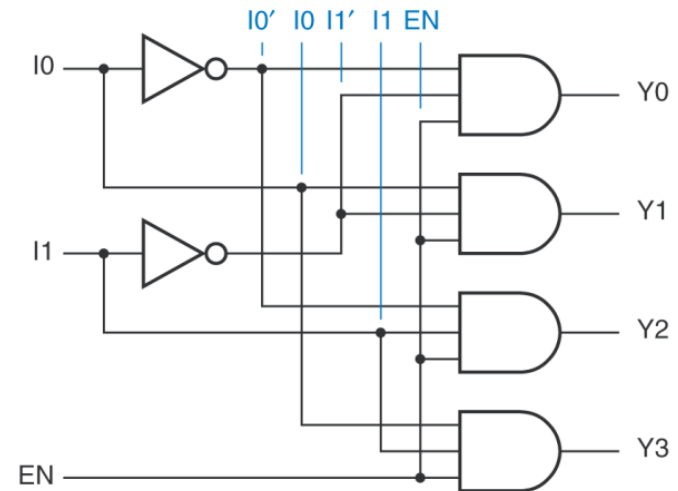
**Structural type  
of description in  
Verilog**

Table 6-20

Structural-style Verilog module for the decoder in Figure 6-32.



(a)



(b)

# Example 2: Position encoding for a 3-bit mechanical encoding disk

| <i>Disk Position</i> | I2 | I1 | I0 | <i>Binary Decoder Output</i> |
|----------------------|----|----|----|------------------------------|
| 0°                   | 0  | 0  | 0  | Y0                           |
| 45°                  | 0  | 0  | 1  | Y1                           |
| 90°                  | 0  | 1  | 1  | Y3                           |
| 135°                 | 0  | 1  | 0  | Y2                           |
| 180°                 | 1  | 1  | 0  | Y6                           |
| 225°                 | 1  | 1  | 1  | Y7                           |
| 270°                 | 1  | 0  | 1  | Y5                           |
| 315°                 | 1  | 0  | 0  | Y4                           |

Table 6-5

Position encoding for a 3-bit mechanical encoding disk.

# Example 2 continued: Using a 3-to-8 decoder to decode a Gray code.

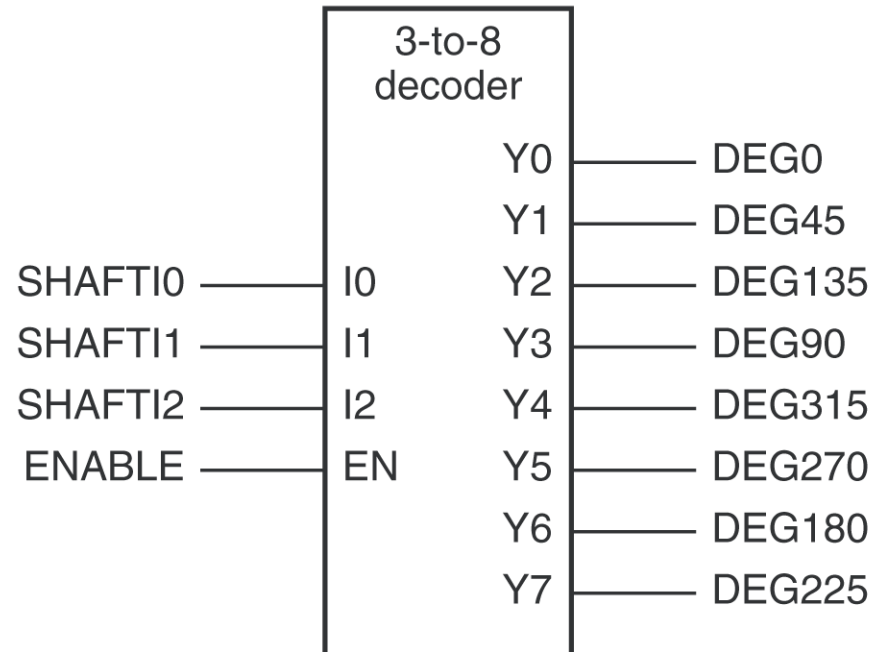


Figure 6-33

Using a 3-to-8 binary decoder to decode a Gray code.

# Example 3:

74x138 3-to-8  
decoder

# Example 3: 74x138 3-to-8 decoder

| <i>Inputs</i> |       |       |   |   |   | <i>Outputs</i> |      |      |      |      |      |      |      |
|---------------|-------|-------|---|---|---|----------------|------|------|------|------|------|------|------|
| G1            | G2A_L | G2B_L | C | B | A | Y7_L           | Y6_L | Y5_L | Y4_L | Y3_L | Y2_L | Y1_L | Y0_L |
| 0             | x     | x     | x | x | x | 1              | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| x             | 1     | x     | x | x | x | 1              | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| x             | x     | 1     | x | x | x | 1              | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| 1             | 0     | 0     | 0 | 0 | 0 | 1              | 1    | 1    | 1    | 1    | 1    | 1    | 0    |
| 1             | 0     | 0     | 0 | 0 | 1 | 1              | 1    | 1    | 1    | 1    | 1    | 0    | 1    |
| 1             | 0     | 0     | 0 | 1 | 0 | 1              | 1    | 1    | 1    | 1    | 0    | 1    | 1    |
| 1             | 0     | 0     | 0 | 1 | 1 | 1              | 1    | 1    | 1    | 0    | 1    | 1    | 1    |
| 1             | 0     | 0     | 1 | 0 | 0 | 1              | 1    | 1    | 0    | 1    | 1    | 1    | 1    |
| 1             | 0     | 0     | 1 | 0 | 1 | 1              | 1    | 0    | 1    | 1    | 1    | 1    | 1    |
| 1             | 0     | 0     | 1 | 1 | 0 | 1              | 0    | 1    | 1    | 1    | 1    | 1    | 1    |
| 1             | 0     | 0     | 1 | 1 | 1 | 0              | 1    | 1    | 1    | 1    | 1    | 1    | 1    |

Table 6-6

Truth table for a 74x138 3-to-8 decoder.



# Example 3 cont: 74X138 3-to-8 decoder

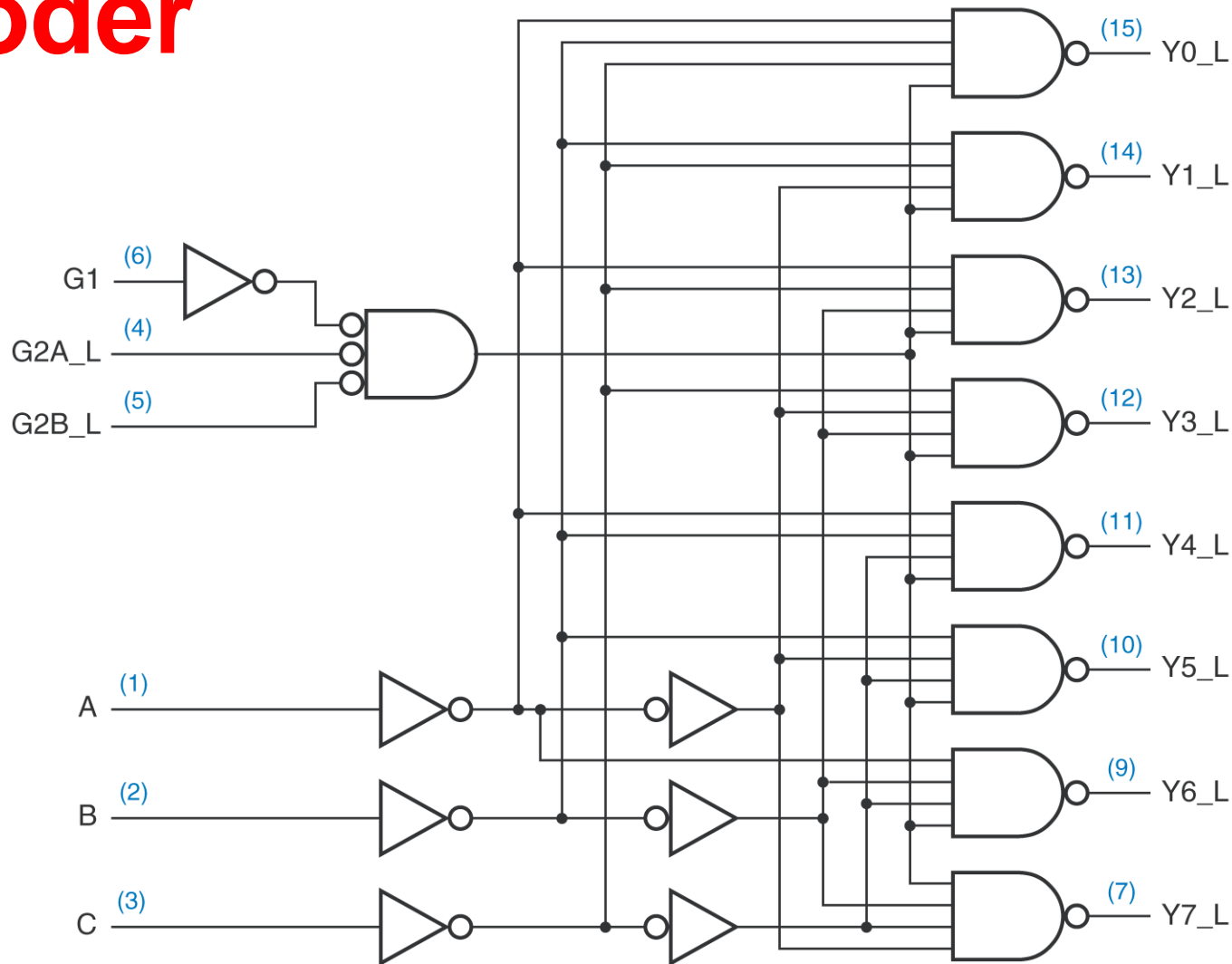


Figure 6-35

Logic diagram for the 74X138 3-to-8 decoder

# Verilog code for 3-to-8 decoder

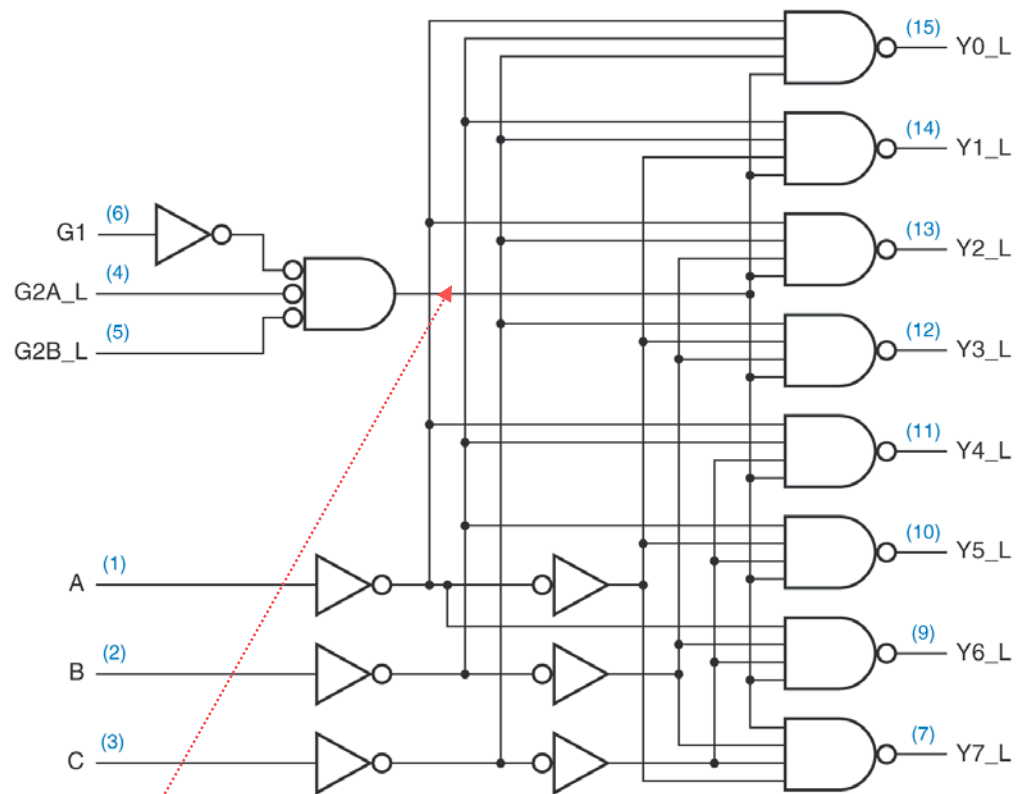


Figure 6-35

Logic diagram for the 74x138 3-to-8 decoder

```
" Input and output pins
A, B, C, !G2A, !G2B, G1
!Y0, !Y1, !Y2, !Y3, !Y4, !Y5, !Y6, !Y7
```

```
pin 1, 2, 3, 4, 5, 6;
pin 15..7 istype 'com';
```

```
" Constant expression
ENB = G1 & G2A & G2B;
```

# Verilog for 3-to-8 decoder

```
module Vr74x138a(G1, G2A_L, G2B_L, A, Y_L);
  input G1, G2A_L, G2B_L;
  input [2:0] A;
  output [0:7] Y_L;
  reg [0:7] Y_L;

  always @ (G1 or G2A_L or G2B_L or A) begin
    if (G1 & ~G2A_L & ~G2B_L)
      case (A)
        0: Y_L = 8'b01111111;
        1: Y_L = 8'b10111111;
        2: Y_L = 8'b11011111;
        3: Y_L = 8'b11101111;
        4: Y_L = 8'b11110111;
        5: Y_L = 8'b11111011;
        6: Y_L = 8'b11111101;
        7: Y_L = 8'b11111110;
        default: Y_L = 8'b11111111;
      endcase
    else Y_L = 8'b11111111;
  end
endmodule
```

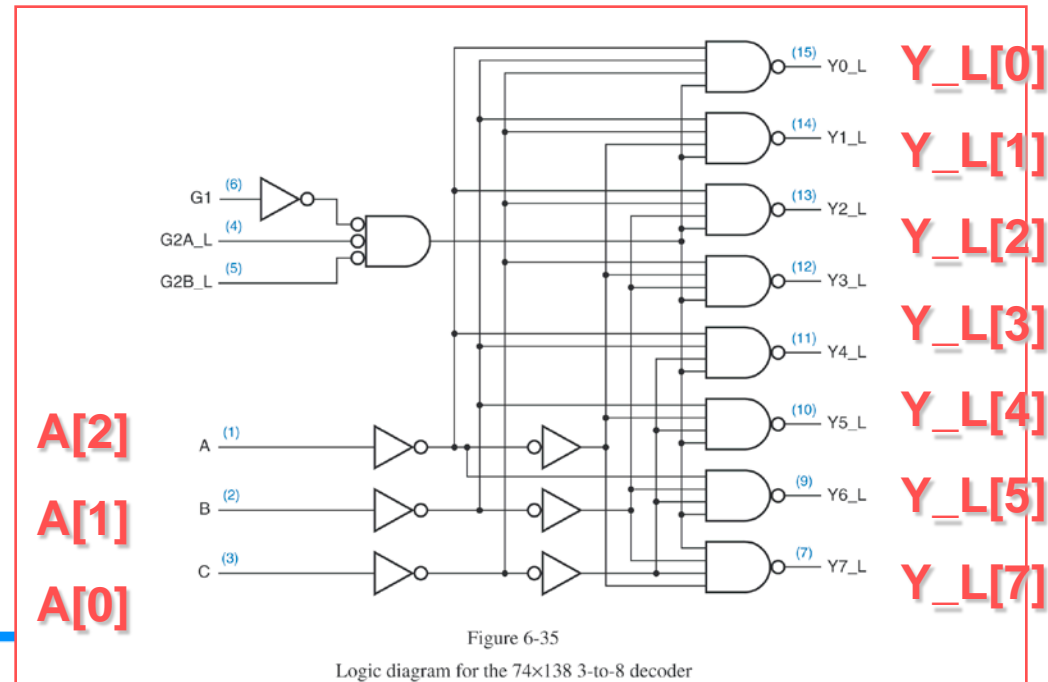


Table 6-21

Functional-style Verilog module for a 74x138-like 3-to-8 binary decoder.

```

module Vr74x138b(G1, G2A_L, G2B_L, A, Y_L);
    input G1, G2A_L, G2B_L;
    input [2:0] A;
    output [0:7] Y_L;
    reg G2A,G2B;
    reg [0:7] Y_L, Y;

    always @ (G1 or G2A_L or G2B_L or A or Y) begin
        G2A = ~G2A_L; // Convert inputs
        G2B = ~G2B_L;
        Y_L = ~Y;      // Convert outputs
        if (G1 & G2A & G2B)
            case (A)
                0: Y = 8'b10000000;
                1: Y = 8'b01000000;
                2: Y = 8'b00100000;
                3: Y = 8'b00010000;
                4: Y = 8'b00001000;
                5: Y = 8'b00000100;
                6: Y = 8'b00000010;
                7: Y = 8'b00000001;
                default: Y = 8'b00000000;
            endcase
        else Y = 8'b00000000;
    end
endmodule

```

# 74X138 Decoder: Active level handling

Table 6-22

Verilog module with a maintainable approach to active-level handling.

# 74x138 like decoder with Active level handling

---

```
module Vr74x138c(G1, G2A_L, G2B_L, A, Y_L);
  input G1, G2A_L, G2B_L;
  input [2:0] A;
  output [0:7] Y_L;
  wire G2A,G2B;
  wire [0:7] Y;

  assign G2A = ~G2A_L; // Convert inputs
  assign G2B = ~G2B_L;
  assign Y_L = ~Y;      // Convert outputs
  Vr3to8deca U1 (G1, G2A, G2B, A, Y);
endmodule
```

---

Table 6-23

Hierarchical definition of 74×138-like decoder with active-level handling.

# Active high 3-to-8 decoder

```

module Vr3to8deca(G1, G2, G3, A, Y);
  input G1, G2, G3;
  input [2:0] A;
  output [0:7] Y;
  reg [0:7] Y;

  always @ (G1 or G2 or G3 or A) begin
    if (G1 & G2 & G3)
      case (A)
        0: Y = 8'b10000000;
        1: Y = 8'b01000000;
        2: Y = 8'b00100000;
        3: Y = 8'b00010000;
        4: Y = 8'b00001000;
        5: Y = 8'b00000100;
        6: Y = 8'b00000010;
        7: Y = 8'b00000001;
        default: Y = 8'b00000000;
      endcase
    else Y = 8'b00000000;
  end
endmodule

```

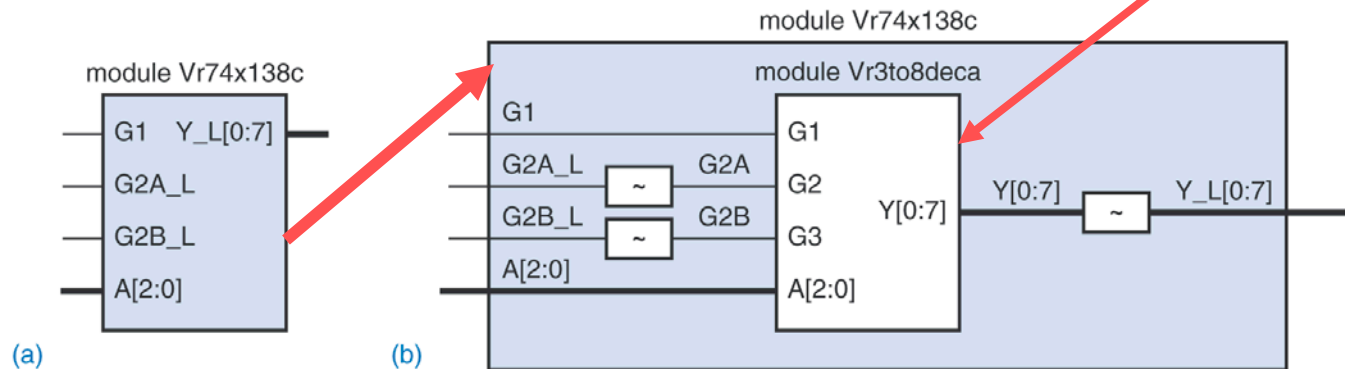


Figure 6-43

9-4.

Verilog module V74x138c: (a) top level; (b) internal structure using module Vr3to8deca.

# Behavioral Verilog for 3-to-8 decoder

---

```
module Vr3to8decb(G1, G2, G3, A, Y);  
    input G1, G2, G3;  
    input [2:0] A;  
    output [0:7] Y;  
    reg [0:7] Y;  
    integer i;  
  
    always @ (G1 or G2 or G3 or A) begin  
        Y = 8'b00000000;  
        if (G1 & G2 & G3)  
            for (i=0; i<=7; i=i+1)  
                if (i == A) Y[i] = 1;  
    end  
endmodule
```

---

Table 6-25

Behavioral Verilog definition for a 3-to-8 decoder.

# Example 3 cont: 74x138 3-to-8 decoder

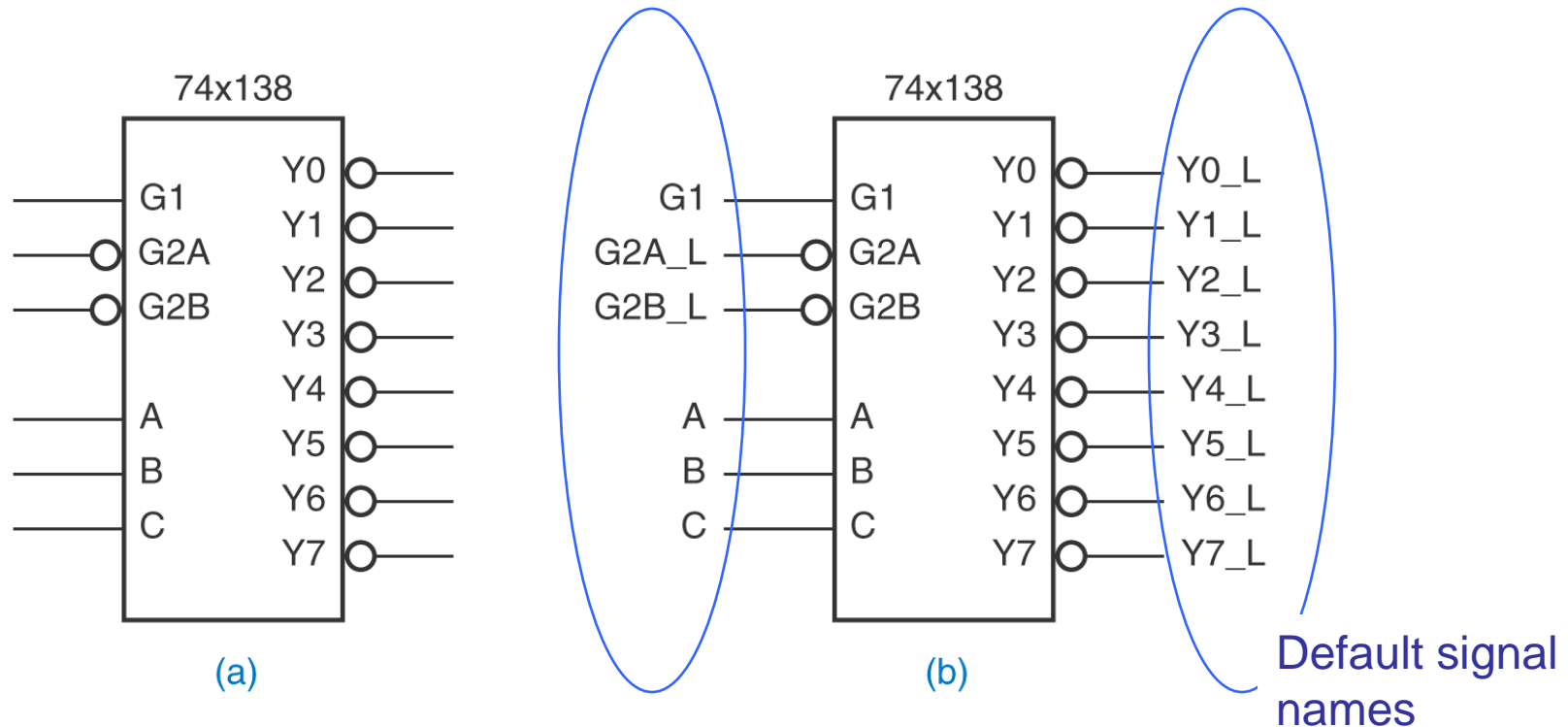


Figure 6-34

Logic symbol for the 74x138 3-to-8 decoder: (a) conventional symbol;  
(b) default signal names associated with external pins.



# Example 3 cont: Symbols for 74x138

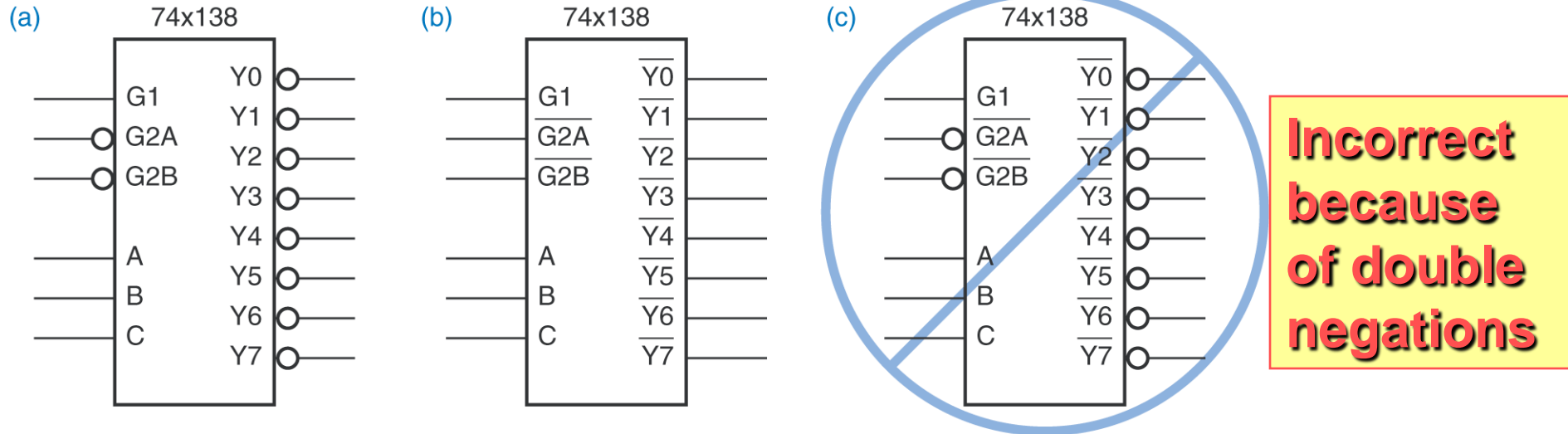
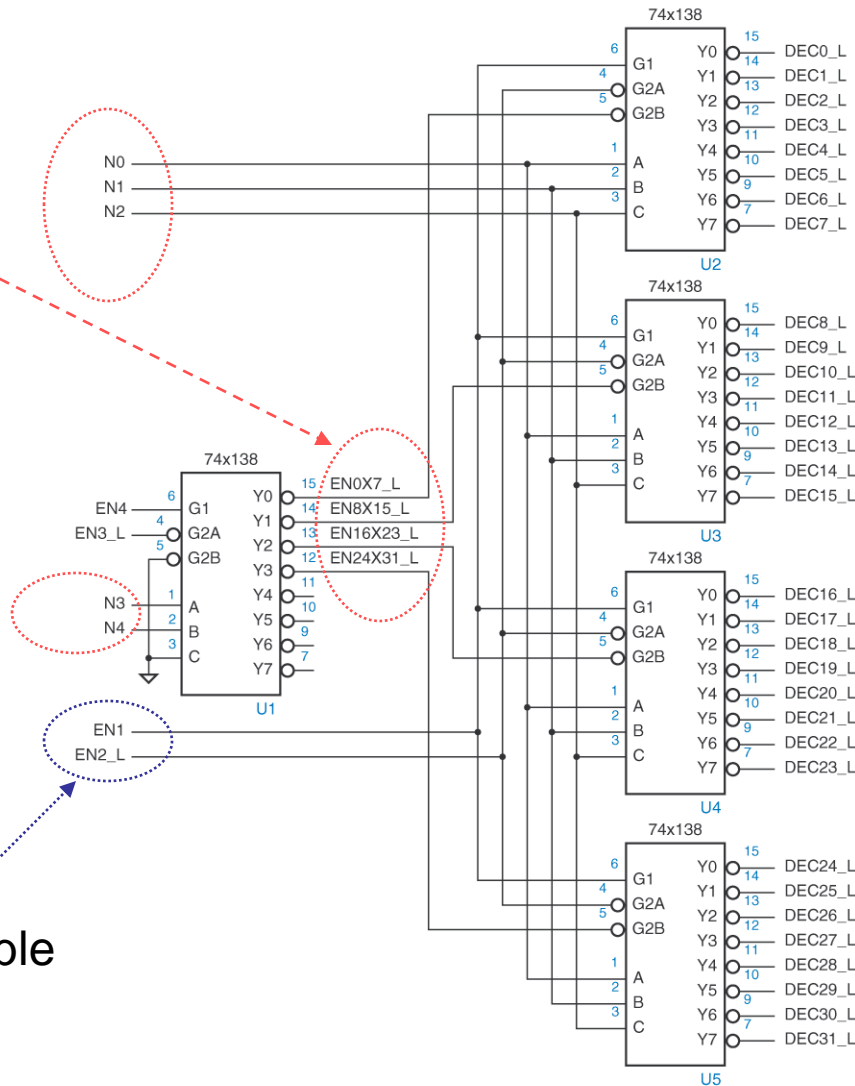


Figure 6-36

Logic symbols for the 74×138: (a) preferred symbol; (b) correct but to be avoided; (c) incorrect because of double negations.

# Example 3 cont: 5-to-32 decoder from 74X138 chips

Chip select  
goes to input  
G2B



Global enable  
goes to  
inputs G1  
and G2A

Figure 6-37

Design of a 5-to-32 decoder using 74x138s.

# Example 3 cont: 4-to-16 decoder using 74X138

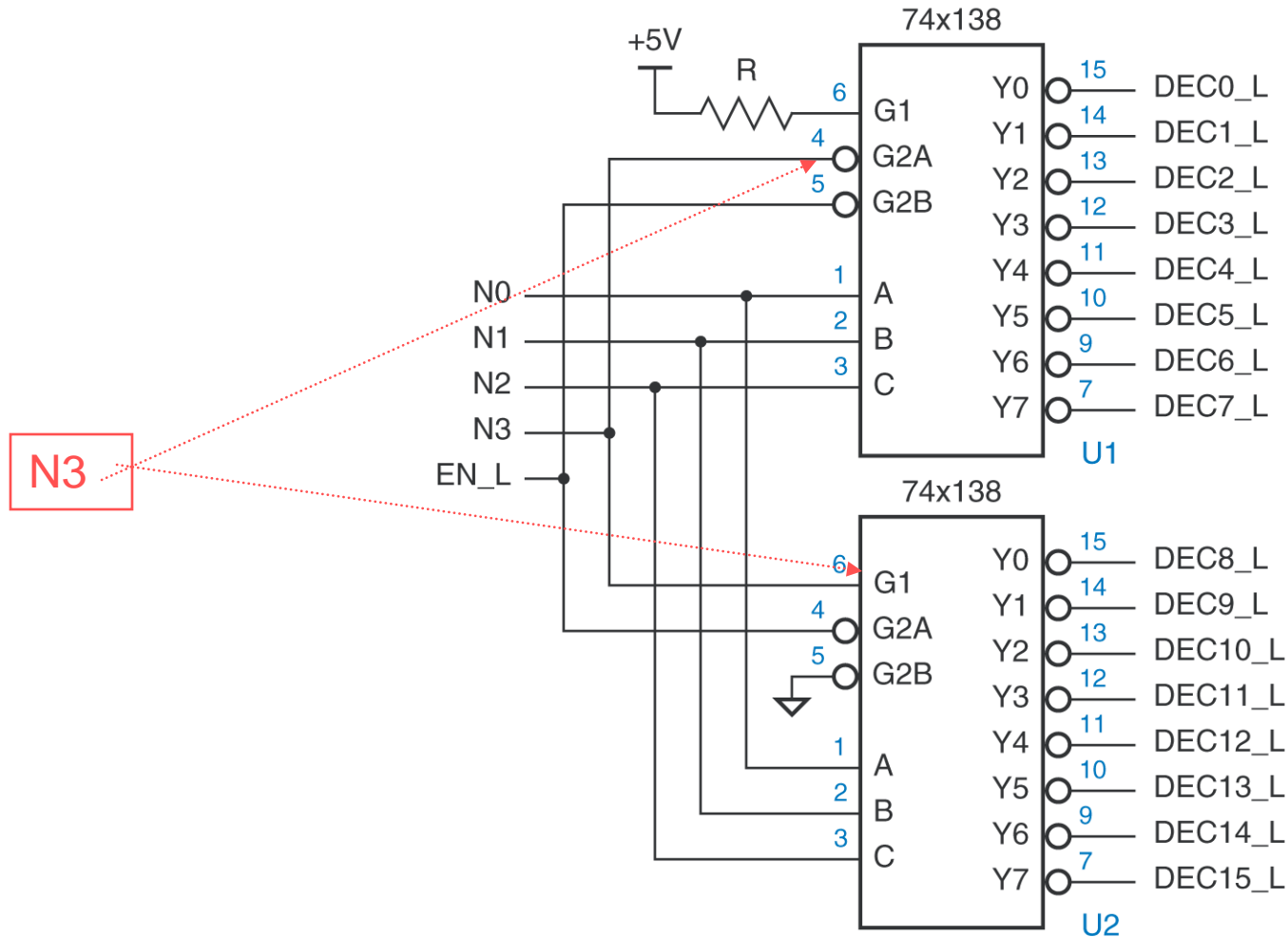


Figure 6-38

Design of a 4-to-16 decoder using 74x138s.

# Example 3 cont: 74x138 decoder using GAL

**74x138 decoder can be built in single GAL 16V8 chip**

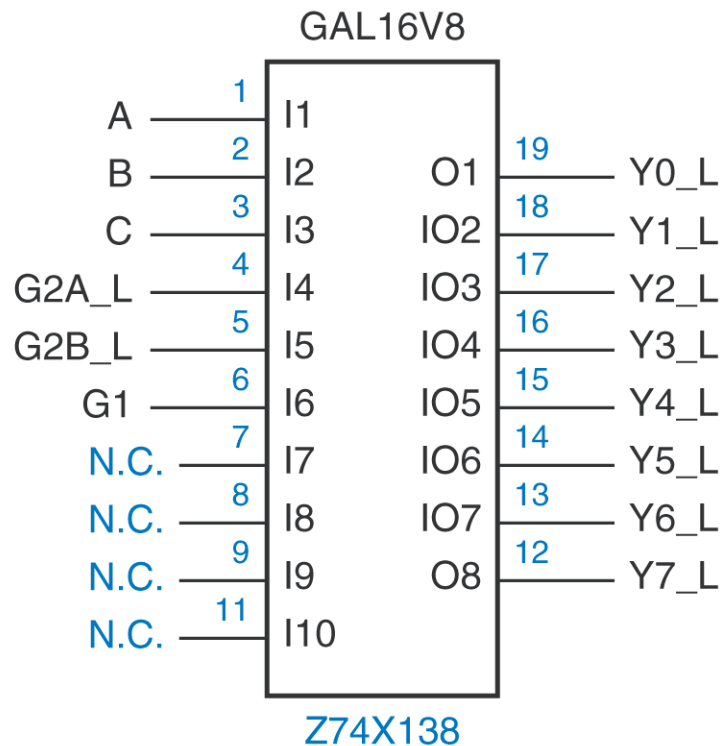


Figure 6-39

Logic diagram for the GAL16V8C used as a 74x138 decoder.

# Figure 6-41. 74x138-like decoder

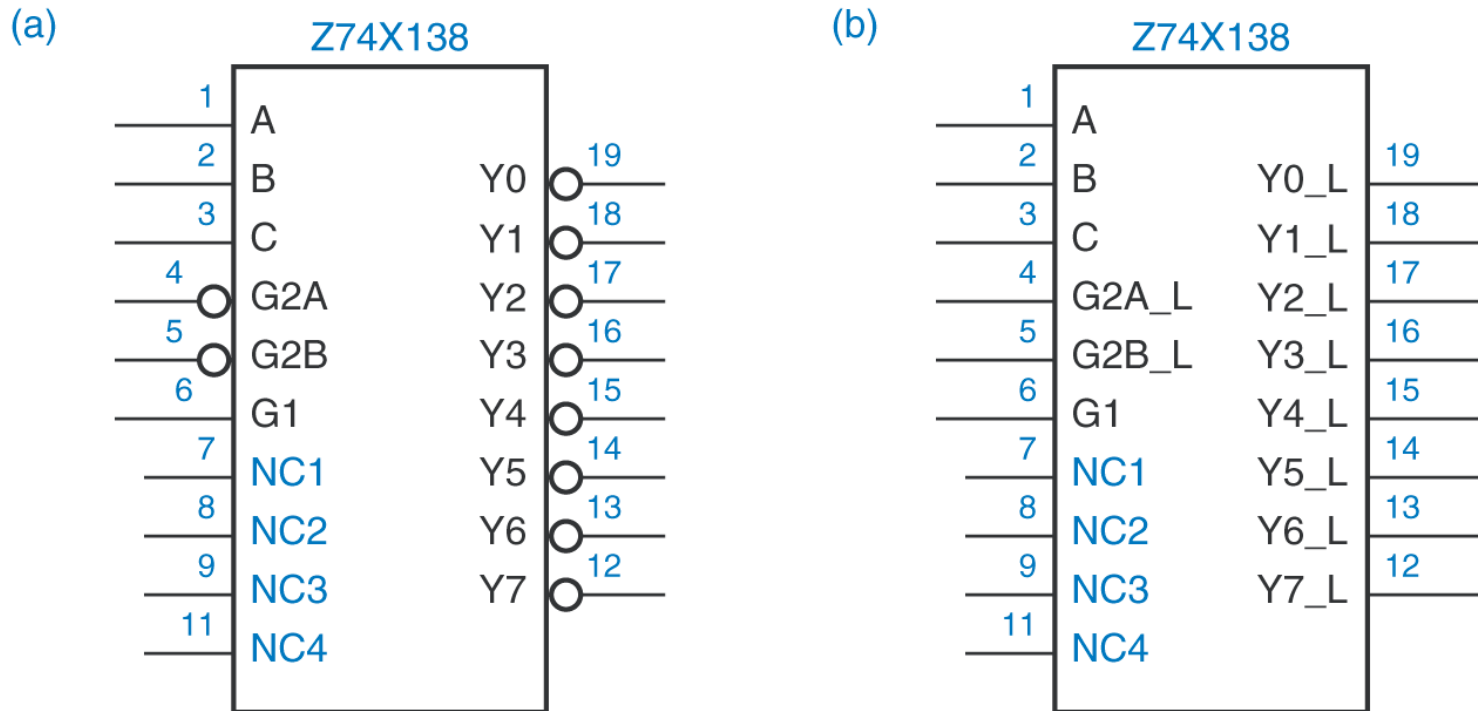


Figure 6-41

Possible CAD-created symbols for the PLD-based, 74×138-like decoder:

(a) based on Table 6-12, after manual insertion of inversion bubbles; (b) based on Table 6-7.

# Example: Customized decoder function

| CS_L | RD_L | A2 | A1 | A0 | <i>Output(s) to Assert</i> |
|------|------|----|----|----|----------------------------|
| 1    | x    | x  | x  | x  | none                       |
| x    | 1    | x  | x  | x  | none                       |
| 0    | 0    | 0  | 0  | 0  | BILL_L, MARY_L             |
| 0    | 0    | 0  | 0  | 1  | MARY_L, KATE_L             |
| 0    | 0    | 0  | 1  | 0  | JOAN_L                     |
| 0    | 0    | 0  | 1  | 1  | PAUL_L                     |
| 0    | 0    | 1  | 0  | 0  | ANNA_L                     |
| 0    | 0    | 1  | 0  | 1  | FRED_L                     |
| 0    | 0    | 1  | 1  | 0  | DAVE_L                     |
| 0    | 0    | 1  | 1  | 1  | KATE_L                     |

Table 6-9

Truth table for a customized decoder function.

# Customized decoder circuit Using 74X138

| CS_L | RD_L | A2 | A1 | A0 | Output(s) to Assert |
|------|------|----|----|----|---------------------|
| 1    | x    | x  | x  | x  | none                |
| x    | 1    | x  | x  | x  | none                |
| 0    | 0    | 0  | 0  | 0  | BILL_L, MARY_L      |
| 0    | 0    | 0  | 0  | 1  | MARY_L, KATE_L      |
| 0    | 0    | 0  | 1  | 0  | JOAN_L              |
| 0    | 0    | 0  | 1  | 1  | PAUL_L              |
| 0    | 0    | 1  | 0  | 0  | ANNA_L              |
| 0    | 0    | 1  | 0  | 1  | FRED_L              |
| 0    | 0    | 1  | 1  | 0  | DAVE_L              |
| 0    | 0    | 1  | 1  | 1  | KATE_L              |

Table 6-9

Truth table for a customized decoder function.

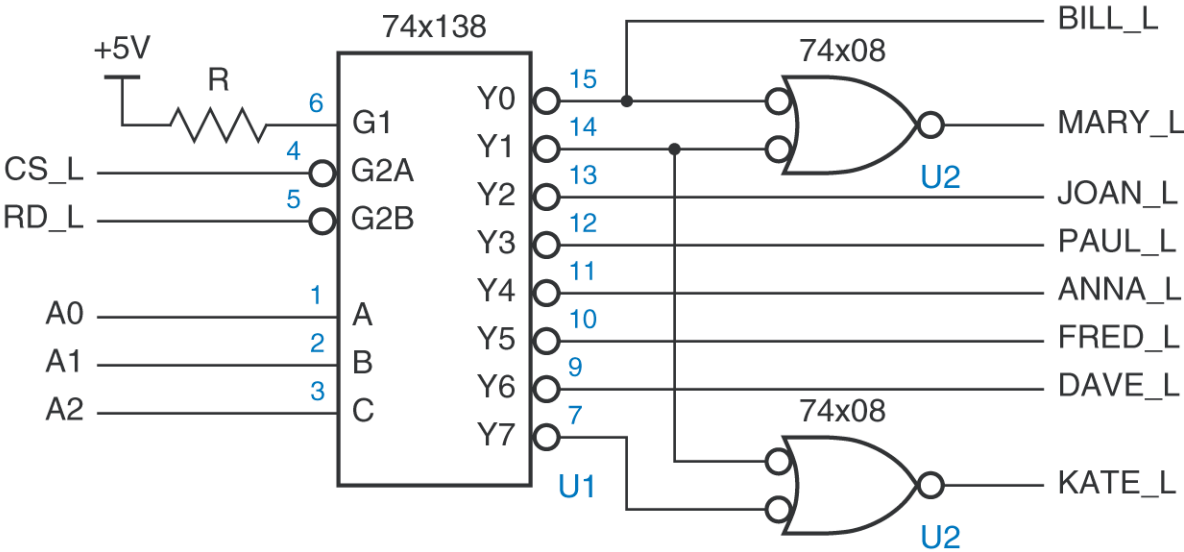


Figure 6-40

Customized decoder circuit

# Seven Segment Display and Decoder



# Seven Segment Display

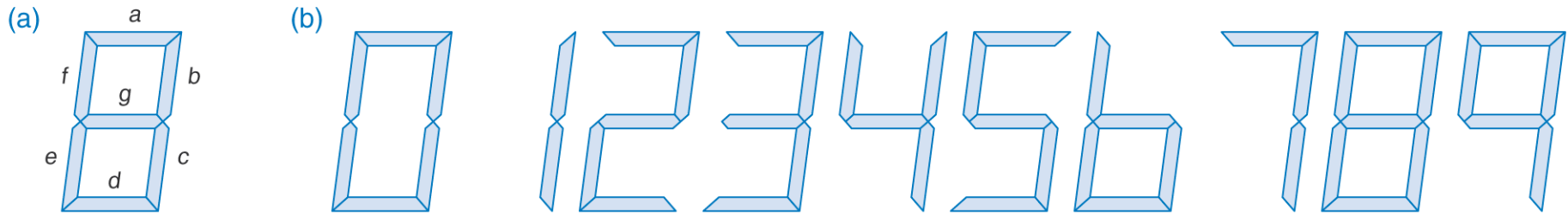


Figure 6-44

Seven-segment display: (a) segment identification; (b) decimal digits.

# Seven Segment Decoder

---

```
module Vr7seg(A, B, C, D, EN,
              SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG);
    input A, B, C, D, EN;
    output SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG;
    reg SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG;
    reg [1:7] SEGS;

    always @ (A or B or C or D or EN) begin
        if (EN)
            case ({D,C,B,A})
                // Segment patterns  abcdefg
                0: SEGS = 7'b1111110; // 0
                1: SEGS = 7'b0110000; // 1
                2: SEGS = 7'b1101101; // 2
                3: SEGS = 7'b1111001; // 3
                4: SEGS = 7'b0110011; // 4
                5: SEGS = 7'b1011011; // 5
                6: SEGS = 7'b0011111; // 6 (no 'tail')
                7: SEGS = 7'b1110000; // 7
                8: SEGS = 7'b1111111; // 8
                9: SEGS = 7'b1110011; // 9 (no 'tail')
                default SEGS = 7'bx;
            endcase
        else SEGS = 7'b0;
        {SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG} = SEGS;
    end
endmodule
```

---

Table 6-26

Verilog program for a seven-segment decoder.

# Encoders

We already used encoders to design control logic for data path blocks

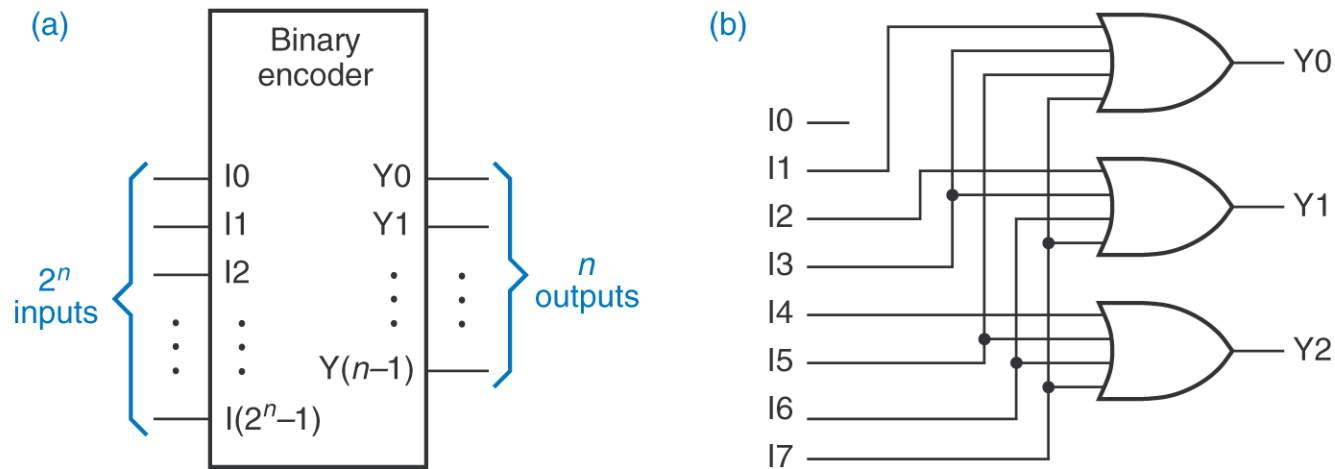


Figure 6-45

Binary encoder: (a) general structure; (b) 8-to-3 encoder.

# Encoders

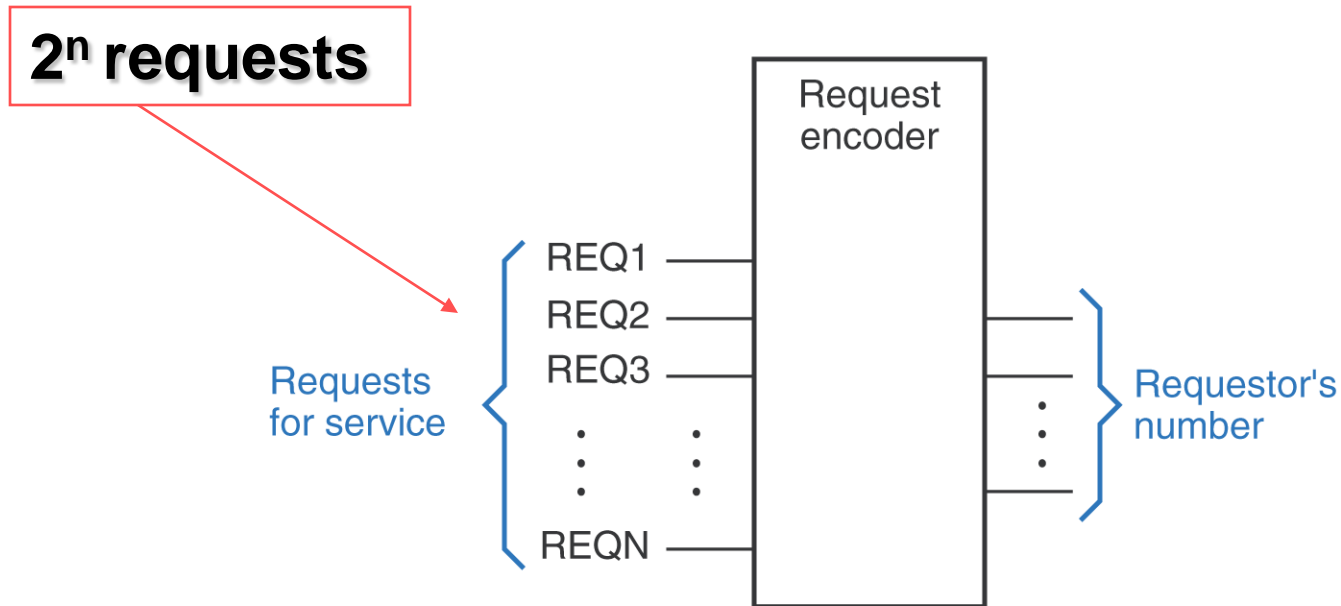


Figure 6-46

A system with  $2^n$  requestors, and a “request encoder” that indicates which request signal is asserted at any time.

# Priority Encoders

# Encoders

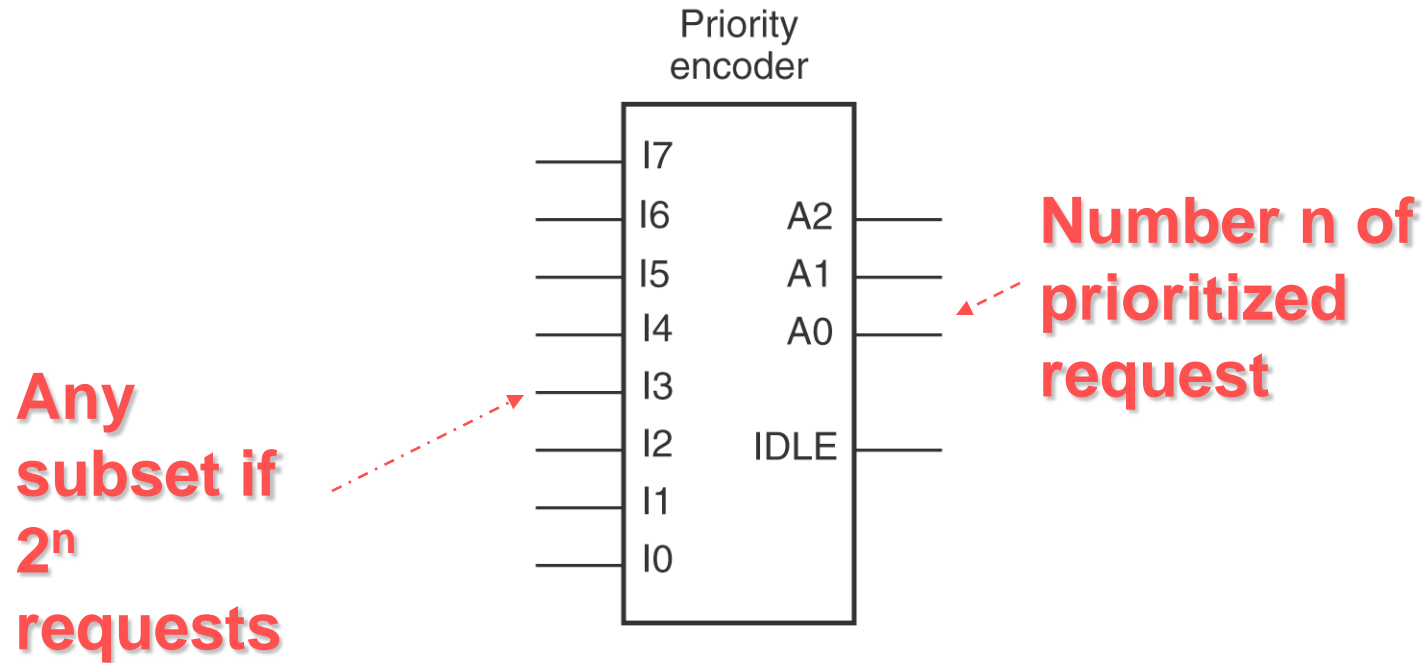


Figure 6-47

Logic symbol for a generic 8-input priority encoder.

# Encoders

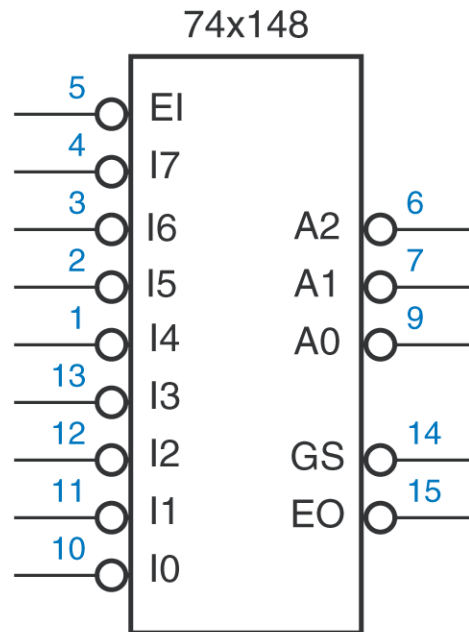


Figure 6-48

Logic symbol for the 74×148 8-input priority encoder.

# 8-input Priority Encoder

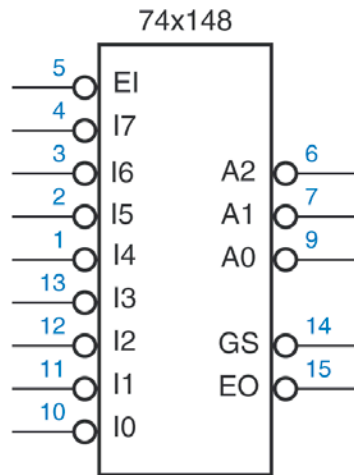


Figure 6-48

Logic symbol for the 74x148 8-input priority encoder.

enable

| Inputs |      |      |      |      |      |      |      |      | Outputs |      |      |      |      |
|--------|------|------|------|------|------|------|------|------|---------|------|------|------|------|
| EI_L   | I0_L | I1_L | I2_L | I3_L | I4_L | I5_L | I6_L | I7_L | A2_L    | A1_L | A0_L | GS_L | EO_L |
| 1      | x    | x    | x    | x    | x    | x    | x    | x    | 1       | 1    | 1    | 1    | 1    |
| 0      | x    | x    | x    | x    | x    | x    | x    | 0    | 0       | 0    | 0    | 0    | 1    |
| 0      | x    | x    | x    | x    | x    | x    | 0    | 1    | 0       | 0    | 1    | 0    | 1    |
| 0      | x    | x    | x    | x    | x    | 0    | 1    | 1    | 0       | 1    | 0    | 0    | 1    |
| 0      | x    | x    | x    | x    | 0    | 1    | 1    | 1    | 0       | 1    | 1    | 0    | 1    |
| 0      | x    | x    | x    | 0    | 1    | 1    | 1    | 1    | 1       | 0    | 0    | 0    | 1    |
| 0      | x    | x    | 0    | 1    | 1    | 1    | 1    | 1    | 1       | 0    | 1    | 0    | 1    |
| 0      | x    | 0    | 1    | 1    | 1    | 1    | 1    | 1    | 1       | 1    | 0    | 0    | 1    |
| 0      | 0    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1       | 1    | 1    | 0    | 1    |
| 0      | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1       | 1    | 1    | 1    | 0    |

Table 6-27

Truth table for a 74x148 8-input priority encoder.



# 15-input Priority Encoder in PLD

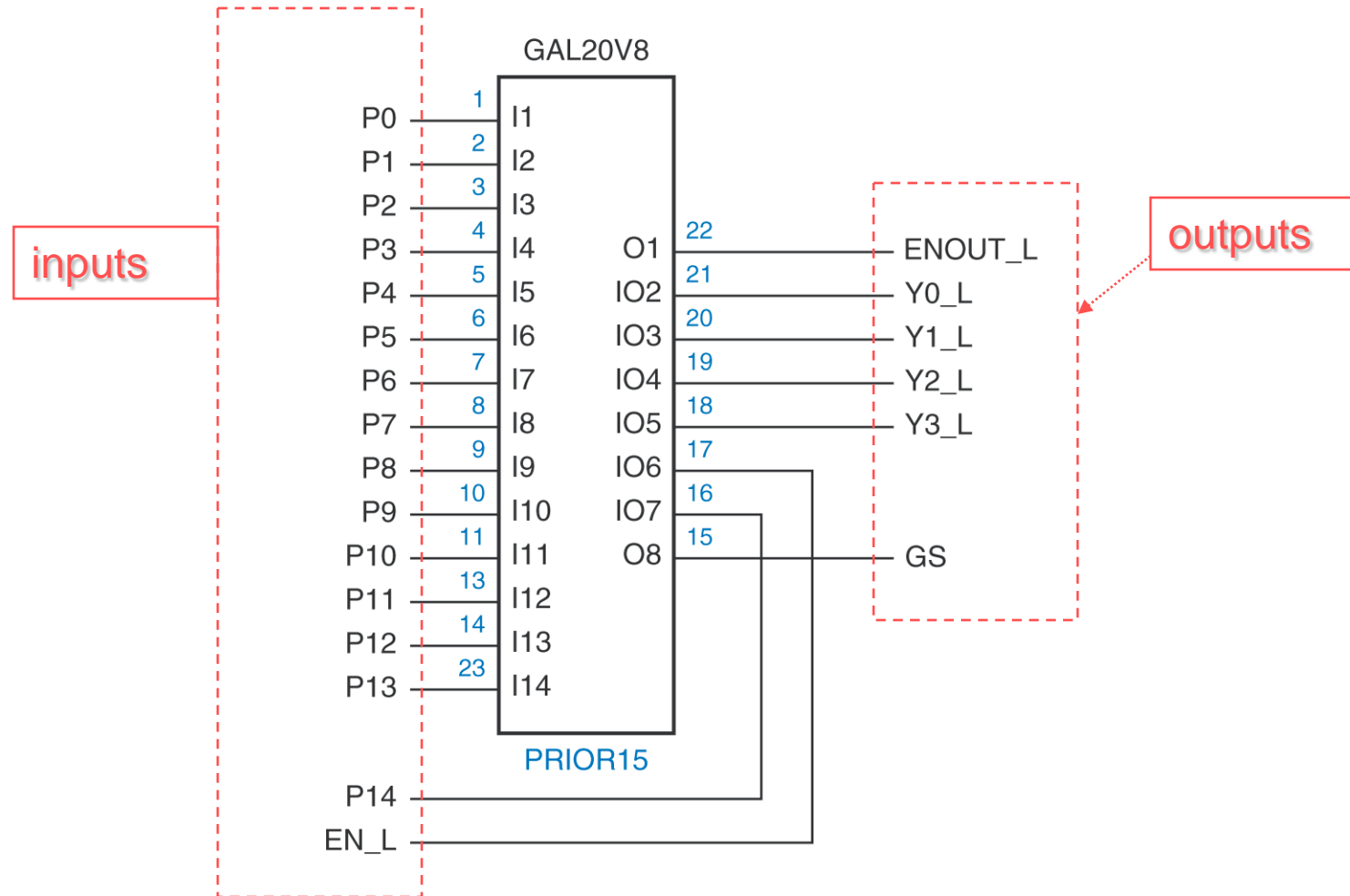


Figure 6-50

Logic diagram for a PLD-based 15-input priority encoder.

# Priority Encoder – handle 32 requests

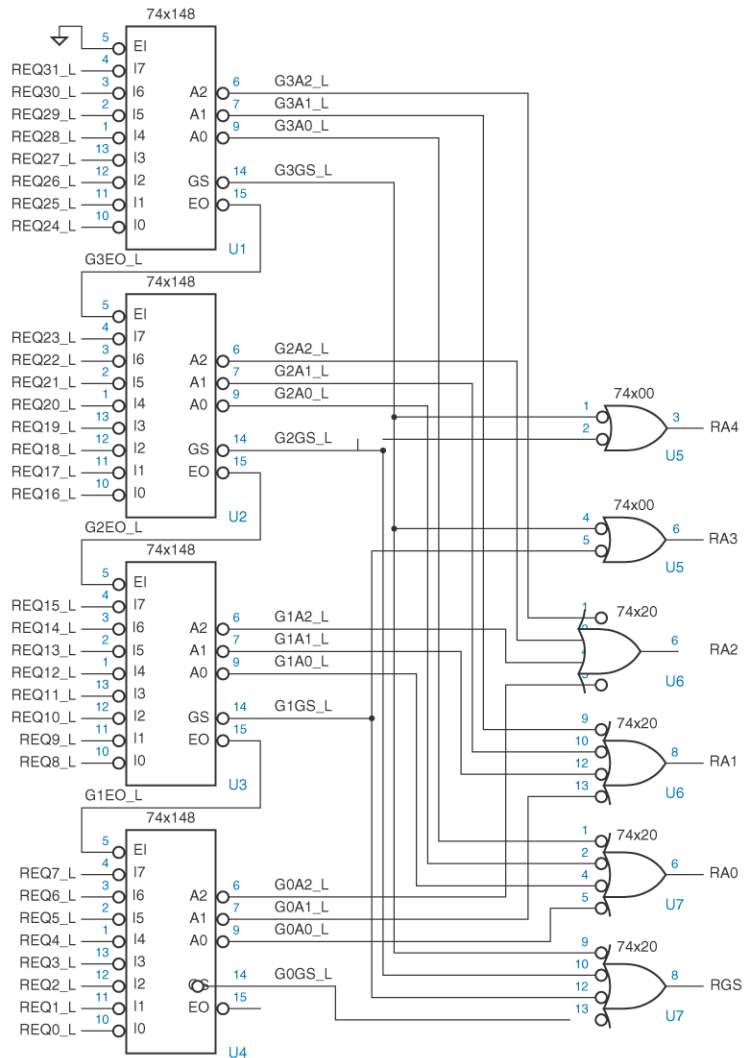


Figure 6-49

Four 74x148s cascaded to handle 32 requests.

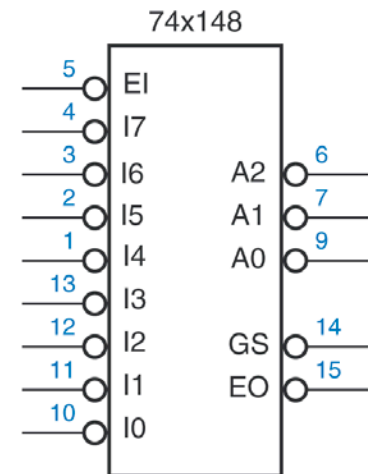


Figure 6-48

Logic symbol for the 74x148 8-input priority encoder.

# 8-input Priority Encoder

```
module Vr74x148(EI_L, I_L, A_L, EO_L, GS_L);
  input EI_L;
  input [7:0] I_L;
  output [2:0] A_L;
  output EO_L, GS_L;
  reg [7:0] I;
  reg [2:0] A, A_L;
  reg EI, EO_L, EO, GS_L, GS;
  integer j;

  always @ (EI_L or EI or I_L or I or A or EO or GS) begin
    EI = ~EI_L; I = ~I_L;           // convert inputs
    EO_L = ~EO; GS_L = ~GS; A_L = ~A; // convert outputs
    EO = 1; GS = 0; A = 0;          // default output values
    begin
      if (EI==0) EO = 0;
      else for (j=0; j<=7; j=j+1) // check low priority first
        if (I[j]==1)
          begin GS = 1; EO = 0; A = j; end
    end
  end
endmodule
```

| Inputs |      |      |      |      |      |      |      |      |  | Outputs |      |      |      |      |
|--------|------|------|------|------|------|------|------|------|--|---------|------|------|------|------|
| EI_L   | I0_L | I1_L | I2_L | I3_L | I4_L | I5_L | I6_L | I7_L |  | A2_L    | A1_L | A0_L | GS_L | EO_L |
| 1      | x    | x    | x    | x    | x    | x    | x    | x    |  | 1       | 1    | 1    | 1    | 1    |
| 0      | x    | x    | x    | x    | x    | x    | x    | 0    |  | 0       | 0    | 0    | 0    | 1    |
| 0      | x    | x    | x    | x    | x    | x    | 0    | 1    |  | 0       | 0    | 1    | 0    | 1    |
| 0      | x    | x    | x    | x    | x    | 0    | 1    | 1    |  | 0       | 1    | 0    | 0    | 1    |
| 0      | x    | x    | x    | x    | 0    | 1    | 1    | 1    |  | 0       | 1    | 1    | 0    | 1    |
| 0      | x    | x    | x    | 0    | 1    | 1    | 1    | 1    |  | 1       | 0    | 0    | 0    | 1    |
| 0      | x    | x    | 0    | 1    | 1    | 1    | 1    | 1    |  | 1       | 0    | 1    | 0    | 1    |
| 0      | x    | 0    | 1    | 1    | 1    | 1    | 1    | 1    |  | 1       | 1    | 0    | 0    | 1    |
| 0      | 0    | 1    | 1    | 1    | 1    | 1    | 1    | 1    |  | 1       | 1    | 1    | 0    | 1    |
| 0      | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    |  | 1       | 1    | 1    | 1    | 0    |

Table 6-31

Behavioral Verilog module for a 74×148-like 8-input priority encoder.

# Three state Buffers

# Various three-state buffers

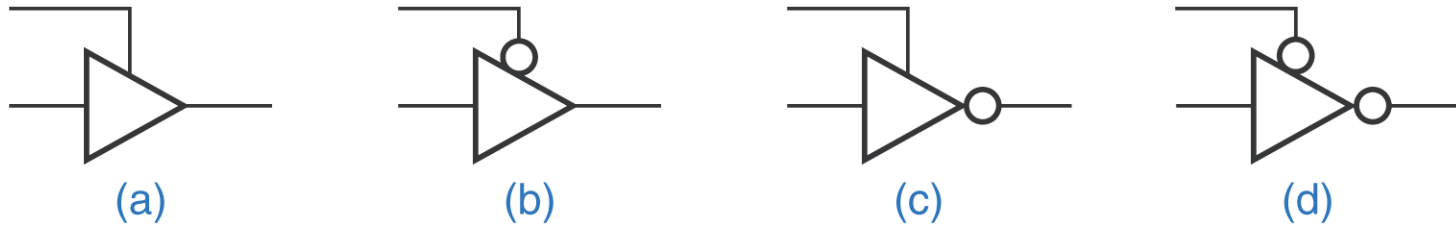


Figure 6-51

Various three-state buffers: (a) noninverting, active-high enable; (b) noninverting, active-low enable; (c) inverting, active-high enable; (d) inverting, active-low enable.

# Use of three-state buffers

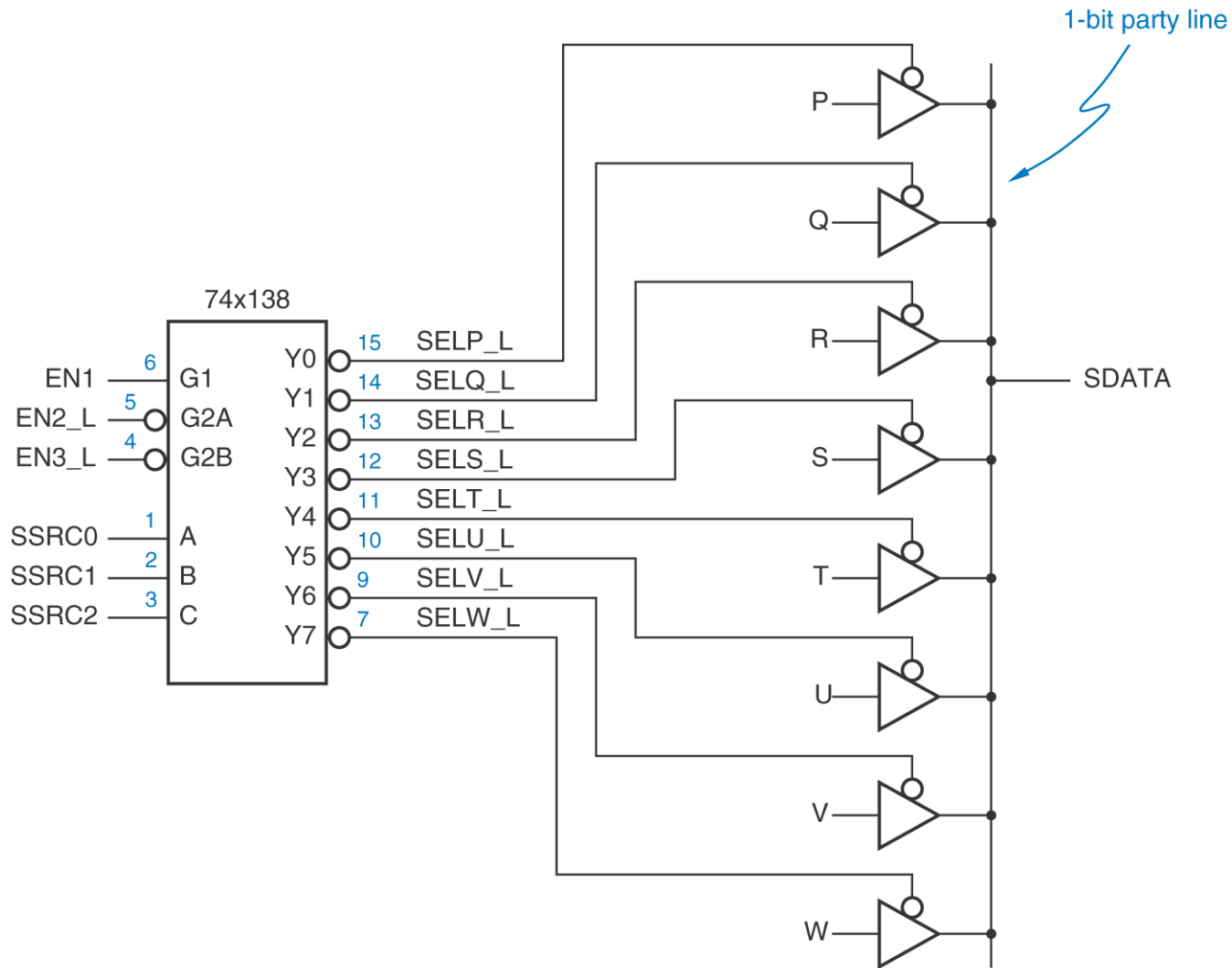


Figure 6-52

**Eight sources sharing a three-state party line**

# Timing diagram for the three-state party line

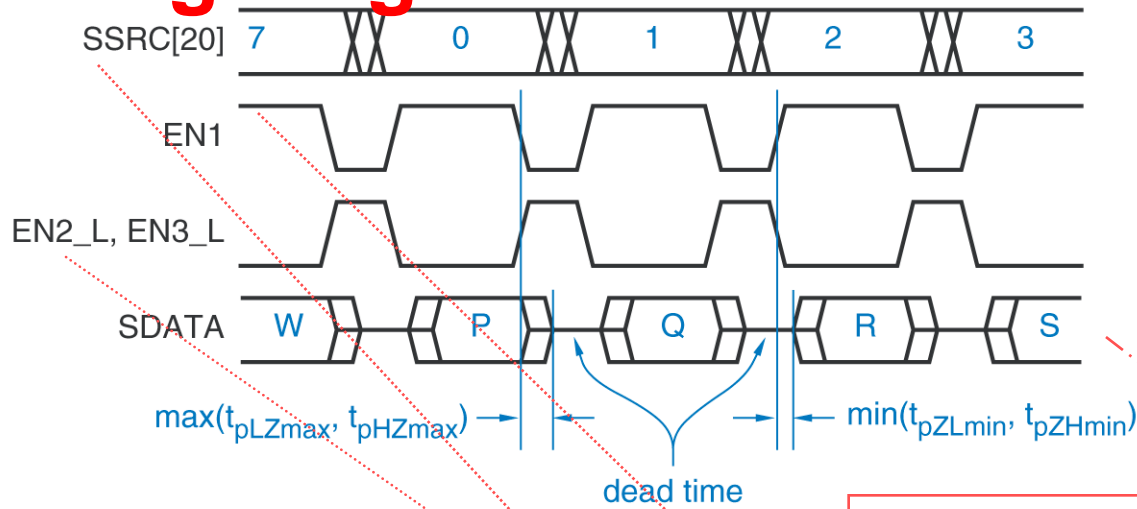


Figure 6-53

Timing diagram for the three-state party

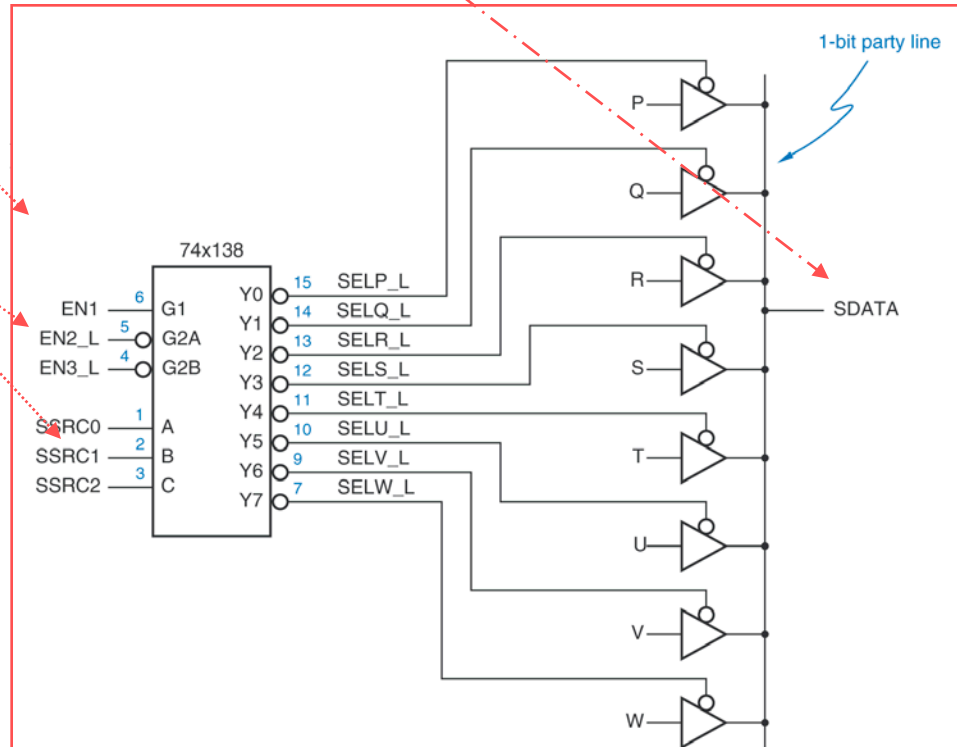


Figure 6-52

Eight sources sharing a three-state party line.

# 74x541 Octal three-state buffer

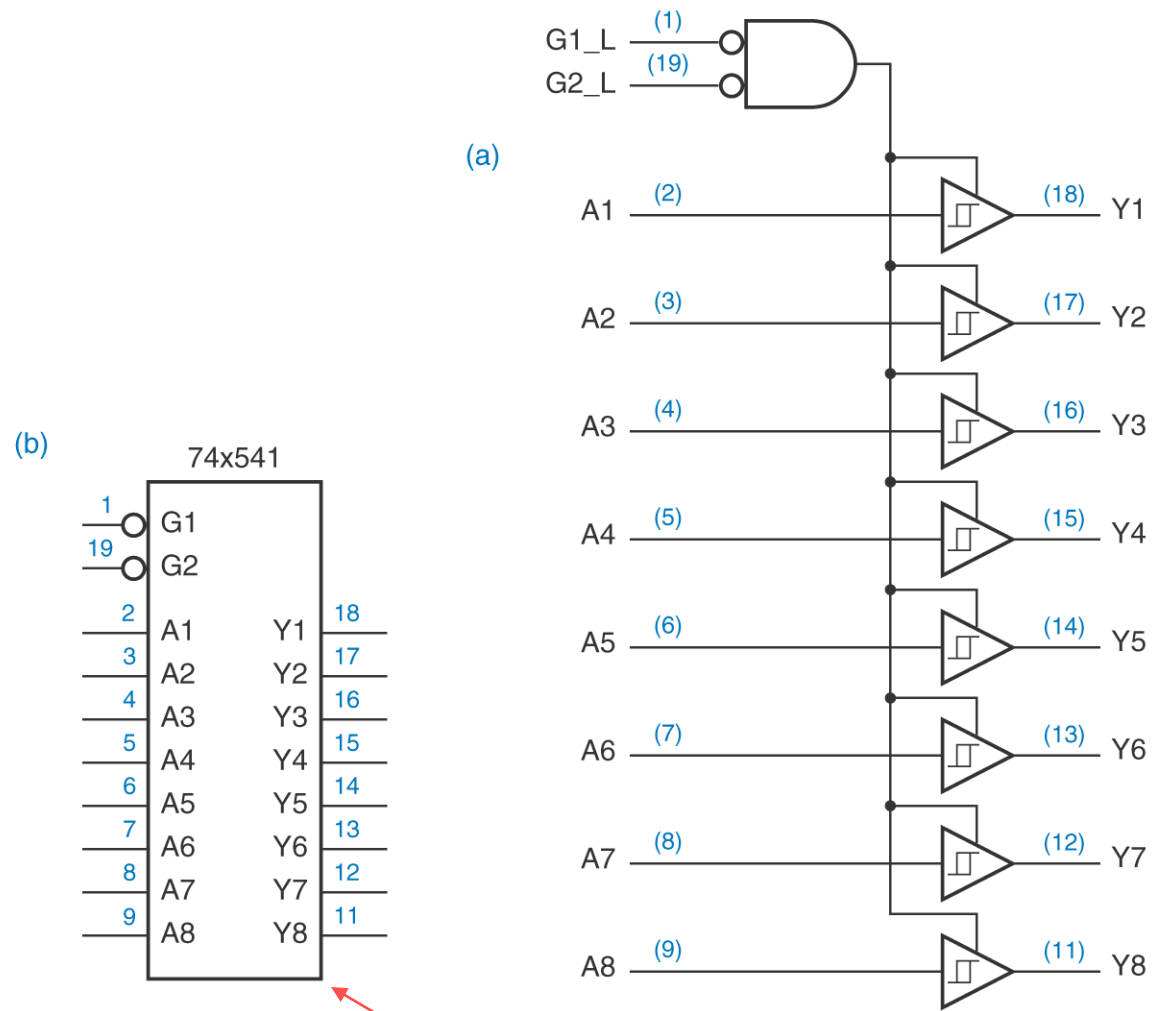


Figure 6-54

The 74x541 octal three-state buffer: (a) logic diagram; (b) traditional logic symbol.



# **Three-State buffers in microprocessors**

# 74x541 as a microprocessor input port.

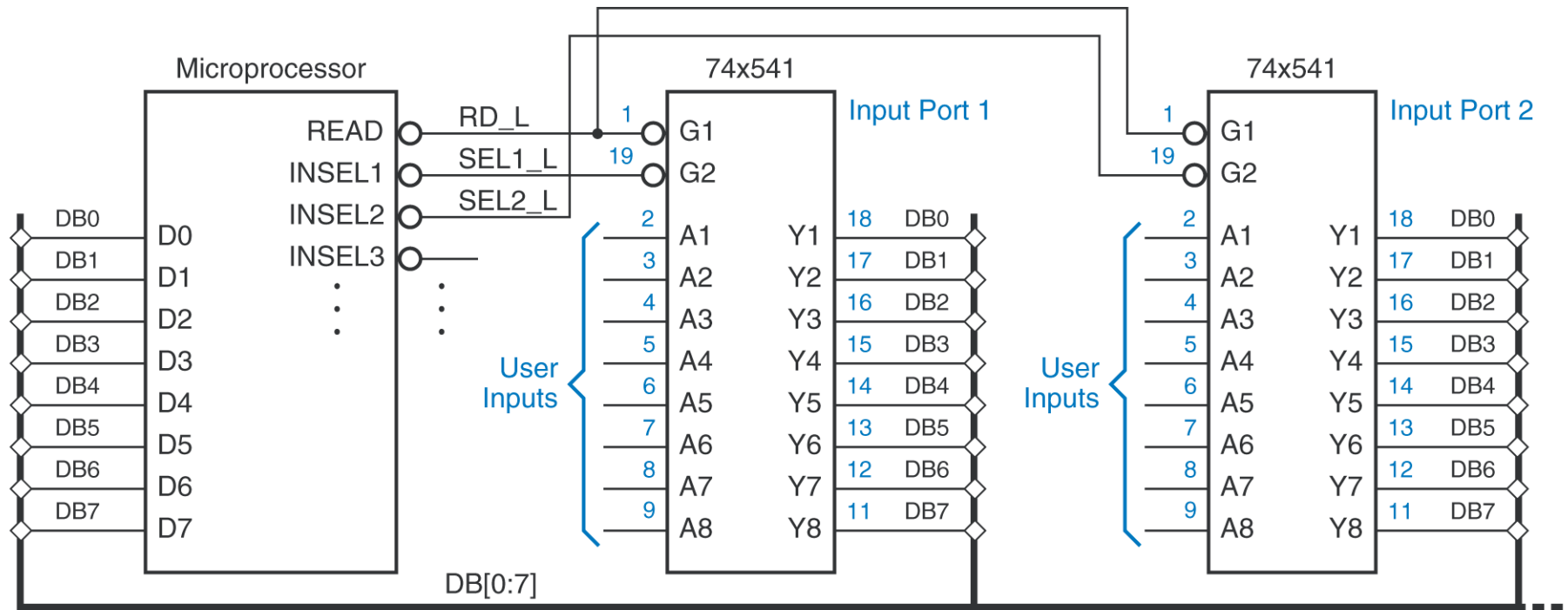


Figure 6-55

Using a 74x541 as a microprocessor input port.

# Verilog module for a 74x540-like 8-bit three-state driver

```
module Vr74x540(G1_L, G2_L, A, Y);  
  input G1_L, G2_L;  
  input [1:8] A;  
  output [1:8] Y;  
  
  assign Y = (~G1_L & ~G2_L) ? A : 8'bz;  
endmodule
```

Table 6-39

Verilog module for a 74x540-like 8-bit three-state driver.

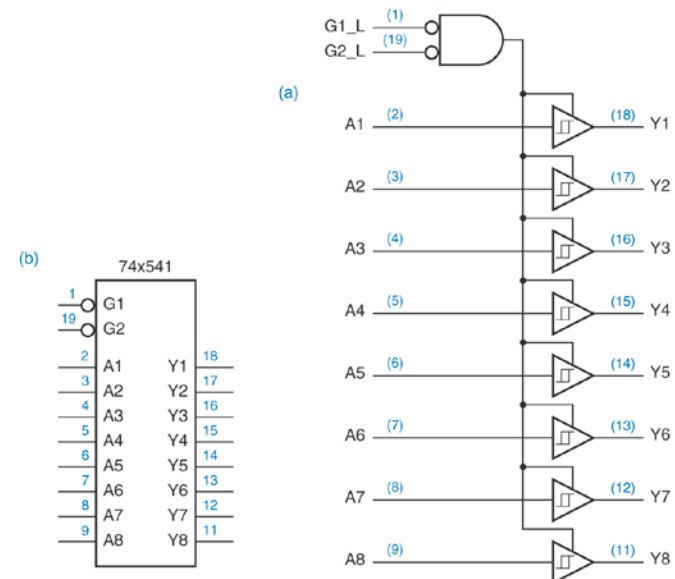


Figure 6-54

The 74x541 octal three-state buffer: (a) logic diagram; (b) traditional logic symbol.

# 74x245 octal three-state transceiver

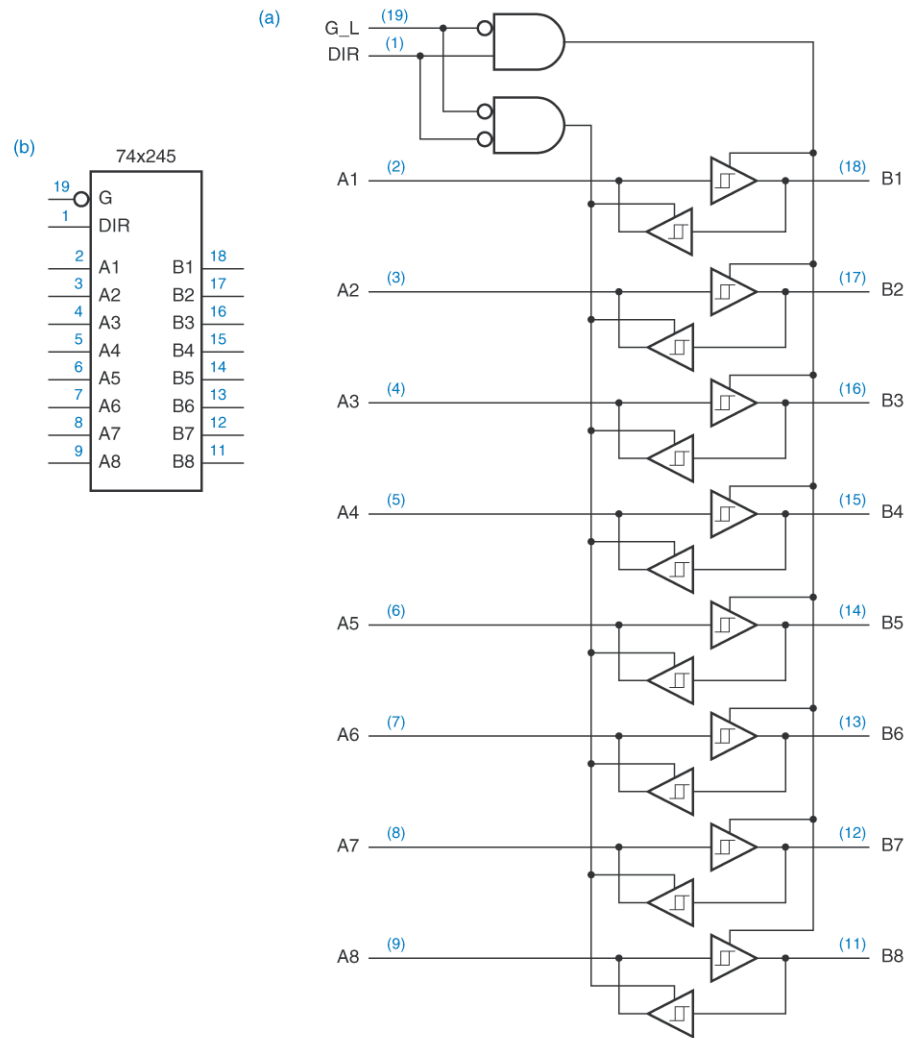


Figure 6-56

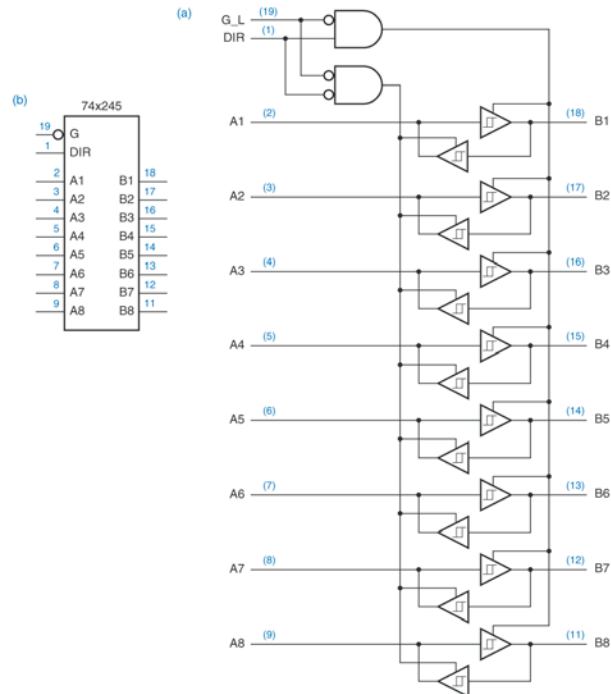
The 74x245 octal three-state transceiver: (a) logic diagram; (b) traditional logic symbol.

# Verilog module for a 74x245-like 8-bit transceiver

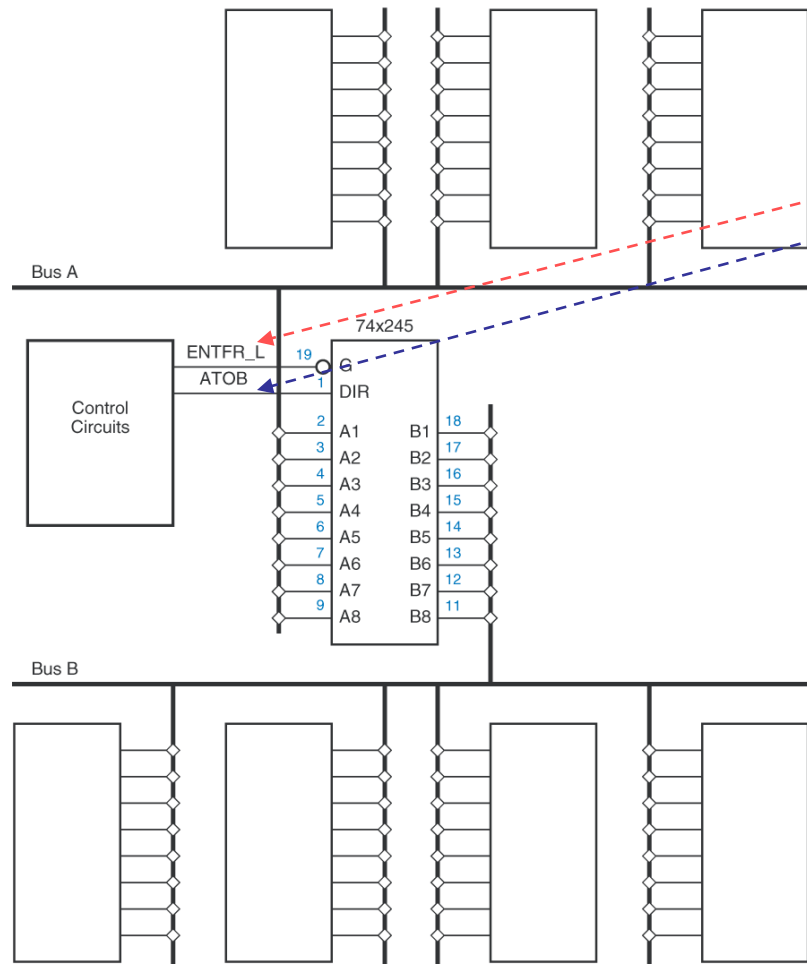
```
module Vr74x245(G_L, DIR, A, B);  
  input G_L, DIR;  
  inout [1:8] A, B;  
  
  assign A = (~G_L & ~DIR) ? B : 8'bz;  
  assign B = (~G_L & DIR) ? A : 8'bz;  
endmodule
```

Table 6-40

Verilog module for a 74×245-like 8-bit transceiver.



# Bidirectional buses and transceiver operation



| ENTFR_L | ATOB | Operation   |
|---------|------|---|
| 0       | 0    | Transfer data from a source on bus B to a destination on bus A. |
| 0       | 1    | Transfer data from a source on bus A to a destination on bus B. |
| 1       | x    | Transfer data on buses A and B independently.                   |

Table 6-32

Modes of operation for a pair of bidirectional buses.

Figure 6-57

Bidirectional buses and transceiver operation.

# Bus selection codes for a four-way bus transceiver

| S2 | S1 | S0 | Source selected |
|----|----|----|-----------------|
| 0  | 0  | 0  | 00              |
| 0  | 0  | 1  | 01              |
| 0  | 1  | 0  | 10              |
| 0  | 1  | 1  | 11              |
| 1  | 0  | 0  | A bus           |
| 1  | 0  | 1  | B bus           |
| 1  | 1  | 0  | C bus           |
| 1  | 1  | 1  | D bus           |

Table 6-34

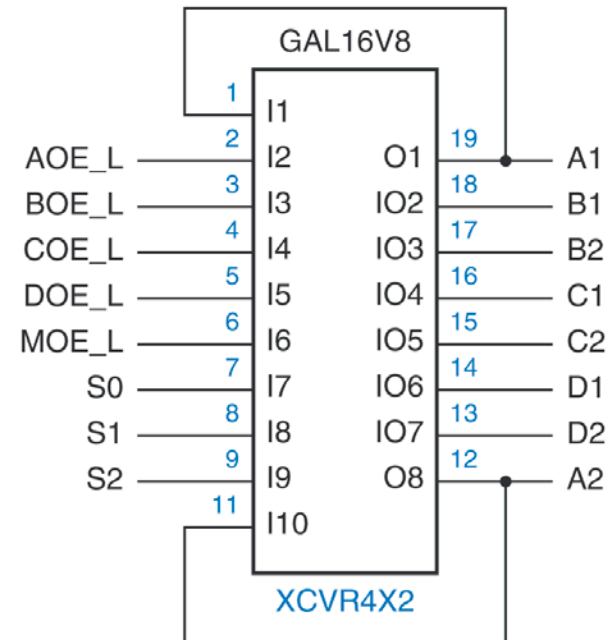


Figure 6-58

PLD inputs and outputs for a four-way, 2-bit bus transceiver.

Bus-selection codes for a four-way bus transceiver.

# PLD inputs and outputs for a four-way, 2-bit bus transceiver

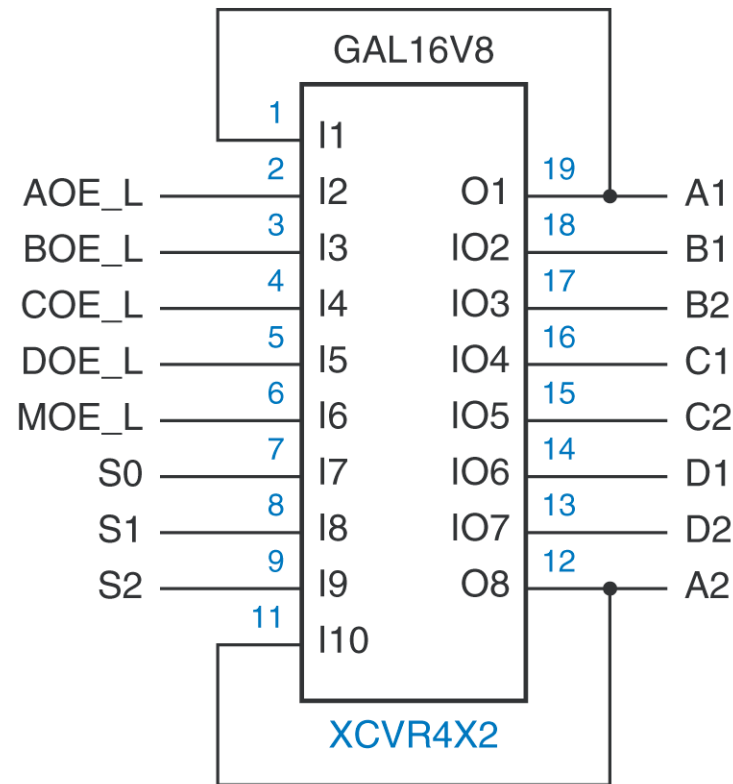


Figure 6-58

PLD inputs and outputs for a four-way, 2-bit bus transceiver.



# Verilog module for a four-way, 2-bit bus transceiver

```
module VrXcvr4x8(A, B, C, D, S, AOE_L, BOE_L, COE_L, DOE_L, MOE_L);
  input [2:0] S;
  input AOE_L, BOE_L, COE_L, DOE_L, MOE_L;
  inout [1:8] A, B, C, D;
  reg [1:8] ibus;

  always @ (A or B or C or D or S) begin
    if (S[2] == 0) ibus = {4{S[1:0]}};
    else case (S[1:0])
      0: ibus = A;
      1: ibus = B;
      2: ibus = C;
      3: ibus = D;
    endcase
  end
  assign A = ((~AOE_L & ~MOE_L) && (S[2:0] != 4)) ? ibus : 8'bz;
  assign B = ((~BOE_L & ~MOE_L) && (S[2:0] != 5)) ? ibus : 8'bz;
  assign C = ((~COE_L & ~MOE_L) && (S[2:0] != 6)) ? ibus : 8'bz;
  assign D = ((~DOE_L & ~MOE_L) && (S[2:0] != 7)) ? ibus : 8'bz;
endmodule
```

| S2 | S1 | S0 | <i>Source<br/>selected</i> |
|----|----|----|----------------------------|
| 0  | 0  | 0  | 00                         |
| 0  | 0  | 1  | 01                         |
| 0  | 1  | 0  | 10                         |
| 0  | 1  | 1  | 11                         |
| 1  | 0  | 0  | A bus                      |
| 1  | 0  | 1  | B bus                      |
| 1  | 1  | 0  | C bus                      |
| 1  | 1  | 1  | D bus                      |

Table 6-34

Bus-selection codes for a four-way bus transceiver.

Table 6-41

Verilog module for four-way, 2-bit bus transceiver.

# Multiplexers

# Multiplexer structure

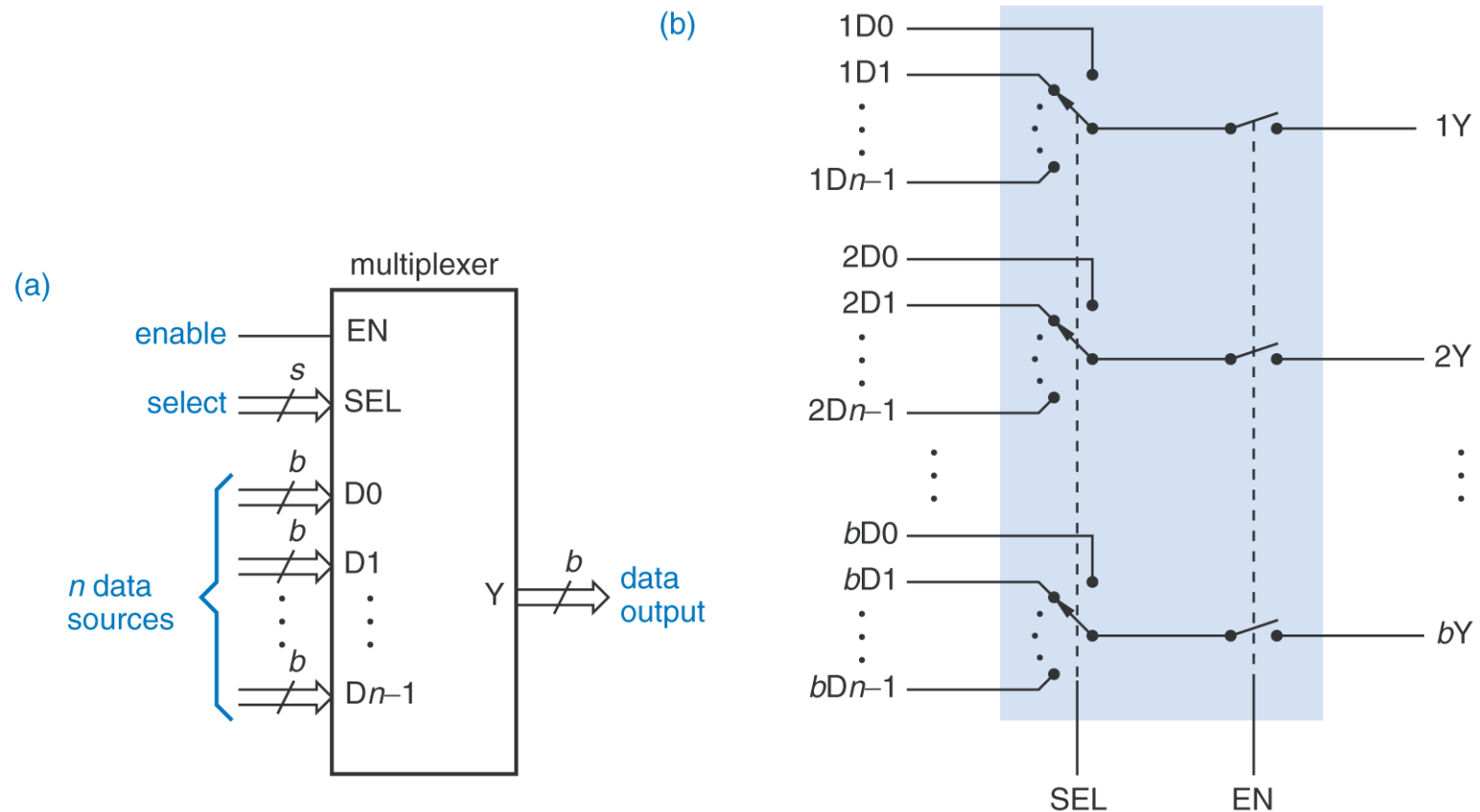
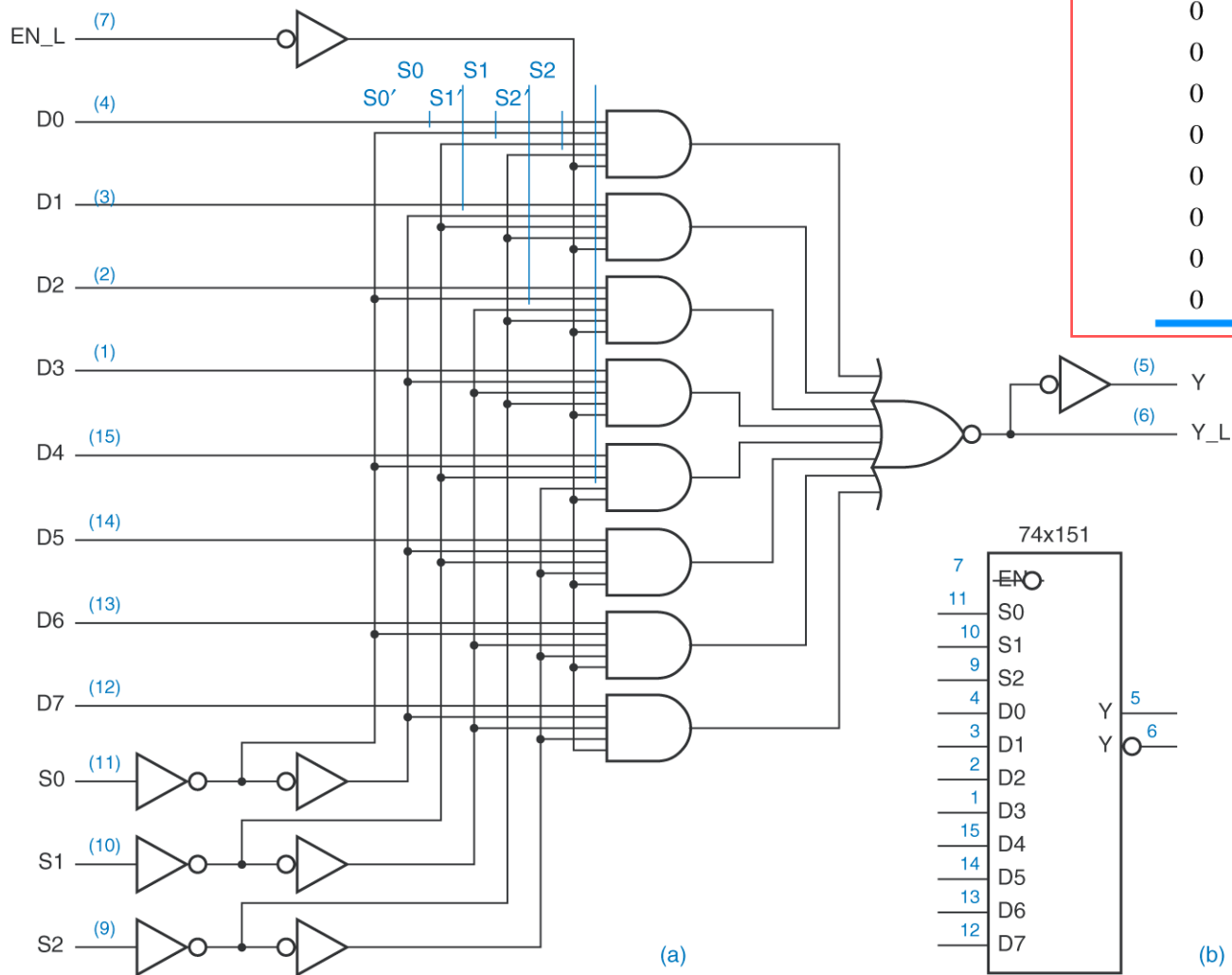


Figure 6-59

Multiplexer structure: (a) inputs and outputs; (b) functional equivalent.

# 74x151 8-input, 1-bit mux



| Inputs |    |    |    | Outputs |     |
|--------|----|----|----|---------|-----|
| EN_L   | S2 | S1 | S0 | Y       | Y_L |
| 1      | x  | x  | x  | 0       | 1   |
| 0      | 0  | 0  | 0  | D0      | D0' |
| 0      | 0  | 0  | 1  | D1      | D1' |
| 0      | 0  | 1  | 0  | D2      | D2' |
| 0      | 0  | 1  | 1  | D3      | D3' |
| 0      | 1  | 0  | 0  | D4      | D4' |
| 0      | 1  | 0  | 1  | D5      | D5' |
| 0      | 1  | 1  | 0  | D6      | D6' |
| 0      | 1  | 1  | 1  | D7      | D7' |

Figure 6-60

The 74x151 8-input, 1-bit multiplexer: (a) logic diagram; (b) traditional logic symbol.

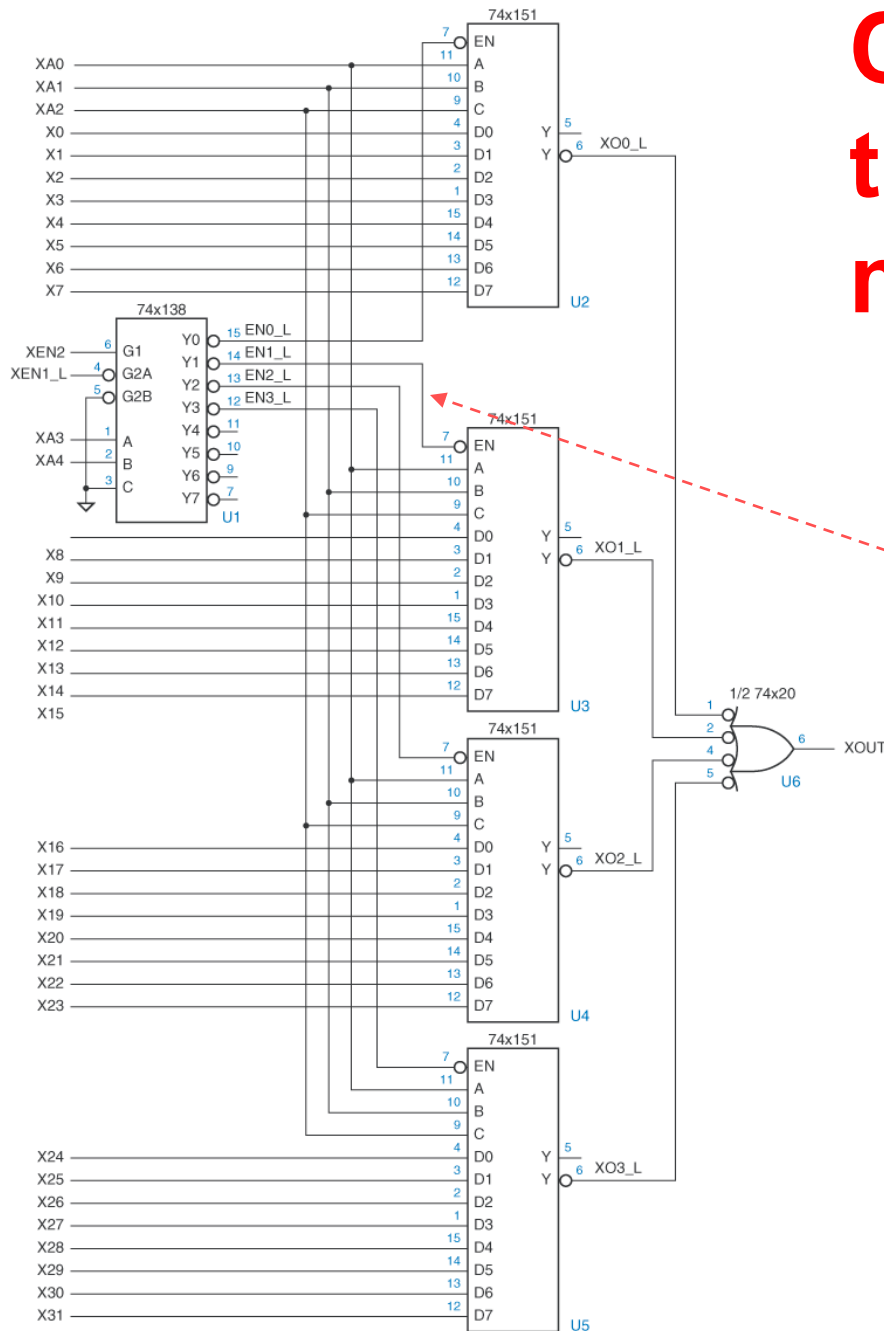
# 74x151 8-input, 1-bit mux

| <i>Inputs</i> |    |    |    | <i>Outputs</i> |     |
|---------------|----|----|----|----------------|-----|
| EN_L          | S2 | S1 | S0 | Y              | Y_L |
| 1             | x  | x  | x  | 0              | 1   |
| 0             | 0  | 0  | 0  | D0             | D0' |
| 0             | 0  | 0  | 1  | D1             | D1' |
| 0             | 0  | 1  | 0  | D2             | D2' |
| 0             | 0  | 1  | 1  | D3             | D3' |
| 0             | 1  | 0  | 0  | D4             | D4' |
| 0             | 1  | 0  | 1  | D5             | D5' |
| 0             | 1  | 1  | 0  | D6             | D6' |
| 0             | 1  | 1  | 1  | D7             | D7' |

Table 6-42

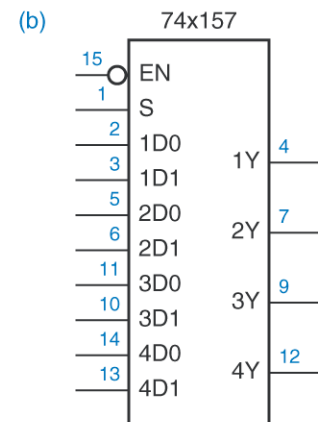
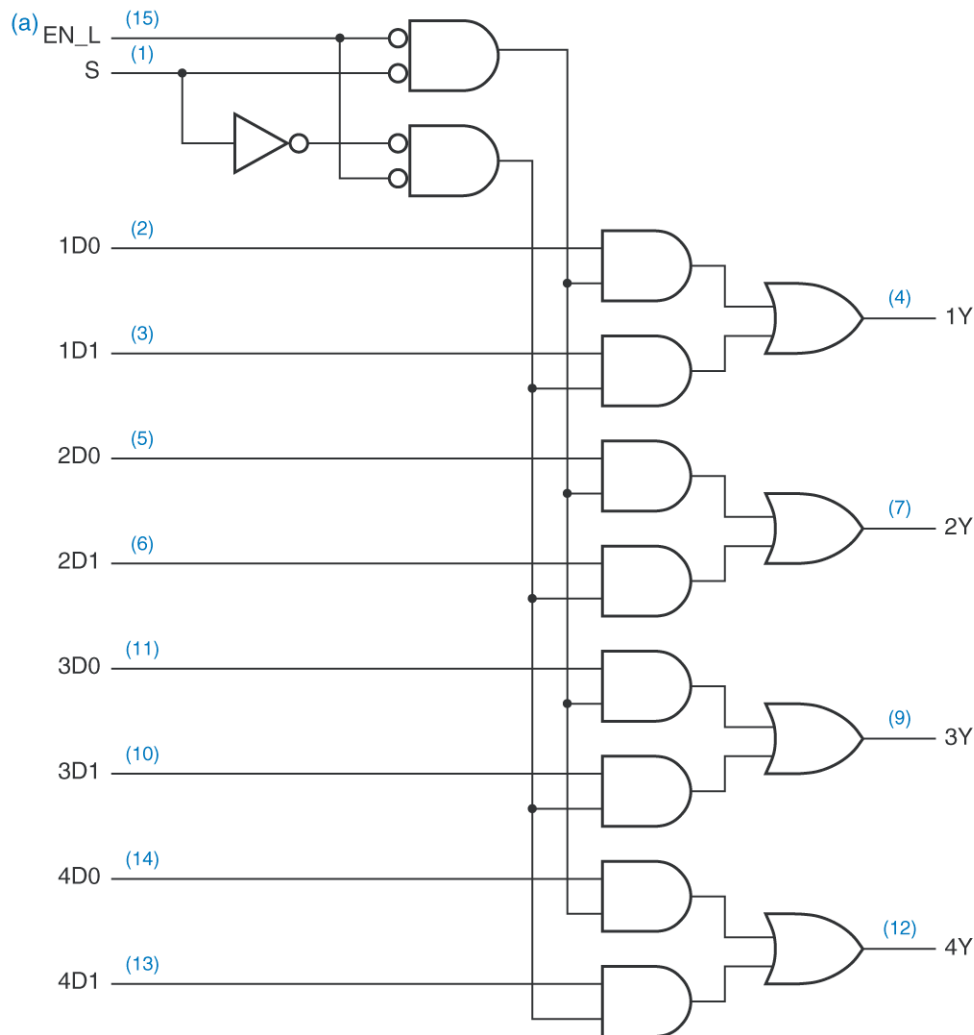
Truth table for a 74×151 8-input, 1-bit multiplexer.

# Combining 74x151s to make a 32-to-1 multiplexer



Decoding and enabling

# 74x157 2-input 4-bit mux



| Inputs |   | Outputs |     |     |     |
|--------|---|---------|-----|-----|-----|
| EN_L   | S | 1Y      | 2Y  | 3Y  | 4Y  |
| 1      | x | 0       | 0   | 0   | 0   |
| 0      | 0 | 1D0     | 2D0 | 3D0 | 4D0 |
| 0      | 1 | 1D1     | 2D1 | 3D1 | 4D1 |

Table 6-43

Truth table for a 74x157 2-input, 4-bit multiplexer.

Figure 6-61

The 74x157 2-input, 4-bit multiplexer: (a) logic diagram; (b) traditional logic symbol.

# GAL16V8 used as a 74x157 multiplexer

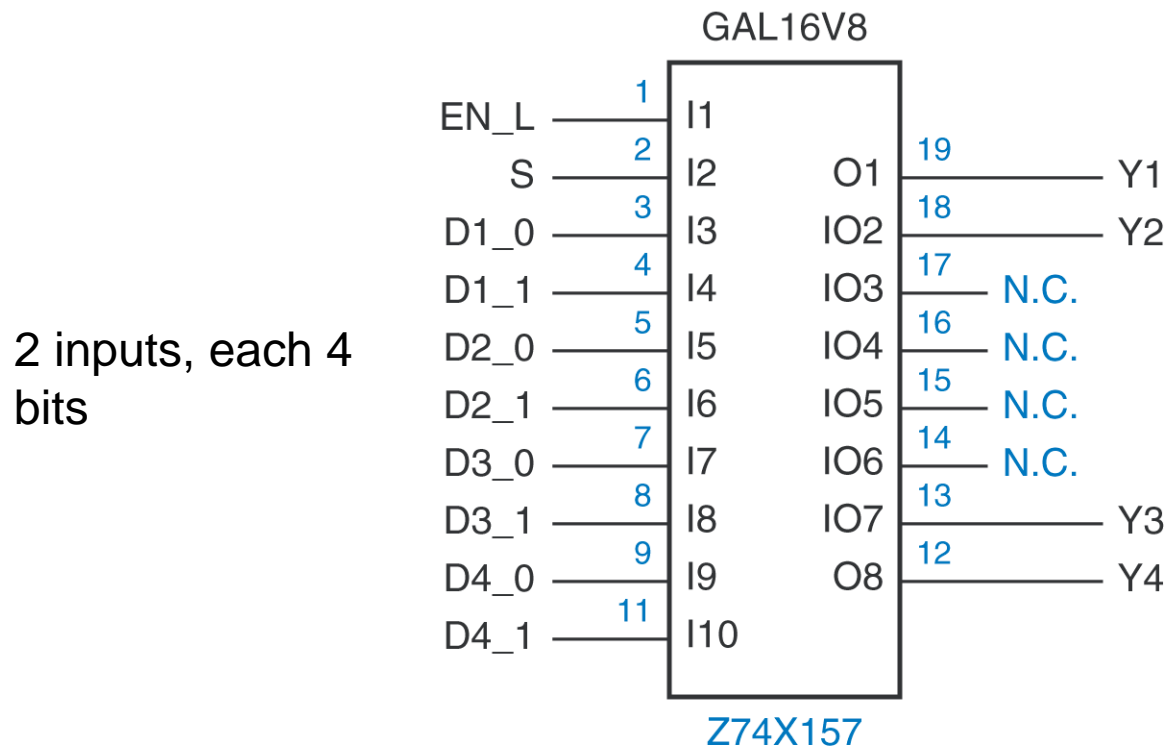


Figure 6-66

Logic diagram for the GAL16V8 used as a 74x157-like multiplexer.



# Buffers to handle large fanout

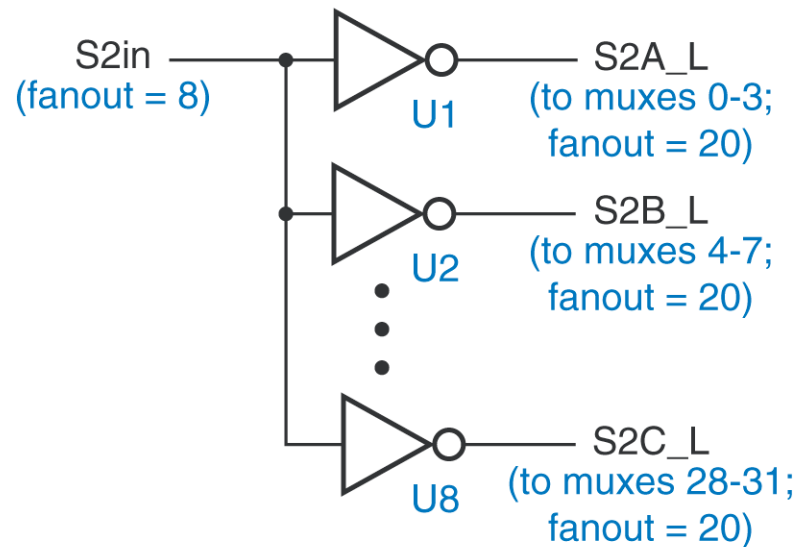


Figure 6-63

Buffers to handle a fanout of 160 in multiplexer control application.

# A mux driving a bus and a demux receiving the bus

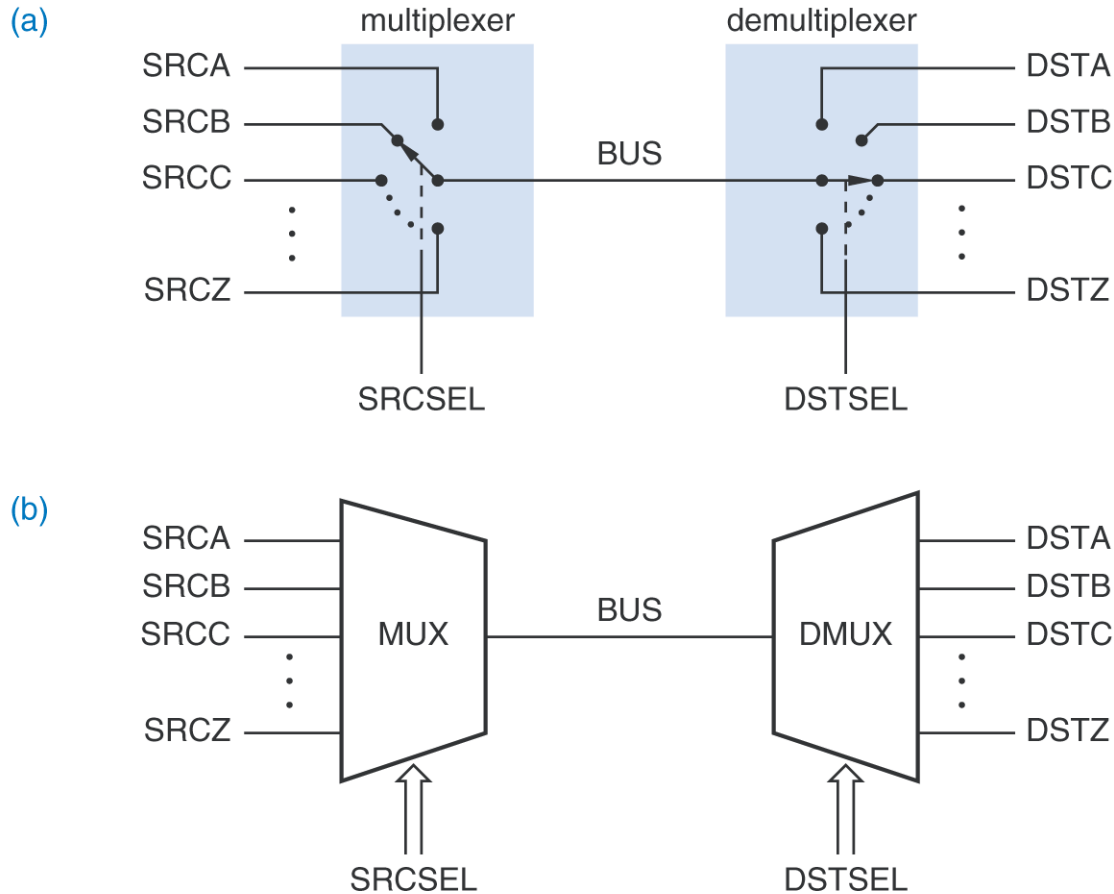


Figure 6-64

A mux driving a bus and a demultiplexer receiving the bus:  
(a) switch equivalent; (b) block-diagram symbols.

# **Decoders as demultiplexers**

# 3-to-8 binary decoder as a demultiplexer

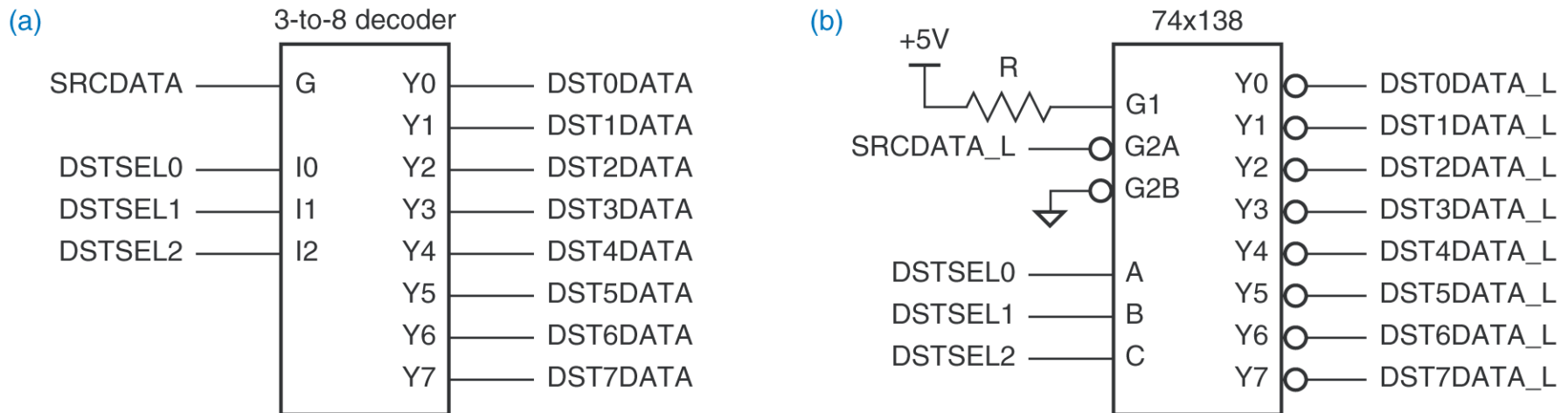


Figure 6-65

Using a 3-to-8 binary decoder as a 1-bit, 8-output demultiplexer: (a) generic decoder; (b) 74x138.

# **GALs as multiplexers**

# Function table for a SPECIALIZED 4-input, 18-bit mux.

| S2 | S1 | S0 | <i>Input to Select</i> |
|----|----|----|------------------------|
| 0  | 0  | 0  | A                      |
| 0  | 0  | 1  | B                      |
| 0  | 1  | 0  | A                      |
| 0  | 1  | 1  | C                      |
| 1  | 0  | 0  | A                      |
| 1  | 0  | 1  | D                      |
| 1  | 1  | 0  | A                      |
| 1  | 1  | 1  | B                      |

Table 6-46

Function table for a specialized 4-input, 18-bit multiplexer.

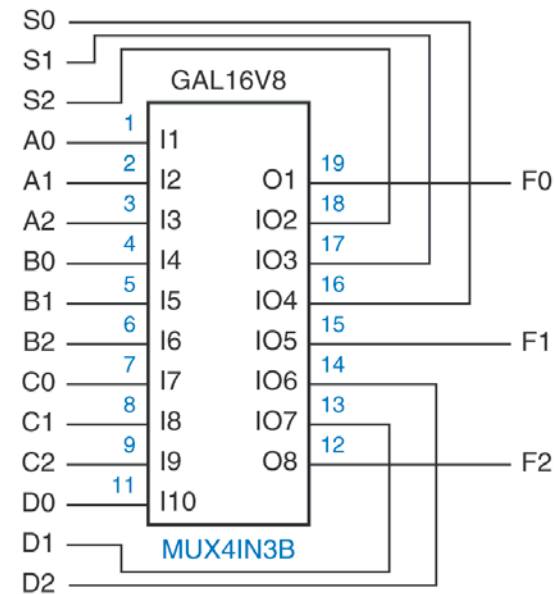


Figure 6-67

Logic diagram for the GAL16V8 used as a specialized 4-input, 3-bit multiplexer.

# Behavioral Verilog for a specialized 4-input 18-bit mux

---

```
module Vrmux4in18b(S, A, B, C, D, Y);
  input [2:0] S;
  input [1:18] A, B, C, D;
  output [1:18] Y;
  reg [1:18] Y;

  always @ (S or A or B or C or D)
    case (S)
      3'd0, 3'd2, 3'd4, 3'd6: Y = A;
      3'd1, 3'd7: Y = B;
      3'd3: Y = C;
      3'd5: Y = D;
      default: Y = 8'bx;
    endcase
endmodule
```

---

Table 6-53

Behavioral Verilog program for a specialized 4-input, 18-bit multiplexer.

From *Digital Design: Principles and Practices*, Fourth Edition, John F. Wakerly, ISBN 0-13-186389-4.

©2006, Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

# Dataflow Verilog for a 4-input, 8-bit mux

```
module Vrmux4in8b(YOE_L, EN_L, S, A, B, C, D, Y);
    input YOE_L, EN_L;
    input [1:0] S;
    input [1:8] A, B, C, D;
    output [1:8] Y;
    assign Y = (~YOE_L == 1'b0) ? 8'bz : (
        (~EN_L == 1'b0) ? 8'b0 : (
            (S == 2'd0) ? A : (
                (S == 2'd1) ? B : (
                    (S == 2'd2) ? C : (
                        (S == 2'd3) ? D : 8'bx)))));
endmodule
```

Table 6-51

Dataflow Verilog program for a 4-input, 8-bit multiplexer.



# Behavioral Verilog for a 4-input, 8-bit mux.

---

```
module Vrmux4in8bc(YOE_L, EN_L, S, A, B, C, D, Y);
  input YOE_L, EN_L;
  input [1:0] S;
  input [1:8] A, B, C, D;
  output [1:8] Y;
  reg [1:8] Y;

  always @ (YOE_L or EN_L or S or A or B or C or D) begin
    if (~YOE_L == 1'b0) Y = 8'bz;
    else if (~EN_L == 1'b0) Y = 8'b0;
    else case (S)
      2'd0: Y = A;
      2'd1: Y = B;
      2'd2: Y = C;
      2'd3: Y = D;
      default: Y = 8'bx;
    endcase
  end
endmodule
```

---

Table 6-52

Behavioral Verilog module for a 4-input, 8-bit multiplexer.