Design of Digital Systems II

Combinational Logic Design Practices (2)

Moslem Amiri, Václav Přenosil

Embedded Systems Laboratory
Faculty of Informatics, Masaryk University
Brno, Czech Republic

amiri@mail.muni.cz
prenosil@fi.muni.cz

November, 2012

Decoders

- A decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where input and output codes are different
  - Input code generally has fewer bits than output code
  - There is a one-to-one mapping from input code words into output code words
    - In a *one-to-one mapping*, each input code word produces a different output code word

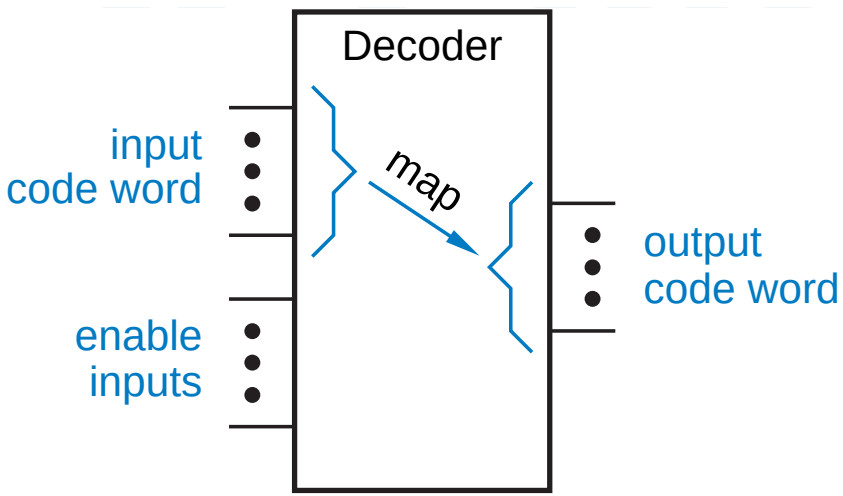


Figure 1: Decoder circuit structure.

Decoders

- Enable inputs must be asserted for decoder to perform its normal mapping function
  - Otherwise, it maps all input code words into a single, "disabled," output code word
- Most commonly used input code is an n-bit binary code
  - An n-bit word represents one of 2^n different coded values
- Most commonly used output code is a 1-out-of-m code
  - m bits where one bit is asserted at any time

Decoders: Binary Decoders

- Binary decoder is an n-to-2^n decoder
  - It has an n-bit binary input code and a 1-out-of-2^n output code

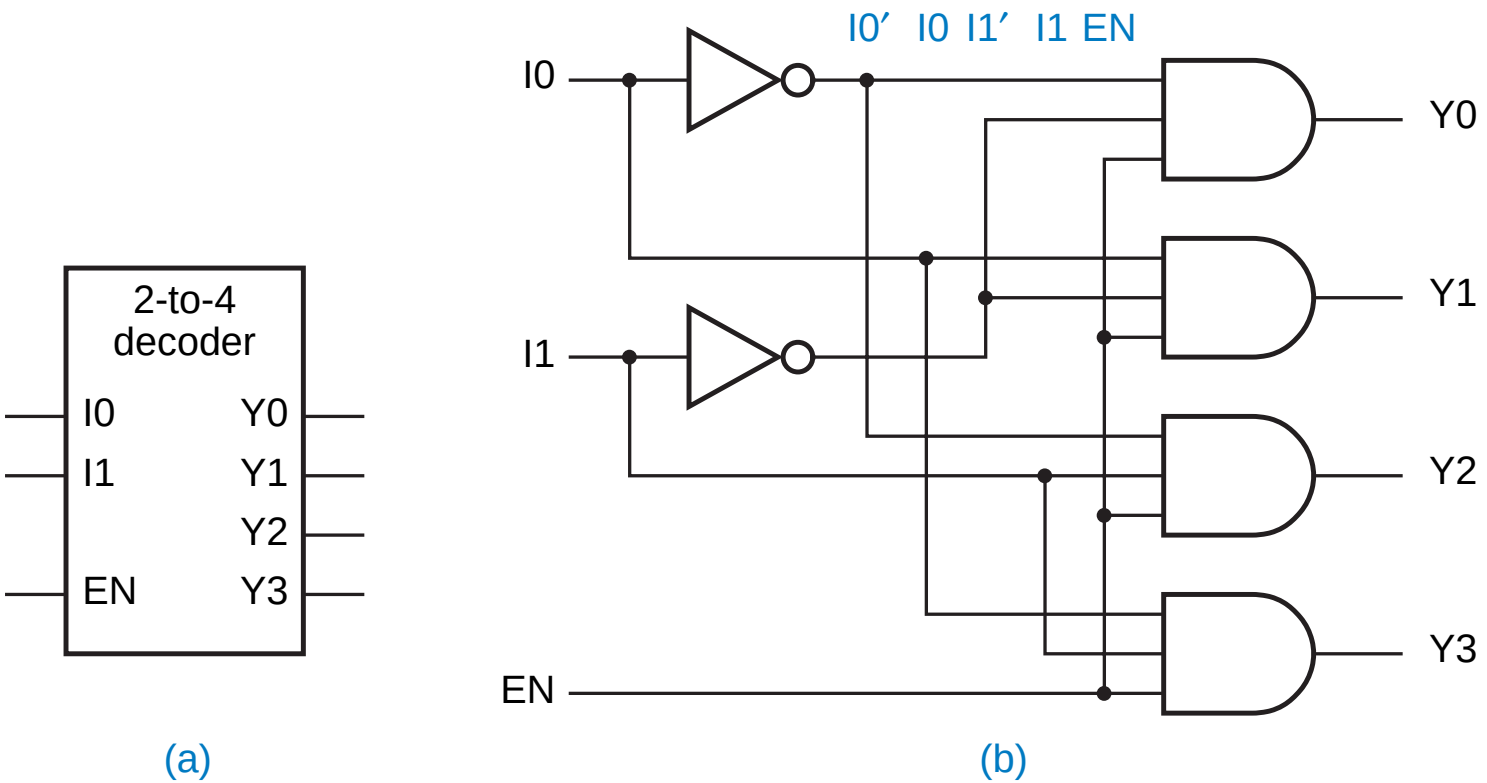


Figure 2: A 2-to-4 decoder: (a) inputs and outputs; (b) logic diagram.

Decoders: Binary Decoders

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

Inputs			Outputs			
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

- Input code of an n-bit binary decoder need not represent integers from 0 through $2^n - 1$
  - E.g., it can be in Gray code (appropriately assign inputs to outputs)
- It is not necessary to use all of outputs of a decoder, or even to decode all possible input combinations
  - E.g., a BCD decoder decodes only first ten binary input combinations 0000-1001 to produce outputs Y0-Y9

Decoders: The 74x139 Dual 2-to-4 Decoder

- 74x139 is a single MSI part containing two independent and identical 2-to-4 decoders

Decoders: The 74x139 Dual 2-to-4 Decoder

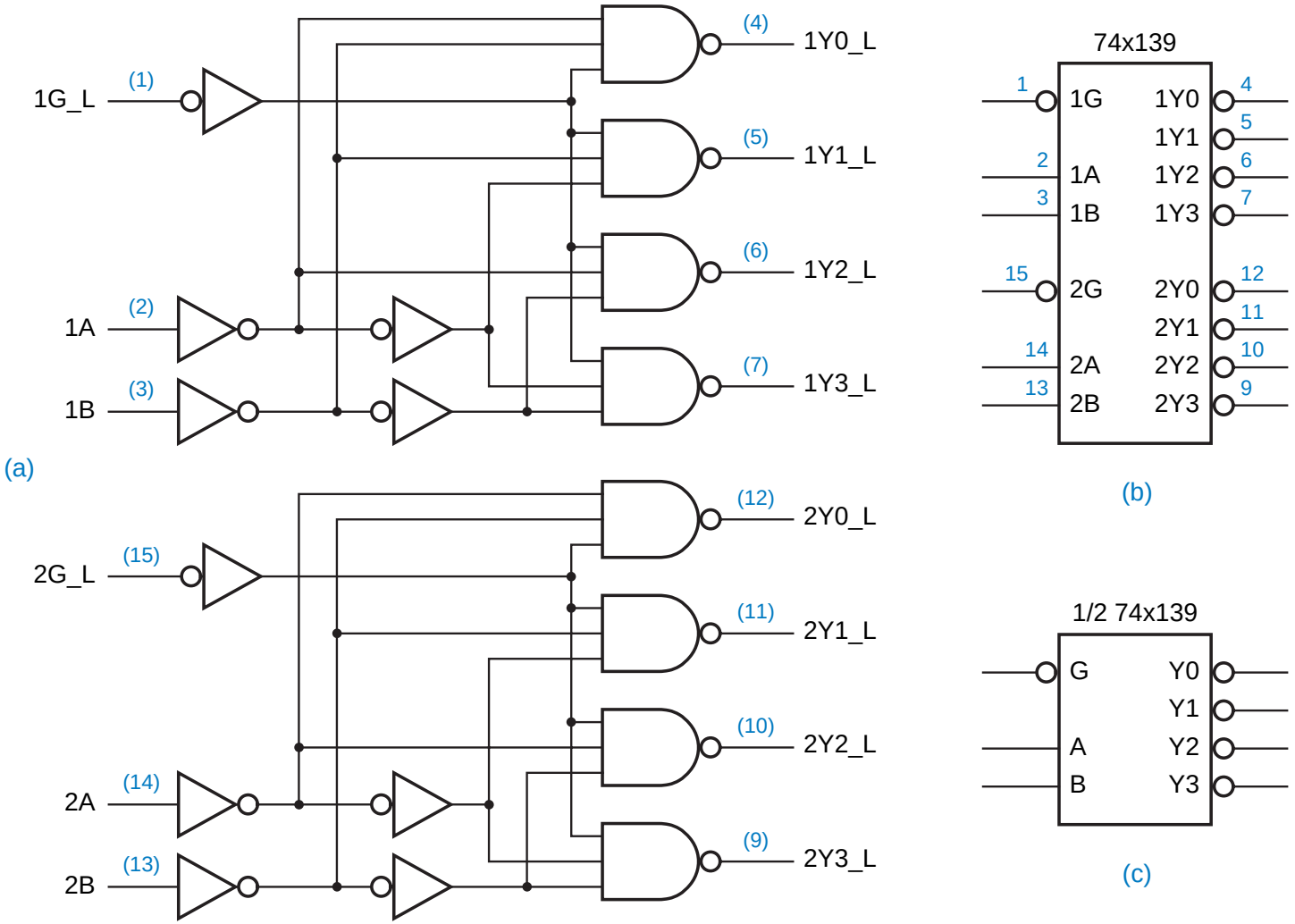


Figure 3: The 74x139 dual 2-to-4 decoder: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol; (c) logic symbol for one decoder.

Decoders: The 74x139 Dual 2-to-4 Decoder

- Outputs and enable input of '139 are active-low
  - Inverting gates are generally faster than noninverting ones
- '139 has extra inverters on its select inputs
  - Without these inverters, each select input would present three AC or DC loads instead of one, consuming much more of fanout budget of device that drives it

Table 2: Truth table for one-half of a 74x139 dual 2-to-4 decoder.

Inputs			Outputs			
G_L	B	A	Y3_L	Y2_L	Y1_L	Y0_L
1	x	x	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

Decoders: The 74x138 3-to-8 Decoder

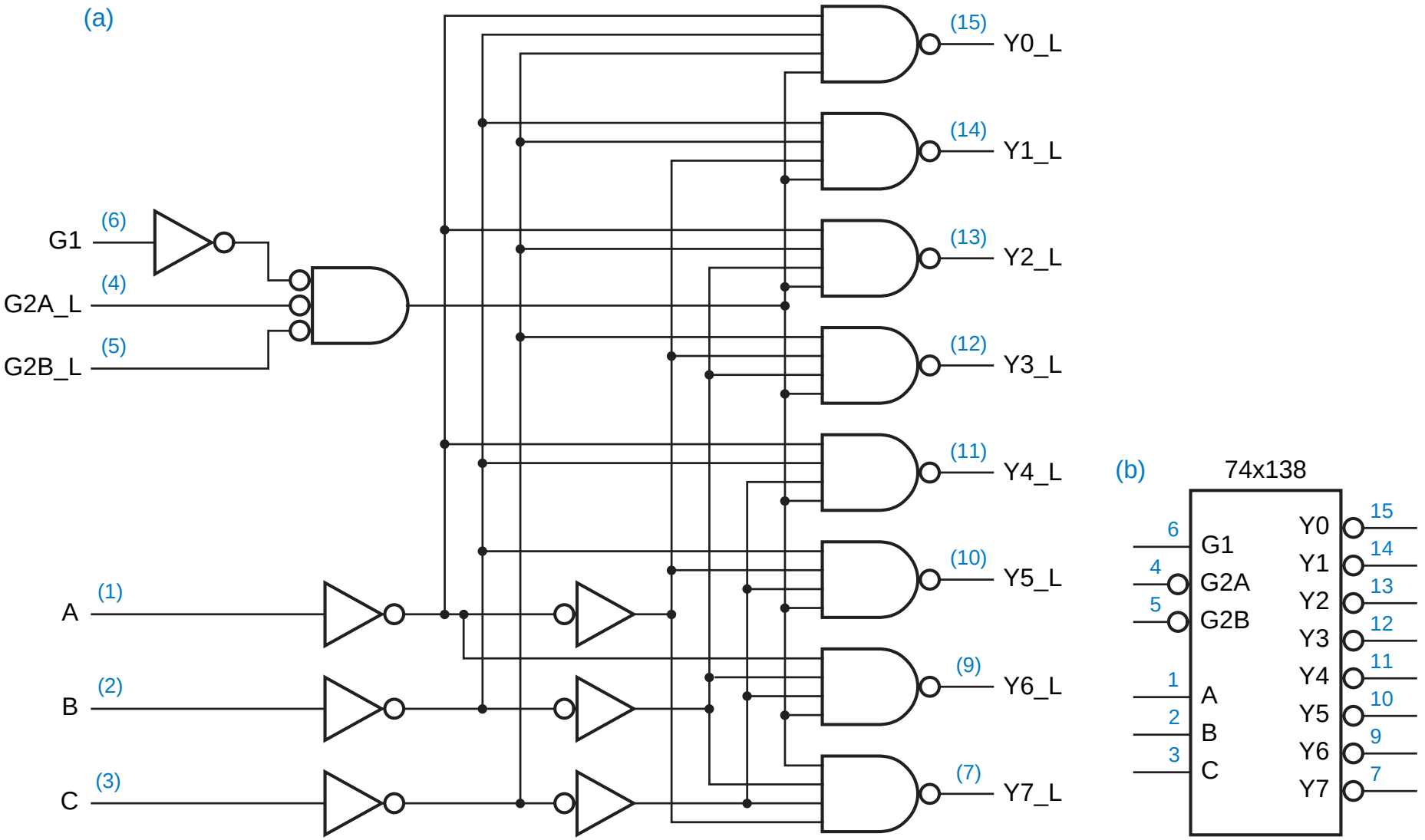


Figure 4: The 74x138 3-to-8 decoder: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol.

Decoders: The 74x138 3-to-8 Decoder

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

Inputs						Outputs							
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

Decoders: Cascading Binary Decoders

- Multiple binary decoders can be used to decode larger code words

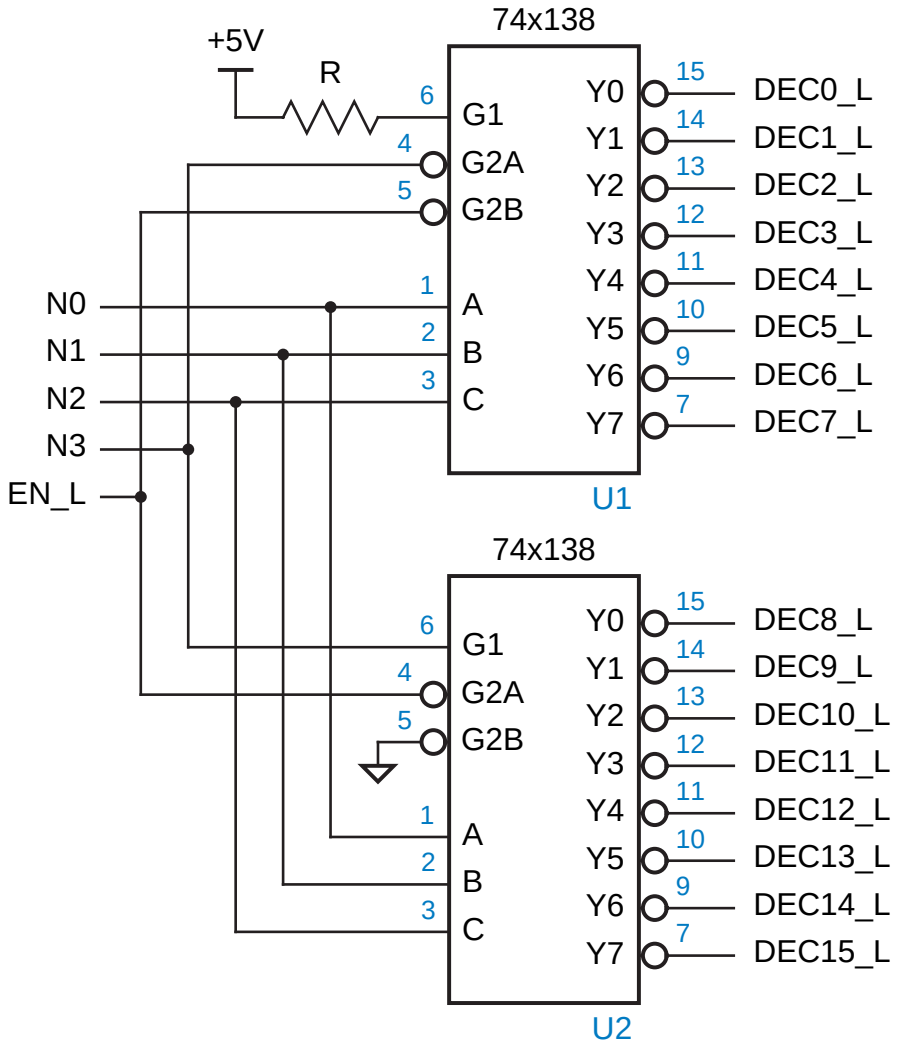


Figure 5: Design of a 4-to-16 decoder using 74x138s.

Decoders: Cascading Binary Decoders

- To handle larger code words, binary decoders can be cascaded hierarchically

Decoders: Cascading Binary Decoders

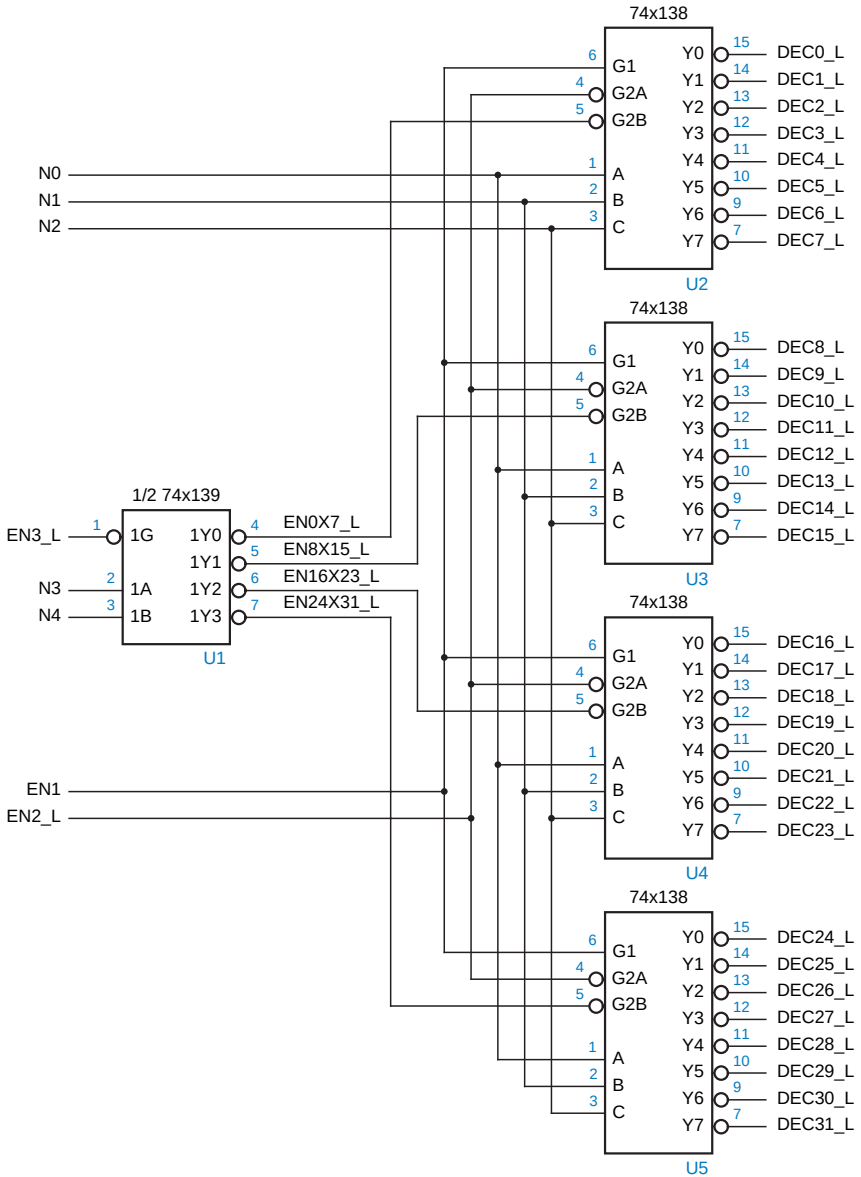


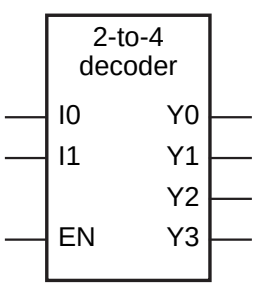
Figure 6: Design of a 5-to-32 decoder using 74x138s and a 74x139.

Decoders in Verilog

Table 4: Structural-style Verilog module for the decoder in Fig. 2.

```
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
```



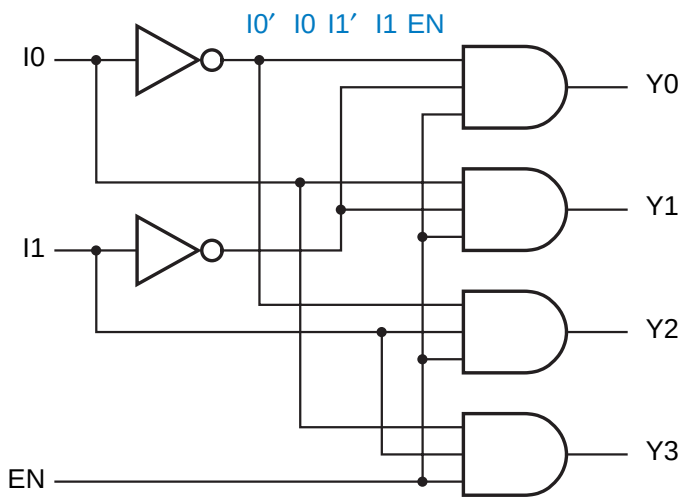
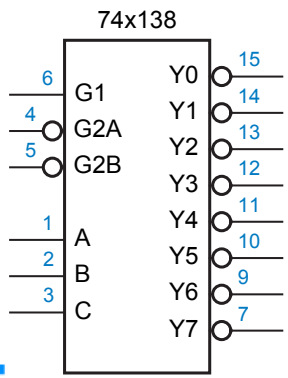


Table 5: Functional-style Verilog module for a 74x138-like 3-to-8 binary decoder.

```
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
```



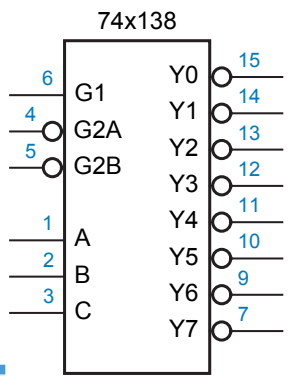
Decoders in Verilog

- In Tab. 5
  - Constants and inversions that handle the fact that two inputs and all outputs are active low are scattered throughout the code
  - While its true that most Verilog programs are written almost entirely with active-high signals, if we are defining a device with active-low external pins, we should handle them in a more systematic and easily maintainable way
- Tab. 6
  - Decoder function is defined in terms of only active-high signals
  - The design can be easily modified in just a few well-defined places if changes are required in external active levels

Table 6: Verilog module with a maintainable approach to active-level handling.

```
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
```



Decoders in Verilog

Table 7: Hierarchical definition of 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 8: Verilog functional definition of an 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
```

Decoders in Verilog

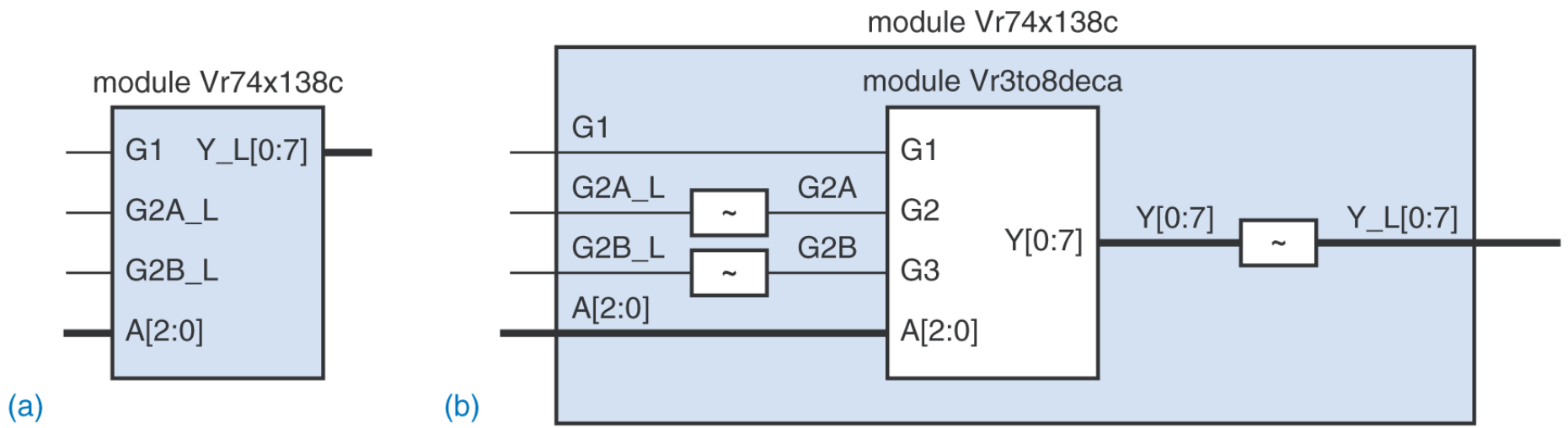


Figure 7: Verilog module 74x138c: (a) top level; (b) internal structure using module Vr3to8deca.

Decoders in Verilog

Table 9: Behavioral Verilog definition for a 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
```

Decoders: Seven-Segment Decoders

- A seven-segment decoder has 4-bit BCD as its input code and "seven-segment code" as its output code

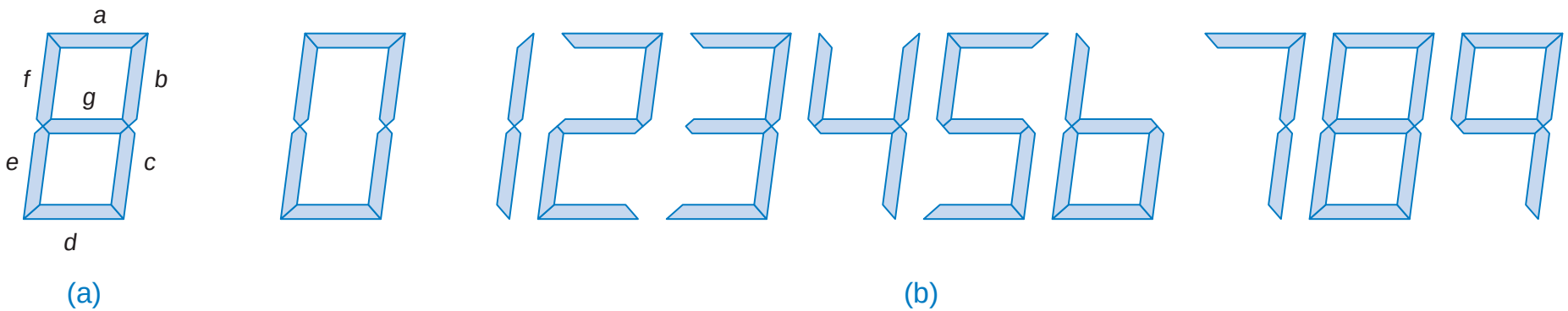


Figure 8: Seven-segment display: (a) segment identification; (b) decimal digits.

Decoders: Seven-Segment Decoders

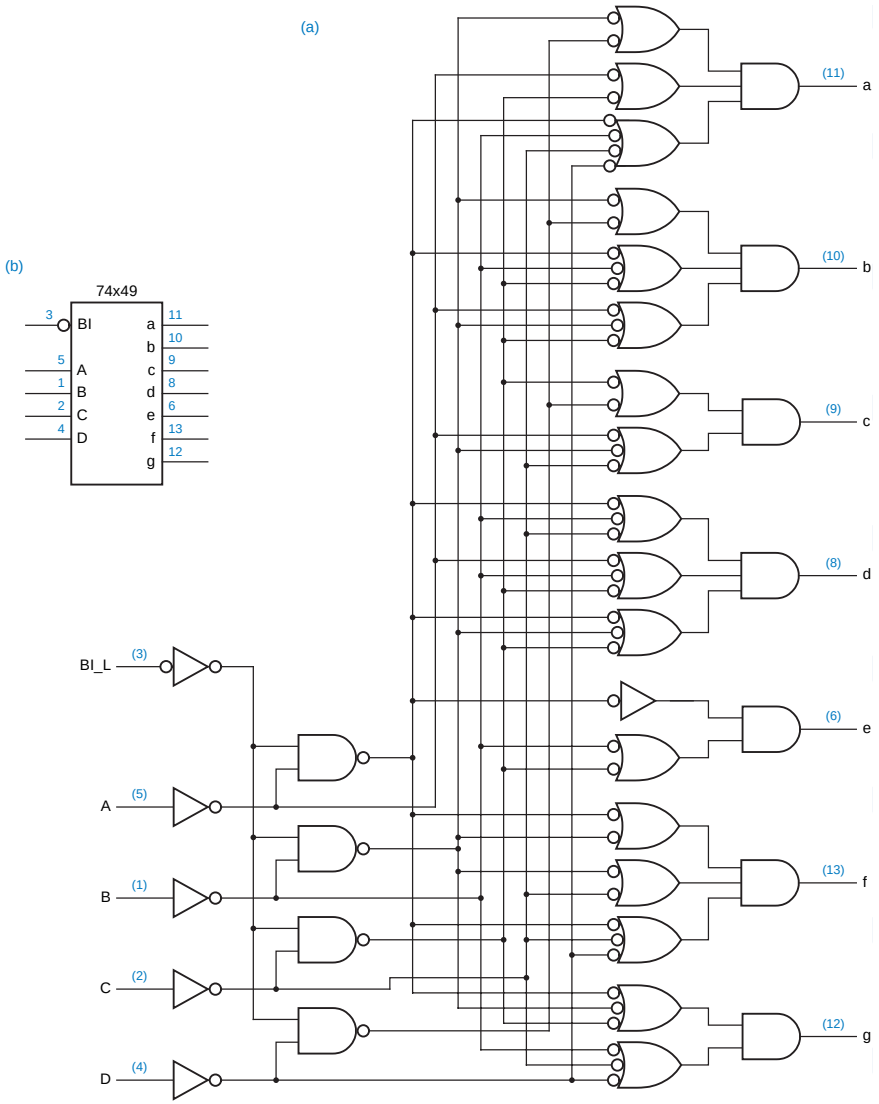


Figure 9: The 74x49 seven-segment decoder: (a) logic diagram, including pin numbers; (b) traditional logic symbol.

Table 10: Truth table for a 74x49 seven-segment decoder.

Inputs					Outputs						
BI_L	D	C	B	A	a	b	c	d	e	f	g
0	x	x	x	x	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
1	0	0	1	0	1	1	0	1	1	0	1
1	0	0	1	1	1	1	1	1	0	0	1
1	0	1	0	0	0	1	1	0	0	1	1
1	0	1	0	1	1	0	1	1	0	1	1
1	0	1	1	0	0	0	1	1	1	1	1
1	0	1	1	1	1	1	1	0	0	0	0
1	1	0	0	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	0	0	1	1
1	1	0	1	0	0	0	0	1	1	0	1
1	1	0	1	1	0	0	1	1	0	0	1
1	1	1	0	0	0	1	0	0	0	1	1
1	1	1	0	1	1	0	0	1	0	1	1
1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0

Decoders: Seven-Segment Decoders

- Each output of 74x49 is a minimal POS realization for corresponding segment, assuming don't-cares for non-decimal input combinations
- INVERT-OR-AND structure used for each output is equivalent to an AND-OR-INVERT gate, which is a fast and compact structure to build in CMOS or TTL
- Modern seven-segment display elements have decoders built into them
  - A 4-bit BCD word can be applied directly to device

Table 11: Verilog program for a 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
```

Encoders

- If a device's output code has fewer bits than input code, it is called an encoder
- Simplest encoder to build is a 2^n-to-n or **binary encoder**
  - Its input code is 1-out-of-2^n code and its output code is n-bit binary

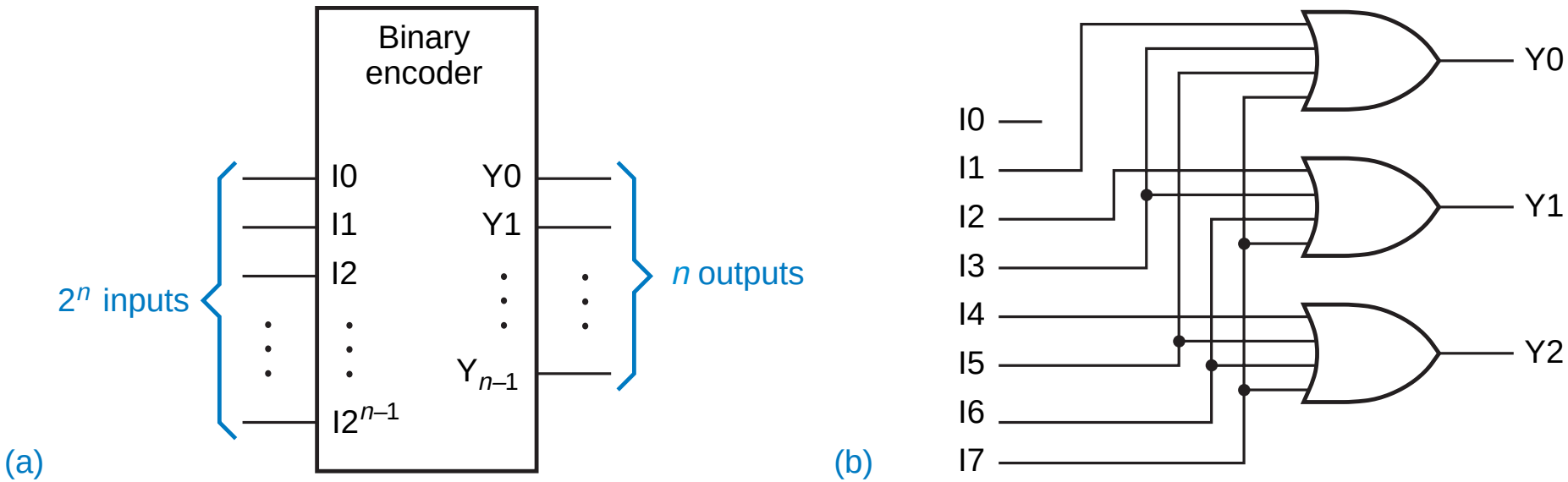


Figure 10: Binary encoder: (a) general structure; (b) 8-to-3 encoder.

$$Y0 = I1 + I3 + I5 + I7$$
$$Y1 = I2 + I3 + I6 + I7$$
$$Y2 = I4 + I5 + I6 + I7$$

Encoders: Priority Encoders

- Consider a system with 2^n inputs, each of which indicates a request for service
  - This structure is often found in microprocessor input/output subsystems where inputs might be interrupt requests
  - Binary encoder works properly only if inputs are guaranteed to be asserted at most one at a time
  - If multiple requests can be made simultaneously, the encoder gives undesirable results

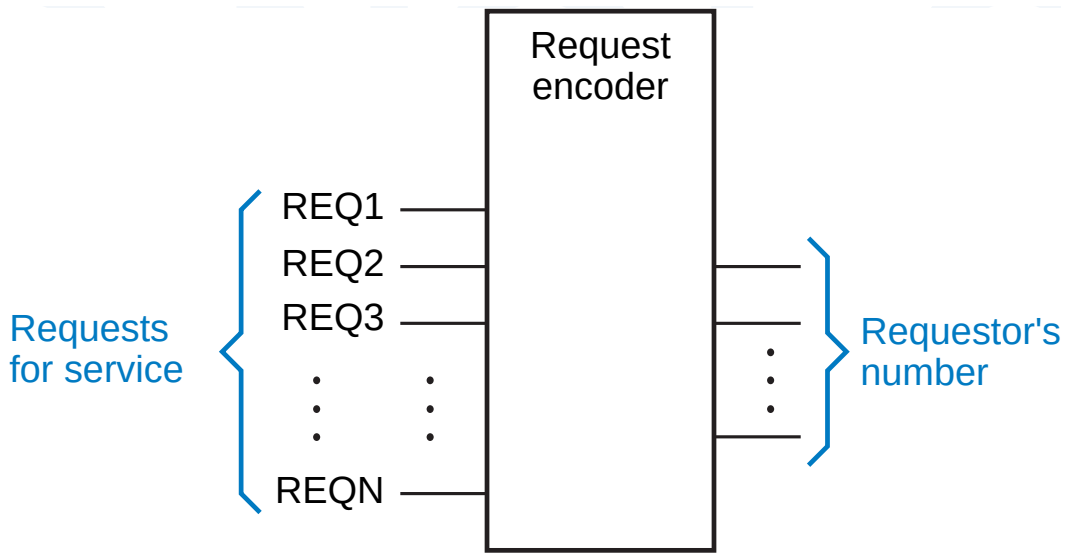


Figure 11: A system with 2^n requestors, and a "request encoder" that indicates which request signal is asserted at any time.

Encoders: Priority Encoders

- We assign priority to input lines, so that when multiple requests are asserted, encoder produces the number of the highest-priority requestor
  - Such a device is called **priority encoder**

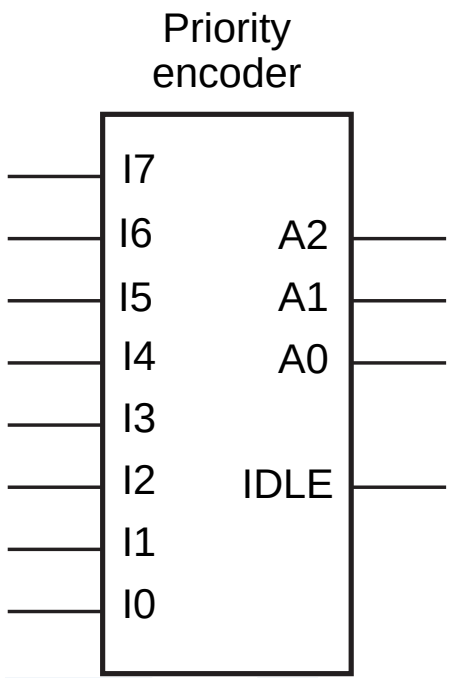


Figure 12: Logic symbol for a generic 8-input priority encoder.

Encoders: Priority Encoders

- Logic equations for priority encoder's outputs (Fig. 12)
  - Input $I7$ has the highest priority
  - Outputs $A2$–$A0$ contain number of the highest-priority asserted input
  - IDLE is asserted if no inputs are asserted
  - First we define eight intermediate variables $H0$–$H7$
  - Using $H0$–$H7$, equations for $A2$–$A0$ are similar to ones for a binary encoder

$$
\begin{aligned}
H7 &= I7 \\
H6 &= I6 \cdot I7' \\
H5 &= I5 \cdot I6' \cdot I7' \\
&\vdots \\
H0 &= I0 \cdot I1' \cdot I2' \cdot I3' \cdot I4' \cdot I5' \cdot I6' \cdot I7' \\
A2 &= H4 + H5 + H6 + H7 \\
A1 &= H2 + H3 + H6 + H7 \\
A0 &= H1 + H3 + H5 + H7 \\
IDLE &= I0' \cdot I1' \cdot I2' \cdot I3' \cdot I4' \cdot I5' \cdot I6' \cdot I7'
\end{aligned}
$$

Encoders: The 74x148 Priority Encoder

- 74x148 is an MSI 8-input priority encoder

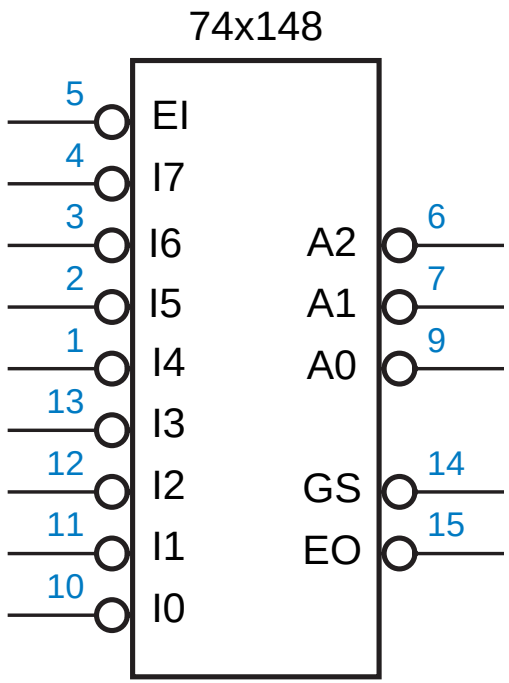


Figure 13: Logic symbol for the 74x148 8-input priority encoder.

Encoders: The 74x148 Priority Encoder

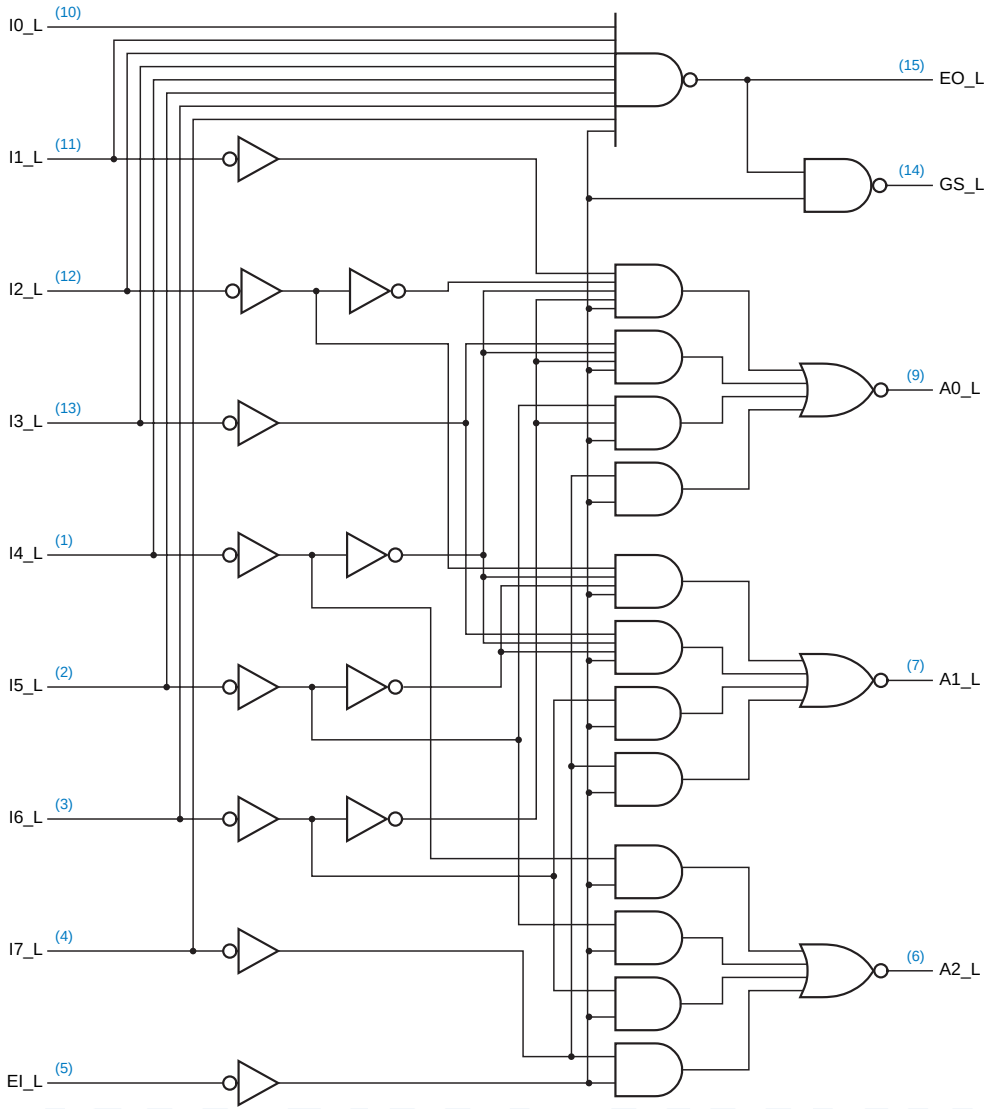


Figure 14: Logic diagram for the 74x148 8-input priority encoder, including pin numbers for a standard 16-pin dual in-line package.

Encoders: The 74x148 Priority Encoder

Table 12: Truth table for a 74x148 8-input priority encoder.

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

Encoders: The 74x148 Priority Encoder

- Instead of an IDLE output, '148 has a GS_L (Group Select) output
  - It is asserted when device is enabled and one or more of request inputs are asserted
- EO_L signal is an enable output used for cascading
  - It is designed to be connected to EI_L input of another '148 that handles lower-priority requests
  - EO_L is asserted if EI_L is asserted but no request input is asserted; thus, a low-priority '148 may be enabled

Encoders: The 74x148 Priority Encoder

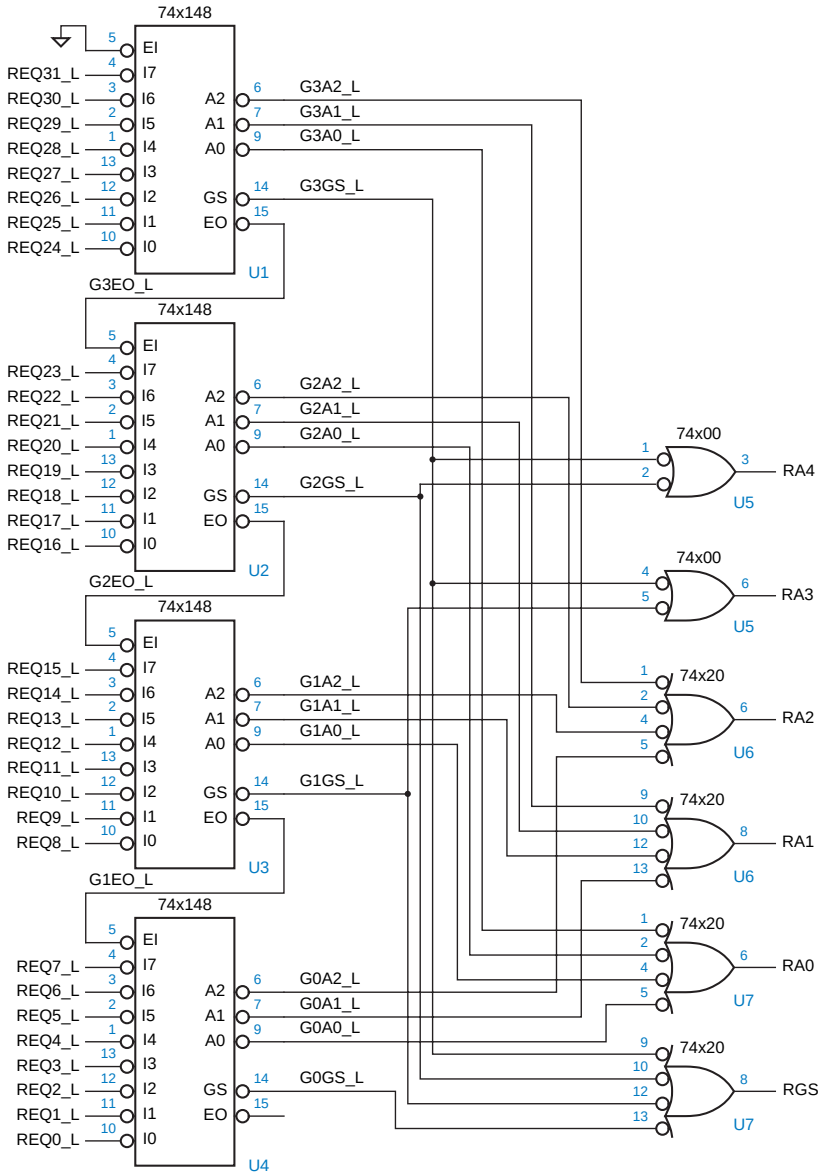


Figure 15: Four 74x148s cascaded to handle 32 requests.

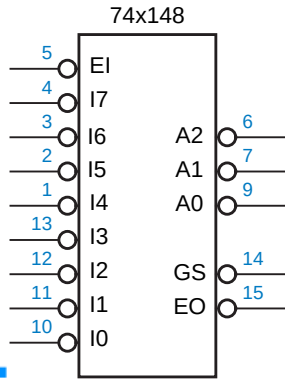
Encoders: The 74x148 Priority Encoder

- In Fig. 15
  - There are 32 request inputs and a 5-bit output, RA4–RA0, indicating the highest-priority requestor
  - Since A2–A0 outputs of at most one '148 will be enabled at any time, outputs of individual '148s can be ORed to produce RA2–RA0
  - Individual GS_L outputs can be combined in a 4-to-2 encoder to produce RA4 and RA3
  - RGS output is asserted if any GS output is asserted

Table 13: Behavioral Verilog module for a 74x148-like 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
```



Three-State Devices: Three-State Buffers

- The most basic three-state device is a *three-state buffer*, often called a *three-state driver*

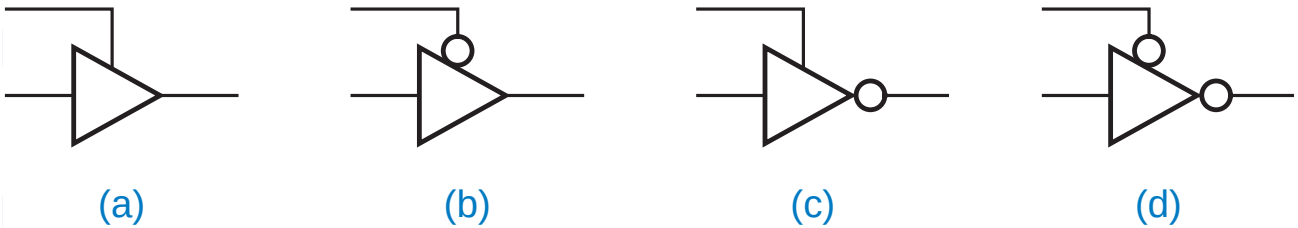


Figure 16: Various three-state buffers: (a) non-inverting, active-high enable; (b) non-inverting, active-low enable; (c) inverting, active-high enable; (d) inverting, active-low enable.

- When enable input is asserted, device behaves like an ordinary buffer or inverter
- When enable input is negated, device output floats
  - It goes to a high-impedance (Hi-Z), disconnected state and functionally behaves as if it were not even there
- Three-state devices allow multiple sources to share a single "party line," as long as only one device talks on the line at a time

Three-State Devices: Three-State Buffers

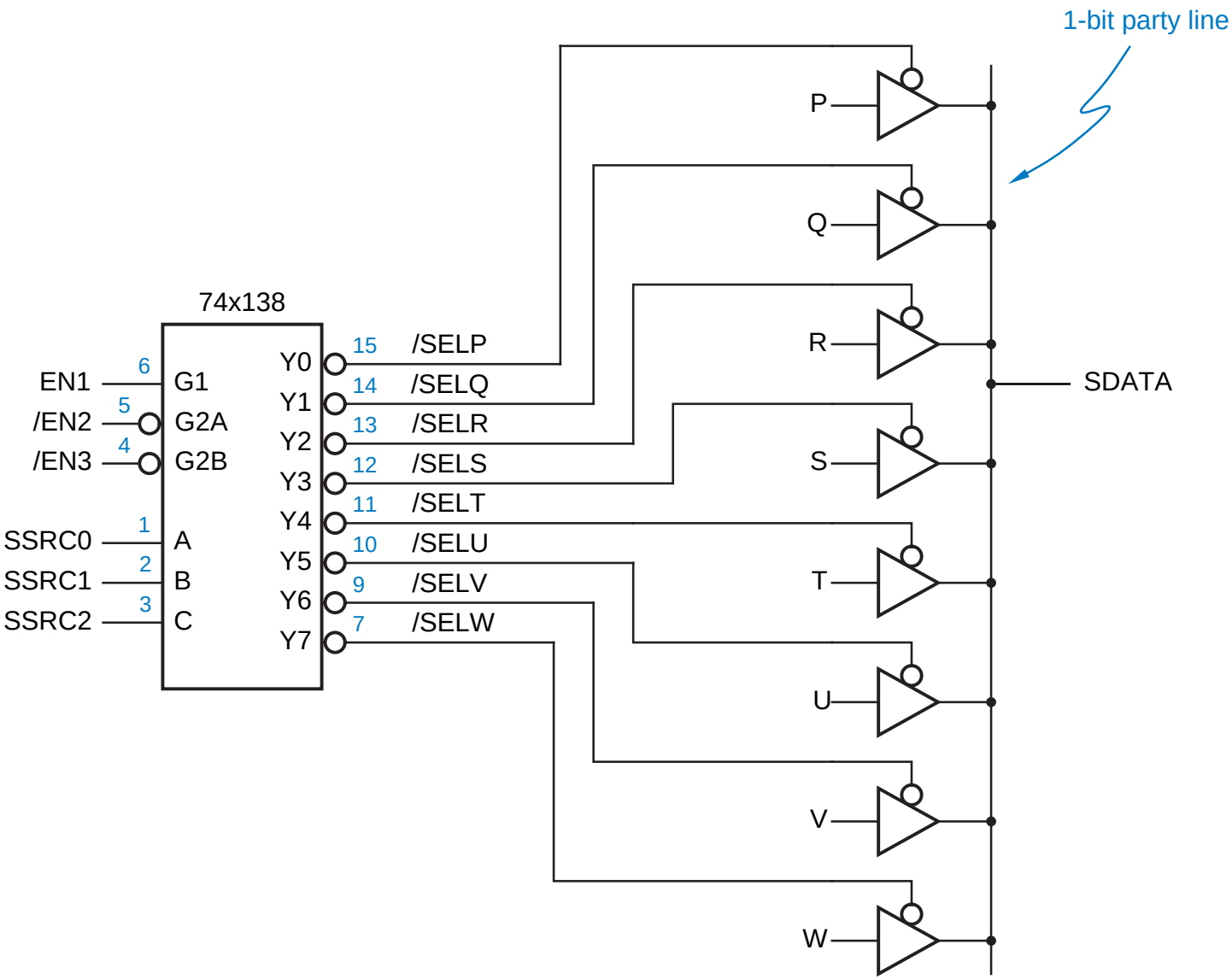


Figure 17: Eight sources sharing a three-state party line.

Three-State Devices: Three-State Buffers

- Three-state devices are designed so that they go into Hi-Z state faster than they come out of Hi-Z state
  - t_{pLZ} and t_{pHZ} are both less than t_{pZL} and t_{pZH}
  - If outputs of two three-state devices are connected to same party line, and we simultaneously disable one and enable other, the first device will get off party line before the second one gets on
  - If both devices were to drive party line at same time, and if both were trying to maintain opposite output values (0 and 1), then excessive current would flow and create noise in system (*fighting*)
- Delays and timing skews in control circuits make it difficult to ensure that enable inputs of different three-state devices change simultaneously
  - Even when this is possible, a problem arises if three-state devices from different-speed logic families are connected to same party line
  - t_{pZL} or t_{pZH} of a fast device may be shorter than t_{pLZ} or t_{pHZ} of a slow one

Three-State Devices: Three-State Buffers

- The only safe way to use three-state devices is to design control logic that guarantees a **dead time** on party line during which no one is driving it
  - Dead time must be long enough to account for worst-case differences between turn-off and turn-on times of devices and for skews in three-state control signals

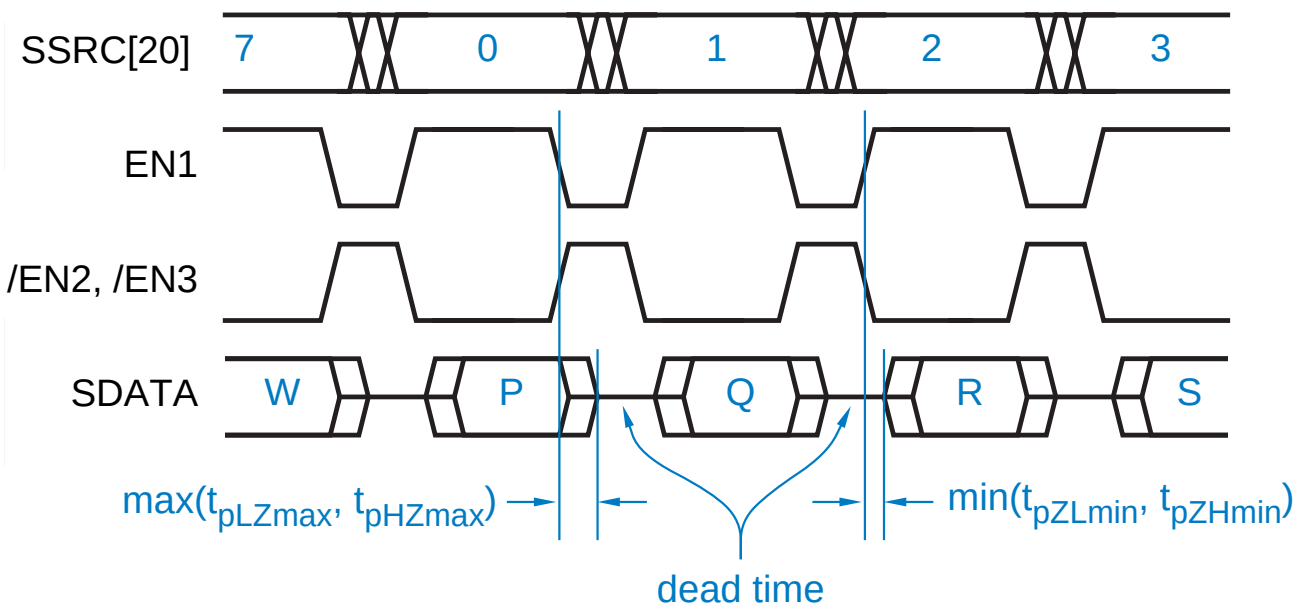


Figure 18: Timing diagram for the three-state party line of Fig. 17.

Three-State Devices: SSI and MSI Three-State Buffers

- Each of 74x125 and 74x126 contains four independent non-inverting three-state buffers in a 14-pin package

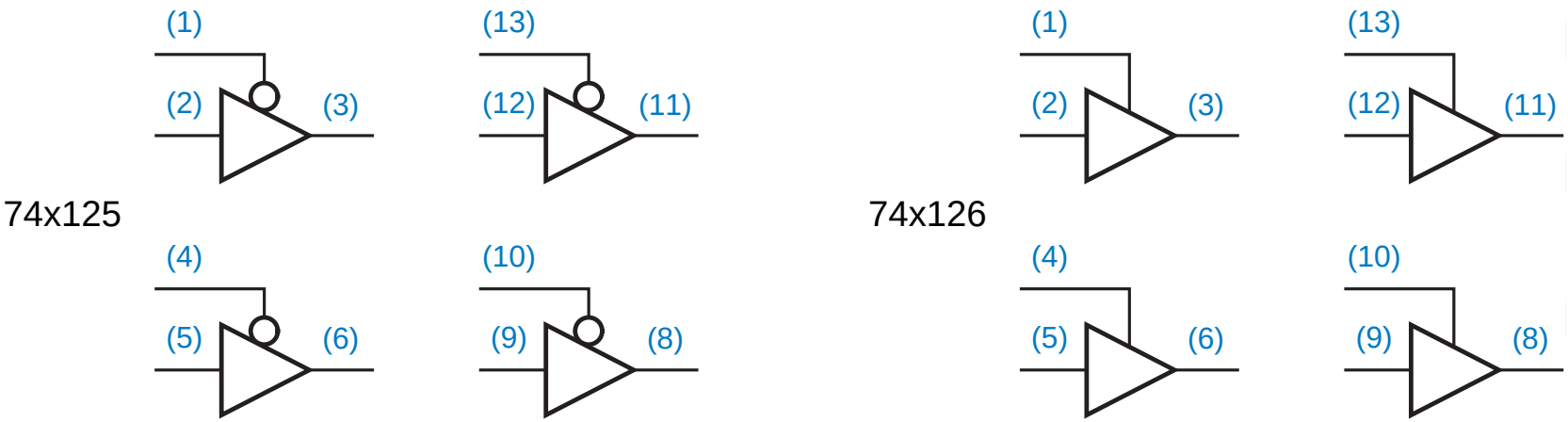


Figure 19: Pinouts of the 74x125 and 74x126 three-state buffers.

Three-State Devices: SSI and MSI Three-State Buffers

- Most party-line applications use a bus with more than one bit of data
  - E.g., in an 8-bit microprocessor system, data bus is eight bits wide, and peripheral devices place data on bus eight bits at a time
  - A peripheral device enables eight three-state drivers to drive bus, all at the same time
- To reduce package size in wide-bus applications, MSI parts contain multiple three-state buffers with common enable inputs

Three-State Devices: SSI and MSI Three-State Buffers

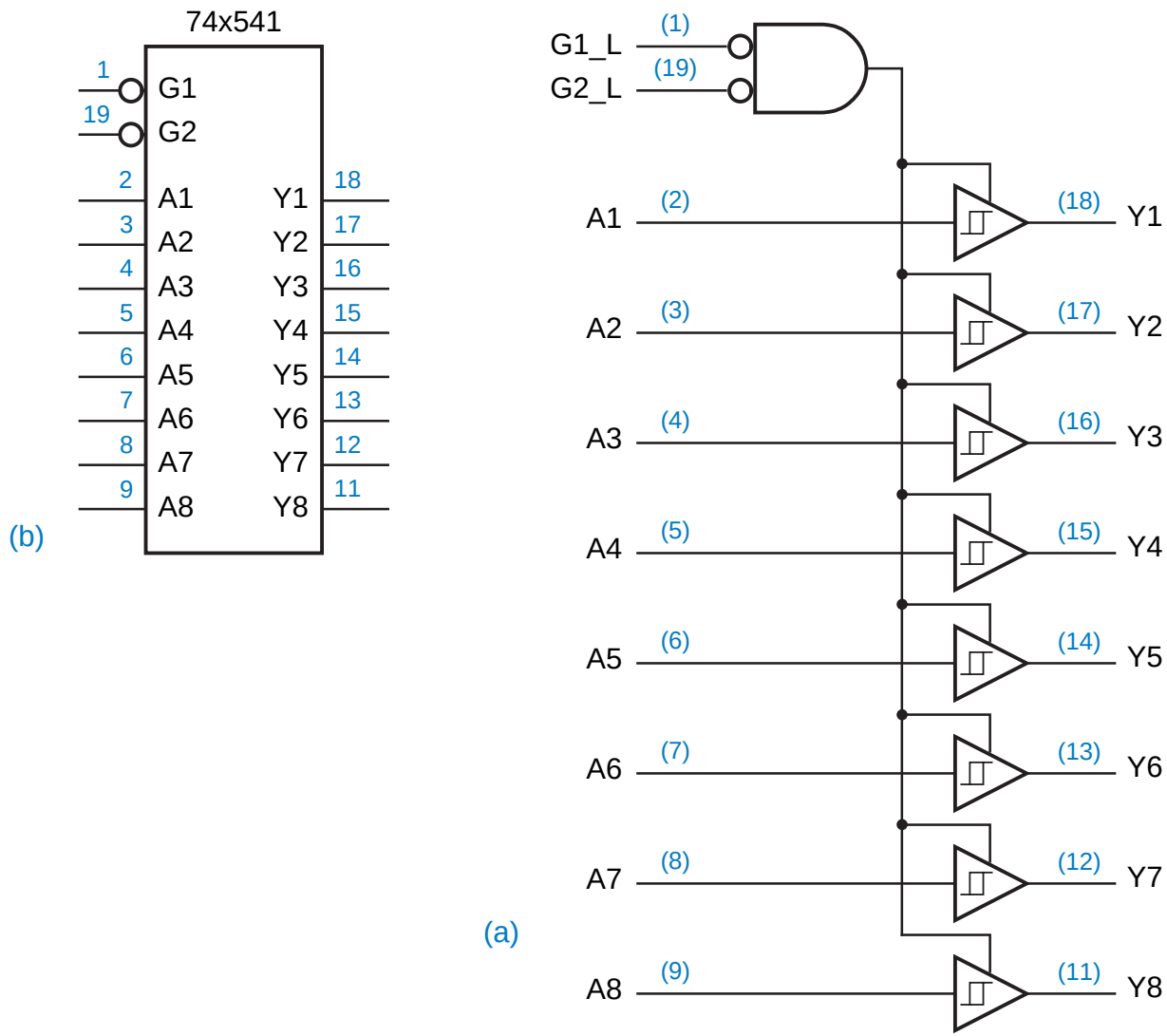


Figure 20: The 74x541 octal three-state buffer: (a) logic diagram, including pin numbers for a standard 20-pin dual in-line package; (b) traditional logic symbol.

Three-State Devices: SSI and MSI Three-State Buffers

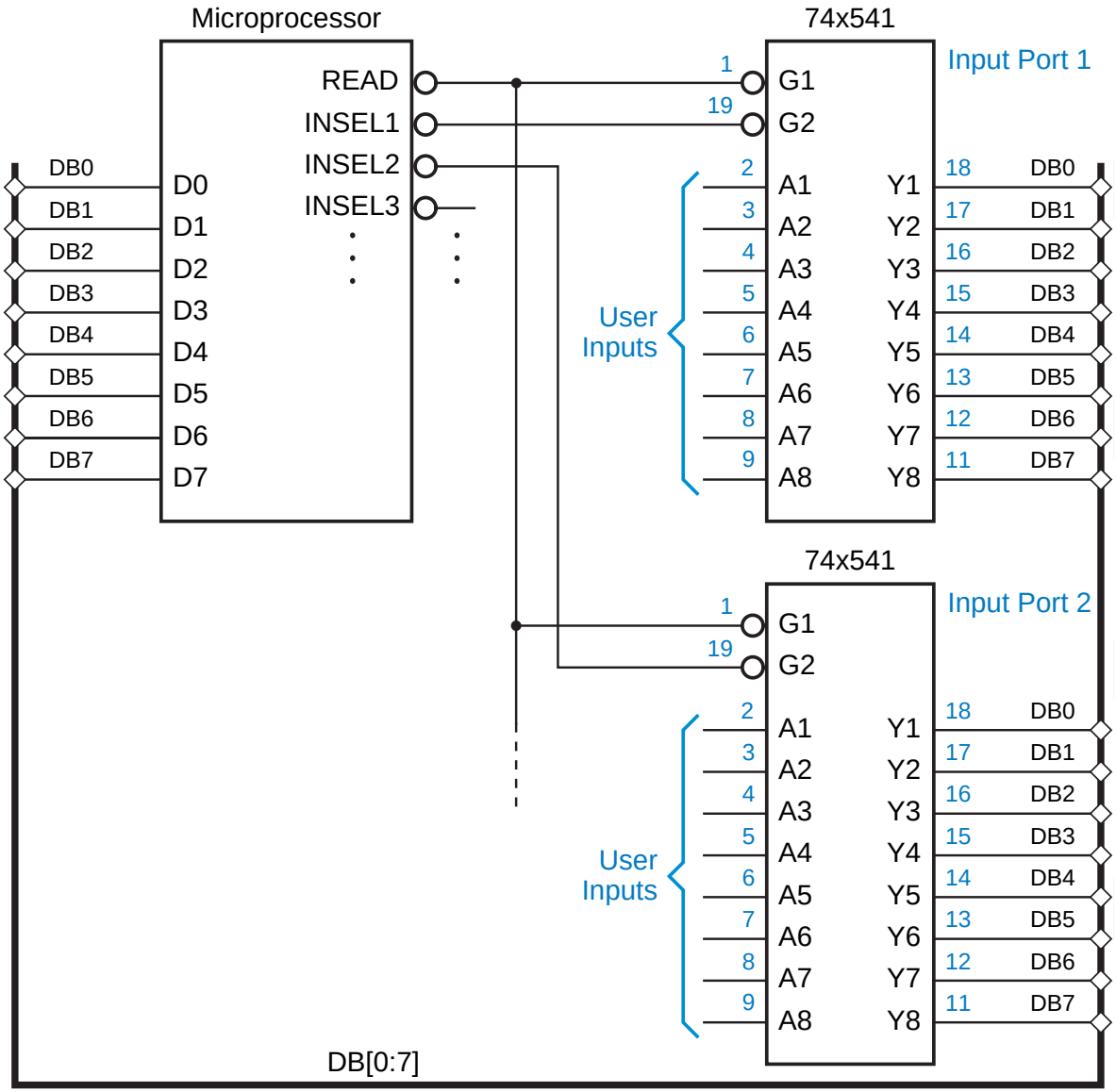


Figure 21: Using a 74x541 as a microprocessor input port.

Three-State Devices: SSI and MSI Three-State Buffers

- In Fig. 21, microprocessor selects Input Port 1 (top 74x541) by asserting INSEL1 and requests a read operation by asserting READ
  - Selected 74x541 responds by driving microprocessor data bus with user-supplied input data
  - Other input ports may be selected when a different INSEL line is asserted along with READ
- A **bus transceiver** contains pairs of three-state buffers connected in opposite directions between each pair of pins, so that data can be transferred in either direction
  - A bus transceiver is typically used between two **bidirectional buses**

Three-State Devices: SSI and MSI Three-State Buffers

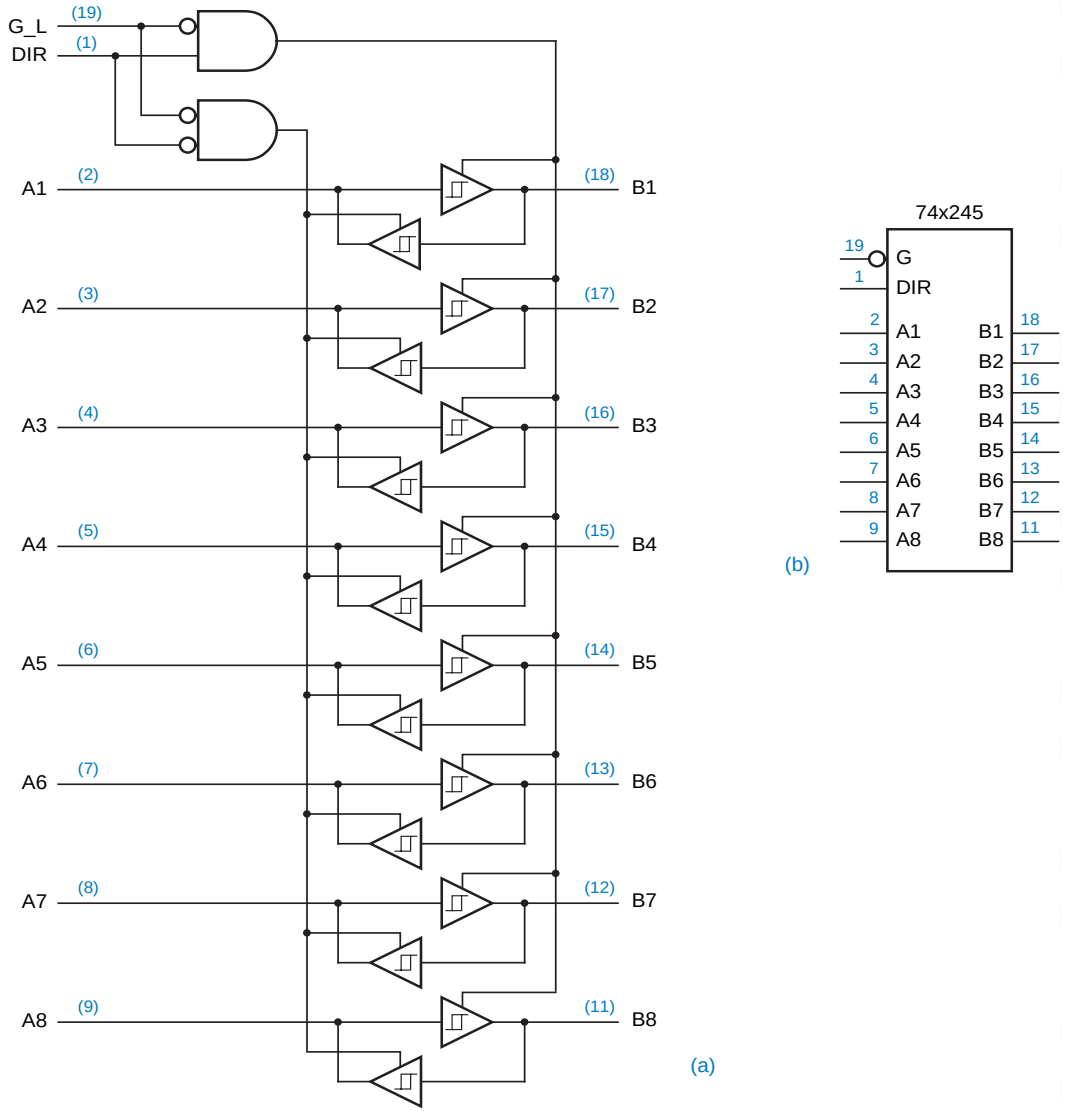


Figure 22: The 74x245 octal three-state transceiver: (a) logic diagram; (b) traditional logic symbol.

Three-State Devices: SSI and MSI Three-State Buffers

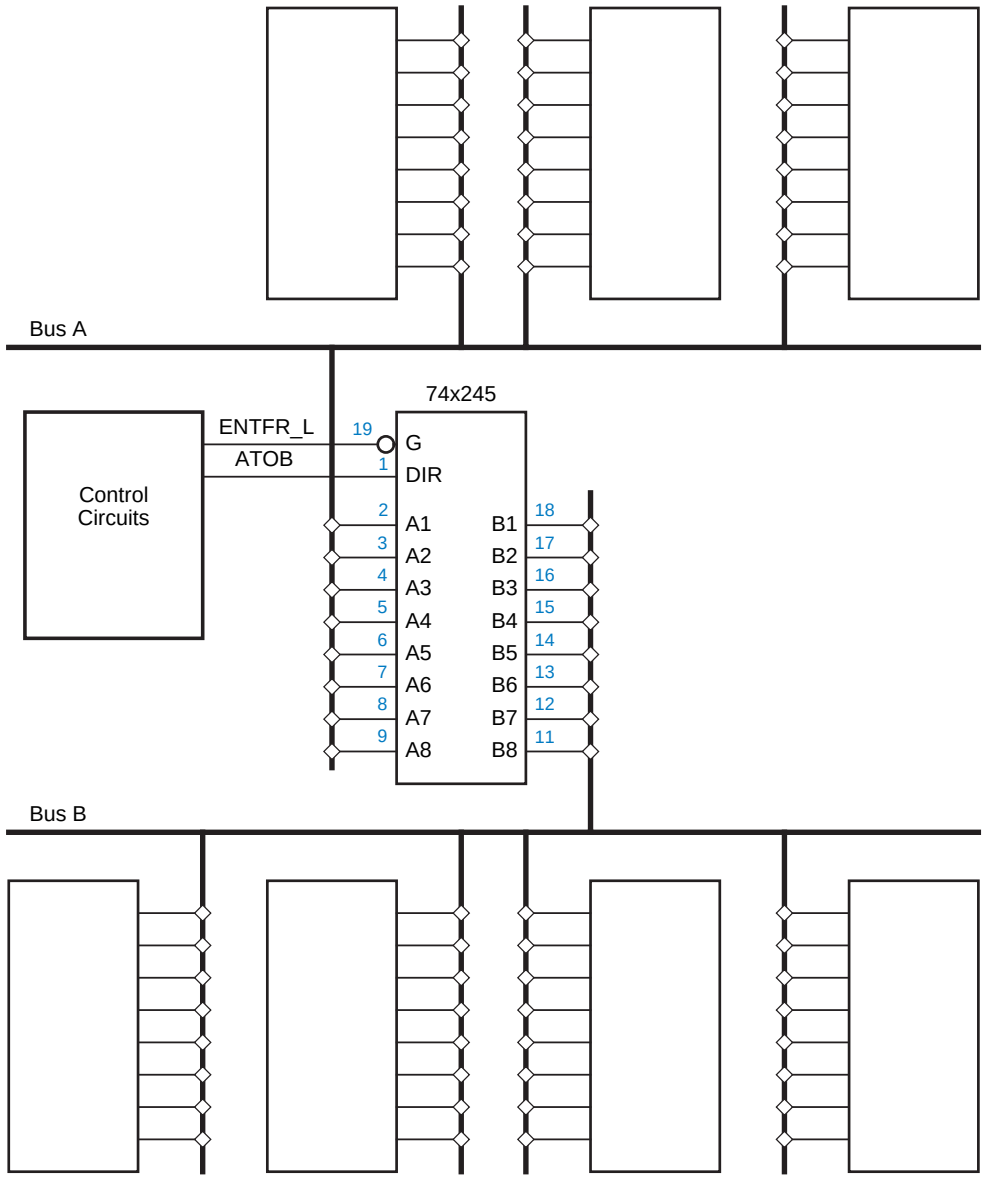


Figure 23: Bidirectional buses and transceiver operation.

Table 14: Modes of operation for a pair of bidirectional buses.

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 15: Verilog module for a 74x541-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
```

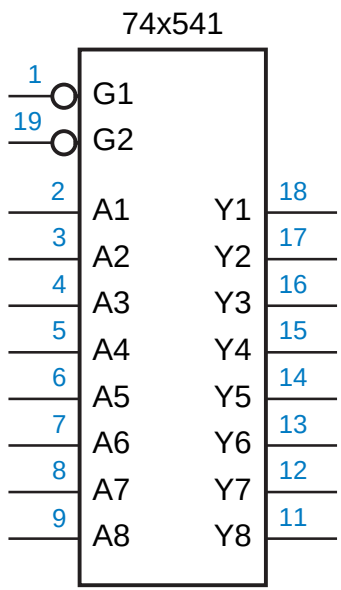


Table 16: Verilog module for a 74x245-like 8-bit transceiver.

```
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
```

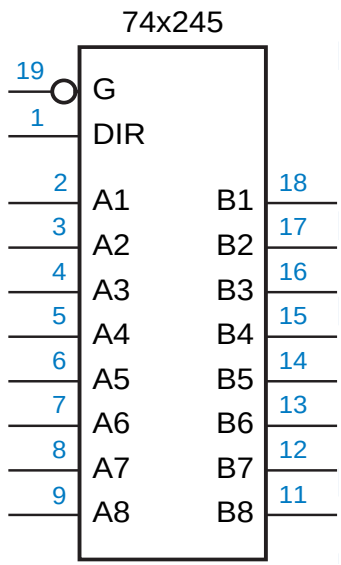


Table 17: Verilog module for a four-way, 8-bit bus transceiver.

```
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
```

Three-State Devices: Three-State Outputs in Verilog

- Tab. 17
  - Transceiver handles four 8-bit bidirectional buses, `A[1:8]`, `B[1:8]`, `C[1:8]`, and `D[1:8]`
  - Each bus has its own output enable input, AOE_L–DOE_L, and a master enable input MOE_L must also be asserted for any bus to be driven
  - The same source of data is driven to all buses, as selected by `S[2:0]`
    - If `S2 = 0`, buses are driven with a constant value
    - When selected source is a bus, the selected source bus cannot be driven, even if it is output-enabled

Table 18: 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

Multiplexers

- A multiplexer is a digital switch
  - It connects data from one of *n* sources to its output

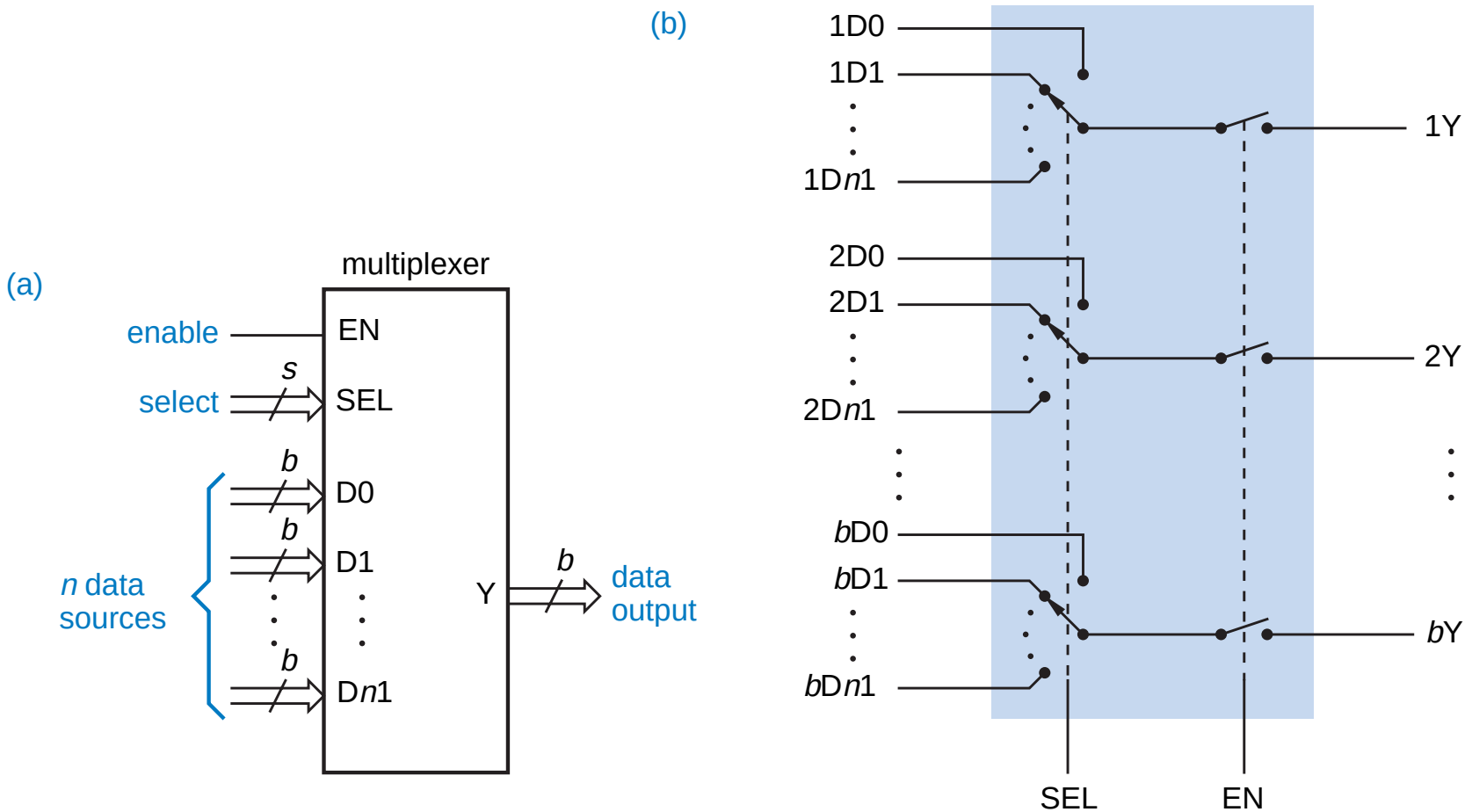


Figure 24: Multiplexer structure: (a) inputs and outputs; (b) functional equivalent.

Multiplexers

- Fig. 24(a) shows inputs and outputs of an n-input, b-bit multiplexer
  - There are s inputs that select among n sources, so $s = \lceil \log_2 n \rceil$
  - When $EN = 0$, all of outputs are 0

Multiplexers: Standard MSI Multiplexers

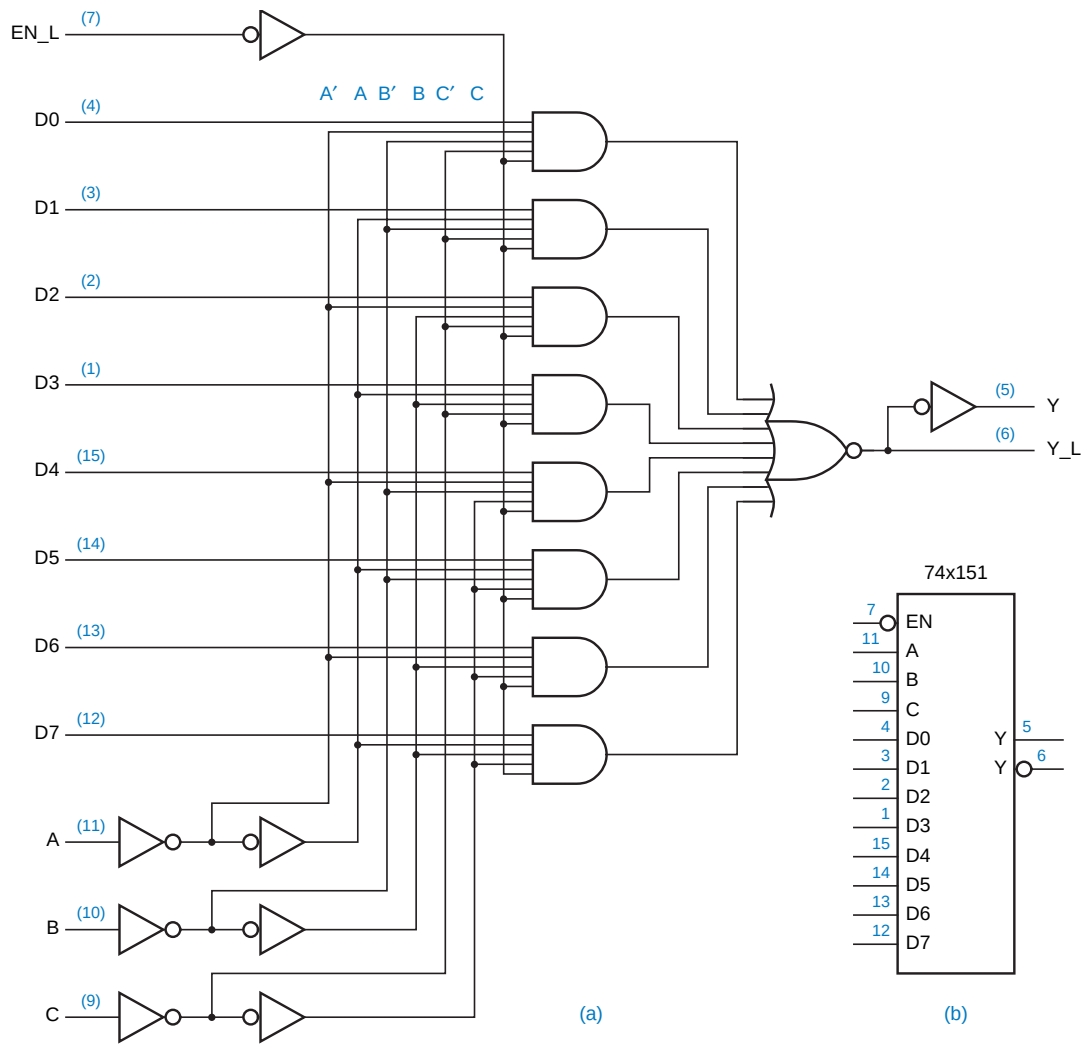


Figure 25: The 74x151 8-input, 1-bit multiplexer: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol.

Table 19: Truth table for a 74x151 8-input, 1-bit multiplexer.

Inputs				Outputs	
EN_L	C	B	A	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’

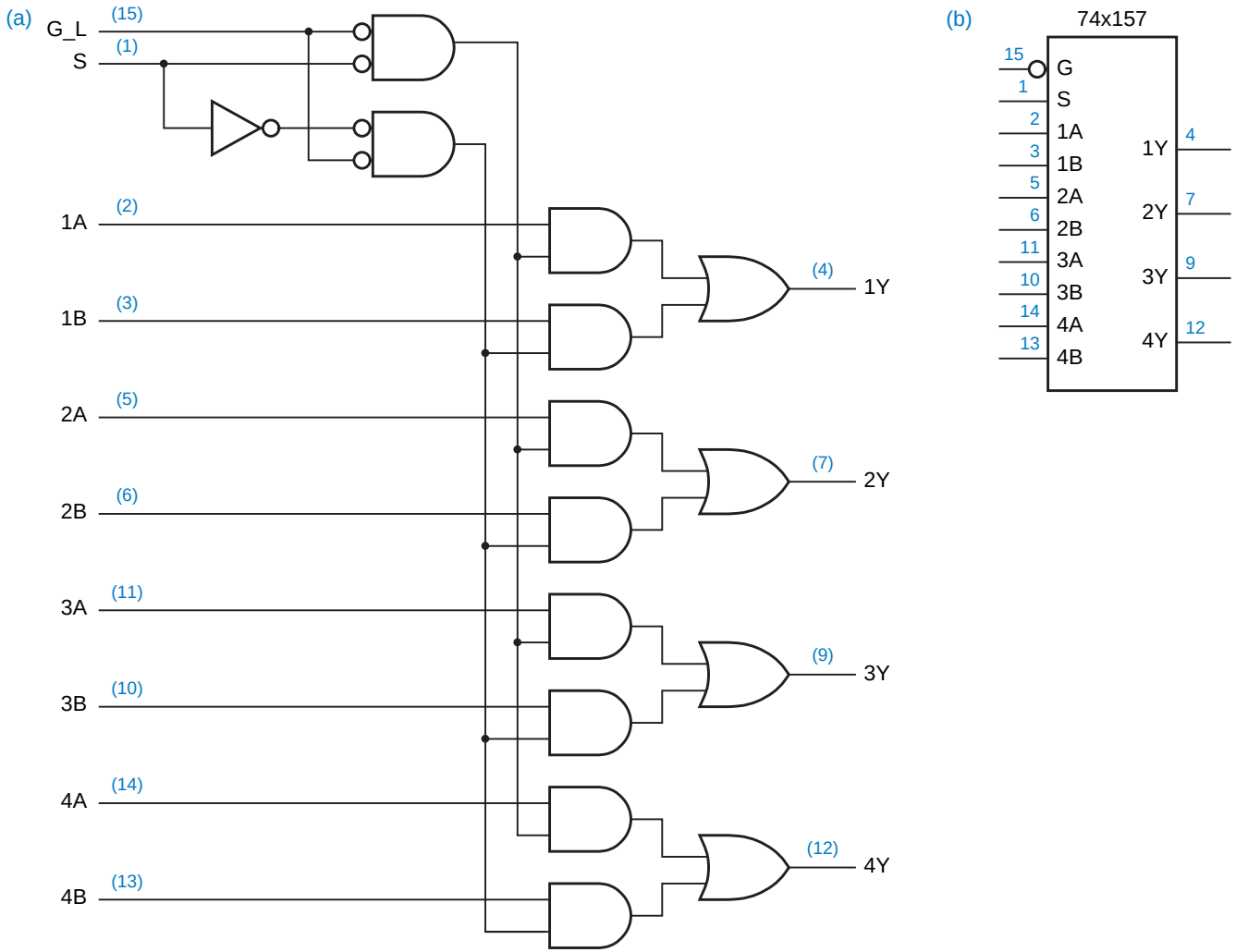


Figure 26: The 74x157 2-input, 4-bit multiplexer: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol.

Table 20: Truth table for a 74x157 2-input, 4-bit multiplexer.

Inputs		Outputs			
G_L	S	1Y	2Y	3Y	4Y
1	x	0	0	0	0
0	0	1A	2A	3A	4A
0	1	1B	2B	3B	4B

Table 21: Truth table for a 74x153 4-input, 2-bit multiplexer.

Inputs				Outputs	
1G_L	2G_L	B	A	1Y	2Y
0	0	0	0	1C0	2C0
0	0	0	1	1C1	2C1
0	0	1	0	1C2	2C2
0	0	1	1	1C3	2C3
0	1	0	0	1C0	0
0	1	0	1	1C1	0
0	1	1	0	1C2	0
0	1	1	1	1C3	0
1	0	0	0	0	2C0
1	0	0	1	0	2C1
1	0	1	0	0	2C2
1	0	1	1	0	2C3
1	1	x	x	0	0

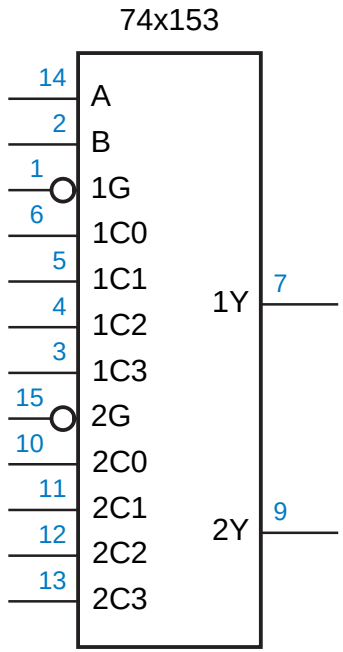


Figure 27: Traditional logic symbol for the 74x153.

Multiplexers: Standard MSI Multiplexers

- Some multiplexers have three-state outputs
  - Enable input, instead of forcing outputs to zero, forces them to Hi-Z state
  - Three-state outputs are useful when *n*-input muxes are combined to form larger muxes

Multiplexers: Expanding Multiplexers

- Size of an MSI multiplexer seldom matches characteristics of problem at hand
  - E.g., an 8-input, 32-bit multiplexer might be used in design of a processor
  - We use 32 74x151 8-input, 1-bit multiplexers, each handling one bit of all inputs and output
  - Processor's 3-bit register-select field is connected to A, B, and C inputs of all 32 muxes, so they all select same register source at any given time
- Another dimension in which multiplexers can be expanded is number of data sources
  - E.g., a 32-input, 1-bit multiplexer

Multiplexers: Expanding Multiplexers

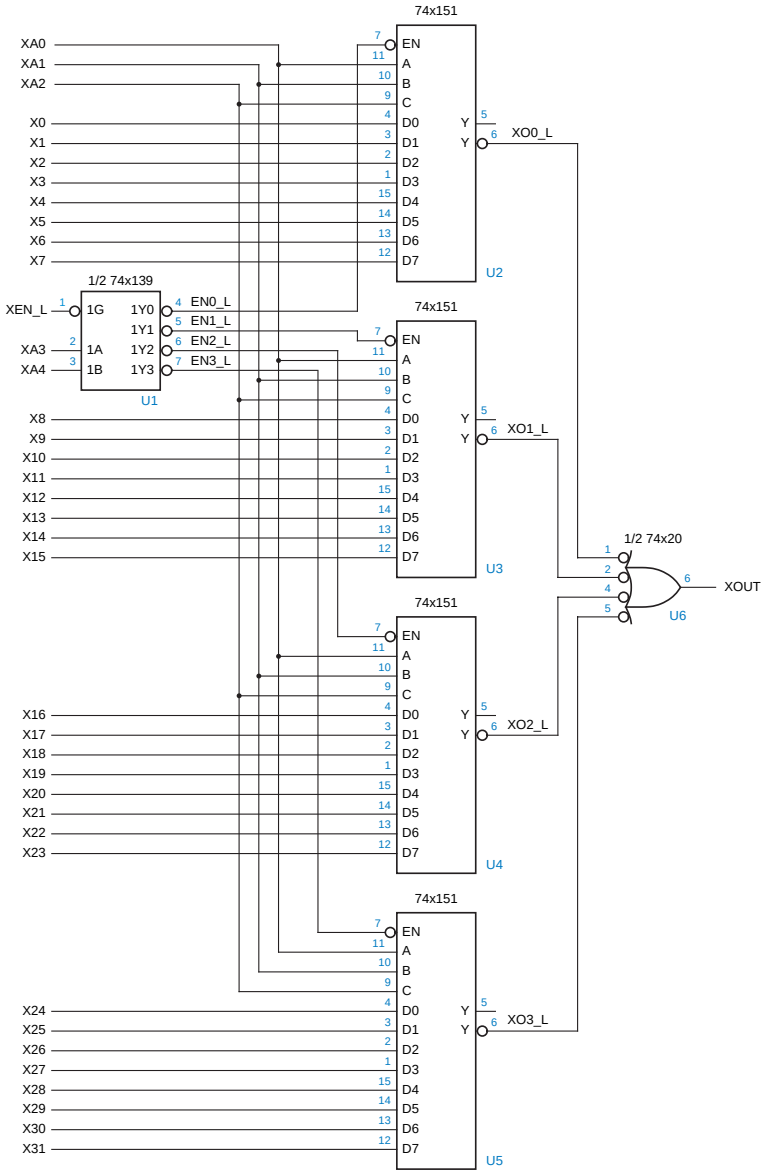


Figure 28: Combining 74x151s to make a 32-to-1 multiplexer.

Multiplexers: Expanding Multiplexers

- 74x251 is identical to '151 in its pinout and its internal logic design, except that Y and Y_L are three-state outputs
  - 32-to-1 multiplexer can also be built using 74x251s
    - The circuit is identical to Fig. 28, except that output NAND gate is eliminated
    - Instead, Y outputs of four '251s are simply tied together

Multiplexers, Demultiplexers, and Buses

- A multiplexer can be used to select one of n sources of data to transmit on a bus
  - At far end of bus, a **demultiplexer** can be used to route bus data to one of m destinations

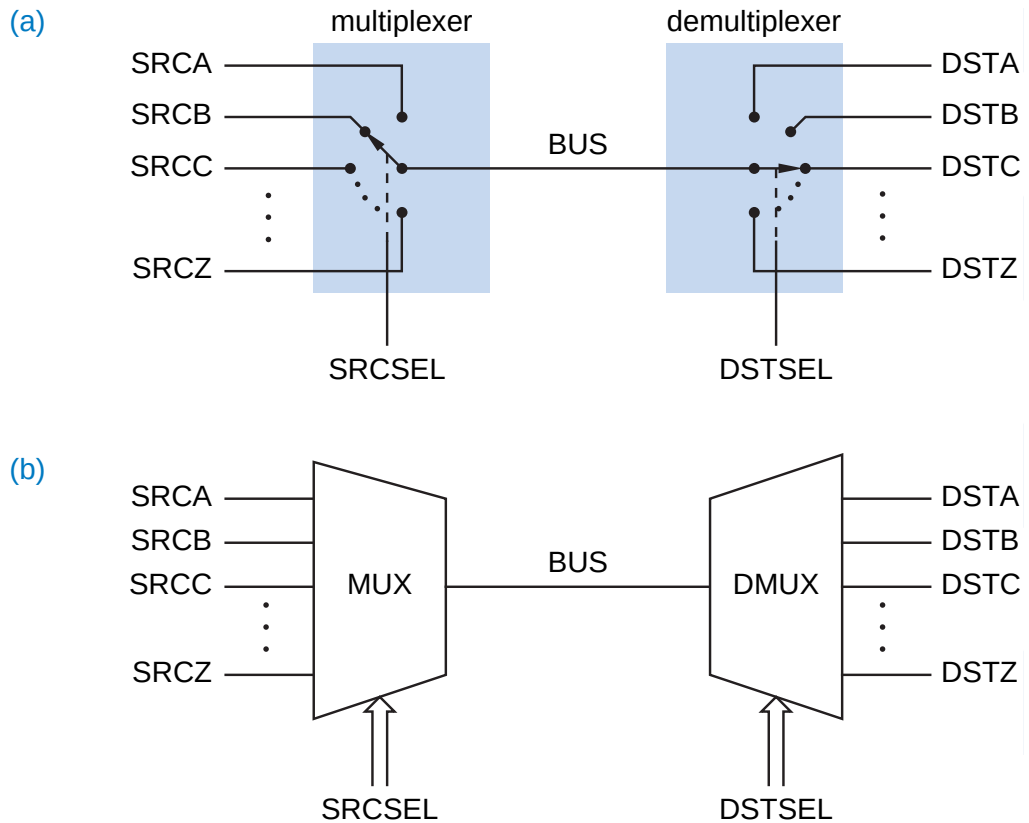


Figure 29: A multiplexer driving a bus and a demultiplexer receiving the bus: (a) switch equivalent; (b) block-diagram symbols.

Multiplexers, Demultiplexers, and Buses

- Function of a demultiplexer is inverse of a multiplexer's
  - A b-bit, n-output demultiplexer has b data inputs and s inputs to select one of $n = 2^s$ sets of b data outputs
- A binary decoder with an enable input can be used as a demultiplexer

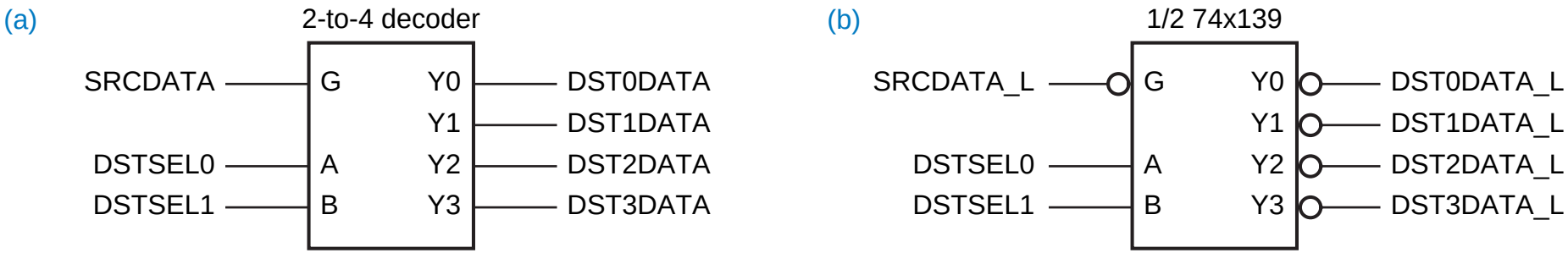


Figure 30: Using a 2-to-4 binary decoder as a 1-bit, 4-output demultiplexer: (a) generic decoder; (b) 74x139.

- Decoder's enable input is connected to data line, and its select inputs determine which of its output lines is driven with data bit

Table 22: Dataflow Verilog program for a 4-input, 8-bit multiplexer.

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

```
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
```

Multiplexers in Verilog

Table 24: Behavioral Verilog program for a specialized 4-input, 18-bit multiplexer.

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

S2	S1	S0	*Input to Select*
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

References

John F. Wakerly, *Digital Design: Principles and Practices (4th Edition)*, Prentice Hall, 2005.