Dec: More Decoders

This section describes a couple of 74-series combinational decoders.

Dec.1 The 74x139 Dual 2-to-4 Decoder

Two independent and identical 2-to-4 decoders are contained in a single MSI part, the *74x139*. The gate-level circuit diagram for this IC is shown in Figure Dec-1(a). Notice that the outputs and the enable input of the ’139 are active-low. Most MSI decoders were originally designed with active-low outputs, since TTL inverting gates are generally faster than noninverting ones. Also notice that the ’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 the fanout budget of the device that drives it.

74x139

A logic symbol for the 74x139 is shown in Figure Dec-1(b). Notice that all of the signal names inside the symbol outline are active high (no “_L”), and that inversion bubbles indicate active-low inputs and outputs. Often a schematic may

Figure Dec-1 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.

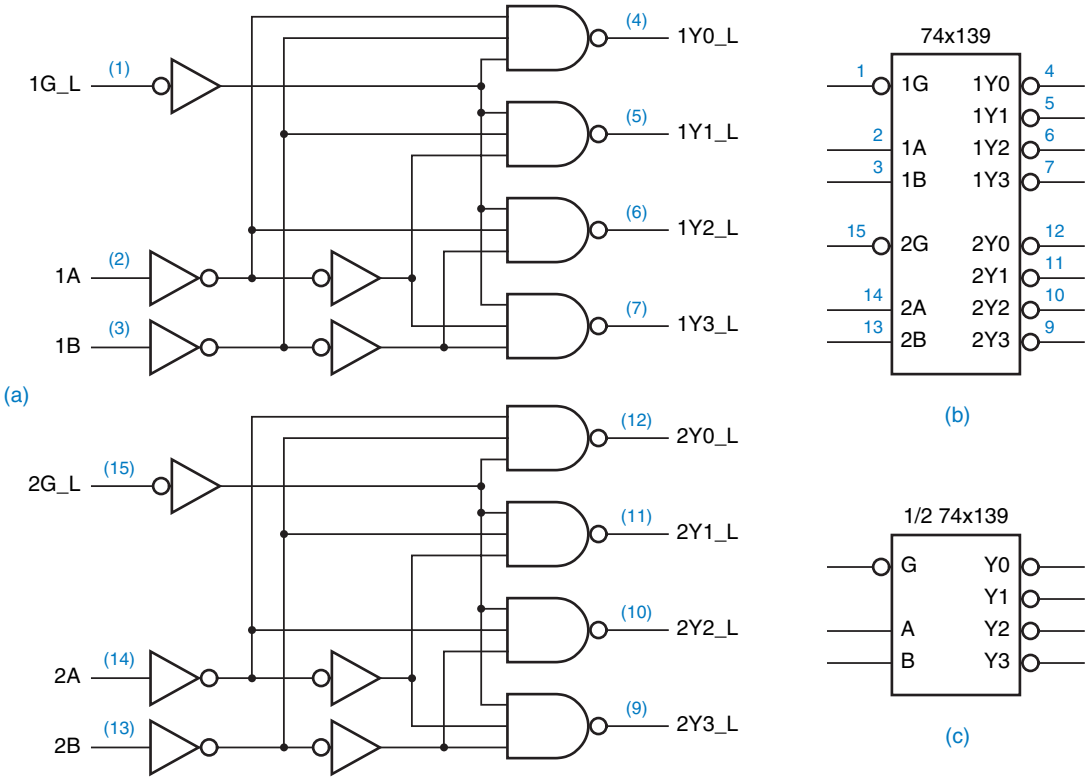


Table Dec-1 Truth table for one-half of a 74x139 dual 2-to-4 decoder.

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

use a generic symbol for just one decoder, one-half of a '139, as shown in (c). In this case, the assignment of the generic function to one half or the other of a particular '139 package can be deferred until the schematic is completed.

Table Dec-1 is the truth table for a 74x139-type decoder. The truth tables in some manufacturers' data books use L and H to denote the input and output signal voltage levels, so there can be no ambiguity about the electrical function of the device; a truth table written this way is sometimes called a *function table*. However, since we use positive logic throughout this book, we can use 0 and 1 without ambiguity. In any case, the truth table gives the logic function in terms of the *external pins* of the device. A truth table for the function performed *inside* the symbol outline would look just like Table 6-4, except that the input signal names would be G, B, A.

function table

Some logic designers draw the symbol for 74x139s and other logic functions without inversion bubbles. Instead, they use an overbar on signal names inside the symbol outline to indicate negation, as shown in Figure Dec-2(a). This notation is self-consistent, but it is inconsistent with our drawing standards for bubble-to-bubble logic design. The symbol shown in (b) is absolutely *incorrect*: according to this symbol, a logic 1, not 0, must be applied to the enable pin to enable the decoder.

Dec.2 Seven-Segment Decoders

Look at your wrist and you'll probably see a *seven-segment display*. This type of display, which normally uses light-emitting diodes (LEDs) or liquid-crystal

seven-segment display

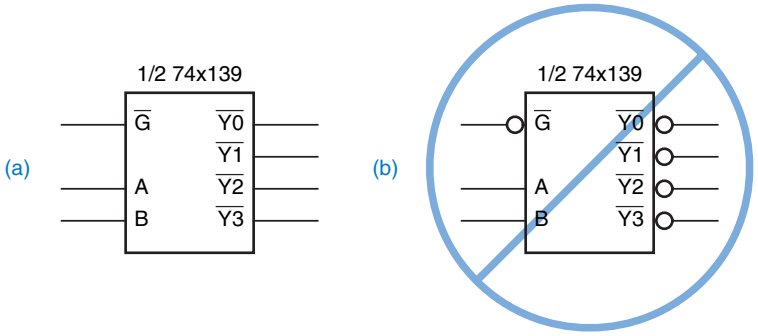


Figure Dec-2 More ways to symbolize a 74x139: (a) correct but to be avoided; (b) incorrect because of double negations.

BAD NAMES Some manufacturers' data sheets have inconsistencies similar to Figure Dec-2(b). For example, Texas Instruments' data sheet for the 74AHC139 uses active-low names like $\overline{1G}$ for the enable inputs, with the overbar indicating an active-low pin, but active-high names like 1Y0 for all the active-low output pins. On the other hand, Motorola's data sheet for the 74VHC139 correctly uses overbars on the names for both the enable inputs and the outputs, but the overbars are barely visible in the device's function table due to a typographical problem.

I've also had the personal experience of building a printed-circuit board with many copies of a new device from a vendor whose documentation clearly indicated that a particular input was active low, only to find out upon the first power-on that the input was active high.

The moral of the story is that you have to study the description of each device to know what's really going on. And if it's a brand-new device, whether from a commercial vendor or your own company's ASIC group, you should double-check all of the signal polarities and pin assignments before committing to a PCB. Rest assured, however, that the signal names in this text *are* consistent and correct.

display (LCD) elements, is used in watches, calculators, and instruments to display decimal data. A digit is displayed by illuminating a subset of the seven line segments shown in Figure Dec-3(a).

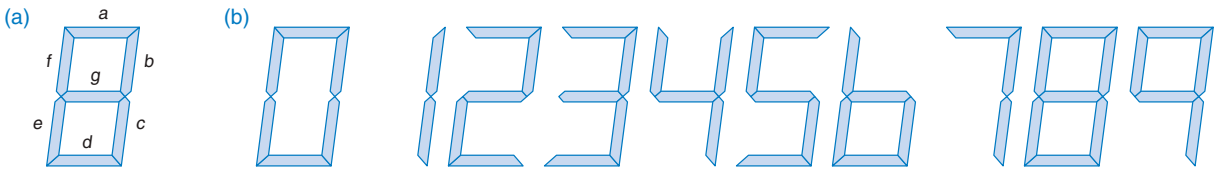
seven-segment decoder

A *seven-segment decoder* has 4-bit BCD as its input code and the "seven-segment code," which is graphically depicted in Figure Dec-3(b), as its output code. Figure Dec-4 and Table Dec-2 on the next pages are the logic diagram and truth table for a *74x49* seven-segment decoder. Except for the strange (clever?) connection of the "blanking input" BI_L, each output of the 74x49 is a minimal product-of-sums realization for the corresponding segment, assuming "don't-cares" for the nondecimal input combinations. The INVERT-OR-AND structure used for each output may seem a little strange, but it is equivalent under the generalized DeMorgan's theorem to an AND-OR-INVERT gate, which is a fairly fast and compact structure to build in CMOS or TTL.

74x49

Most modern seven-segment display elements have decoders built into them, so that a 4-bit BCD word can be applied directly to the device. Many of the older, discrete seven-segment decoders have special high-voltage or high-

Figure Dec-3 Seven-segment display: (a) segment identification; (b) decimal digits.



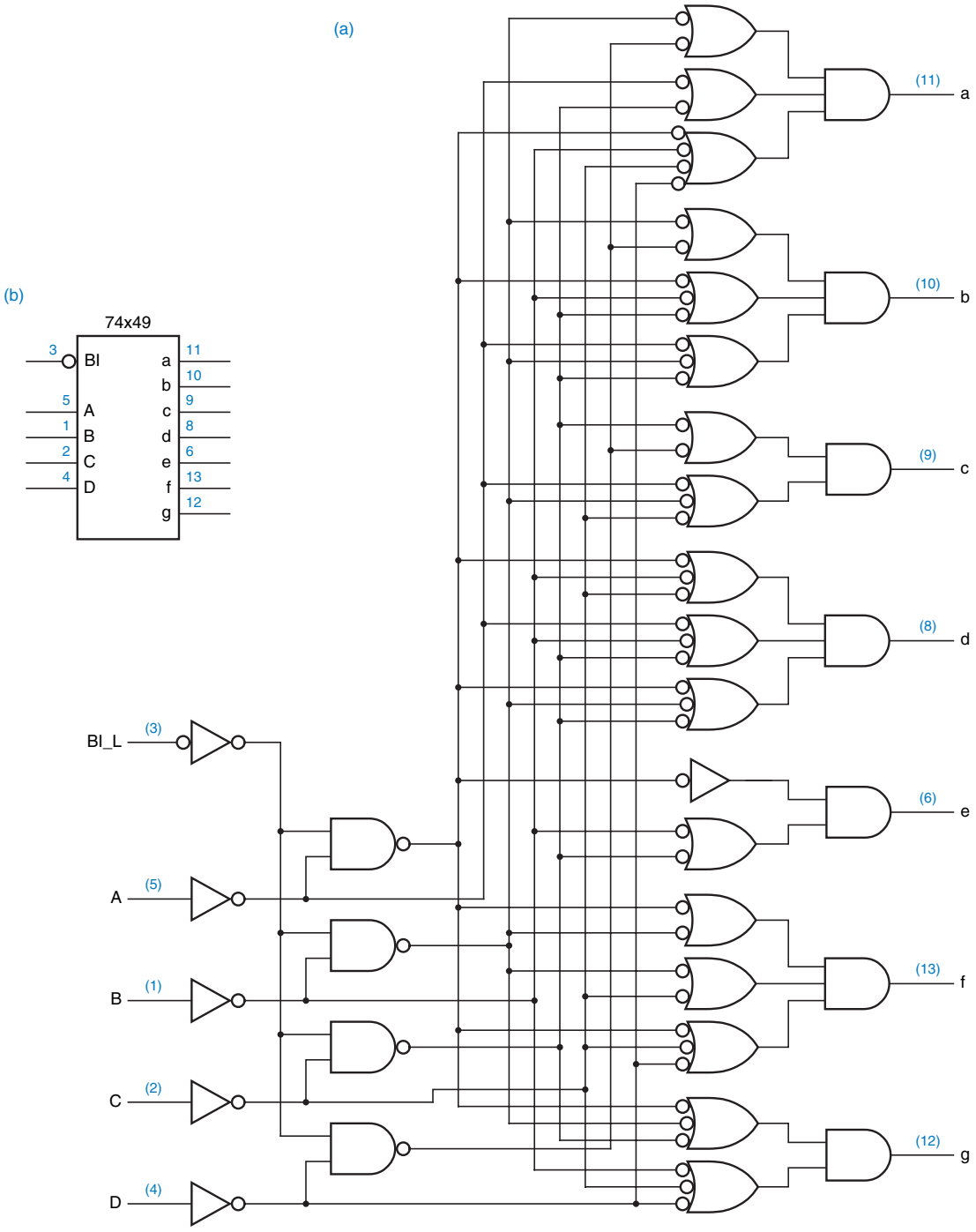


Figure Dec-4 The 74x49 seven-segment decoder: (a) logic diagram, including pin numbers; (b) traditional logic symbol.

Table Dec-2 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

current outputs that are well suited for driving large, high-powered display elements.

Table Dec-3 on the next page is an ABEL program for a seven-segment decoder. Sets are used to define the digit patterns to make the program more readable.

Exercises

Dec.1 Draw the digits created by a 74x49 seven-segment decoder for the non-decimal inputs 1010 through 1111.

Dec.2 Modify the seven-segment-decoder ABEL program in Table Dec-3 so that the digits 6 and 9 have tails as shown in Figure xDec.2. In addition, display the character "E" for nondecimal inputs 1010 through 1111. Synthesize both the original program and the modified program for your favorite CPLD, and compare the number of product terms in the two versions.

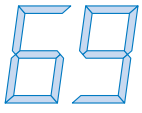


Figure xDec.2

Table Dec-3 ABEL program for a 74x49-like seven-segment decoder.

```
module Z74X49H
title 'Seven-Segment_Decoder
J. Wakerly, Micro Design Resources, Inc.'
Z74X49H device 'P16V8C';

" Input pins
A, B, C, D                  pin 1, 2, 3, 4;
BI_L                        pin 5;
" Output pins
SEGA, SEGB, SEGC, SEGD      pin 19, 18, 17, 16 istype 'com';
SEGE, SEGF, SEGG            pin 15, 14, 13     istype 'com';

" Definitions
BI = !BI_L;
DIGITIN = [D,C,B,A];
SEGOUT = [SEGA,SEGB,SEGC,SEGD,SEGE,SEGF,SEGG];

" Segment encodings for digits
DIG0 = [1,1,1,1,1,1,0];  " 0
DIG1 = [0,1,1,0,0,0,0];  " 1
DIG2 = [1,1,0,1,1,0,1];  " 2
DIG3 = [1,1,1,1,0,0,1];  " 3
DIG4 = [0,1,1,0,0,1,1];  " 4
DIG5 = [1,0,1,1,0,1,1];  " 5
DIG6 = [1,0,1,1,1,1,1];  " 6  'tail' included
DIG7 = [1,1,1,0,0,0,0];  " 7
DIG8 = [1,1,1,1,1,1,1];  " 8
DIG9 = [1,1,1,1,0,1,1];  " 9  'tail' included
DIGA = [1,1,1,0,1,1,1];  " A
DIGB = [0,0,1,1,1,1,1];  " b
DIGC = [1,0,0,1,1,1,0];  " C
DIGD = [0,1,1,1,1,0,1];  " d
DIGE = [1,0,0,1,1,1,1];  " E
DIGF = [1,0,0,0,1,1,1];  " F

equations

SEGOUT = !BI & ( (DIGITIN ==  0) & DIG0 # (DIGITIN ==  1) & DIG1
               # (DIGITIN ==  2) & DIG2 # (DIGITIN ==  3) & DIG3
               # (DIGITIN ==  4) & DIG4 # (DIGITIN ==  5) & DIG5
               # (DIGITIN ==  6) & DIG6 # (DIGITIN ==  7) & DIG7
               # (DIGITIN ==  8) & DIG8 # (DIGITIN ==  9) & DIG9
               # (DIGITIN == 10) & DIGA # (DIGITIN == 11) & DIGB
               # (DIGITIN == 12) & DIGC # (DIGITIN == 13) & DIGD
               # (DIGITIN == 14) & DIGE # (DIGITIN == 15) & DIGF );

end Z74X49H
```

Dec.3 Enhance the ABEL program in Table Dec-3 by providing an input signal TAILS which, when asserted, causes the digits 6 and 9 have tails as shown in Figure xDec.2.

Dec.4 Starting with the ABEL program in Table Dec-3, write a program for a seven-segment decoder with the following enhancements:

- The outputs are all active low.
- Two new inputs, ENHEX and ERRDET, control segment-output decoding.
- If ENHEX = 0, the outputs match the behavior of a 74x49.
- If ENHEX = 1, then the outputs for digits 6 and 9 have tails, and the outputs for digits A–F are controlled by ERRDET.
- If ENHEX = 1 and ERRDET = 0, then the outputs for digits A–F look like the letters A–F, as in the original program.
- If ENHEX = 1 and ERRDET = 1, then digits A–F look like the letter U.

Dec.5 Determine whether the a, b, and c output circuits in the 74x49 seven-segment decoder correspond to minimal product-of-sums expressions for these segments, assuming that the nondecimal input combinations are "don't cares" and BI_L = 1.

Dec.6 Redesign the gate-level circuit of the MSI 74x49 seven-segment decoder so that the digits 6 and 9 have tails as shown in Figure xDec.2.