CHAPTER 12

Digital Electronics

12.1 The Basics of Digital Electronics

Until now I have mainly covered the analog realm of electronics—circuits that accept and respond to voltages that vary continuously over a given range. Such analog circuits included rectifiers, filters, amplifiers, simple *RC* timers, oscillators, simple transistor switches, etc. Although each of these analog circuits is fundamentally important in its own right, these circuits lack an important feature—they cannot store and process bits of information needed to make complex logical decisions. To incorporate logical decision-making processes into a circuit, you need to use digital electronics.

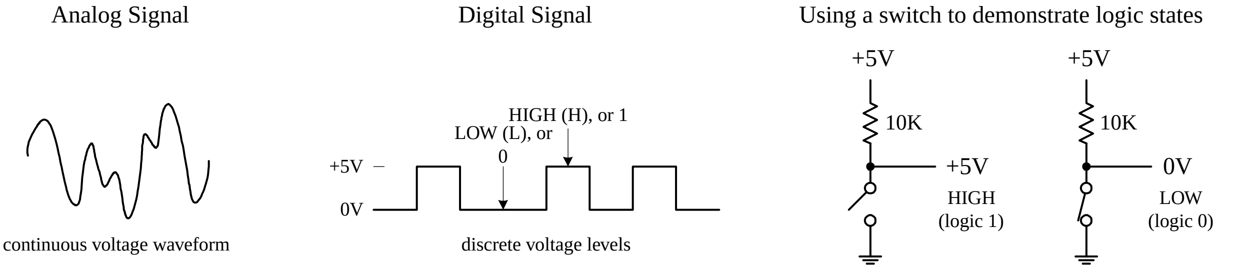


FIGURE 12.1

12.1.1 Digital Logic States

In digital electronics there are only two voltage states present at any point within a circuit. These voltage states are either *high* or *low*. The meaning of a voltage being high or low at a particular location within a circuit can signify a number of things. For example, it may represent the on or off state of a switch or saturated transistor. It may represent one bit of a number, or whether an event has occurred, or whether some action should be taken.

The high and low states can be represented as true and false statements, which are used in Boolean logic. In most cases, high = true and low = false. However, this does not have to be the case—you could make high = false and low = true. The decision to use one convention over the other is a matter left ultimately to the designer. In digital lingo, to avoid people getting confused over which convention is in use, the term *positive true logic* is used when high = true, while the term *negative true logic* is used when high = false.

In Boolean logic, the symbols 1 and 0 are used to represent true and false, respectively. Now, unfortunately, 1 and 0 are also used in electronics to represent high and low voltage states, where high = 1 and low = 0. As you can see, things can get a bit confusing, especially if you are not sure which type of logic convention is being used, positive true or negative true logic. I will give some examples later on in this chapter that deal with this confusing issue.

The exact voltages assigned to a high or low voltage states depend on the specific logic IC that is used (as it turns out, digital components are entirely IC based). As a general rule of thumb, +5 V is considered high, while 0 V (ground) is considered low. However, as you will see in Section 12.4, this does not have to be the case. For example, some logic ICs may interrupt a voltage from +2.4 to +5 V as high and a voltage from +0.8 to 0 V as low. Other ICs may use an entirely different range. Again, I will discuss these details later.

12.1.2 Number Codes Used in Digital Electronics

Binary

Because digital circuits work with only two voltage states, it is logical to use the binary number system to keep track of information. A binary number is composed of two binary digits, 0 and 1, which are also called *bits* (e.g., 0 = low voltage, 1 = high voltage). By contrast, a decimal number such as 736 is represented by successive powers of 10:

$$736_{10} = 7 \times 10^2 + 3 \times 10^1 + 6 \times 10^0$$

Similarly, a binary number such as 11100 (28_{10}) can be expressed as successive powers of 2:

$$11100_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

The subscript tells what number system is in use (X_{10} = decimal number, X_2 = binary number). The highest-order bit (leftmost bit) is called the *most significant bit* (MSB), while the lowest-order bit (rightmost bit) is called the *least significant bit* (LSB). Methods used to convert from decimal to binary and vice versa are shown below.

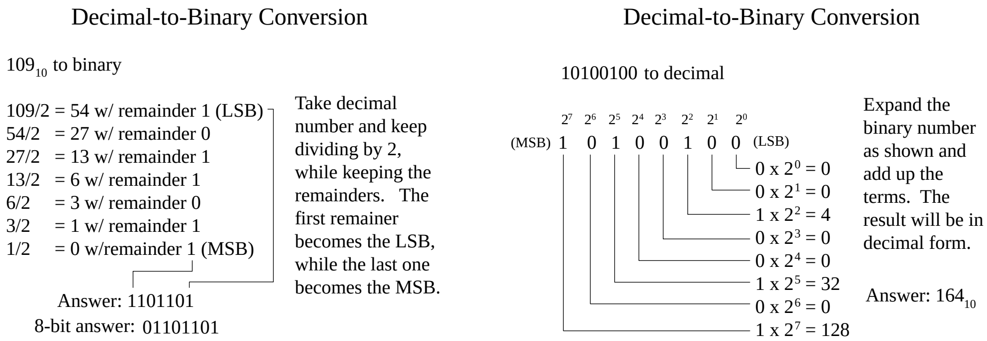


FIGURE 12.2

It should be noted that most digital systems deal with 4, 8, 16, 32, etc., bit strings. In the decimal-to-binary conversion example given here, you had a 7-bit answer. In an 8-bit system, you would have to add an additional 0 in front of the MSB (e.g., 01101101). In a 16-bit system, 9 additional 0s would have to be added (e.g., 0000000001101101).

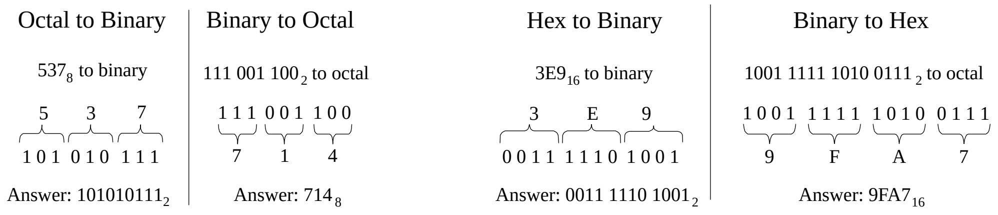
As a practical note, the easiest way to convert a number from one base to another is to use a calculator. For example, to convert a decimal number into a binary number, type in the decimal number (in base 10 mode) and then change to binary mode (which usually entails a 2d-function key). The number will now be in binary (1s and 0s). To convert a binary number to a decimal number, start out in binary mode, type in the number, and then switch to decimal mode.

Octal and Hexadecimal

Two other number systems used in digital electronics include the octal and hexadecimal systems. In the octal system (base 8), there are 8 allowable digits: 0, 1, 2, 3, 4, 5, 6, 7. In the hexadecimal system, there (base 16) there are 16 allowable digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Here are example octal and hexadecimal numbers with decimal equivalents:

247_8 (octal) $= 2 \times 8^2 + 4 \times 8^1 + 9 \times 8^0 = 167_{10}$ (decimal)
$2D5_{16}$ (hex) $= 2 \times 16^2 + D\ (=13_{10}) \times 16^1 + 9 \times 16^0 = 725_{10}$ (decimal)

Now, binary numbers are of course the natural choice for digital systems, but since these binary numbers can become long and difficult to interpret by our decimal-based brains (a result of our 10 fingers), it is common to write them out in hexadecimal or octal form. Unlike decimal numbers, octal and hexadecimal numbers can be translated easily to and from binary. This is so because a binary number, no matter how long, can be broken up into 3-bit groupings (for octal) or 4-bit groupings (for hexadecimal)—you simply add zero to the beginning of the binary number if the total numbers of bits is not divisible by 3 or 4. Figure 12.3 should paint the picture better than words.



A 3-digit binary number is replaced for each octal digit, and vise versa. The 3-digit terms are then grouped (or octal terms are grouped).

A 4-digit binary number is replaced for each hex digit, and vise versa. The 4-digit terms are then grouped (or hex terms are grouped).

FIGURE 12.3

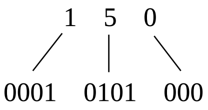
Today, the hexadecimal system has essentially replaced the octal system. The octal system was popular at one time, when microprocessor systems used 12-bit and 36-bit words, along with a 6-bit alphanumeric code—all these are divisible by 3-bit units (1 octal digit). Today, microprocessor systems mainly work with 8-bit, 16-bit, 20-bit, 32-bit, or 64-bit words—all these are divisible by 4-bit units (1 hex digit). In other words, an 8-bit word can be broken down into 2 hex digits, a 16-bit word into 4 hex digits, a 20-bit word into 5 hex digits, etc. Hexadecimal representation of binary numbers pops up in many memory and microprocessor applications that use programming codes (e.g., within assembly language) to address memory locations and initiate other specialized tasks that would otherwise require typing in long binary numbers. For example, a 20-bit address code used to identify 1 of 1 million memory locations can be replaced with a hexadecimal code (in the assembly program) that reduces the count to 5 hex digits. [Note that a compiler program later converts the hex numbers within the assembly language program into binary numbers (machine code) which the microprocessor can use.] Table 12.1 gives a conversion table.

TABLE 12.1 Decimal, Binary, Octal, Hex, BCD Conversion Table

DECIMAL	BINARY	OCTAL	HEXADECIMAL	BCD
00	0000 0000	00	00	0000 0000
01	0000 0001	01	01	0000 0001
02	0000 0010	02	02	0000 0010
03	0000 0011	03	03	0000 0011
04	0000 0100	04	04	0000 0100
05	0000 0101	05	05	0000 0101
06	0000 0110	06	06	0000 0110
07	0000 0111	07	07	0000 0111
08	0000 1000	10	08	0001 1000
09	0000 1001	11	09	0000 1001
10	0000 1010	12	0A	0001 0000
11	0000 1011	13	0B	0001 0001
12	0000 1100	14	0C	0001 0010
13	0000 1101	15	0D	0001 0011
14	0000 1110	16	0E	0001 0100
15	0000 1111	17	0F	0001 0101
16	0001 0000	20	10	0001 0110
17	0001 0001	21	11	0001 0111
18	0001 0010	22	12	0001 1000
19	0001 0011	23	13	0001 1001
20	0001 0100	24	14	0010 0000

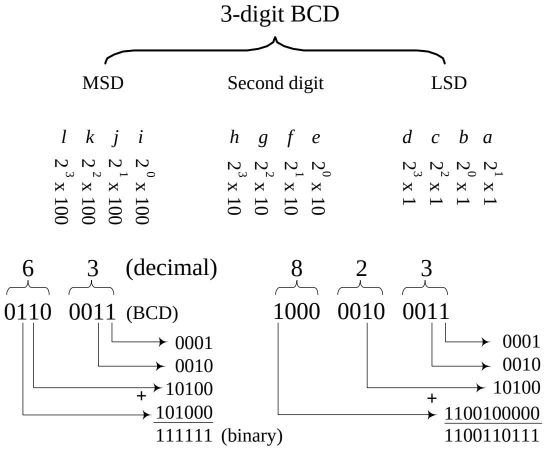
BCD Code

Binary-coded decimal (BCD) is used to represent each digit of a decimal number as a 4-bit binary number. For example, the number 150_{10} in BCD is expressed as



$150_{10} = 0001\ 0101\ 000_{(BCD)}$

To convert from BCD to binary is vastly more difficult, as shown in Fig. 12.4. Of course, you could cheat by converting the BCD into decimal first and then convert to binary, but that does not show you the mechanics of how machines do things with 1s and 0s. You will rarely have to do BCD-to-binary conversion, so I will not dwell on this topic—I will leave it to you to figure out how it works (see Fig. 12.4).



bit position	Weighting factor decimal	 binary
a	1	1
b	2	10
c	4	100
d	8	1000
e	10	1010
f	20	10100
g	40	101000
h	80	1010000
i	100	1100100
j	200	11001000
k	400	110010000
l	800	1100100000

FIGURE 12.4

BCD is commonly used when outputting to decimal (0–9) displays, such as those found in digital clocks and multimeters. BCD will be discussed later in Section 12.3.

Sign-Magnitude and 2's Complement Numbers

Up to now I have not considered negative binary numbers. How do you represent them? A simple method is to use *sign-magnitude representation*. In this method, you simply reserve a bit, usually the MSB, to act as a sign bit. If the sign bit is 0, the number is positive; if the sign bit is 1, the number is negative (see Fig. 12.5). Although the sign-magnitude representation is simple, it is seldom used because adding requires a different procedure than subtracting (as you will see in the next section). Occasionally, you will see sign-magnitude numbers used in display and analog-to-digital applications but hardly ever in circuits that perform arithmetic.

A more popular choice when dealing with negative numbers is to use *2's complement representation*. In 2's complement, the positive numbers are exactly the same as unsigned binary numbers. A negative number, however, is represented by a binary number, which when added to its corresponding positive equivalent results in zero. In this way, you can avoid two separate procedures for doing addition and subtraction. You will see how this works in the next section. A simple procedure outlining how to convert a decimal number into a binary number and then into a 2's complement number, and vice versa, is outlined in Fig. 12.5.

Arithmetic with Binary Numbers

Adding, subtracting, multiplying, and dividing binary numbers, hexadecimal numbers, etc., can be done with a calculator set to that particular base mode. But that's

Decimal, Sign-Magnitude, 2's Complement Conversion Table

DECIMAL	SIGN-MAGNITUDE	2'S COMPLEMENT
+7	0000 0111	0000 0111
+6	0000 0110	0000 0110
+5	0000 0101	0000 0101
+4	0000 0100	0000 0100
+3	0000 0011	0000 0011
+2	0000 0010	0000 0010
+1	0000 0001	0000 0001
0	0000 0000	0000 0000
–1	1000 0001	1111 1111
–2	1000 0010	1111 1110
–3	1000 0011	1111 1101
–4	1000 0100	1111 1100
–5	1000 0101	1111 1011
–6	1000 0110	1111 1010
–7	1000 0111	1111 1001
–8	1000 1000	1111 1000

Decimal to 2's complement

+ 4_{10} to 2's complement

```
true binary  = 0010 1001
2's comp     = 0010 1001
```

If the decimal number is positive, the 2's complement number is equal to the true binary equivalent of the decimal number.

-41_{10} to 2's complement

```
true binary  = 0010 1001
1's comp     = 1101 0110
Add 1        =        +1
2's comp     =  0010 1001
```

If the decimal number is negative, the 2's complement number is found by:
1) Complementing each bit of the true binary equivalent of the decimal (making 1's into 0's and vise versa). This is is called taking the 1's complement.
2) Adding 1 to the 1's complement number to get the magnitude bits. The sign bit will always end up being 1.

2's complement to decimal

1100 1101 (2's comp) to decimal

```
2's comp       =  1100 1101
Complement  =  0011 0010
Add 1          =          +1
True binary   =  0011 0011
Decimal  eq.  =      –51₁₀
```

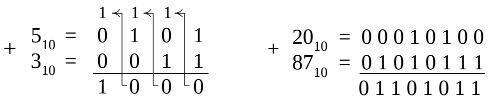
If the 2's complement number is positive (sign bit = 0), perform a regular binary-to-decimal conversion.

If the 2's complement number is negative (sign bit = 1), the decimal sign will be negative. The decimal is found by:
1) Complementing each bit of the 2's complement number.
2) Adding 1 to get the true binary equivalent.
3) Performing a true binary-to-decimal conversion.

FIGURE 12.5

cheating, and it doesn't help you understand the "mechanics" of how it is done. The mechanics become important when designing the actual arithmetical circuits. Here are the basic techniques used to add and subtract binary numbers.

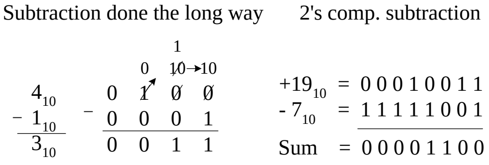
ADDING



Adding binary numbers is just like adding decimal numbers; whenever the result of adding one column of numbers is greater than one digit, a 1 is carried over to the next column to be added.

FIGURE 12.6

SUBTRACTION



Subtracting decimal numbers is not as easy as it looks. It is similar to decimal subtraction but can be confusing. For example, you might think that if you were to subtract a 1 from a 0, you would borrow a 1 from the column to the left. No! You must borrow a 10 (2_{10}). It becomes a headache if you try to do this by hand. The trick to subtracting binary numbers is to use the 2's complement representation that provides the sign bit and then just add the positive number with the negative number to get the sum. This method is often used by digital circuits because it allows both addition and subtraction, without the headache of having to subtract the smaller number from the larger number.

FIGURE 12.7

ASCII

ASCII (American Standard Code for Information Interchange) is an alphanumeric code used to transmit letters, symbols, numbers, and special nonprinting characters between computers and computer peripherals (e.g., printer, keyboard, etc.). ASCII consists of 128 different 7-bit codes. Codes from 000 0000 (or hex 00) to 001 1111 (or hex 1F) are reserved for nonprinting characters or special machine commands like ESC (escape), DEL (delete), CR (carriage return), LF (line feed), etc. Codes from 010 0000 (or hex 20) to 111 1111 (or hex 7F) are reserved for printing characters like a, A, #, &, {, @, 3, etc. See Tables 12.2 and 12.3. In practice, when ASCII code is sent, an additional bit is added to make it compatible with 8-bit systems. This bit may be set to 0 and ignored, it may be used as a parity bit for error detection (I will cover parity bits in Section 12.3), or it may act as a special function bit used to implement an additional set of specialized characters.

TABLE 12.2 ASCII Nonprinting Characters

DEC	HEX	7-BIT CODE	CONTROL CHAR	CHAR	MEANING	DEC	HEX	7-BIT	CONTROL CHAR	CHAR	MEANING
00	00	000 0000	ctrl-@	NUL	Null	16	10	001 0000	ctrl-P	DLE	Data line escape
01	01	000 0001	ctrl-A	SOH	Start of heading	17	11	001 0001	ctrl-Q	DC1	Device control 1
02	02	000 0010	ctrl-B	STX	Start of text	18	12	001 0010	ctrl-R	DC2	Device control 2
03	03	000 0011	ctrl-C	ETX	End of text	19	13	001 0011	ctrl-S	DC3	Device control 3
04	04	000 0100	ctrl-D	EOT	End of xmit	20	14	001 0100	ctrl-T	DC4	Device control 4
05	05	000 0101	ctrl-E	ENQ	Enquiry	21	15	001 0101	ctrl-U	NAK	Neg acknowledge
06	06	000 0110	ctrl-F	ACK	Acknowledge	22	16	001 0110	ctrl-V	SYN	Synchronous idle
07	07	000 0111	ctrl-G	BEL	Bell	23	17	001 0111	ctrl-W	ETB	End of xmit block
08	08	000 1000	ctrl-H	BS	Backspace	24	18	001 1000	ctrl-X	CAN	Cancel
09	09	000 1001	ctrl-I	HT	Horizontal tab	25	19	001 1001	ctrl-Y	EM	End of medium
10	0A	000 1010	ctrl-J	LF	Line feed	26	1A	001 1010	ctrl-Z	SUB	Substitute
11	0B	000 1011	ctrl-K	VT	Vertical tab	27	1B	001 1011	ctrl-[	ESC	Escape
12	0C	000 1100	ctrl-L	FF	Form feed	28	1C	001 1100	ctrl-\	FS	File separator
13	0D	000 1101	ctrl-M	CR	Carriage return	29	1D	001 1101	ctrl-]	GS	Group separator
14	0E	000 1110	ctrl-N	S0	Shift out	30	1E	001 1110	ctrl-^	RS	Record separator
15	0F	000 1111	ctrl-O	SI	Shift in	31	1F	001 1111	ctrl-_	US	Unit separator

TABLE 12.3 ASCII Printing Characters

DEC	HEX	7-BIT CODE	CHAR	DEC	HEX	7-BIT	CHAR	DEC	HEX	7-BIT CODE	CHAR
32	20	010 0000	SP	64	40	100 0000	@	96	60	110 0000	’
33	21	010 0001	!	65	41	100 0001	A	97	61	110 0001	a
34	22	010 0010	“	66	42	100 0010	B	98	62	110 0010	b
35	23	010 0011	#	67	43	100 0011	C	99	63	110 0011	c
36	24	010 0100	$	68	44	100 0100	D	100	64	110 0100	d
37	25	010 0101	%	69	45	100 0101	E	101	65	110 0101	e
38	26	010 0110	&	70	46	100 0110	F	102	66	110 0110	f
39	27	010 0111	‘	71	47	100 0111	G	103	67	110 0111	g
40	28	010 1000	(	72	48	100 1000	H	104	68	110 1000	h
41	29	010 1001	)	73	49	100 1001	I	105	69	110 1001	i
42	2A	010 1010	*	74	4A	100 1010	J	106	6A	110 1010	j
43	2B	010 1011	+	75	4B	100 1011	K	107	6B	110 1011	k
44	2C	010 1100	,	76	4C	100 1100	L	108	6C	110 1100	l
45	2D	010 1101	-	77	4D	100 1101	M	109	6D	110 1101	m
46	2E	010 1110	.	78	4E	100 1110	N	110	6E	110 1110	n
47	2F	010 1111	/	79	4F	100 1111	O	111	6F	110 1111	o
48	30	011 0000	0	80	50	101 0000	P	112	70	111 0000	p
49	31	011 0001	1	81	51	101 0001	Q	113	71	111 0001	q
50	32	011 0010	2	82	52	101 0010	R	114	72	111 0010	r
51	33	011 0011	3	83	53	101 0011	S	115	73	111 0011	s
52	34	011 0100	4	84	54	101 0100	T	116	74	111 0100	t
53	35	011 0101	5	85	55	101 0101	U	117	75	111 0101	u
54	36	011 0110	6	86	56	101 0110	V	118	76	111 0110	v
55	37	011 0111	7	87	57	101 0111	W	119	77	111 0111	w
56	38	011 1000	8	88	58	101 1000	X	120	78	111 1000	x
57	39	011 1001	9	89	59	101 1001	Y	121	79	111 1001	y
58	3A	011 1010	:	90	5A	101 1010	Z	122	7A	111 1010	z
59	3B	011 1011	;	91	5B	101 1011	[	123	7B	111 1011	{
60	3C	011 1100	<	92	5C	101 1100	\	124	7C	111 1100	|
61	3D	011 1101	=	93	5D	101 1101	]	125	7D	111 1101	}
62	3E	011 1110	>	94	5E	101 1110	^	126	7E	111 1110	~
63	3F	011 1111	?	95	5F	101 1111	_	127	7F	111 1111	DEL

12.1.3 Clock Timing and Parallel versus Serial Transmission

Before moving on to the next section, let's take a brief look at three important items: clock timing, parallel transmission, and serial transmission.

Clock Timing

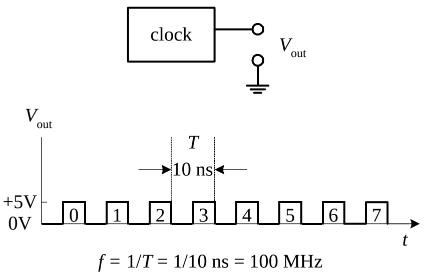
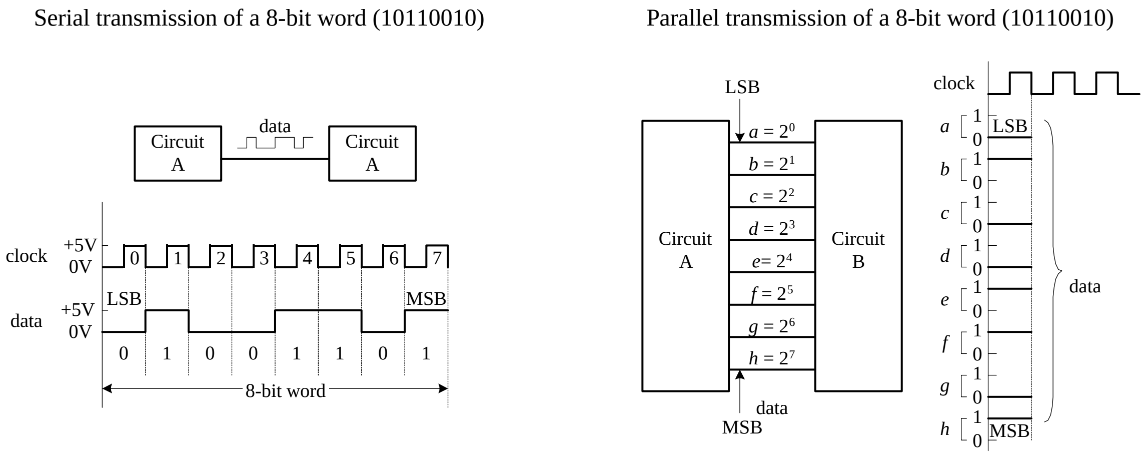


FIGURE 12.8

Digital circuits require precise timing to function properly. Usually, a clock circuit that generates a series of high and low pulses at a fixed frequency is used as a reference on which to base all critical actions executed within a system. The clock is also used to push bits of data through the digital circuitry. The period of a clock pulse is related to its frequency by $T = 1/f$. So, if T = 10 ns, then f = 1/(10 ns) = 100 MHz.

Serial versus Parallel Representation

Binary information can be transmitted from one location to another in either a serial or parallel manner. The serial format uses a single electrical conductor (and a common ground) for data transfer. Each bit from the binary number occupies a separate clock period, with the change from one bit to another occurring at each falling or leading clock edge—the type of edge depends on the circuitry used. Figure 12.9 shows an 8-bit (10110010) word that is transmitted from circuit A to circuit B in 8 clock pulses (0–7). In computer systems, serial communications are used to transfer data between keyboard and computer, as well as to transfer data between two computers via a telephone line.


Serial transmission of a 8-bit word (10110010)

Parallel transmission of a 8-bit word (10110010)

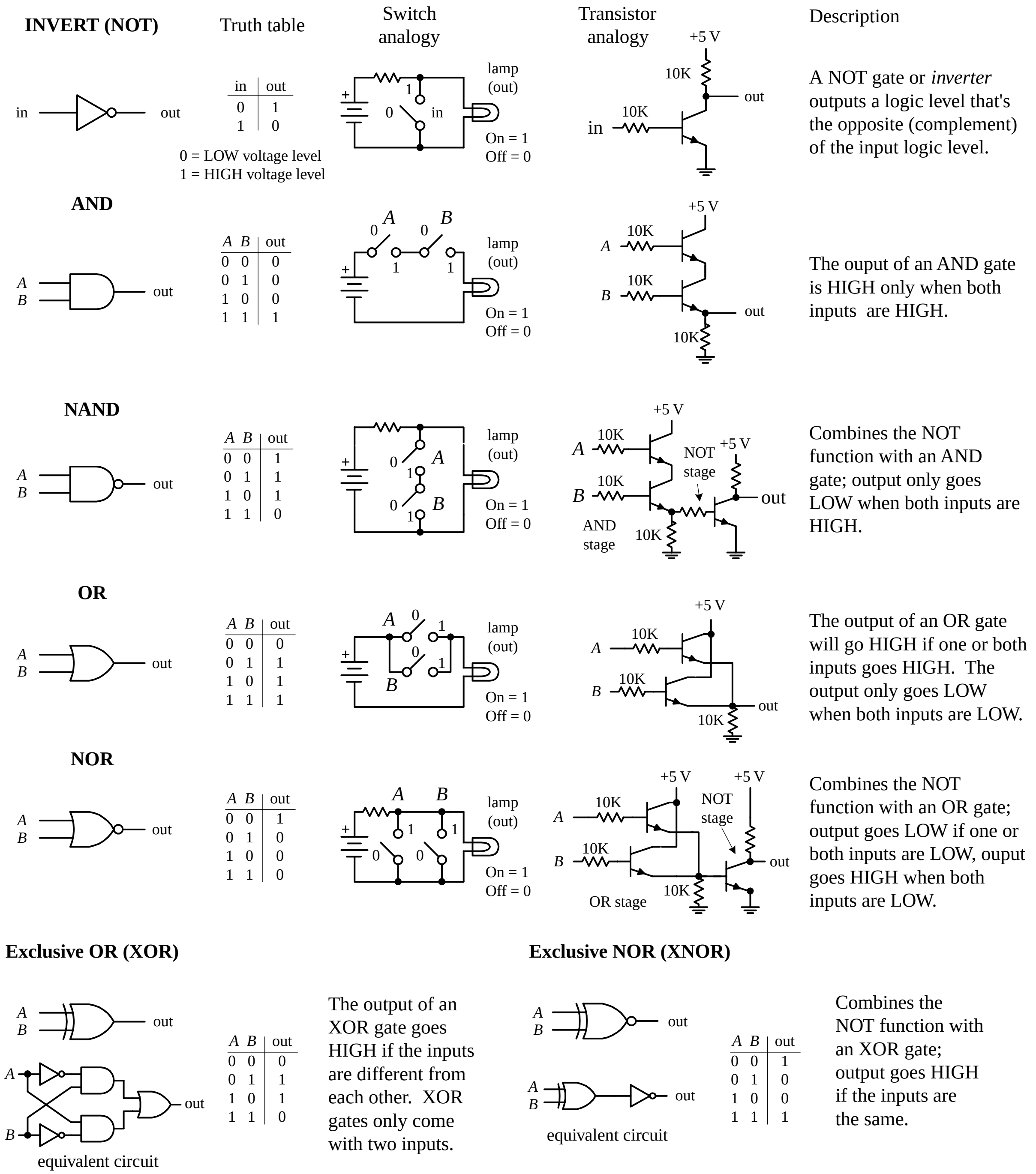
FIGURE 12.9

Parallel transmission uses separate electrical conductors for each bit (and a common ground). In Fig. 12.9, an 8-bit string (01110110) is sent from circuit A to circuit B. As you can see, unlike serial transmission, the entire word is transmitted in only one clock cycle, not 8 clock cycles. In other words, it is 8 times faster. Parallel communications are most frequently found within microprocessor systems that use multiline data and control buses to transmit data and control instructions from the microprocessor to other microprocessor-based devices (e.g., memory, output registers, etc.).

12.2 Logic Gates

Logic gates are the building blocks of digital electronics. The fundamental logic gates include the INVERT (NOT), AND, NAND, OR, NOR, exclusive OR (XOR), and exclusive NOR (XNOR) gates. Each of these gates performs a different logical operation. Figure 12.10 provides a description of what each logic gate does and gives a switch and transistor analogy for each gate.

FIGURE 12.10



INVERT (NOT)

in	out
0	1
1	0

0 = LOW voltage level
1 = HIGH voltage level

A NOT gate or *inverter* outputs a logic level that's the opposite (complement) of the input logic level.

AND

A	*B*	out
0	0	0
0	1	0
1	0	0
1	1	1

The ouput of an AND gate is HIGH only when both inputs are HIGH.

NAND

A	*B*	out
0	0	1
0	1	1
1	0	1
1	1	0

Combines the NOT function with an AND gate; output only goes LOW when both inputs are HIGH.

OR

A	*B*	out
0	0	0
0	1	1
1	0	1
1	1	1

The output of an OR gate will go HIGH if one or both inputs goes HIGH. The output only goes LOW when both inputs are LOW.

NOR

A	*B*	out
0	0	1
0	1	0
1	0	0
1	1	0

Combines the NOT function with an OR gate; output goes LOW if one or both inputs are LOW, ouput goes HIGH when both inputs are LOW.

Exclusive OR (XOR)

A	*B*	out
0	0	0
0	1	1
1	0	1
1	1	0

The output of an XOR gate goes HIGH if the inputs are different from each other. XOR gates only come with two inputs.

Exclusive NOR (XNOR)

A	*B*	out
0	0	1
0	1	0
1	0	0
1	1	1

Combines the NOT function with an XOR gate; output goes HIGH if the inputs are the same.

12.2.1 Multiple-Input Logic Gates

AND, NAND, OR, and NOR gates often come with more than two inputs (this is not the case with XOR and XNOR gates, which require two inputs only). Figure 12.11 shows a 4-input AND, an 8-input AND, a 3-input NOR, and an 8-input NOR gate. With the 8-input AND gate, all inputs must be high for the output to be high. With the 8-input OR gate, at least one of the inputs must be high for the output to go high.

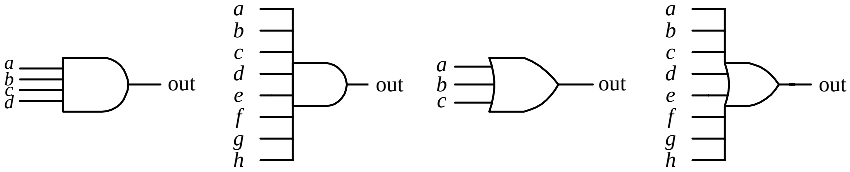


FIGURE 12.11

12.2.2 Digital Logic Gate ICs

The construction of digital gates is best left to the IC manufacturers. In fact, making gates from discrete components is highly impractical in regard to both overall performance (power consumption, speed, drive capacity, etc.) and overall cost and size.

There are a number of technologies used in the fabrication of digital logic. The two most popular technologies include TTL (transistor-transistor logic) and CMOS (complementary MOSFET) logic. TTL incorporates bipolar transistors into its design, while CMOS incorporates MOSFET transistors. Both technologies perform the same basic functions, but certain characteristics (e.g., power consumption, speed, output drive capacity, etc.) differ. There are many subfamilies within both TTL and CMOS. These subfamilies, as well as the various characteristics associated with each subfamily, will be discussed in greater detail in Section 12.4.

A logic IC, be it TTL or CMOS, typically houses more than one logic gate (e.g., quad 2-input NAND, hex inverter, etc.). Each of the gates within the IC shares a common supply voltage that is implemented via two supply pins, a positive supply pin ($+V_{CC}$ or $+V_{DD}$) and a ground pin (GND). The vast majority of TTL and CMOS ICs are designed to run off a +5-V supply. (This does not apply for all the logic families, but I will get to that later.)

Generally speaking, input and output voltage levels are assumed to be 0 V (low) and +5 V (high). However, the actual input voltage required and the actual output voltage provided by the gate are not set in stone. For example, the 74*xx* TTL series will recognize a high input from 2.0 to 5 V, a low from 0 to 0.8 V, and will guarantee a high output from 2.4 to 5 V and a low output from 0 to 0.4 V. However, for the CMOS 4000B series ($V_{CC} = +5$ V), recognizable input voltages range from 3.3 to 5 V for high, 0 to 1.7 V for low, while guaranteed high and low output levels range from 4.9 to 5 V and 0 to 0.1 V, respectively. Again, I will discuss specifics later in Section 12.4. For now, let’s just get acquainted with what some of these ICs look like—see Figs. 12.12 and 12.13. [CMOS devices listed in the figures include 74HC*xx*, 4000(B), while TTL devices shown include the 74*xx*, 74F*xx*, 74LS.]

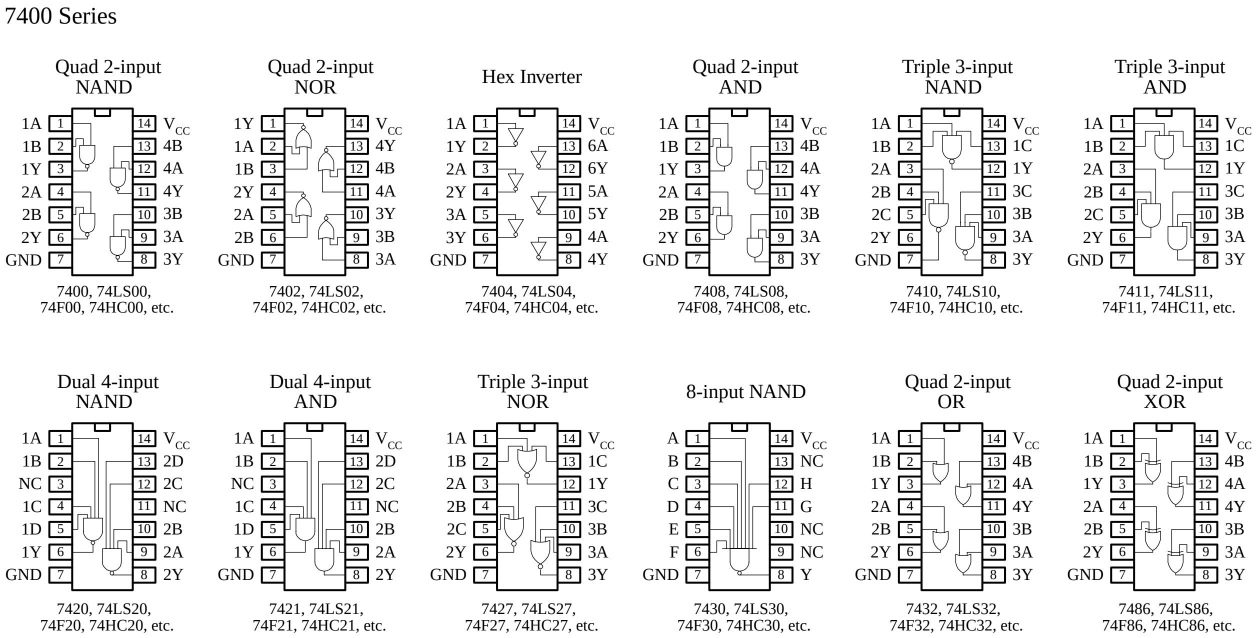

7400 Series
Quad 2-input NAND
7400, 74LS00, 74F00, 74HC00, etc.
Quad 2-input NOR
7402, 74LS02, 74F02, 74HC02, etc.
Hex Inverter
7404, 74LS04, 74F04, 74HC04, etc.
Quad 2-input AND
7408, 74LS08, 74F08, 74HC08, etc.
Triple 3-input NAND
7410, 74LS10, 74F10, 74HC10, etc.
Triple 3-input AND
7411, 74LS11, 74F11, 74HC11, etc.
Dual 4-input NAND
7420, 74LS20, 74F20, 74HC20, etc.
Dual 4-input AND
7421, 74LS21, 74F21, 74HC21, etc.
Triple 3-input NOR
7427, 74LS27, 74F27, 74HC27, etc.
8-input NAND
7430, 74LS30, 74F30, 74HC30, etc.
Quad 2-input OR
7432, 74LS32, 74F32, 74HC32, etc.
Quad 2-input XOR
7486, 74LS86, 74F86, 74HC86, etc.
1A 1B 1Y 2A 2B 2Y GND
V_{CC} 4B 4A 4Y 3B 3A 3Y

FIGURE 12.12

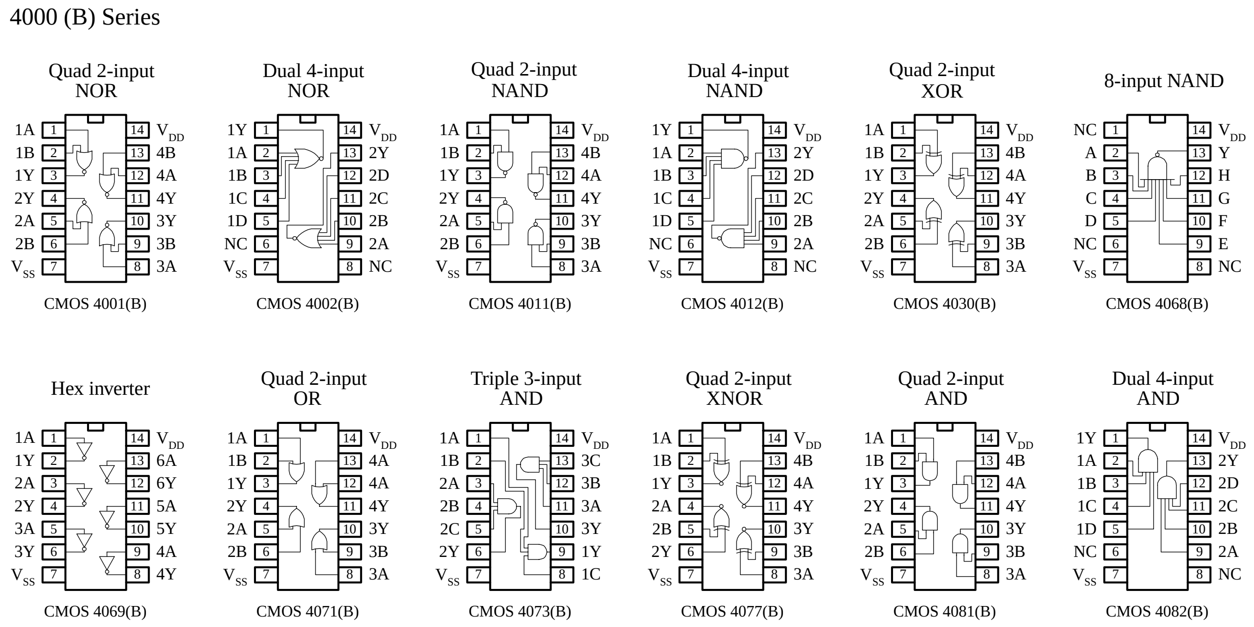


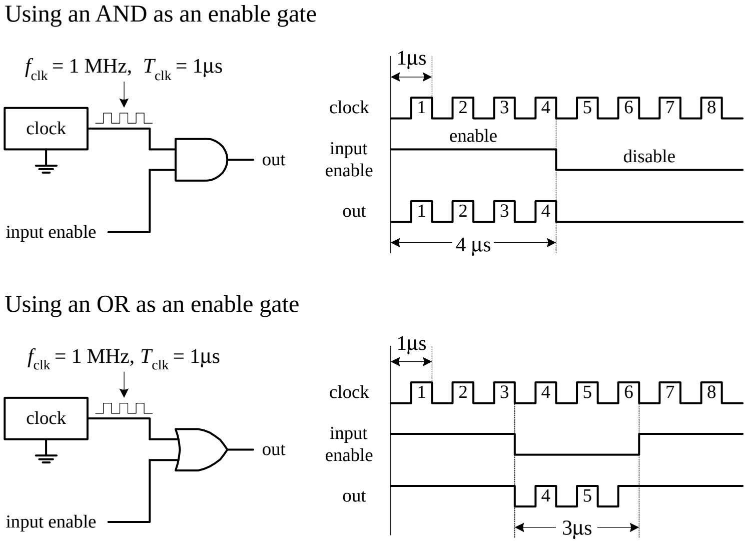
FIGURE 12.13

12.2.3 Applications for a Single Logic Gate

Before we jump into the heart of logic gate applications that involve combining logic gates to form complex decision-making circuits, let's take a look at a few simple applications that require the use of a single logic gate.

Enable/Disable Control

An enable / disable gate is a logic gate that acts to control the passage of a given waveform. The waveform, say, a clock signal, is applied to one of the gate's inputs, while the other input acts as the enable / disable control lead. Enable / disable gates are used frequently in digital systems to enable and disable control information from reaching various devices. Figure 12.14 shows two enable / disable circuits; the first uses an AND gate, and the second uses an OR gate. NAND and NOR gates are also frequently used as enable gates.


Using an AND as an enable gate

Using an OR as an enable gate

FIGURE 12.14

In the upper part of the figure, an AND gate acts as the enable gate. When the input enable lead is made high, the clock signal will pass to the output. In this example, the input enable is held high for 4 µs, allowing 4 clock pulses (where T_{clk} = 1 µs) to pass. When the input enable lead is low, the gate is disabled, and no clock pulses make it through to the output.

Below, an OR gate is used as the enable gate. The output is held high when the input enable lead is high, even as the clock signal is varying. However, when the enable input is low, the clock pulses are passed to the output.

Waveform Generation

By using the basic enable / disable function of a logic gate, as illustrated in the last example, it is possible, with the help of a repetitive waveform generator circuit, to create specialized waveforms that can be used for the digital control of sequencing circuits. An example waveform generator circuit is the Johnson counter, shown below. The Johnson counter will be discussed in Section 12.8—for now let's simply focus on the outputs. In the figure below, a Johnson counter uses clock pulses to generate different output waveforms, as shown in the timing diagram. Outputs A, B, C, and D go high for 4 µs (four clock periods) and are offset from each other by 1 µs. Outputs $\overline{A}$, $\overline{B}$, $\overline{C}$, and $\overline{D}$ produce waveforms that are complements of outputs A, B, C, and D, respectively.

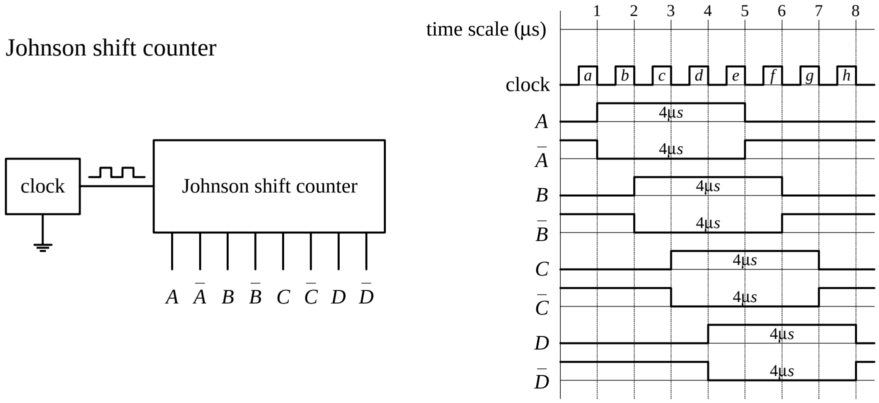
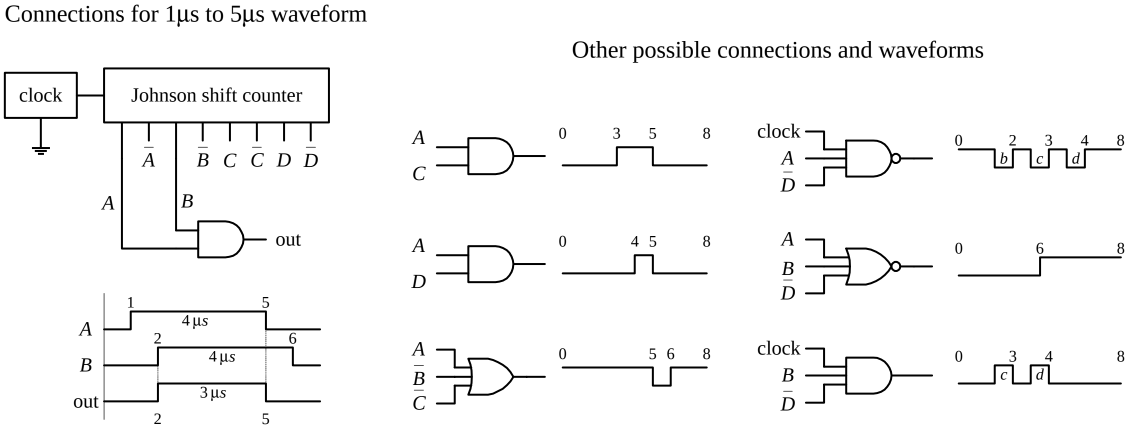

Johnson shift counter

FIGURE 12.15

Now, there may be certain applications that require 4-µs high / low pulses applied at a given time—as the counter provides. However, what would you do if the application required a 3-μs high waveform that begins at 2 μs and ends at 5 μs (relative to the time scale indicated in the figure above)? This is where the logic gates come in handy. For example, if you attach an AND gate's inputs to the counter's A and B outputs, you will get the desired 2- to 5-μs high waveform at the AND gate's output: from 1 to 2 μs the AND gate outputs a low ($A = 1, B = 0$), from 2 to 5 μs the AND gate outputs a high ($A = 1$, $B = 1$), and from 5 to 6 μs the AND gate outputs a low ($A = 0$, $B = 1$). See the leftmost figure below.


Connections for 1μs to 5μs waveform

Other possible connections and waveforms

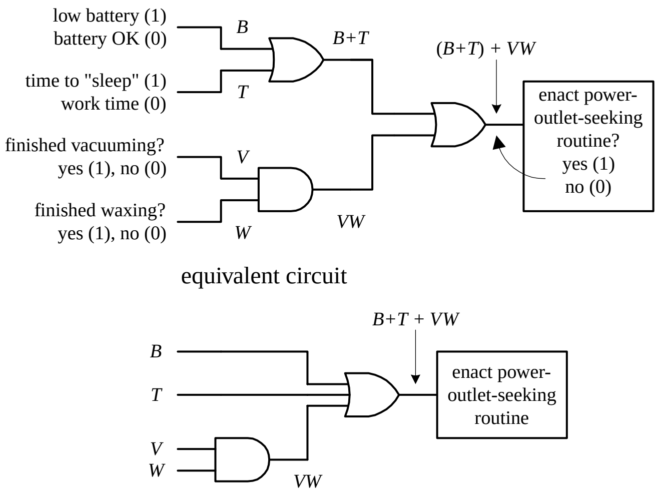
FIGURE 12.16

Various other specialized waveforms can be generated by using different logic gates and tapping different outputs of the Johnson shift counter. In the figure above and to the left, six other possibilities are shown.

12.2.4 Combinational Logic

Combinational logic involves combining logic gates together to form circuits capable of enacting more useful, complex functions. For example, let's design the logic used to instruct a janitor-type robot to recharge itself (seek out a power outlet) only when a specific set of conditions is met. The "recharge itself" condition is specified as follows: when its battery is low (indicated by a high output voltage from a battery-monitor circuit), when the workday is over (indicated by a high output voltage from a timer circuit), when vacuuming is complete (indicated by a high voltage output from a vacuum-completion monitor circuit), and when waxing is complete (indicated by a high output voltage from a wax-completion monitor circuit). Let's also assume that the power-outlet-seeking routine circuit is activated when a high is applied to its input.

Two simple combinational circuits that perform the desired logic function for the robot are shown in Fig. 12.17. The two circuits use a different number of gates but perform the same function. Now, the question remains, how did we come up with these circuits? In either circuit, it is not hard to predict what gates are needed. You simply exchange the word *and* present within the conditional statement with an AND gate within the logic circuit and exchange the word *or* present within the conditional statement with an OR gate within the logic circuit. Common sense takes care of the rest.



B	T	V	W	$B+T$	VW	$(B+T)+VW$
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	1	1	1	1

FIGURE 12.17

However, when you begin designing more complex circuits, using intuition to figure out what kind of logic gates to use and how to join them together becomes exceedingly difficult. To make designing combinational circuits easier, a special symbolic language called *Boolean algebra* is used, which only works with true and false variables. A Boolean expression for the robot circuit would appear as follows:

$$E = (B + T) + VW$$

This expression amounts to saying that if B (battery-check circuit's output) *or* T (timer circuit's output) is true *or* V *and* W (vacuum and waxing circuit outputs) are true, than E (enact power-outlet circuit input) is true. Note that the word *or* is replaced by the symbol + and the word *and* is simply expressed in a way similar to multiplying two variables together (placing them side-by-side or using a dot between variables). Also, it is important to note that the term *true* in Boolean algebra is expressed as a 1, while *false* is expressed as a 0. Here we are assuming positive logic, where true = high voltage. Using the Boolean expression for the robot circuit, we can come up with some of the following results (the truth table in Fig. 12.17 provides all possible results):

$E = (B + T) + VW$

$E = (1 + 1) + (1 \cdot 1) = 1 + 1 = 1$ (battery is low, time to sleep, finished with chores = go recharge).

$E = (1 + 0) + (0 \cdot 0) = 1 + 0 = 1$ (battery is low = go recharge).

$E = (0 + 0) + (1 \cdot 0) = 0 + 0 = 0$ (hasn't finished waxing = don't recharge yet).

$E = (0 + 0) + (1 \cdot 1) = 0 + 1 = 1$ (has finished all chores = go recharge).

$E = (0 + 0) + (0 \cdot 0) = 0 + 0 = 0$ (hasn't finished vacuuming and waxing = don't recharge yet).

The robot example showed you how to express AND and OR functions in Boolean algebraic terms. But what about the negation operations (NOT, NAND, NOR) and the exclusive operations (XOR, XNOR)? How do you express these in Boolean terms? To represent a NOT condition, you simply place a line over the NOT'ed variable or variables. For a NAND expression, you simply place a line over an AND expression. For a NOR expression, you simply place a line over an OR expression. For exclusive operations, you use the symbol $\oplus$. Here's a rundown of all the possible Boolean expressions for the various logic gates.

Boolean expressions for the logic gates

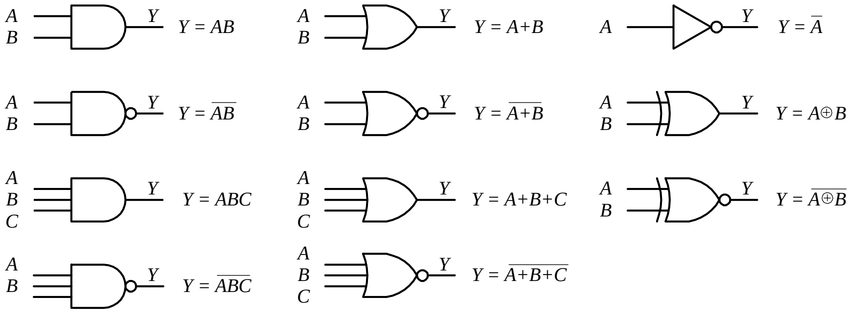


FIGURE 12.18

Like conventional algebra, Boolean algebra has a set of logic identities that can be used to simplify the Boolean expressions and thus make circuits more compact. These identities go by such names as the *commutative law of addition, associate law of addition, distributive law,* etc. Instead of worrying about what the various identities are called, simply make reference to the list of identities provided below and to the left. Most of these identities are self-explanatory, although a few are not so obvious, as you will see in a minute. The various circuits below and to the right show some of the identities in action.

LOGIC IDENTITIES

1) $A + B = B + A$

2) $AB = BA$

3) $A + (B + C) = (A + B) + C$

4) $A(BC) = (AB)C$

5) $A(B + C) = AB + AC$

6) $(A + B)(C + D) = AC + AD + BC + BD$

7) $\overline{1} = 0$

8) $\overline{0} = 1$

9) $A \cdot 0 = 0$

10) $A \cdot 1 = A$

11) $A + 0 = A$

12) $A + 1 = 1$

13) $A + A = A$

14) $AA = A$

15) $\overline{\overline{A}} = A$

16) $A + \overline{A} = 1$

17) $A\overline{A} = 0$

18) $\overline{A + B} = \overline{A}\,\overline{B}$

19) $\overline{AB} = \overline{A} + \overline{B}$

20) $A + \overline{A}B = A + B$

21) $\overline{A} + AB = \overline{A} + B$

22) $A \oplus B = \overline{A}B + A\overline{B} = (A + B)(\overline{AB})$

23) $\overline{A \oplus B} = AB + \overline{A}\,\overline{B}$

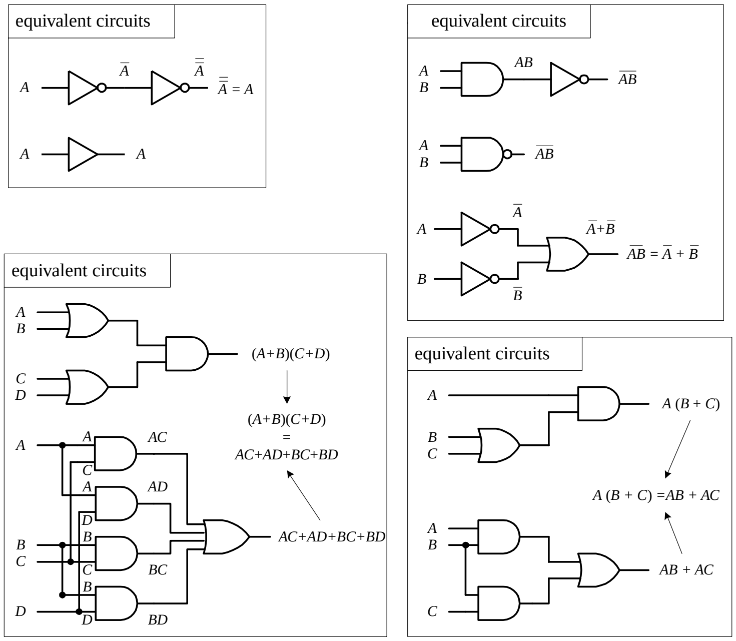


FIGURE 12.19

EXAMPLE

Let's find the initial Boolean expression for the circuit in Fig. 12.20 and then use the logic identities to come up with a circuit that requires fewer gates.

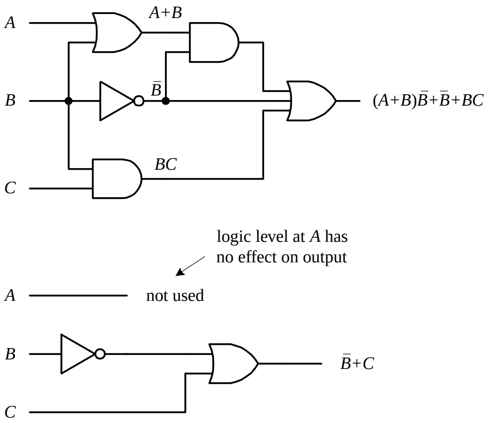


FIGURE 12.20

The circuit shown here is expressed by the following Boolean expression:

$out = (A + B)\overline{B} + \overline{B} + BC$

This expression can be simplified by using Identity 5:

$(A + B)\overline{B} = A\overline{B} + B\overline{B}$

This makes

$out = A\overline{B} + B\overline{B} + \overline{B} + BC$

Using Identities 17 ($B\overline{B} = 0$) and 11 ($B + 0 = B$), you get

$out = A\overline{B} + 0 + \overline{B} + BC = A\overline{B} + BC$

Factoring a $\overline{B}$ from the preceding term gives

$out = \overline{B}(A + 1) + BC$

Using Identities 12 ($A + 1 = 1$) and 10, you get

$out = B(1) + BC = \overline{B} + BC$

Finally, using Identity 21, you get the simplified expression

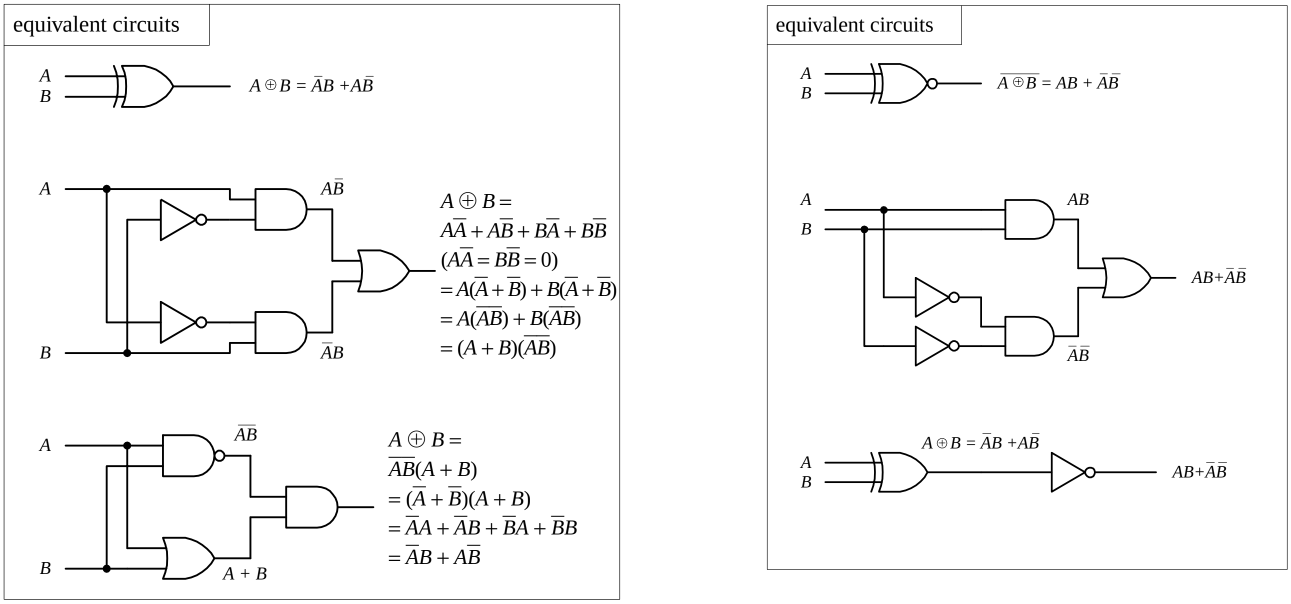
$out = \overline{B} + C$

Notice that A is now missing. This means that the logic input at A has no effect on the output and therefore can omitted. From the reduction, you get the simplified circuit in the bottom part of the figure.

Dealing with Exclusive Gates (Identities 22 and 23)

Now let's take a look at a couple of not so obvious logic identities I mentioned a second ago, namely, those which involve the XOR (Identity 22) and XNOR (Identity 23) gates. The leftmost section below shows equivalent circuits for the XOR gate. In the lower two equivalent circuits, Identity 22 is proved by Boolean reduction. Equivalent circuits for the XNOR gate are show in the rightmost section below. To prove Identity 23, you can simply invert Identity 22.

FIGURE 12.21

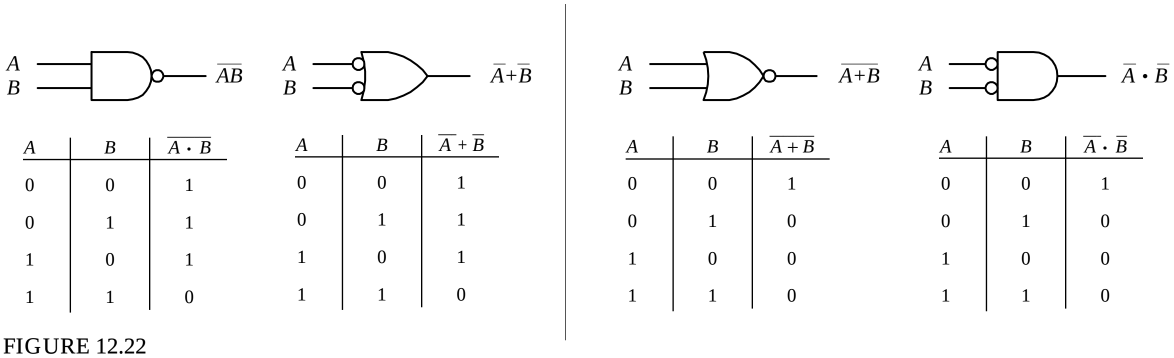

equivalent circuits
A
B
$A \oplus B = \overline{A}B + A\overline{B}$
A
$A\overline{B}$
$A \oplus B =$
$A\overline{A} + A\overline{B} + B\overline{A} + B\overline{B}$
$(A\overline{A} = B\overline{B} = 0)$
$= A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B})$
$= A(\overline{AB}) + B(\overline{AB})$
$= (A + B)(\overline{AB})$
B
$\overline{A}B$
A
$\overline{AB}$
$A \oplus B =$
$\overline{AB}(A + B)$
$= (\overline{A} + \overline{B})(A + B)$
$= \overline{A}A + \overline{A}B + \overline{B}A + \overline{B}B$
$= \overline{A}B + A\overline{B}$
B
A + B
equivalent circuits
A
B
$\overline{A \oplus B} = AB + \overline{A}\overline{B}$
A
B
AB
$AB+\overline{A}\overline{B}$
$\overline{A}\overline{B}$
$A \oplus B = \overline{A}B + A\overline{B}$
A
B
$AB+\overline{A}\overline{B}$

De Morgan's Theorem (Identities 18 and 19)

To simplify circuits containing NANDs and NORs, you can use an incredibly useful theorem known as *De Morgan's theorem*. This theorem allows you to convert an expression having an inversion bar over two or more variables into an expression having inversion bars over single variables only. De Morgan's theorem (Identities 18 and 19) is as follows:

$$\overline{A \cdot B} = \overline{A} + \overline{B} \quad \text{(2 variables)} \qquad \overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C} \quad \text{(3 or more variables)}$$
$$\overline{A + B} = \overline{A} \cdot \overline{B} \qquad\qquad\qquad\qquad \overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$$

The easiest way to prove that these identities are correct is to use the figure below, noting that the truth tables for the equivalent circuits are the same. Note the inversion bubbles present on the inputs of the corresponding leftmost gates. The inversion bubbles mean that before inputs *A* and *B* are applied to the base gate, they are inverted (negated). In other words, the bubbles are simplified expressions for NOT gates.


A
B
$\overline{AB}$
A
B
$\overline{A}+\overline{B}$
A
B
$\overline{A+B}$
A
B
$\overline{A} \cdot \overline{B}$

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

A	B	$\overline{A} + \overline{B}$
0	0	1
0	1	1
1	0	1
1	1	0

A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

A	B	$\overline{A} \cdot \overline{B}$
0	0	1
0	1	0
1	0	0
1	1	0

FIGURE 12.22

Why do you use the inverted-input OR gate symbol instead of a NAND gate symbol? Or why would you use the inverted-input AND gate symbol instead of a NOR gate symbol? This is a choice left up to the designer—whatever choice seems most logical to use. For example, when designing a circuit, it may be easier to think about ORing or ANDing inverted inputs than to think about NANDing or NORing inputs. Similarly, it may be easier to create truth tables or work with Boolean expressions using the inverted-input gate—it is typically easier to create truth tables and Boolean expressions that do not have variables joined under a common inversion bar. Of course, when it comes time to construct the actual working circuit, you probably will want to convert to the NAND and NOR gates because they do not require additional NOT gates at their inputs.

Bubble Pushing

A shortcut method for forming equivalent logic circuits, based on De Morgan's theorem, is to use what's called *bubble pushing.*

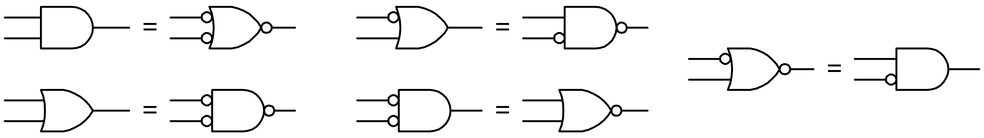


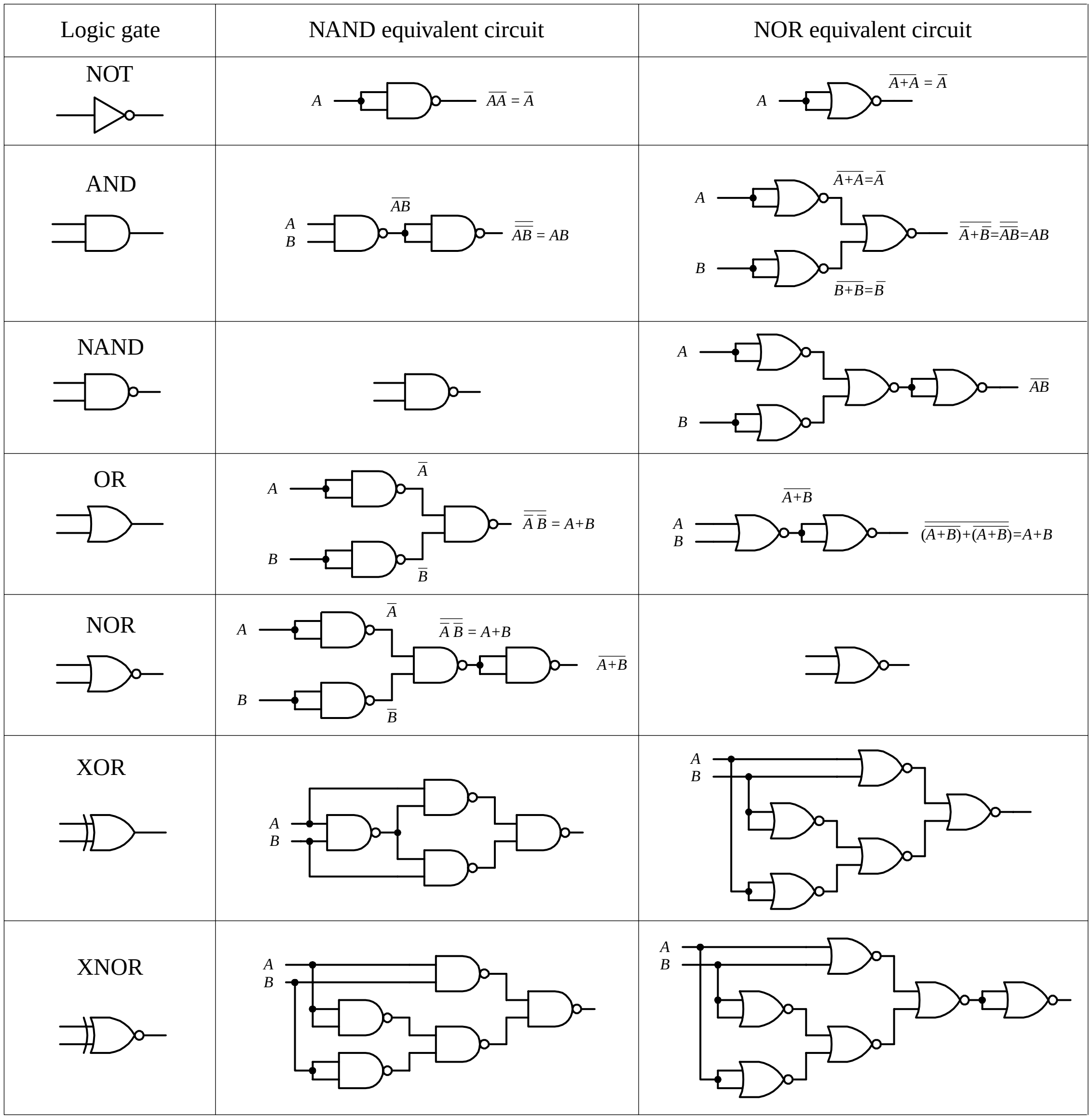
FIGURE 12.23

Bubble pushing involves the flowing tricks: First, change an AND gate to an OR gate or change an OR gate to an AND gate. Second, add inversion bubbles to the inputs and outputs where there were none, while removing the original bubbles. That's it. You can prove to yourself that this works by examining the corresponding truth tables for the original gate and the bubble-pushed gate, or you can work out the Boolean expressions using De Morgan's theorem. Figure 12.23 shows examples of bubble pushing.

Universal Capability of NAND and NOR Gates

NAND and NOR gates are referred to as *universal gates* because each alone can be combined together with itself to form all other possible logic gates. The ability to create any logic gate from NAND or NOR gates is obviously a handy feature. For example, if you do not have an XOR IC handy, you can use a single multigate NAND gate (e.g., 74HC00) instead. The figure below shows how to wire NAND or NOR gates together to create equivalent circuits of the various logic gates.

FIGURE 12.24


Logic gate
NAND equivalent circuit
NOR equivalent circuit
NOT
A
$\overline{AA} = \overline{A}$
A
$\overline{A+A} = \overline{A}$
AND
A
B
$\overline{AB}$
$\overline{\overline{AB}} = AB$
A
B
$\overline{A+A}=\overline{A}$
$\overline{B+B}=\overline{B}$
$\overline{\overline{A}+\overline{B}}=\overline{\overline{AB}}=AB$
NAND
A
B
$\overline{AB}$
OR
A
B
$\overline{A}$
$\overline{B}$
$\overline{\overline{A}\,\overline{B}} = A+B$
A
B
$\overline{A+B}$
$\overline{\overline{(A+B)}+\overline{(A+B)}}=A+B$
NOR
A
B
$\overline{A}$
$\overline{B}$
$\overline{\overline{A}\,\overline{B}} = A+B$
$\overline{A+B}$
XOR
A
B
A
B
XNOR
A
B
A
B

AND-OR-INVERT Gates (AOIs)

When a Boolean expression is reduced, the equation that is left over typically will be of one of the following two forms: *product-of-sums* (POS) or *sum-of-products* (SOP). A POS expression appears as two or more ORed variables ANDed together with two or more additional ORed variables. An SOP expression appears as two or more ANDed variables ORed together with additional ANDed variables. The figure below shows two circuits that provide the same logic function (they are equivalent), but the circuit to the left is designed to yield a POS expression, while the circuit to the right is designed to yield a SOP expression.

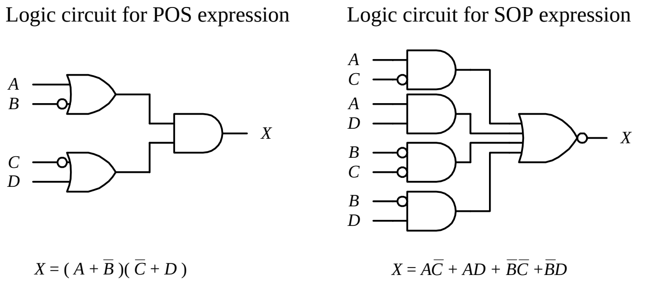

Logic circuit for POS expression
A
B
C
D
X
$X = (A + \overline{B})(\overline{C} + D)$
Logic circuit for SOP expression
A
C
A
D
B
C
B
D
X
$X = A\overline{C} + AD + \overline{B}\overline{C} + \overline{B}D$

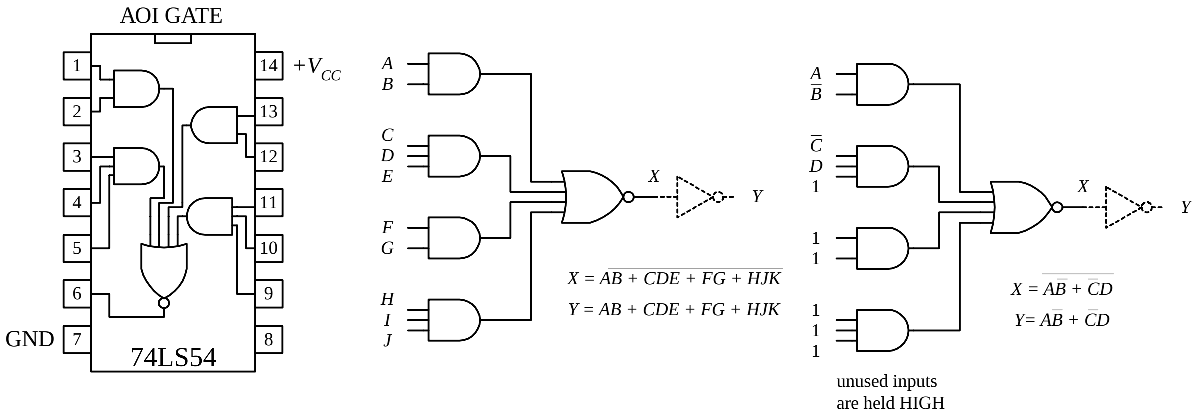
Table made using SOP expression
(it's easier than POS)

A	B	C	D	$A\overline{C}$	AD	$\overline{B}\,\overline{C}$	$\overline{B}D$	X
0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	1	1	1
0	0	1	0	0	0	0	0	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0
1	0	0	0	1	0	1	0	1
1	0	0	1	1	1	1	1	1
1	0	1	0	0	0	0	0	0
1	0	1	1	0	1	0	1	1
1	1	0	0	1	0	0	0	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	0	0	0
1	1	1	1	0	1	0	0	1

FIGURE 12.25

In terms of design, which circuit is best, the one that implements the POS expression or the one that implements SOP expression? The POS design shown here would appear to be the better choice because it requires fewer gates. However, the SOP design is nice because it is easy to work with the Boolean expression. For example, which Boolean expression above (POS or SOP) would you rather use to create a truth table? The SOP expression seems the obvious choice. A more down-to-earth reason for using an SOP design has to do with the fact that special ICs called AND-OR-INVERT (AOI) gates are designed to handle SOP expressions. For example, the 74LS54 AOI IC shown below creates an inverted SOP expression at its output, via two 2-input AND gates and two 3-input AND gates NORed together. A NOT gate can be attached to the output to get rid of the inversion bar, if desired. If specific inputs are not used, they should be held high, as shown in the example circuit below and to the far left. AOI ICs come in many different configurations—check out the catalogs to see what's available.

FIGURE 12.26


AOI GATE
1
2
3
4
5
6
7
GND
14
+V_{CC}
13
12
11
10
9
8
74LS54
A
B
C
D
E
F
G
H
I
J
X
Y
$X = \overline{AB + CDE + FG + HJK}$
$Y = AB + CDE + FG + HJK$
A
$\overline{B}$
$\overline{C}$
D
1
1
1
1
1
1
X
Y
$X = \overline{A\overline{B} + \overline{C}D}$
$Y = A\overline{B} + \overline{C}D$
unused inputs
are held HIGH

12.2.5 Keeping Circuits Simple (Karnaugh Maps)

We have just seen how using the logic identities can simplify a Boolean expression. This is important because it reduces the number of gates needed to construct the logic circuit. However, as I am sure you will agree, having to work out Boolean problems in longhand is not easy. It takes time and ingenuity. Now, a simple way to avoid the unpleasant task of using your ingenuity is to get a computer program that accepts a truth table or Boolean expression and then provides you with the simplest expression and perhaps even the circuit schematic. However, let's assume that you do not have such a program to help you out. Are you stuck with the Boolean longhand approach? No. What you do is use a technique referred to as *Karnaugh mapping.* With this technique, you take a given truth table (or Boolean expression that can be converted into a truth table), convert it into a Karnaugh map, apply some simple graphic rules, and come up with the simplest (most of the time) possible Boolean expression for your final circuit. Karnaugh mapping works best for circuits with three to four inputs—below this, things usually do not require much thought anyway; beyond four inputs, things get quite tricky. Here's a basic outline showing how to apply Karnaugh mapping to a three-input system:

1. First, select a desired truth table. Let's choose the one shown in Fig. 12.27. (If you only have a Boolean expression, transform it into an SOP expression and use the SOP expression to create the truth table—refer to Fig. 12.26 to figure out how this is done.)
2. Next, translate the truth table into a Karnaugh map. A Karnaugh map is similar to a truth table but has its variables represented along two axes. Translating the truth

FIGURE 12.27

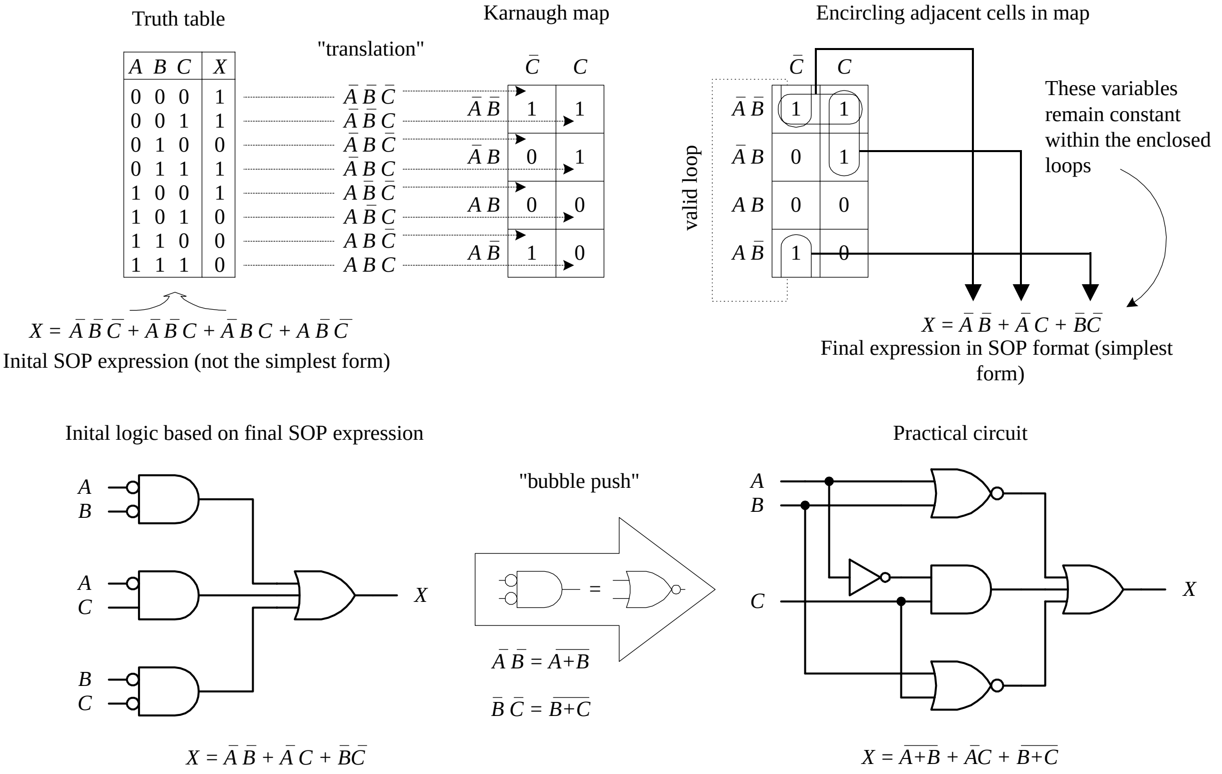

Truth table
"translation"
Karnaugh map
Encircling adjacent cells in map
A B C | X
0 0 0 | 1
0 0 1 | 1
0 1 0 | 0
0 1 1 | 1
1 0 0 | 1
1 0 1 | 0
1 1 0 | 0
1 1 1 | 0
$\bar{A}\bar{B}\bar{C}$
$\bar{A}\bar{B}C$
$\bar{A}B\bar{C}$
$\bar{A}BC$
$A\bar{B}\bar{C}$
$A\bar{B}C$
$AB\bar{C}$
ABC
$\bar{C}$ C
$\bar{A}\bar{B}$ 1 1
$\bar{A}B$ 0 1
AB 0 0
$A\bar{B}$ 1 0
valid loop
These variables remain constant within the enclosed loops
$X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C}$
Inital SOP expression (not the simplest form)
$X = \bar{A}\bar{B} + \bar{A}C + \bar{B}\bar{C}$
Final expression in SOP format (simplest form)
Inital logic based on final SOP expression
A
B
A
C
B
C
X
$X = \bar{A}\bar{B} + \bar{A}C + \bar{B}\bar{C}$
"bubble push"
=
$\bar{A}\bar{B} = \overline{A+B}$
$\bar{B}\bar{C} = \overline{B+C}$
Practical circuit
A
B
C
X
$X = \overline{A+B} + \bar{A}C + \overline{B+C}$

table into a Karnaugh map reduces the number of 1s and 0s needed to present the information. Figure 12.27 shows how the translation is carried out.

3. After you create the Karnaugh map, you proceed to encircle adjacent cells of 1s into groups of 2, 4, or 8. The more groups you can encircle, the simpler the final equation will be. In other words, take all possible loops.
4. Now, identify the variables that remain constant within each loop, and write out an SOP equation by ORing these variables together. Here, *constant* means that a variable and its inverse are not present together within the loop. For example, the top horizontal loop in Fig. 12.27 yields $\overline{A}\,\overline{B}$ (the first term in the SOP expression), since $\overline{A}$'s and $\overline{B}$'s inverses (A and B) are not present. However, the C variable is omitted from this term because C and $\overline{C}$ are both present.
5. The SOP expression you end up with is the simplest possible expression. With it you can create your logic circuit. You may have to apply some bubble pushing to make the final circuit practical, as shown in the figure below.

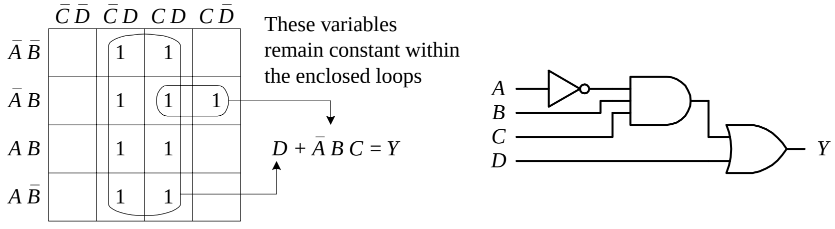
To apply Karnaugh mapping to four-input circuits, you apply the same basic steps used in the three-input scheme. However, now you use must use a 4 × 4 Karnaugh map to hold all the necessary information. Here is an example of how a four-input truth table (or unsimplified four-variable SOP expression) can be mapped and converted into a simplified SOP expression that can be used to create the final logic circuit:

inputs				out
A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Unsimplified SOP expression:

$$\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D+\overline{A}\cdot\overline{B}\cdot C\cdot D+\overline{A}\cdot B\cdot\overline{C}\cdot D+\overline{A}\cdot B\cdot C\cdot\overline{D}$$
$$+\overline{A}\cdot B\cdot C\cdot D+A\cdot\overline{B}\cdot\overline{C}\cdot D+A\cdot\overline{B}\cdot C\cdot D+A\cdot B\cdot\overline{C}\cdot D+A\cdot B\cdot C\cdot D=Y$$

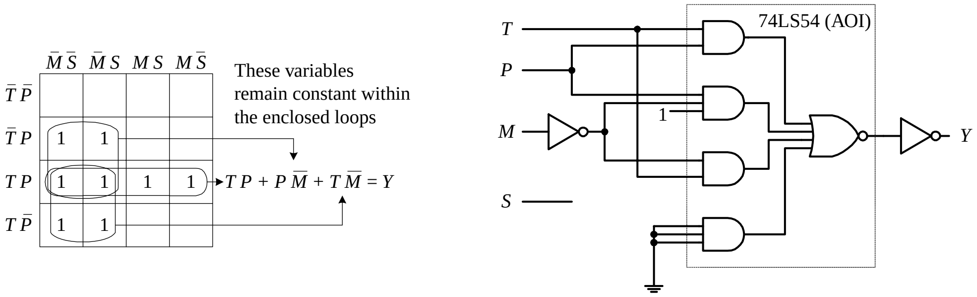
Simplified SOP expression and circuit



$\overline{C}\,\overline{D}$ $\overline{C}D$ CD $C\overline{D}$
$\overline{A}\,\overline{B}$
$\overline{A}B$
AB
$A\overline{B}$
1 1
1 1 1
1 1
1 1
These variables remain constant within the enclosed loops
$D+\overline{A}BC=Y$
A
B
C
D
Y

FIGURE 12.28

Here's an example that uses an AOI IC to implement the final SOP expression after mapping. I've thrown in variables other than the traditional *A*, *B*, *C*, and *D* just to let you know you are not limited to them alone. The choice of variables is up to you and usually depends on the application.



$\overline{M}\,\overline{S}$ $\overline{M}S$ MS $M\overline{S}$
$\overline{T}\,\overline{P}$
$\overline{T}P$
TP
$T\overline{P}$
1 1
1 1 1 1
1 1
These variables remain constant within the enclosed loops
$TP+P\overline{M}+T\overline{M}=Y$
74LS54 (AOI)
T
P
M
S
1
Y

FIGURE 12.29

Other Looping Configurations

Here are examples of other looping arrangements used with 4 × 4 Karnaugh maps:

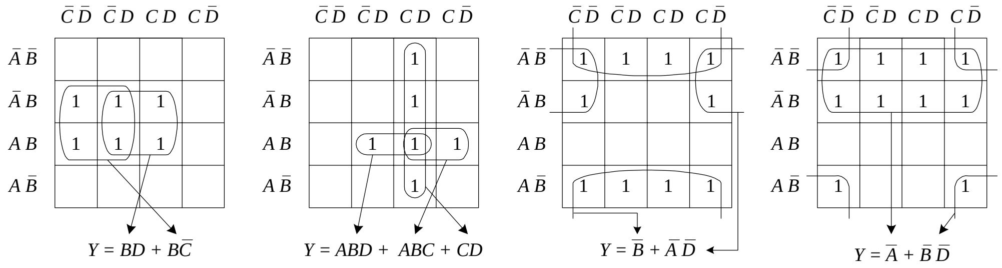

$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$
$\bar{A}B$
AB
$A\bar{B}$
$Y = BD + B\bar{C}$
$Y = ABD + ABC + CD$
$Y = \bar{B} + \bar{A}\bar{D}$
$Y = \bar{A} + \bar{B}\bar{D}$

FIGURE 12.30

12.3 Combinational Devices

Now that you know a little something about how to use logic gates to enact functions represented within truth tables and Boolean expressions, it is time to take a look at some common functions that are used in the real world of digital electronics. As you will see, these functions are usually carried out by an IC that contains all the necessary logic.

A word on IC part numbers before I begin. As with the logic gate ICs, the combinational ICs that follow will be of either the 4000 or 7400 series. It is important to note that an original TTL IC, like the 74138, is essentially the same device (same pinouts and function—usually, but not always) as its newer counterparts, the 74F138, 74HC128 (CMOS), 74LS138, etc. The practical difference resides in the overall performance of the device (speed, power dissipation, voltage level rating, etc.). I will get into these gory details in a bit.

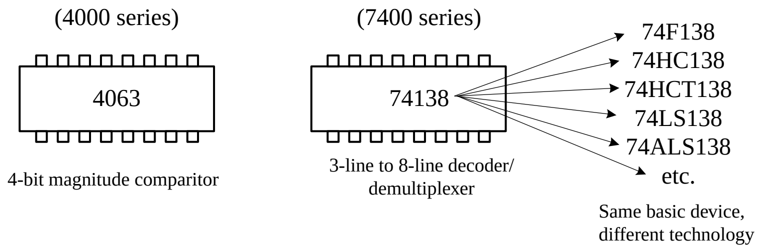

(4000 series)
4063
4-bit magnitude comparitor
(7400 series)
74138
3-line to 8-line decoder/ demultiplexer
74F138
74HC138
74HCT138
74LS138
74ALS138
etc.
Same basic device, different technology

FIGURE 12.31

12.3.1 Multiplexers (Data Selectors) and Bilateral Switches

Multiplexers or data selectors act as digitally controlled switches. (The term *data selector* appears to be the accepted term when the device is designed to act like an SPDT switch, while the term *multiplexer* is used when the throw count of the “switch” exceeds two, e.g., SP8T. I will stick with this convention, although others may not.) A simple 1-of-2 data selector built from logic gates is shown in Fig. 12.32. The data select input of this circuit acts to control which input (*A* or *B*) gets passed to the output: When data select is high, input *A* passes while *B* is blocked. When data select is low, input *B* is passed while *B* is blocked. To understand how this circuit works, think of the AND gates as enable gates.

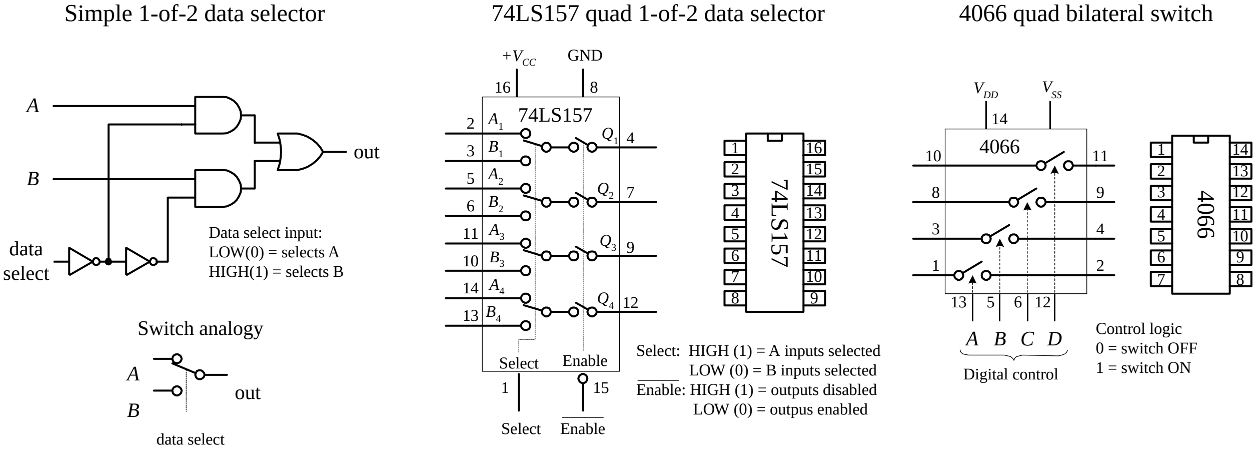
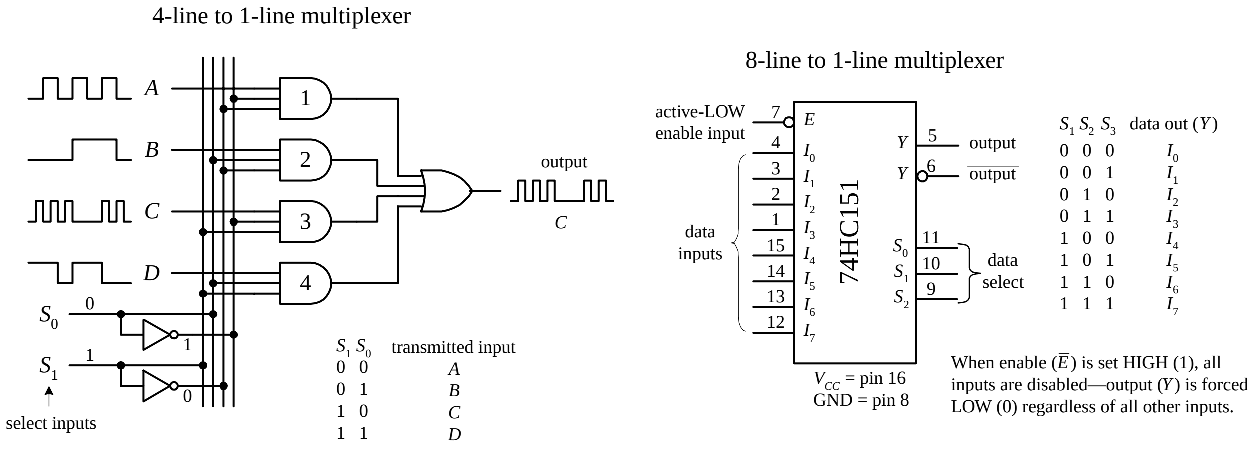


FIGURE 12.32

There are a number of different types of data selectors that come in IC form. For example, the 74LS157 quad 1-of-2 data selector IC, shown in Fig. 12.32, acts like an electrically controlled quad SPDT switch (or if you like, a 4PDT switch). When its select input is set high (1), inputs A_1, A_2, A_3, and A_4 are allowed to pass to outputs Q_1, Q_2, Q_3, and Q_4. When its select input is low (0), inputs B_1, B_2, B_3, and B_4 are allowed to pass to outputs Q_1, Q_2, Q_3, and Q_4. Either of these two conditions, however, ultimately depends on the state of the enable input. When the enable input is low, all data input signals are allowed to pass to the output; however, if the enable is high, the signals are not allowed to pass. This type of enable control is referred to as *active-low* enable, since the active function (passing the data to the output) occurs only with a low-level input voltage. The active-low input is denoted with a bubble (inversion bubble), while the outer label of the active-low input is represented with a line over it. Sometimes people omit the bubble and place a bar over the inner label. Both conventions are used commonly.

Figure 12.33 shows a 4-line-to-1-line multiplexer built with logic gates. This circuit resembles the 2-of-1 data selector shown in Fig. 12.32 but requires an additional select input to provide four address combinations.

FIGURE 12.33



S_1 S_0	transmitted input
0 0	A
0 1	B
1 0	C
1 1	D

S_1 S_2 S_3	data out (Y)
0 0 0	I_0
0 0 1	I_1
0 1 0	I_2
0 1 1	I_3
1 0 0	I_4
1 0 1	I_5
1 1 0	I_6
1 1 1	I_7

When enable ($\overline{E}$) is set HIGH (1), all inputs are disabled—output (Y) is forced LOW (0) regardless of all other inputs.

In terms of ICs, there are multiplexers of various input line capacities. For example, the 74151 8-line-to-1-line multiplexer uses three select inputs (S_0, S_1, S_2) to choose among 1 of 8 possible data inputs (I_0 to I_7) to be funneled to the output. Note that this device actually has two outputs, one true (pin 5) and one inverted (pin 6). The active-low enable forces the true output low when set high, regardless of the inputs.

To create a larger multiplexer, you combine two smaller multiplexers together. For example, Fig. 12.34 shows two 8-line-to-1-line 74HC151s combined to create a 16-line-to-1-line multiplexer. Another alternative is to use a 16-line-to-1-line multiplexer IC like the 74HC150 shown below. Check the catalogs to see what other kinds of multiplexers are available.

Combining two 8-line-to-1-line multiplexers to create a 16-line-to-1-line multiplexer

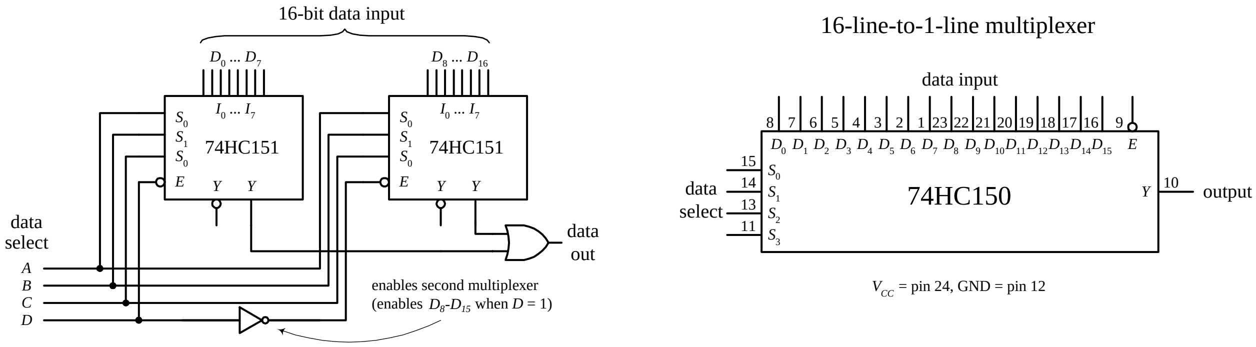


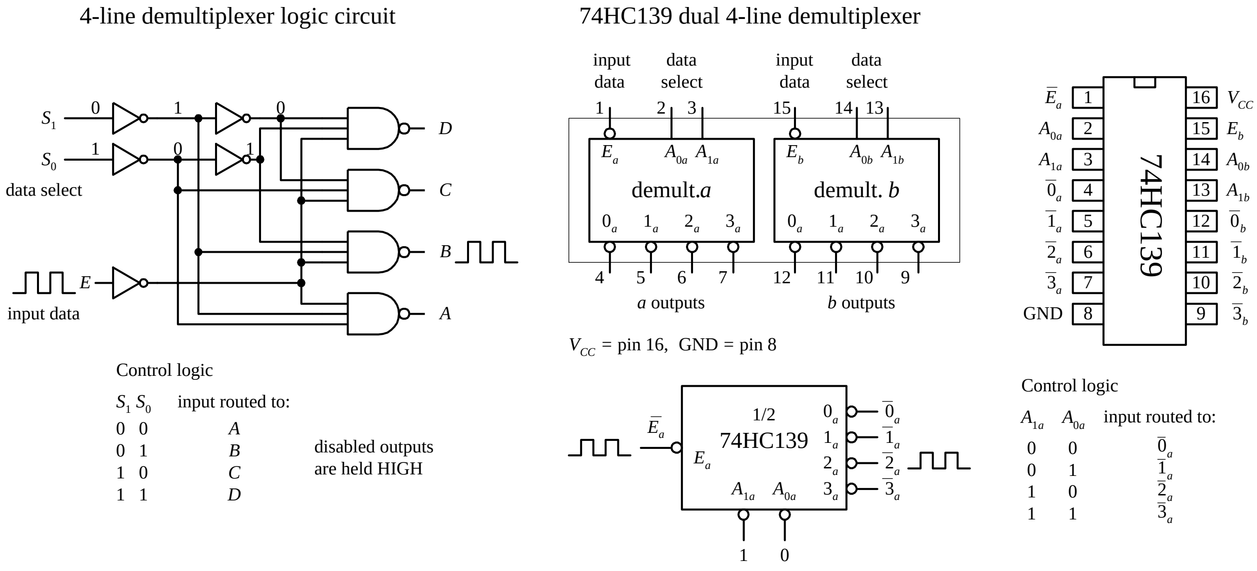
FIGURE 12.34

Finally, let's take a look at a very useful device called a *bilateral switch*. An example bilateral switch IC is the 4066, shown to the far left in Fig. 12.32. Unlike the multiplexer, this device merely acts as a digitally controlled quad SPST switch or quad transmission gate. Using a digital control input, you select which switches are on and which switches are off. To turn on a given switch, apply a high level to the corresponding switch select input; otherwise, keep the select input low.

Later in this chapter you come across analog switches and multiplexers. These devices use digital select inputs to control analog signals. Analog switches and multiplexers become important when you start linking the digital world to the analog world.

12.3.2 Demultiplexers (Data Disctributors) and Decoders

A demultiplexer (or data distributor) is the opposite of a multiplexer. It takes a single data input and routes it to one of several possible outputs. A simple four-line demultiplexer built from logic gates is shown in Fig. 12.35 *left*. To select the output (*A, B, C,* or *D*) to which you want to send the input signal (applied at *E*), you apply logic levels to the data select inputs (S_0, S_1), as shown in the truth table. Notice that the unselected outputs assume a high level, while the selected output varies with the input signal. An IC that contains two functionally separate four-line demultiplexers is the 74HC139, shown in Fig. 12.35 *right*. If you need more outputs, check out the 75*xx*154 16-line demultiplexer. This IC uses four data select inputs to choose from 1 of 16 possible outputs. Check out the catalogs to see what other demultiplexers exist.



Control logic

S_1 S_0	input routed to:
0 0	*A*
0 1	*B*
1 0	*C*
1 1	*D*

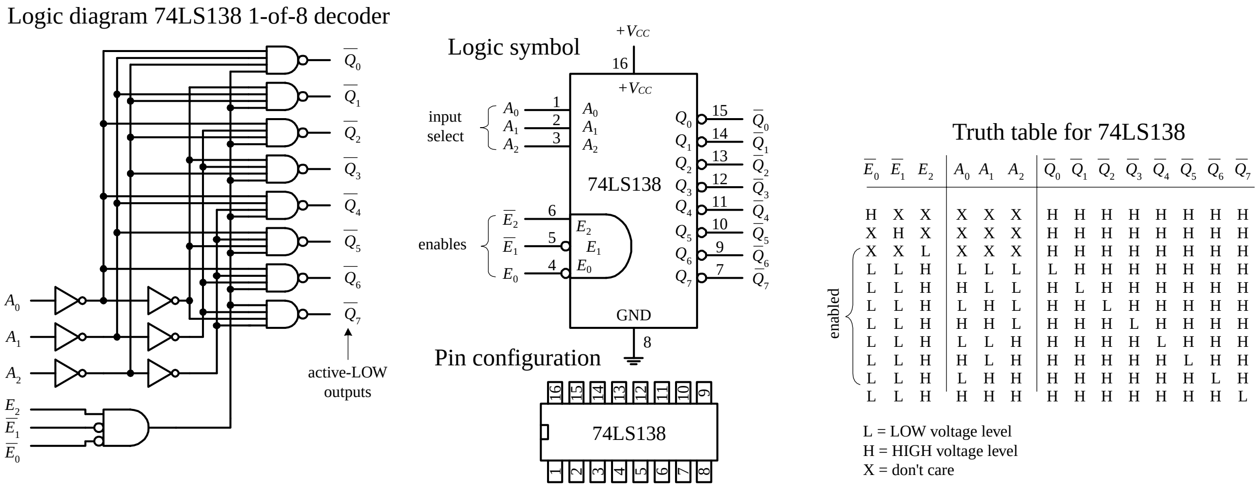
Control logic

A_{1a}	A_{0a}	input routed to:
0	0	$\overline{0}_a$
0	1	$\overline{1}_a$
1	0	$\overline{2}_a$
1	1	$\overline{3}_a$

FIGURE 12.35

A decoder is somewhat like a demultiplexer, but it does not route input data to a specific output via data select inputs. Instead, it simply uses the data select inputs to choose which output (or outputs) among many are to be made high or low. The number of address inputs, the number of outputs, and the active state of the selected output vary from decoder to decoder. The variance is of course based on what the decoder is designed to do.

For example, the 74LS138 1-of-8 decoder shown in Fig. 12.36 uses a 3-bit address input to select which of 8 outputs will be made low—all other outputs are held high. Like the demultiplexer in Fig. 12.35, this decoder has active-low outputs.



Truth table for 74LS138

$\overline{E}_0$	$\overline{E}_1$	E_2	A_0	A_1	A_2	$\overline{Q}_0$	$\overline{Q}_1$	$\overline{Q}_2$	$\overline{Q}_3$	$\overline{Q}_4$	$\overline{Q}_5$	$\overline{Q}_6$	$\overline{Q}_7$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

L = LOW voltage level
H = HIGH voltage level
X = don't care

FIGURE 12.36

Now what exactly does it mean to say an output is an active-low output? It simply means that when an active-low output is selected, it is forced to a low logic state; otherwise, it is held high. Active-high outputs behave in the opposite manner. An active-low output is usually indicated with a bubble, although often it is indicated with a bared variable within the IC logic symbol—no bubble included. Active-high outputs have no bubbles. Both active-low and active-high outputs are equally common among ICs. By placing a load (e.g., warning LED) between $+V_{CC}$ and an active-low output, you can sink current through the load and into the active-low output when the output is selected. By placing a load between an active-high output and ground, you can source current from the active-high output and sink it through the load when the output is selected. There are of course limits to how much current an IC can source or sink. I will discuss these limits in Section 12.4, and I will present various schemes used to drive analog loads in Section 12.10.

Now let's get back to the 74LS138 decoder and discuss the remaining enable inputs ($\overline{E}_0$, $\overline{E}_1$, E_2). For the 74LS138 to "decode," you must make the active-low inputs $\overline{E}_0$ and $\overline{E}_1$ low while making the active-high input E_2 high. If any other set of enable inputs is applied, the decoder is disabled, making all active-low outputs high regardless of the selected inputs.

Other common decoders include the 7442 BCD-to-DEC (decimal) decoder, the 74154 1-of-16 (Hex) decoder, and the 7447 BCD-to-seven-segment decoder shown below. Like the preceding decoder, these devices also have active-low outputs. The 7442 uses a binary-coded decimal input to select 1 of 10 (0 through 9) possible outputs. The 74154 uses a 4-bit binary input to address 1 of 16 (or 0 of 15) outputs, making that output low (all others high), provided the enables are both set low.

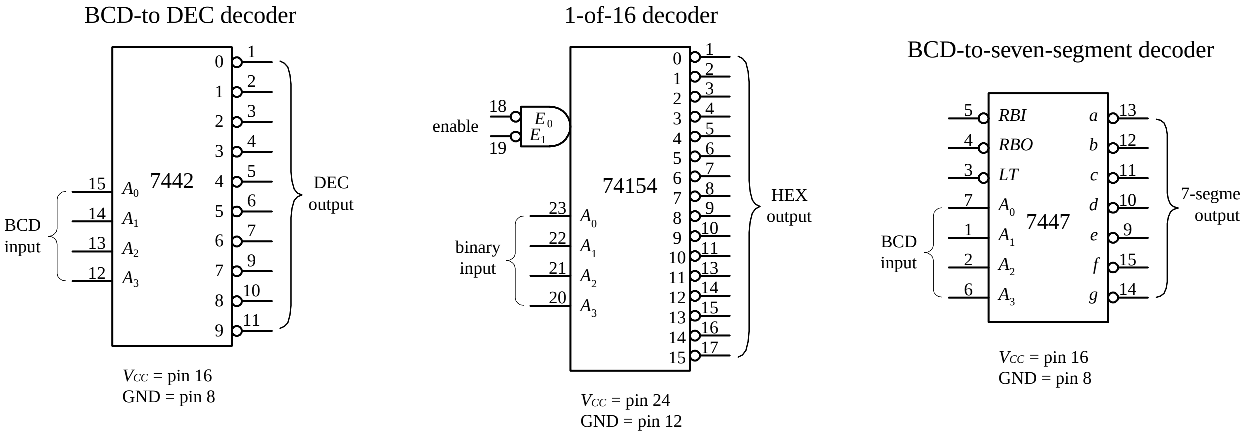
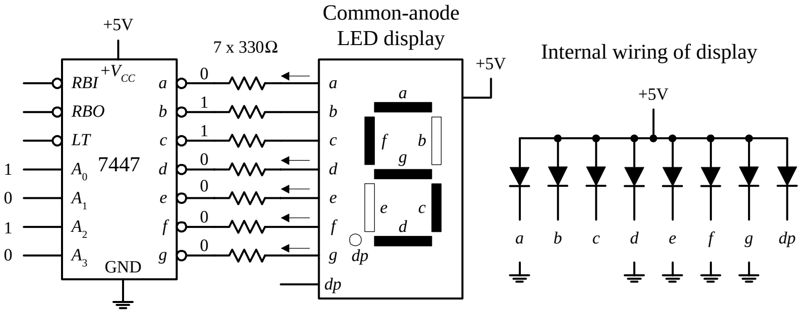


FIGURE 12.37

Now the 7447 is a bit different from the other decoders. With this device, more than one output can be driven low at a time. This is important because it allows the 7447 to drive a seven-segement LED display; to create different numbers requires driving more than one LED segment at a time. For example, in Fig. 12.38, when the BCD number for 5 (0101) is applied to the 7447's inputs, all outputs except $\overline{b}$ and $\overline{c}$ go low. This causes LED segments *a, d, e, f,* and *g* to light up—the 7447 sinks current through these LED segments, as indicated by the internal wiring of the display and the truth table.

7447 BCD-to-7-segment decoder/LED driver IC



Inputs				Digit	7447 outputs							Lit LED segment						
A_3	A_2	A_1	A_0		$\bar{a}$	$\bar{b}$	$\bar{c}$	$\bar{d}$	$\bar{e}$	$\bar{f}$	$\bar{g}$	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1	☀	☀	☀	☀	☀	☀	○
0	0	0	1	1	1	0	0	1	1	1	1	○	☀	☀	○	○	○	○
0	0	1	0	2	0	0	1	0	0	1	0	☀	☀	○	☀	☀	○	☀
0	0	1	1	3	0	0	0	0	1	1	0	☀	☀	☀	☀	○	○	☀
0	1	0	0	4	1	0	0	1	1	0	0	○	☀	☀	○	○	☀	☀
0	1	0	1	5	0	1	0	0	1	0	0	☀	○	☀	☀	○	☀	☀
0	1	1	0	6	0	1	0	0	0	0	0	☀	○	☀	☀	☀	☀	☀
0	1	1	1	7	0	0	0	1	1	1	1	☀	☀	☀	○	○	○	○
1	0	0	0	8	0	0	0	0	0	0	0	☀	☀	☀	☀	☀	☀	☀
1	0	0	1	9	0	0	0	1	1	0	0	☀	☀	☀	○	○	☀	☀

1 = HIGH voltage level, 0 = LOW voltage level,
☀ = LED segment ON, ○ = LED segment OFF

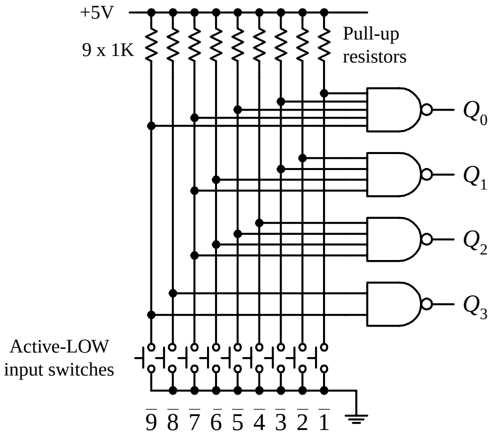
FIGURE 12.38

The 7447 also comes with a lamp test active-low input ($\overline{LT}$) that can be used to drive all LED segments at once to see if any of the segments are faulty. The ripple blanking input ($\overline{RBI}$) and ripple blanking output ($\overline{RBO}$) can be used in multistage display applications to suppress a leading-edge and/or trailing-edge zero in a multidigit decimal. For example, using the ripple blanking inputs and outputs, it is possible to take an 8-digit expression like 0056.020 and display 56.02, suppressing the two leading zeros and the one trailing zero. Leading-edge zero suppression is obtained by connecting the ripple blanking output of a decoder to the ripple blanking input of the next lower-stage device. The most significant decoder stage should have its ripple blanking input grounded. A similar procedure is used to provide automatic suppression of trailing zeros in the fractional part of the decimal.

12.3.3 Encoders and Code Converters

Encoders are the opposite of decoders. They are used to generate a coded output from a single active numeric input. To illustrate this in a simple manner, let's take a look at the simple decimal-to-BCD encoder circuit shown below.

Simple decimal-to-BCD encoder


+5V
9 x 1K
Pull-up
resistors
Q_0
Q_1
Q_2
Q_3
Active-LOW
input switches
$\overline{9}$ $\overline{8}$ $\overline{7}$ $\overline{6}$ $\overline{5}$ $\overline{4}$ $\overline{3}$ $\overline{2}$ $\overline{1}$

Truth table

$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{8}$	$\overline{9}$	Q_0	Q_1	Q_2	Q_3	BCD (pos. logic)
H	H	H	H	H	H	H	H	H	L	L	L	L	0000 (0_{10})
L	H	H	H	H	H	H	H	H	L	L	L	H	0001 (1_{10})
H	L	H	H	H	H	H	H	H	L	L	H	L	0010 (2_{10})
H	H	L	H	H	H	H	H	H	L	L	H	H	0011 (3_{10})
H	H	H	L	H	H	H	H	H	L	H	L	L	0100 (4_{10})
H	H	H	H	L	H	H	H	H	L	H	L	H	0101 (5_{10})
H	H	H	H	H	L	H	H	H	L	H	H	L	0110 (6_{10})
H	H	H	H	H	H	L	H	H	L	H	H	H	0111 (7_{10})
H	H	H	H	H	H	H	L	H	H	L	L	L	1000 (8_{10})
H	H	H	H	H	H	H	H	L	H	L	L	H	1001 (9_{10})

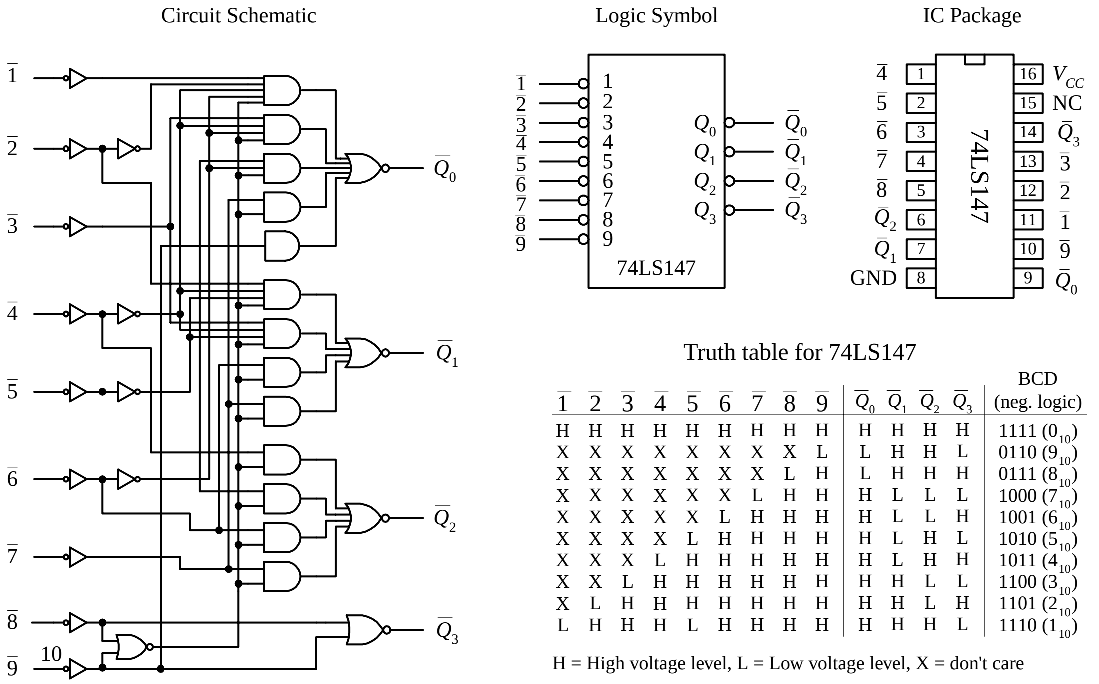
H = High voltage level, L = Low voltage level

FIGURE 12.39

In this circuit, normally all lines are held high by the pull-up resistors connected to +5 V. To generate a BCD output that is equivalent to a single selected decimal input, the switch corresponding to that decimal is closed. (The switch acts as an active-low input.) The truth table in Fig. 12.39 explains the rest.

Figure 12.40 shows a 74LS147 decimal-to-BCD (10-line-to-4-line) priority encoder IC. The 74LS147 provides the same basic function as the circuit shown in Fig. 12.39, but it has active-low outputs. This means that instead of getting an LLHH output when “3” is selected, as in the previous encoder, you get HHLL. The two outputs represent the same thing (“3”); one is expressed in positive true logic, and the other (the 74LS147) is expressed in negative true logic. If you do not like negative true logic, you can slap inverters on the outputs of the 74LS147 to get positive true logic. The choice to use positive or negative true logic really depends on what you are planning to drive. For example, negative true logic is useful when the device that you wish to drive uses active-low inputs.

74LS147 decimal-to-4-bit BCD Priority Encoder IC



Truth table for 74LS147

$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{8}$	$\overline{9}$	$\overline{Q}_0$	$\overline{Q}_1$	$\overline{Q}_2$	$\overline{Q}_3$	BCD (neg. logic)
H	H	H	H	H	H	H	H	H	H	H	H	H	1111 (0_{10})
X	X	X	X	X	X	X	X	L	L	H	H	L	0110 (9_{10})
X	X	X	X	X	X	X	L	H	L	H	H	H	0111 (8_{10})
X	X	X	X	X	X	L	H	H	H	L	L	L	1000 (7_{10})
X	X	X	X	X	L	H	H	H	H	L	L	H	1001 (6_{10})
X	X	X	X	L	H	H	H	H	H	L	H	L	1010 (5_{10})
X	X	X	L	H	H	H	H	H	H	L	H	H	1011 (4_{10})
X	X	L	H	H	H	H	H	H	H	H	L	L	1100 (3_{10})
X	L	H	H	H	H	H	H	H	H	H	L	H	1101 (2_{10})
L	H	H	H	L	H	H	H	H	H	H	H	L	1110 (1_{10})

H = High voltage level, L = Low voltage level, X = don't care

FIGURE 12.40

Another important difference between the two encoders is the term *priority* that is used with the 74LS147 and not used with the encoder in Fig. 12.39. The term *priority* is applied to the 74LS147 because this encoder is designed so that if two or more inputs are selected at the same time, it will only select the larger-order digit. For example, if 3, 5, and 8 are selected at the same time, only the 8 (negative true BCD LHHH or 0111) will be output. The truth table in Fig. 12.40 demonstrates this—look at the “don’t care” or “X” entries. With the nonpriority encoder, if two or more inputs are applied at the same time, the output will be unpredictable.

The circuit shown in Fig. 12.41 provides a simple illustration of how an encoder and a decoder can be used together to drive an LED display via a 0-to-9 keypad. The 74LS147 encodes a keypad’s input into BCD (negative logic). A set of inverters then converts the negative true BCD into positive true BCD. The transformed BCD is then fed into a 7447 seven-segment LED display decoder / driver IC.

A decimal-to-BCD encoder being used to convert keypad instructions into BCD instructions used to drive a LED display circuit (7447 decoder plus common-anode display)

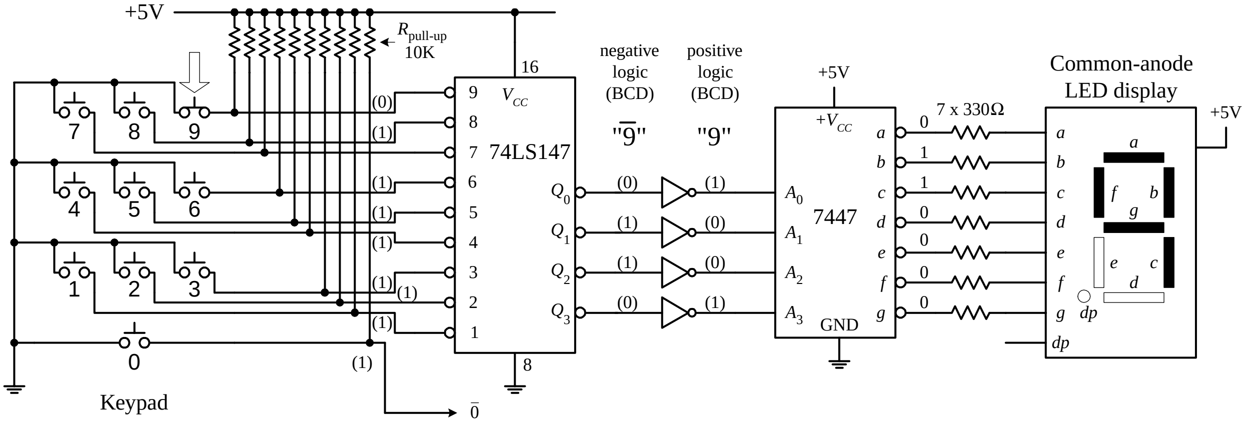
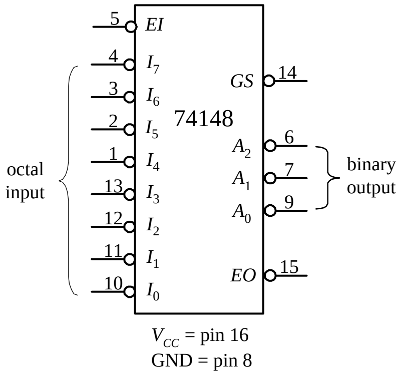


FIGURE 12.41

Figure 12.42 shows a 74148 octal-to-binary priory encoder IC. It is used to transform a specified single octal input into a binary 3-bit output code. As with the 74LS147, the 74148 comes with a priority feature, meaning, again, that if two or more inputs are selected at the same time, only the higher order number is selected.

74148 octal-to-binary priority encoder



74148 truth table

$\overline{EI}$	$\overline{I_0}$	$\overline{I_1}$	$\overline{I_2}$	$\overline{I_3}$	$\overline{I_4}$	$\overline{I_5}$	$\overline{I_6}$	$\overline{I_7}$	$\overline{GS}$	$\overline{A_0}$	$\overline{A_1}$	$\overline{A_2}$	$\overline{EO}$
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	H	L	L	H
L	X	X	X	X	X	L	H	H	L	L	H	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	L	L	L	H	H
L	X	X	L	H	H	H	H	H	L	H	L	H	H
L	X	L	H	H	H	H	H	H	L	L	H	H	H
L	L	H	H	H	H	H	H	H	L	H	H	H	H

FIGURE 12.42

A high applied to the input enable ($\overline{EI}$) forces all outputs to their inactive (high) state and allows new data to settle without producing erroneous information at the outputs. A group signal output ($\overline{GS}$) and an enable output ($\overline{EO}$) are also provided to allow for system expansion. The $\overline{GS}$ output is active level low when any input is low (active). The $\overline{EO}$ output is low (active) when all inputs are high. Using the output enable along with the input enable allows priority coding of N input signals. Both $\overline{EO}$ and $\overline{GS}$ are active high when the input enable is high (device disabled).

Figure 12.43 shows a 74184 BCD-to-binary converter (encoder) IC. This device has eight active-high outputs (Y_1–Y_8). Outputs Y_1 to Y_5 are outputs for regular BCD-to-binary conversion, while outputs Y_6 to Y_8 are used for a special BDC code called *nine's complement* and *ten's complement.* The active-high BCD code is applied to inputs A through E. The $\overline{G}$ input is an active-low enable input.

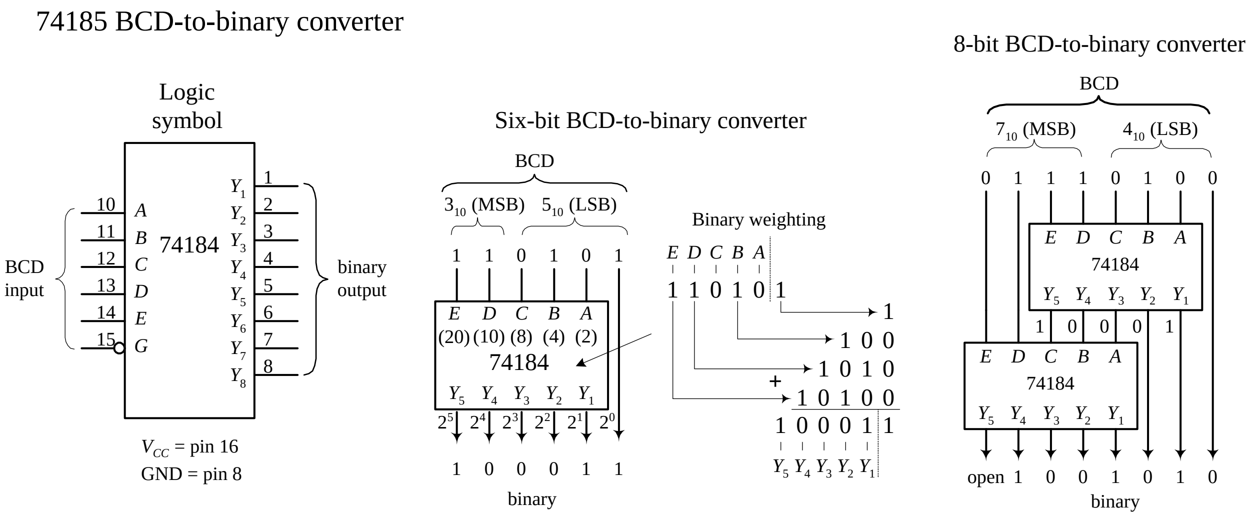


FIGURE 12.43

A sample 6-bit BCD-to-binary converter and a sample 8-bit BCD-to-binary converter that use the 74184 are shown to the right in Fig. 12.43. In the 6-bit circuit, since the LSB of the BCD input is always equal to the LSB of the binary output, the connection is made straight from input to output. The other BCD bits are applied directly to inputs *A* through *E*. The binary weighing factors for each input are *A* = 2, *B* = 4, *C* = 8, *D* = 10, and *E* = 20. Because only 2 bits are available for the MSD BCD input, the largest BCD digit in that position is 3 (binary 11). To get a complete 8-bit BCD converter, you connect two 74184s together, as shown to the far right in Fig. 12.43.

Figure 12.44 shows a 74185 binary-to-BCD converter (encoder). It is essentially the same as the 74184 but in reverse. The figure shows 6-bit and 8-bit binary-to-BCD converter arrangements.

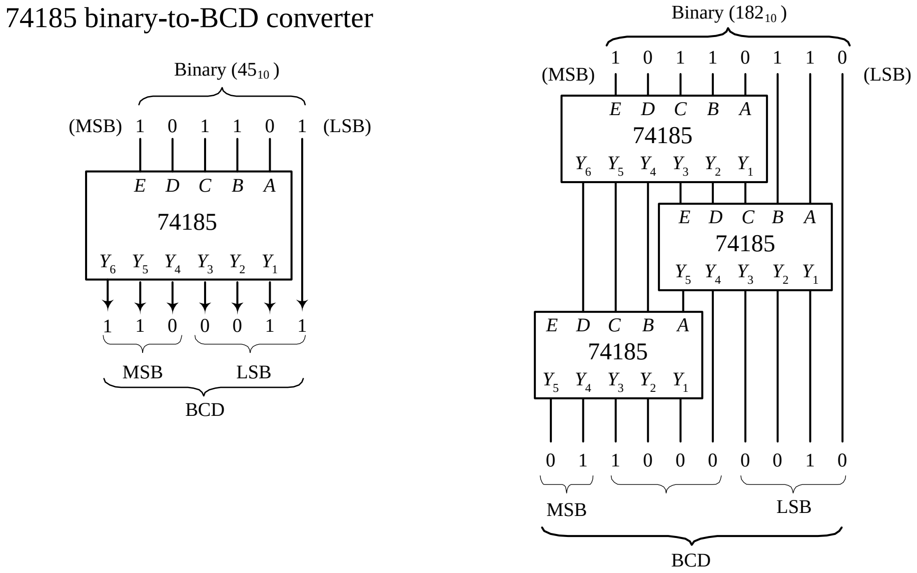
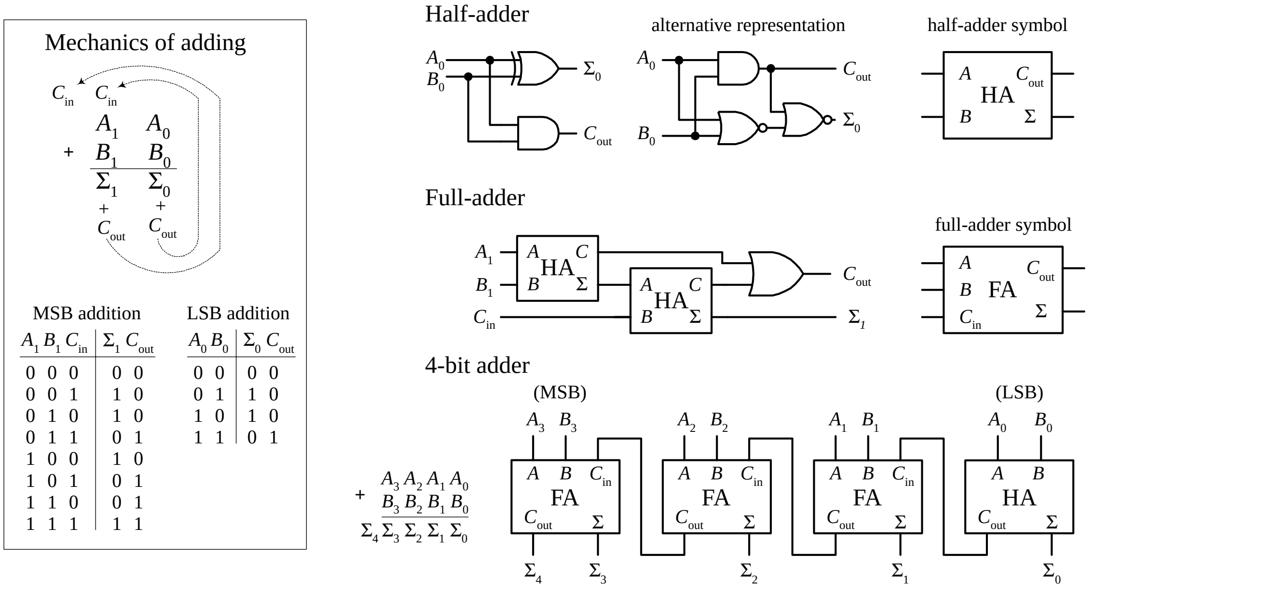


FIGURE 12.44

12.3.4 Binary Adders

With a few logic gates you can create a circuit that adds binary numbers. The mechanics of adding binary numbers is basically the same as that of adding decimal numbers. When the first digit of a two-digit number is added, a 1 is carried and added to the next row whenever the count exceeds binary 2 (e.g., 1 + 1 = 10, or = 0 carry a 1). For numbers with more digits, you have multiple carry bits. To demonstrate how you can use logic gates to perform basic addition, start out by considering the half-adder circuits below. Both half-adders shown are equivalent; one simply uses XOR/AND logic, while the other uses NOR/AND logic. The half-adder adds two single-bit numbers *A* and *B* and produces a 2-bit number; the LSB is represented as Σ_0, and the MSB or carry bit is represented as C_{out}.



MSB addition

A_1 B_1 C_{in}	Σ_1 C_{out}
0 0 0	0 0
0 0 1	1 0
0 1 0	1 0
0 1 1	0 1
1 0 0	1 0
1 0 1	0 1
1 1 0	0 1
1 1 1	1 1

LSB addition

A_0 B_0	Σ_0 C_{out}
0 0	0 0
0 1	1 0
1 0	1 0
1 1	0 1

FIGURE 12.45

The most complicated operation the half-adder can do is 1 + 1. To perform addition on a two-digit number, you must attach a full-adder circuit (shown in Fig. 12.45) to the output of the half-adder. The full-adder has three inputs; two are used to input the second digits of the two binary numbers (A_1, B_1), while the third accepts the carry bit from the half-adder (the circuit that added the first digits, A_0 and B_0, of the two numbers). The two outputs of the full-adder will provide the 2d-place digit sum Σ_1 and another carry bit that acts as the 3d-place digit of the final sum. Now, you can keep adding more full-adders to the half-adder/full-adder combination to add larger number, linking the carry bit output of the first full-adder to the next full-adder, and so forth. To illustrate this point, a 4-bit adder is shown in Fig. 12.45.

There are a number of 4-bit full-adder ICs available such as the 74LS283 and 4008. These devices will add two 4-bit binary number and provide an additional input carry bit, as well as an output carry bit, so you can stack them together to get 8-bit, 12-bit, 16-bit, etc. adders. For example, the figure below shows an 8-bit adder made by cascading two 74LS283 4-bit adders.

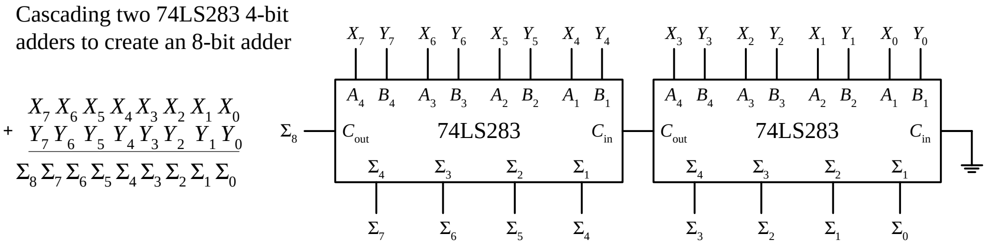

Cascading two 74LS283 4-bit adders to create an 8-bit adder
$X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0$
+ $Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0$
$\Sigma_8 \Sigma_7 \Sigma_6 \Sigma_5 \Sigma_4 \Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0$
X_7 Y_7 X_6 Y_6 X_5 Y_5 X_4 Y_4
X_3 Y_3 X_2 Y_2 X_1 Y_1 X_0 Y_0
A_4 B_4 A_3 B_3 A_2 B_2 A_1 B_1
Σ_8 C_{out} 74LS283 C_{in}
Σ_4 Σ_3 Σ_2 Σ_1
Σ_7 Σ_6 Σ_5 Σ_4 Σ_3 Σ_2 Σ_1 Σ_0

FIGURE 12.46

12.3.5 Binary Adder/Subtractor

Figure 12.47 shows how two 74LS283 4-bit adders can be combined with an XOR array to yield an 8-bit 2's complement adder/subtractor. The first number *X* is applied to the X_0-X_7 inputs, while the second number *Y* is applied to the Y_0-Y_7 inputs.

To add *X* and *Y*, the add/subtract switch is thrown to the add position, making one input of all XOR gates low. This has the effect of making the XOR gates appear transparent, allowing *Y* values to pass to the 74LS283s' *B* inputs (*X* values are passed to the *A* inputs). The 8-bit adder then adds the numbers and presents the result to the Σ outputs.

To subtract *Y* from *X*, you must first convert *Y* into 1's complement form; then you must add 1 to get *Y* into 2's complement form. After that you simply add *X* to the 2's complemented form of *Y* to get *X* – *Y*. When the add/subtract switch is thrown to the subtract position, one input to each XOR gate is set high. This causes the *Y* bits that are applied to the other XOR inputs to become inverted at the XOR outputs—you have just taken the 1's complement of *Y*. The 1's complement bits of *Y* are then presented to the inputs of the 8-bit adder. At the same time, C_{in} of the left 74LS283 is set high via the wire (see figure) so that a 1 is added to the 1's complement number to yield a 2's complement number. The 8-bit adder then adds *X* and the 2's complement of *Y* together. The final result is presented at the Σ outputs. In the figure, 76 is subtracted from 28.

8-bit 2's complement adder/subtractor

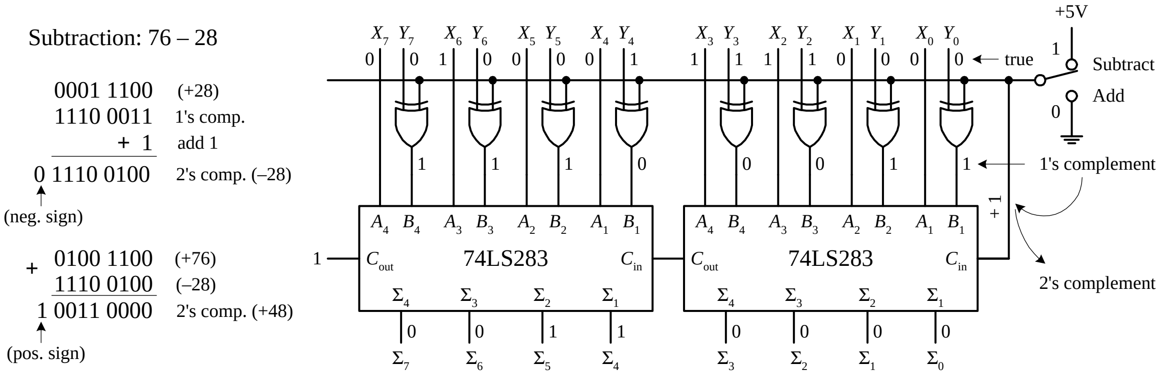
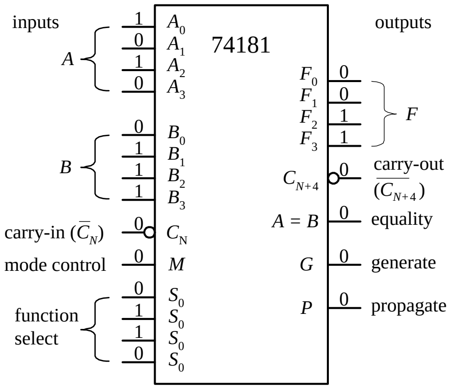


FIGURE 12.47

12.3.6 Arithmetic/Logic Units (ALUs)

An arithmetic/logic unit (ALU) is a multipurpose integrated circuit capable of performing various arithmetic and logic operations. To choose a specific operation to be performed, a binary code is applied to the IC's mode select inputs. The 74181, shown in Fig. 12.48, is a 4-bit ALU that provides 16 arithmetic and 16 logic operations.

To select an arithmetic operation, the 74181's mode control input (*M*) is set low. To select a logic operation, the mode control input is set high. Once you have decided whether you want to perform a logic or arithmetic operation, you apply a



Mode select				Logic functions	Arithmetic operations
S_3	S_2	S_1	S_0	($M = 1$)	($M = 0$, $\overline{C}_n = 1$)
0	0	0	0	$F = \overline{A}$	$F = A$
0	0	0	1	$F = \overline{A + B}$	$F = A + B$
0	0	1	0	$F = \overline{A}B$	$F = A + \overline{B}$
0	0	1	1	$F = 0$	$F =$ minus 1 (2's comp.)
0	1	0	0	$F = \overline{AB}$	$F = A$ plus $A\overline{B}$
0	1	0	1	$F = \overline{B}$	$F = (A + B)$ plus $A\overline{B}$
0	1	1	0	$F = A \oplus B$	$F = A$ minus B minus 1
0	1	1	1	$F = A\overline{B}$	$F = A\overline{B}$ minus 1
1	0	0	0	$F = \overline{A} + B$	$F = A$ plus AB
1	0	0	1	$F = \overline{A \oplus B}$	$F = A$ plus B
1	0	1	0	$F = B$	$F = (A + \overline{B})$ plus AB
1	0	1	1	$F = AB$	$F = AB$ minus 1
1	1	0	0	$F = 1$	$F = A$ plus A
1	1	0	1	$F = A + B$	$F = (A + B)$ plus A
1	1	1	0	$F = A + B$	$F = (A + \overline{B})$ plus A
1	1	1	1	$F = A$	$F = A$ minus 1

FIGURE 12.48

4-bit code to the mode select inputs (S_0, S_1, S_2, S_3) to specify which specific operation, as indicated within the truth table, is to be performed. For example, if you select $S_3 = 1$, $S_2 = 1$, $S_1 = 1$, $S_0 = 0$, while $M = 1$, then you get $F_0 = A_0 + B_0$, $F_1 = A_1 + B_1$, $F_2 = A_2 + B_2$, $F_3 = A_3 + B_3$. Note that the + shown in the truth table does not represent addition; it is used to represent the OR function—for addition, you use "plus." Carry-in ($\overline{C}_N$) and carry-out (C_{N+4}) leads are provided for use in arithmetic operations. All arithmetic results generated by this device are in 2's complement notation.

12.3.7 Comparators and Magnitude Comparator ICs

A digital comparator is a circuit that accepts two binary numbers and determines whether the two numbers are equal. For example, the figure below shows a 1-bit and a 4-bit comparator. The 1-bit comparator outputs a high (1) only when the two 1-bit numbers A and B are equal. If A is not equal to B, then the output goes low (0). The 4-bit is basically four 1-bit comparators in one. When all individual digits of each number are equal, all XOR gates output a high, which in turn enables the AND gate, making the output high. If any two corresponding digits of the two numbers are not equal, the output goes low.

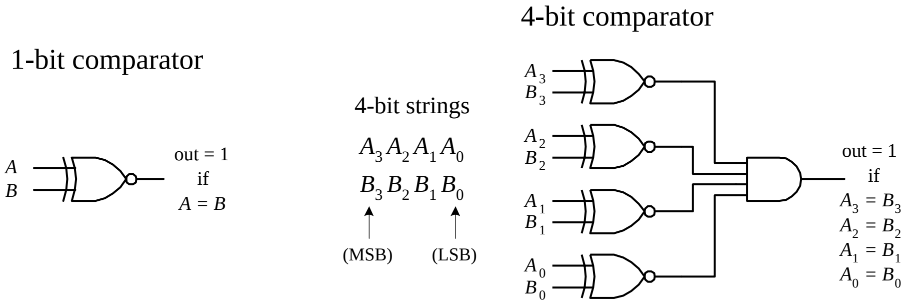
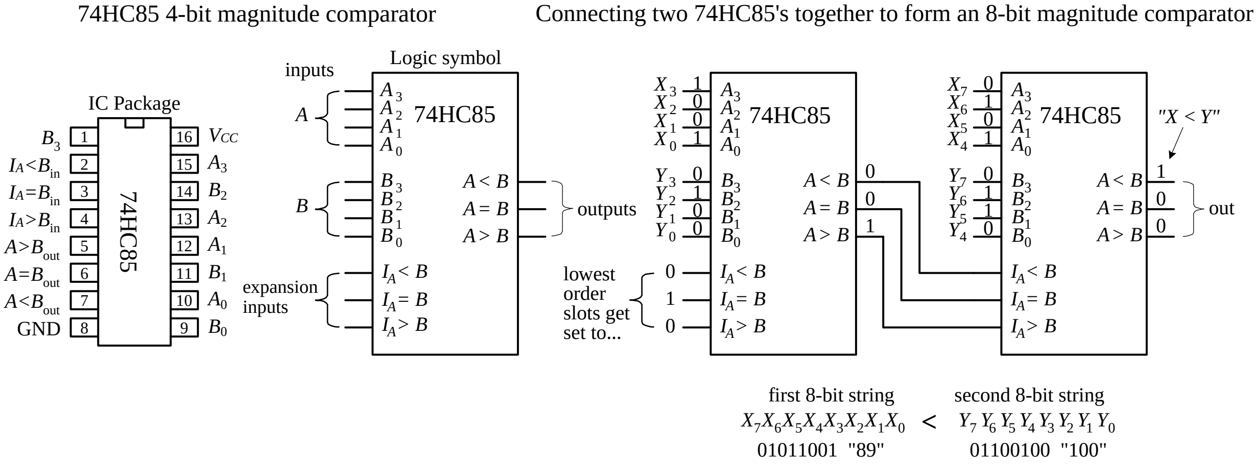


FIGURE 12.49

Now, say you want to know which number, A or B, is larger. The circuits in Fig. 12.49 will not do the trick. What you need instead is a *magnitude comparator* like the 74HC85 shown in Fig. 12.50. This device not only tells you if two numbers are equal; it also tells you which number is larger. For example, if you apply a 1001 (9_{10}) to the $A_3A_2A_1A_0$ inputs and a second number 1100 (12_{10}) to the $B_3B_2B_1B_0$ inputs, the $A < B$ output will go high (the other two outputs, $A > B$ and $A = B$, will remain low). If A and B were equal, the $A = B$ output would have gone high, etc. If you wanted to compare a larger number, say, two 8-bit numbers, you simply cascade two 74HC85's together, as shown to the right in Fig. 12.50. The leftmost 74HC85 compares the lower-order bits, while the rightmost 74HC85 compares the higher-order bits. To link the two devices together, you connect the output of the lower-order device to the expansion inputs of the higher-order device, as shown. The lower-order device's expansion inputs are always set low ($I_A < B$), high ($I_A = B$), low ($I_A > B$).


74HC85 4-bit magnitude comparator

Connecting two 74HC85's together to form an 8-bit magnitude comparator

FIGURE 12.50

12.3.8 Parity Generator/Checker

Often, external noise will corrupt binary information (cause a bit to flip from one logic state to the other) as it travels along a conductor from one device to the next. For example, in the 4-bit system shown in Fig. 12.51, a BCD 4 (0100) picks up noise and becomes 0101 (or 5) before reaching its destination. Depending on the application, this type of error could lead to some serious problems.

To avoid problems caused by unwanted data corruption, a parity generator/checker system, like the one shown in Fig. 12.51, can be used. The basic idea is to add an extra bit, called a *parity bit*, to the digital information being transmitted. If the parity bit makes the sum of all transmitted bits (including the parity bit) odd, the transmitted information is of odd parity. If the parity bit makes the sum even, the transmitted information is of even parity. A parity generator circuit creates the parity bit, while the parity checker on the receiving end determines if the information sent is of the proper parity. The type of parity (odd or even) is agreed to beforehand, so the parity checker knows what to look for. The parity bit can be placed next to the MSB or the LSB, provided the device on the receiving end knows which bit is the parity bit and which bits are the data. The arrangement shown in Fig. 12.51 is designed with an even-parity error-detection system.

Even vs. odd parity (examples)

4-bit string	parity bit	parity
1 0 0 1	0	even (two 1's)
1 1 0 0	1	odd (three 1's)
0 0 0 0	1	odd (one 1's)
1 1 1 0	1	even (four 1's)

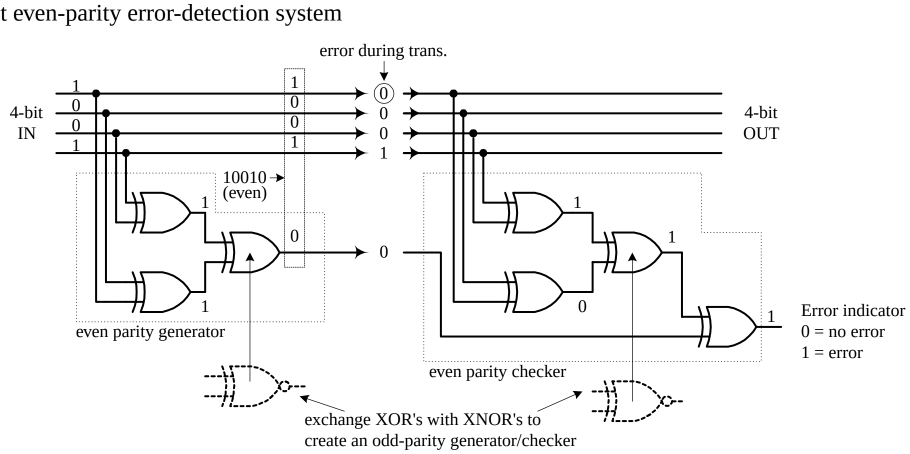
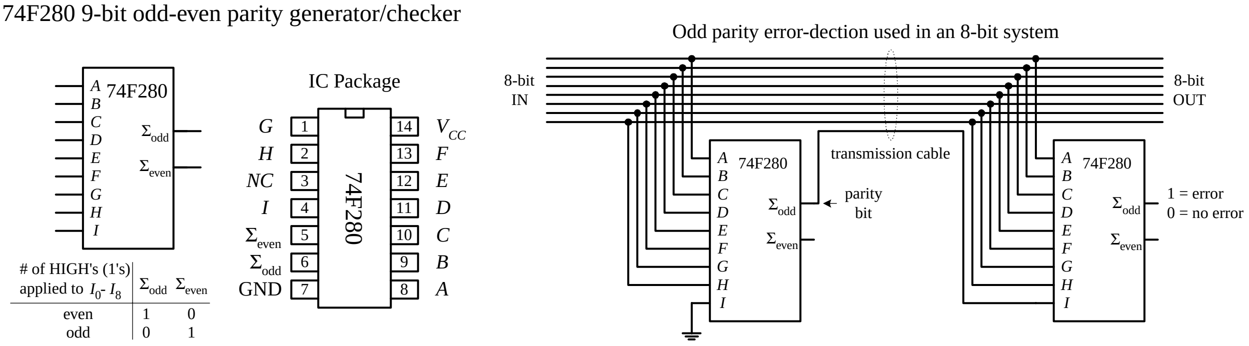


FIGURE 12.51

If you want to avoid building parity generators and checkers from scratch, use a parity generator/checker IC like the 74F280 9-bit odd-even parity generator/checker shown below. To make a complete error-detection system, two 74F280s are used—one acts as the parity generator; the other acts as the parity checker. The generator's inputs *A* through *H* are connected to the eight data lines of the transmitting portion of the circuit. The ninth input (*I*) is grounded when the device is used as a generator. If you want to create an odd-parity generator, you tap the Σ_{odd} output; for even parity, you tap Σ_{even}. The 74F280 checker taps the main line at the receiving end and also accepts the parity bit line at input *I*. The figure below shows an odd-parity error-detection system used with an 8-bit system. If an error occurs, a high (1) is generated at the Σ_{odd} output.



# of HIGH's (1's) applied to I_0- I_8	Σ_{odd}	Σ_{even}
even	1	0
odd	0	1

FIGURE 12.52

12.3.9 A Note on Obsolescence and the Trend Toward Microcontroller Control

You have just covered most of the combinational devices you will find discussed in textbooks and find listed within electronic catalogs. Many of these devices are still used. However, some devices such as the binary adders and code converters are becoming obsolete.

Today, the trend is to use software-controlled devices such as microprocessors and microcontrollers to carry out arithmetic operations and code conversions. Before you attempt to design any logic circuit, I suggest jumping to Section 12.12. In that section, pay close attention to microcontrollers. These devices are quite amazing. They are essentially microprocessors but are significantly easier to program and are easier to interface with other circuits and devices.

Microcontrollers can be used to collect data, store data, and perform logical operations using the input data. They also can generate output signals that can be used to control displays, audio devices, stepper motors, servos, etc. The specific functions a microcontroller is designed to perform depend on the program you store in its internal ROM-type memory. Programming the microcontroller typically involves simply using a special programming unit provided by the manufacturer. The programming unit usually consists of a special prototyping platform that is linked to a PC (via a serial or parallel port) that is running a host program. In the host program, you typically write out a program in a high-level language such as C, or some other specialized language designed for a certain microcontroller, and then, with the press of a key, the program is converted into machine language (1s and 0s) and downloaded into the microcontroller's memory.

In many applications, a single microcontroller can replace entire logic circuits comprised of numerous discrete components. For this reason, it is tempting to skip the rest of the sections of this chapter and go directly to the section on microcontrollers. However, there are three basic problems with this approach. First, if you are a beginner, you will miss out on many important principles behind digital control that are most easily understood by learning how the discrete components work. Second, many digital circuits that you can build simply do not require the amount of sophistication a microcontroller provides. Finally, you may feel intimidated by the electronics catalogs that list every conceivable component available, be it obsolete or not. Knowing what's out there and knowing what to avoid are also important parts of the learning process.

12.4 Logic Families

Before moving on to sequential logic, let's touch on a few practical matters regarding the various logic families available and what kind of operating characteristics these families have. In this section you will also encounter unique logic gates that have open-collector output stages and logic gates that have Schmitt-triggered inputs.

The key ingredient within any integrated logic device, be it a logic gate, a multiplexer, or a microprocessor, is the transistor. The kinds of transistors used within the integrated circuit, to a large extend, specify the type of logic family. The two most popular transistors used in ICs are bipolar and MOSFET transistors. In general, ICs made from MOSFET transistors use less space due to their simpler construction, have very high noise immunity, and consume less power than equivalent bipolar transistor ICs. However, the high input impedance and input capacitance of the MOSFET transistors (due to their insulated gate leads) results in longer time constants for transistor on/off switching speeds when compared with bipolar gates and therefore typically result in a slower device. Over years of development, however, the performance gap between these two technologies has narrowed considerably.

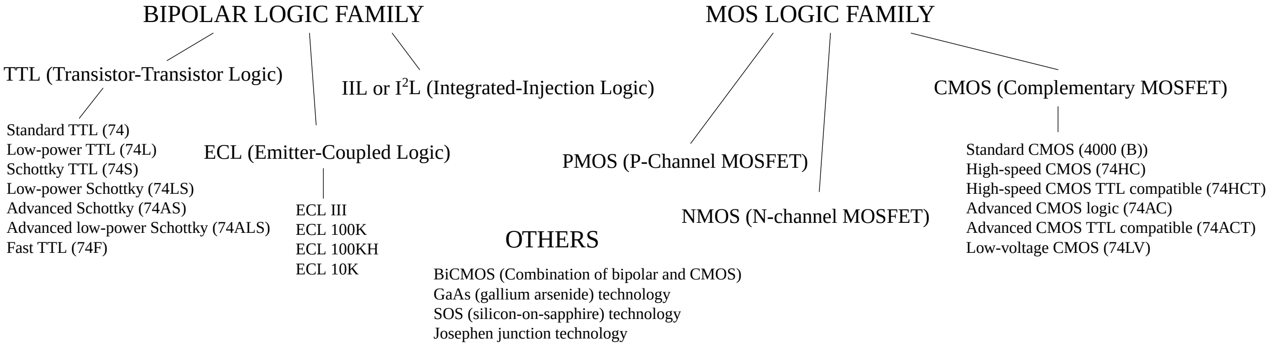

BIPOLAR LOGIC FAMILY
MOS LOGIC FAMILY
TTL (Transistor-Transistor Logic)
IIL or I^{2}L (Integrated-Injection Logic)
CMOS (Complementary MOSFET)
Standard TTL (74)
Low-power TTL (74L)
Schottky TTL (74S)
Low-power Schottky (74LS)
Advanced Schottky (74AS)
Advanced low-power Schottky (74ALS)
Fast TTL (74F)
ECL (Emitter-Coupled Logic)
ECL III
ECL 100K
ECL 100KH
ECL 10K
PMOS (P-Channel MOSFET)
NMOS (N-channel MOSFET)
Standard CMOS (4000 (B))
High-speed CMOS (74HC)
High-speed CMOS TTL compatible (74HCT)
Advanced CMOS logic (74AC)
Advanced CMOS TTL compatible (74ACT)
Low-voltage CMOS (74LV)
OTHERS
BiCMOS (Combination of bipolar and CMOS)
GaAs (gallium arsenide) technology
SOS (silicon-on-sapphire) technology
Josephen junction technology

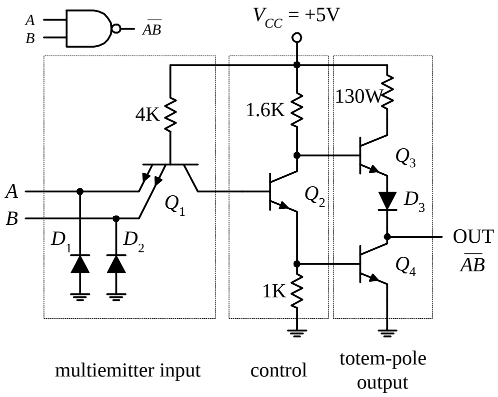
FIGURE 12.53

Both the bipolar and MOSFET logic families can be divided into a number of subclasses. The major subclasses of the bipolar family include TTL (transistor-transistor logic), ECL (emitter-coupled logic), and IIL or I^{2}L (integrated-injection logic). The major subclasses of the MOSFET logic include PMOS (P-channel MOSFET logic), NMOS (N-channel MOSFET logic), and CMOS (complementary MOSFET logic). CMOS uses both NMOS and PMOS technologies (it uses both N-channel and P-channel MOSFETs). The two most popular technologies are TTL and CMOS, while the other technologies are typically used in large-scale integration devices, such as microprocessors and memories. There are new technologies popping up all the time, which yield faster, more energy-efficient devices. Some examples include BiCMOS, GaAS, SOS, and Josephen junction technologies.

As you have already learned, TTL and CMOS devices are grouped into functional categories that get placed into either the 7400 series [74F, 74LS, 74HC (CMOS), etc.] or 4000 CMOS series (or the improved 4000B series). Now, another series you will run into is the 5400 series. This series is essentially equivalent to the 7400 series (same pinouts, same basic logic function), but it is a more expansive chip because it is designed for military applications that require increased supply voltage tolerances and temperature tolerances. For example, a 7400 IC typically has a supply voltage range from 4.75 to 5.25 V with a temperature range from 0 to 70°C, while a 5400 IC typically will have a voltage range between 4.5 and 5.5 V and a temperature range from –55 to 125°C.

12.4.1 TTL Family of ICs

The original TTL series, referred to as the *standard TTL series* (74*xx*), was developed early in the 1960s. This series is still in use, even though its overall performance is inferior to the newer line of TTL devices, such as the 74LS*xx*, 74ALS*xx*, and 74F*xx*. The internal circuitry of a standard TTL 7400 NAND gate, along with a description of how it works, is provided next.



The TTL NAND gate is broken up into three basic sections: multiemitter input, control section, and totem-pole output stage. In the multiemitter input section, a multiemitter bipolar transistor Q_1 acts like a two-input AND gate, while diodes D_1 and D_2 act as negative clamping diodes used to protect the inputs from any short-term negative input voltages that could damage the transistor. Q_2 provides control and current boosting to the totem-pole output stage; when the output is high (1), Q_4 is off (open) and Q_3 is on (short). When the output is low (0), Q_4 is on and Q_3 is off. Because one or the other transistor is always off, the current flow from V_{CC} to ground in that section of the circuit is minimized. The lower figures show both a high and low output state, along with the approximate voltages present at various locations. Notice that the actual output voltages are not exactly 0 or +5V—a result of internal voltage drops across resistor, transistor, and diode. Instead, the outputs are around 3.4V for high and 0.3V for low. As a note, to create, say, an eight-input NAND gate, the multiemitter input transistor would have eight emitters instead of just two as shown.

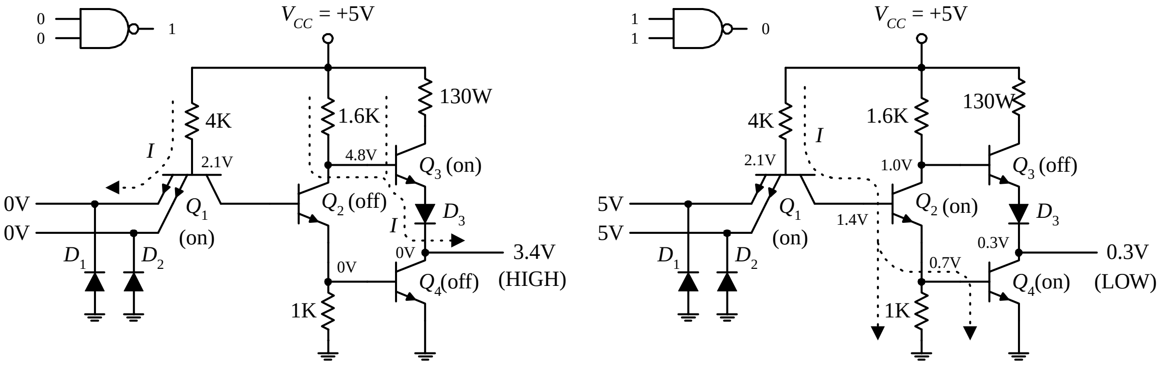


FIGURE 12.54

A simple modification to the standard TTL series was made early on by reducing all the internal resistor values in order to reduce the *RC* time constants and thus increase the speed (reduce propagation delays). This improvement to the original TTL series marked the 74H series. Although the 74H series offered improved speed (about twice as fast) over the 74 series, it had more than double the power consumption. Later, the 74L series emerged. Unlike the 74H, the 74L took the 74 and increased all internal resistances. The net effect lead to a reduction in power but increased propagation delay.

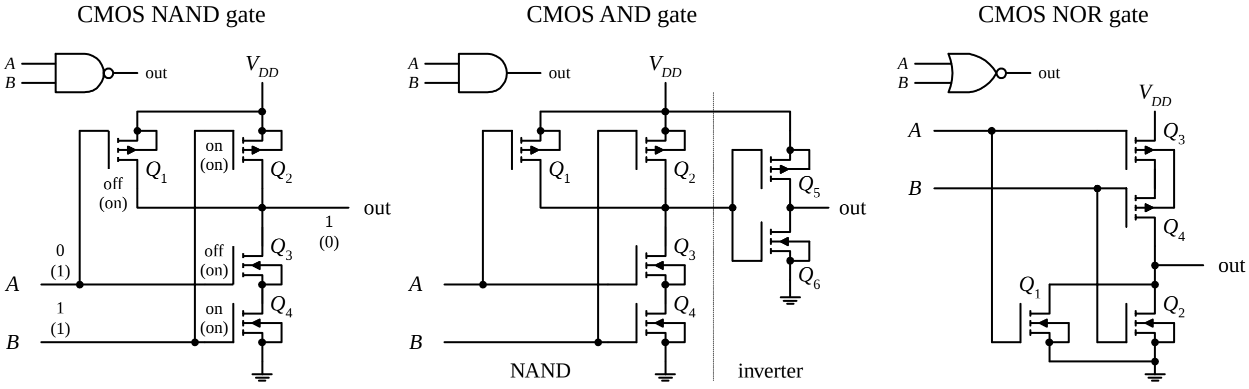
A significant improvement in speed within the TTL line emerged with the development of the 74S*xx* series (Schottky TTL series). The key modifications involved placing Schottky diodes across the base-to-collector junctions of the transistors. These Schottky diodes eliminated capacitive effects caused by charge buildup in the transistor's base region by passing the charge to the collector region. Schottky diodes were the best choice because of their inherent low charge buildup characteristics. The overall effect was an increase in speed by a factor of 5 and only a doubling in power.

Continually over time, by using different integration techniques and increasing the values of the internal resistors, more power-efficient series emerged, like the low-power Schottky 74LS series, with about one-third the power dissipation of the 74S. After the 74LS, the advanced-low-power Schottky 74ALS series emerged, which had even better performance. Another series developed around this time was the 74F series, or FAST logic, which used a new process of integration called *oxide isolation* (also used in the ALS series) that led to reduced propagation delays and decreased the overall size.

Today you will find many of the older series listed in electronics catalogs. Which series you choose ultimately depends on what kind of performance you are looking for.

12.4.2 CMOS Family of ICs

While the TTL series was going through its various transformations, the CMOS series entered the picture. The original CMOS 4000 series (or the improved 4000B series) was developed to offer lower power consumption than the TTL series of devices—a feature made possible by the high input impedance characteristics of its MOSFET transistors. The 4000B series also offered a larger supply voltage range (3 to 18 V), with minimum logic high = ⅔V_{DD}, and maximum logic low = ⅓V_{DD}. The 4000B series, though more energy efficient than the TTL series, was significantly slower and more susceptible to damage due to electrostatic discharge. The figure below shows the internal circuitry of CMOS NAND, AND, and NOR gates. To figure out how the gates work, apply high (logic 1) or low (logic 0) levels to the inputs and see which transistor gates turn on and which transistor gates turn off.


CMOS NAND gate

CMOS AND gate

CMOS NOR gate

FIGURE 12.55

A further improvement in speed over the original 4000B series came with the introduction of the 40H00 series. Although this series was faster than the 4000B series, it was not quite as fast as the 74LS TTL series. The 74C CMOS series also emerged on the scene, which was designed specifically to be pin-compatible with the TTL line. Another significant improvement in the CMOS family came with the development of the 74HC and the 74HCT series. Both these series, like the 74C series, were pin-compatible with the TTL 74 series. The 74HC (high-speed CMOS) series had the same speed as the 74LS as well as the traditional CMOS low-power consumption. The 74HCT (high-speed CMOS TTL compatible) series was developed to be interchangeable with TTL devices (same input/output voltage level characteristics). The 74HC series is very popular today. Still further improvements in 74HC/74HCT series led to the advanced CMOS logic (74AC/74ACT) series. The 74AC (advanced CMOS) series approached speeds comparable with the 74F TTL series, while the 74ACT (advanced CMOS TTL compatible) series was designed to be TTL compatible.

12.4.3 Input/Output Voltages and Noise Margins

The exact input voltage levels required for a logic IC to perceive a high (logic 1) or low (logic 0) input level differ between the various logic families. At the same time, the high and low output levels provided by a logic IC vary among the logic families. For example, the figure below shows valid input and output voltage levels for both the 74LS (TTL) and 74HC (CMOS) families.

Valid input/output logic levels for the TTL 74LS and the CMOS 74HC

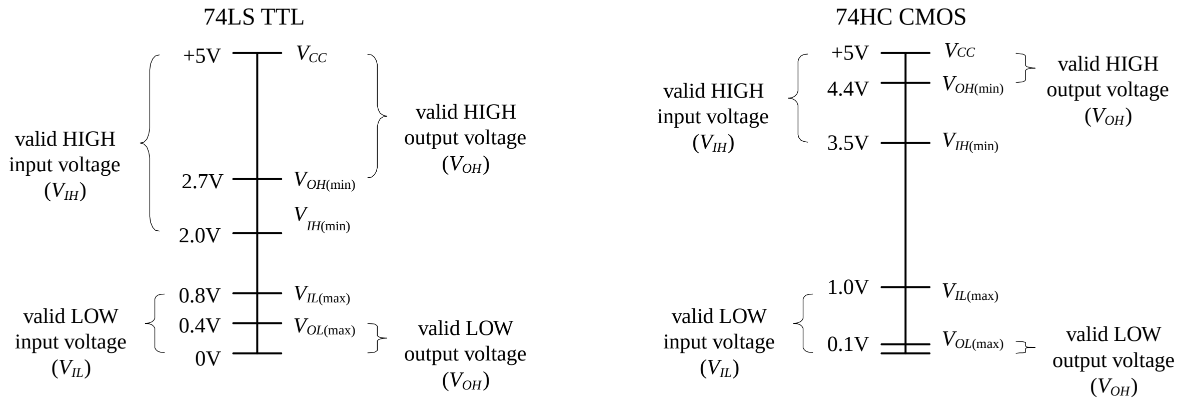

74LS TTL
+5V
V_{CC}
valid HIGH input voltage (V_{IH})
valid HIGH output voltage (V_{OH})
2.7V
$V_{OH(\mathrm{min})}$
$V_{IH(\mathrm{min})}$
2.0V
0.8V
$V_{IL(\mathrm{max})}$
0.4V
$V_{OL(\mathrm{max})}$
0V
valid LOW input voltage (V_{IL})
valid LOW output voltage (V_{OH})
74HC CMOS
+5V
V_{CC}
4.4V
$V_{OH(\mathrm{min})}$
valid HIGH output voltage (V_{OH})
valid HIGH input voltage (V_{IH})
3.5V
$V_{IH(\mathrm{min})}$
1.0V
$V_{IL(\mathrm{max})}$
0.1V
$V_{OL(\mathrm{max})}$
valid LOW input voltage (V_{IL})
valid LOW output voltage (V_{OH})

FIGURE 12.56

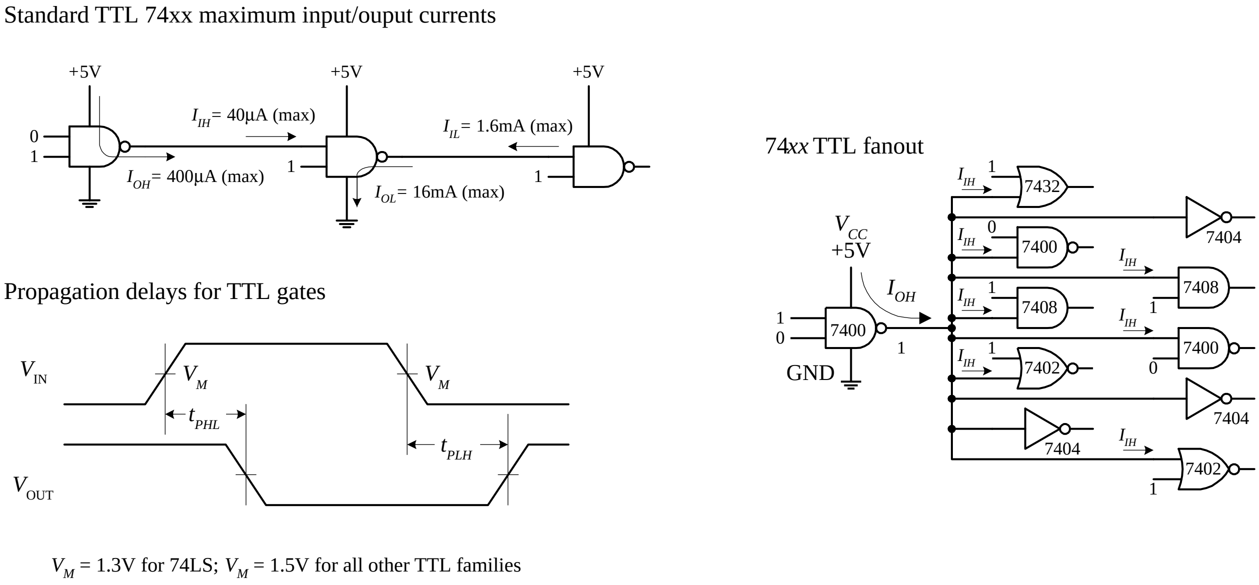
In Fig. 12.56, V_{IH} represents the valid voltage range that will be interpreted as a high logic input level. V_{IL} represents the valid voltage range that will be interpreted as a low logic input level. V_{OL} represents the valid voltage range that will be guaranteed as a low logic output level, while V_{OH} represents the valid voltage range that will be guaranteed as a high logic output level.

As you can see from Fig. 12.56, if you connect the output of a 74HC device to the input of a 74LS device, there is no problem—the output logic levels of the 74HC are within the valid input range of the 74LS. However, if you turn things around, driving a 74HC device's inputs from a 74LS's output, you have problems—a high output level from the 74LS is too small to be interpreted as a high input level for the 74HC. I will discuss tricks used to interface the various logic families together in a moment.

12.4.4 Current Ratings, Fanout, and Propagation Delays

Logic IC inputs and outputs can only sink or source a given amount of current. I_{IL} is defined as the maximum low-level input current, I_{IH} as the maximum high-level input current, I_{OH} as the maximum high-level output current, and I_{OL} as the maximum low-level output current. As an example, a standard 74*xx* TTL gate may have an $I_L = -1.6$ mA and $I_{IH} = 40$ µA while having an $I_{OL} = 16$ mA and $I_{OH} = -400$ µA. The negative sign means that current is leaving the gate (the gate is acting as a source), while a positive sign means that current in entering the gate (the gate is acting as sink).

The limit to how much current a device can sink or source determines the size of loads that can be attached. The term *fanout* is used to specify the total number of gates that can be driven by a single gate of the same family without exceeding the current rating of the gate. The fanout is determined by taking the smaller result of I_{OL}/I_{IL} or I_{OH}/I_{IH}. For the standard 74 series, the fanout is 10 (16 mA/1.6 mA). For the 74LS



Standard TTL 74xx maximum input/ouput currents

Propagation delays for TTL gates

V_M = 1.3V for 74LS; V_M = 1.5V for all other TTL families

FIGURE 12.57

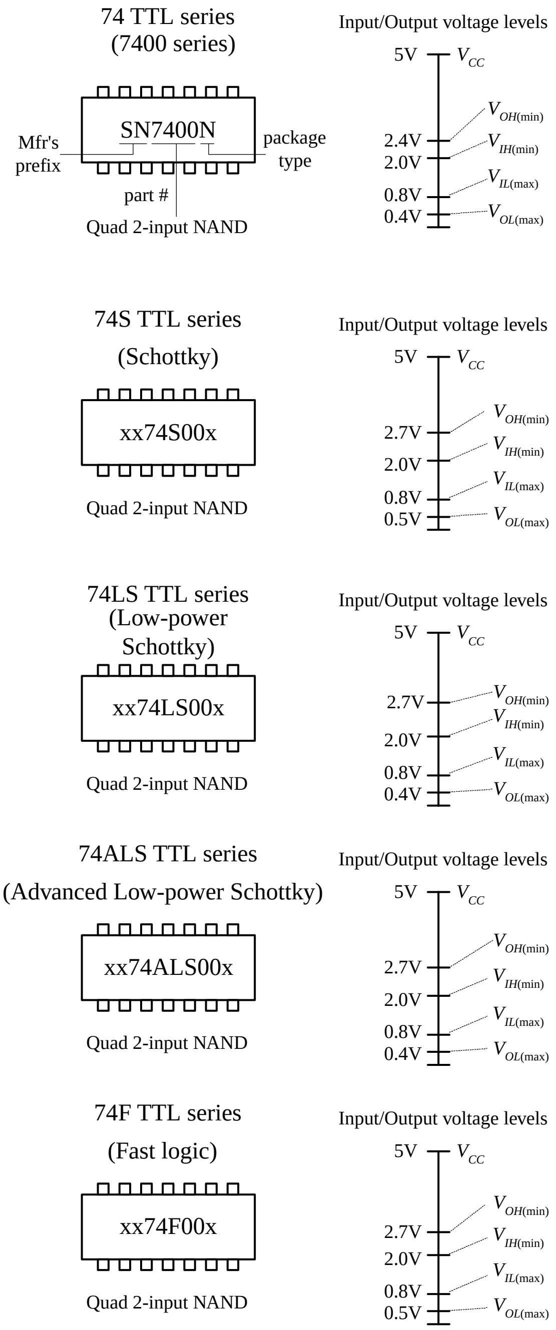
series, the fanout is around 20; for the 74F, it is around 33; and for the 7HC, it is around 50.

If you apply a square pulse to the input of a logic gate, the output signal will experience a sloping rise time and fall time, as shown in the graph in Fig. 12.57. The *rise time* (t_r) is the length of time it takes for a pulse to rise from 10 to 90 percent of its high level (e.g., 5 V = high: 0.5 V = 10%, 4.5 V = 90%). The *fall time* t_f is the length of time it takes for a high level to fall from the 90 to 10 percent. The rise and fall times, however, are not as significant when compared with propagation delays between input transition and output response. Propagation delay results from the limited switching speeds of the internal transistors within the logic device. The low-to-high propagation delay T_{PHL} is the time it takes for the output of a device to switch from low to high after the input transition. The high-to-low propagation delay T_{PLH} is the time it takes for the output to switch from high to low after the input transition. When designing circuits, it is important to take into account these delays, especially when you start dealing with sequential logic, where timing is everything. Figures 12.58 and 12.59 provide typical propagation delays for various TTL and CMOS devices. Manufacturers will provide more accurate propagation information in their data sheets.

12.4.5 A Detailed Look at the Various TTL and CMOS Subfamilies

The following information, shown in Figs. 12.58 and 12.59, especially the data pertaining to propagation delays and current ratings, represents *typical values* for a given logic series. For more accurate data about a specific device, you must consult the manufacturer's literature. In other words, only use the provided information as a rough guide to get a feeling for the overall performance of a given logic series.

TTL Series



74 TTL series
(7400 series)

Year introduced: 1968
Voltage supply: 4.5 to 5.5V
P_D = 10 mW/gate
t_{PHL} = 8ns, t_{PLH} = 13ns
$f_{\max}$ = 35 MHz
$I_{IH(\max)}$ = 40µA (sink)
$I_{IL(\max)}$ = -1.6mA (source)
$I_{OH(\max)}$ = -0.4mA (source)
$I_{OL(\max)}$ = 16mA (sink)

Comments

The 74*xx* TTL series has been the foundation and reference for virtually every new logic series of IC's. This series is easy to use and has excellent output drive capability. However, this series has high-power consumption and is relatively slow, making the series obsolete when compared with the newer TTL series.

74S TTL series
(Schottky)

Year introduced: 1974
Voltage supply: 4.5 to 5.5V
P_D = 20 mW/gate
t_{PHL} = 5ns, t_{PLH} = 5ns
$f_{\max}$ = 125 MHz
$I_{IH(\max)}$ = 50µA (sink)
$I_{IL(\max)}$ = -2.0mA (source)
$I_{OH(\max)}$ = -1.0mA (source)
$I_{OL(\max)}$ = 20mA (sink)

Comments

Increased speed over the standard 74xx TTL. Incorporates a Schottky diode across base-to-collector junctions of bipolar transistors within the IC. This reduces propagation delay (by factor of 4 over 74xx series) by eliminating charge build-up at base region of transistors (a build-up of charge takes time to dissipate, thus causing propagation delay).

74LS TTL series
(Low-power
Schottky)

Year introduced: 1976
Voltage supply: 4.5 to 5.5V
P_D = 2 mW/gate
t_{PHL} = 8ns, t_{PLH} = 8ns
$f_{\max}$ = 45 MHz
$I_{IH(\max)}$ = 20µA (sink)
$I_{IL(\max)}$ = -400µA (source)
$I_{OH(\max)}$ = -0.4mA (source)
$I_{OL(\max)}$ = 4mA (sink)

Comments

A very popular TTL series used today. Improved power dissipation over the 74Sxx series, but isn't as fast.

74ALS TTL series
(Advanced Low-power Schottky)

Year introduced: 1979
Voltage supply: 4.5 to 5.5V
P_D = 1 mW/gate
t_{PHL} = 7ns, t_{PLH} = 5ns
$f_{\max}$ = 35 MHz
$I_{IH(\max)}$ = 20µA (sink)
$I_{IL(\max)}$ = -0.1mA (source)
$I_{OH(\max)}$ = -0.4mA (source)
$I_{OL(\max)}$ = 8mA (sink)

Comments

Further improvement of the 74LS*xx* series; propagation delay and power dissipation are reduced. Rapidly replacing the 74*xx* and 74LS*xx* series.

74F TTL series
(Fast logic)

Year introduced: 1983
Voltage supply: 4.5 to 5.5V
P_D = 4 mW/gate
t_{PHL} = 3.7ns, t_{PLH} = 3.2ns
$f_{\max}$ = 100 MHz
$I_{IH(\max)}$ = 20µA (sink)
$I_{IL(\max)}$ = -1mA (source)
$I_{OH(\max)}$ = -0.6mA (source)
$I_{OL(\max)}$ = 20mA (sink)

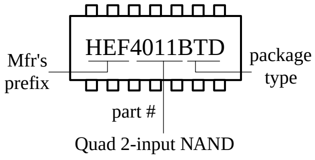
Comments

Faster than the 74ALSxx series. It also has improved noise margins, reduced input currents and better driving capabilities. Handles heavier capacitive loads than eariler TTL families.

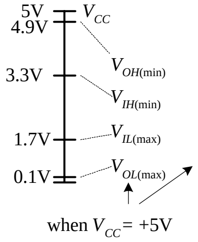
FIGURE 12.58

CMOS Series

4000 CMOS series
(4000B improved version)



Input/Output voltage levels

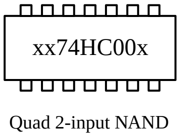


Year introduced: 1970
Voltage supply: 3 to 18V
P_D = 1mW/gate at 1 MHz
t_{PHL} = 50ns, t_{PLH} = 65ns
$f_{\max}$ = 6 MHz
$I_{IH(\max)}$ = 1µA (sink)
$I_{IL(\max)}$ = -1µA (source)
$I_{OH(\max)}$ = -3.0mA (source)
$I_{OL(\max)}$ = 3.0mA (sink)

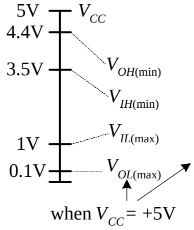
Comments

The 4000 series is the orignal CMOS line--the 4000B series is an improvement. Supply voltages range from 3 to 18V, with V_{IH} = 2/3V_{CC} and V_{OH} = 1/3V_{CC} . Was popular due to low power consumption when compared to TTL. Considered obsolete when compared to the newer CMOS line of IC's, and is much slower than any of the TTL series. The 4000 series is also susceptible to damage from electrostatic discharge.

74HC CMOS series
(High-speed CMOS)



Input/Output voltage levels

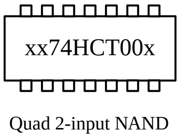


Year introduced: 1975
Voltage supply: 2 to 6V
P_D = 0.5mW/gate at 1 MHz
t_{PHL} = 20ns, t_{PLH} = 20ns
$f_{\max}$ = 20 MHz
$I_{IH(\max)}$ = 1µA (sink)
$I_{IL(\max)}$ = -1µA (source)
$I_{OH(\max)}$ = -4mA (source)
$I_{OL(\max)}$ = 4mA (sink)

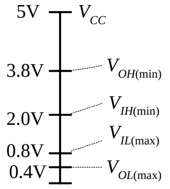
Comments

Improvement in speed over the 4000 CMOS series; as speedy as the 74LS*xx* TTL series, depending on the operating frequency. Also provides greater noise immunity and greater voltage and temperature operating ranges than TTL. This is a very popular series used today.

74HCT CMOS series
(High-speed CMOS, TTL compatible)



Input/Output voltage levels

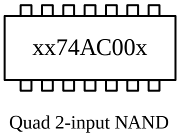


Year introduced: 1975
Voltage supply: 4.5 to 5.5V
P_D = 0.5mW/gate at 1 MHz
t_{PHL} = 40ns, t_{PLH} = 40ns
$f_{\max}$ = 24 MHz
$I_{IH(\max)}$ = 1µA (sink)
$I_{IL(\max)}$ = -1µA (source)
$I_{OH(\max)}$ = -4mA (source)
$I_{OL(\max)}$ = 4mA (sink)

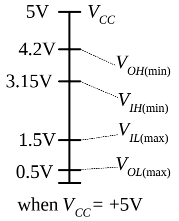
Comments

Like the 74HCxx CMOS series but is also TTL compatable (it's pin compatible, as well as input/ output voltage-level compatable).

74AC CMOS series
(Advanced CMOS)



Input/Output voltage levels

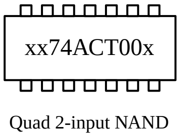


Year introduced: 1985
Voltage supply: 3 to 5.5V
P_D = 0.5mW/gate at 1 MHz
t_{PHL} = 3ns, t_{PLH} = 3ns
$f_{\max}$ = 125 MHz
$I_{IH(\max)}$ = 1µA (sink)
$I_{IL(\max)}$ = -1µA (source)
$I_{OH(\max)}$ = -24mA (source)
$I_{OL(\max)}$ = 24mA (sink)

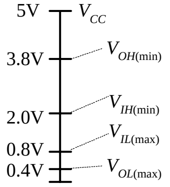
Comments

Further improvement of the 74HCxx series; it's faster, has a higher output current drive capacity, and provides shorter propagation delays.

74ACT CMOS series
(Advanced CMOS, TTL compatible)



Input/Output voltage levels

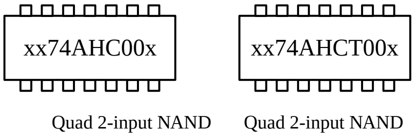


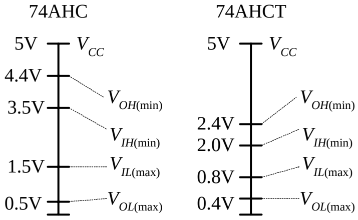
Year introduced: 1985
Voltage supply: 4.5 to 5.5V
P_D = 0.5mW/gate at 1 MHz
t_{PHL} = 5ns, t_{PLH} = 5ns
$f_{\max}$ = 125 MHz
$I_{IH(\max)}$ = 1µA (sink)
$I_{IL(\max)}$ = -1µA (source)
$I_{OH(\max)}$ = -24mA (source)
$I_{OL(\max)}$ = 24mA (sink)

Comments

Like the 74ACxx CMOS series but is also TTL compatable (it's pin compatible, as well as input/ output voltage-level compatable).

74AHC / 74AHCT CMOS series
(Advanced High-speed/ TTL compatible CMOS)





Advanced high-speed CMOS is an enhanced version of the 74HC and 74HCT series; the 74AHC has half the static power consumption, one-third the propagation delay. The 74AHCT is TTL compatible. Introduced in 1996.

FIGURE 12.59

12.4.6 A Look at a Few Other Logic Series

The 74-BiCMOS Series

The 74-BiCMOS series of devices incorporates the best features of bipolar and CMOS technology together in one package. The overall effect is an extremely high-speed, low-power digital logic family. This product line is especially well suited for and is mostly limited to microprocessor bus interface logic. Each manufacturer uses a different suffix to identify its BiCMOS line. For example, Texas Instruments uses 74BCT*xx*, while Signetics (Phillips) uses 74ABT*xx*.

74-Low-Voltage Series

The 74-low-voltage series is a relatively new series that uses a nominal supply voltage of 3.3 V. Members of this series include the 74LV (low-voltage HCMOS), 74LVC (low-voltage CMOS), the 74LVT (low-voltage technology), and the 74ALVC (advanced low-voltage CMOS). See Fig. 12.60.

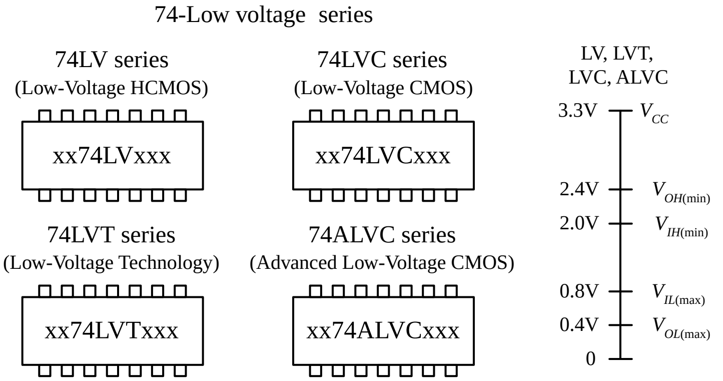


FIGURE 12.60

A relatively new series of logic using a nominal supply voltage of 3.3 V which are designed for extremely low-power and low-voltage applications (e.g., battery-powered devices). The switching speed of LV logic is extremely fast, ranging from about 9 ns for LB series down to 2.1 ns for ALVC. Another nice feature of LV logic is high output drive capability. The LVT, for example, can sink up to 64 mA and source up to 32 mA. LVT 1992 BiCMOS, LVC/ALVC 1993 CMOS.

Emitter-Coupled Logic

Emitter-coupled logic (ECL), a member of the bipolar family, is used for extremely high-speed applications, reaching speeds up to 500 MHz with propagation delays as low as 0.8 ns. There is one problem with ECL—it consumes a considerable amount of power when compared with the TTL and CMOS series. ECL is best suited for use in computer systems, where power consumption is not as big an issue as speed. The trick to getting the bipolar transistors in an ECL device to respond so quickly is to never let the transistors saturate. Instead, high and low levels are determined by which transistor in a differential amplifier is conducting more. Figure 12.61 shows the internal circuitry of an OR / NOR ECL gate. The high and low logic-level voltages (−0.8 and −17 V, respectively) and the supply voltage (−5.2 V / 0 V) are somewhat unusual and cause problems when interfacing with TTL and CMOS.

Internal circuitry of ECL OR/NOR gate

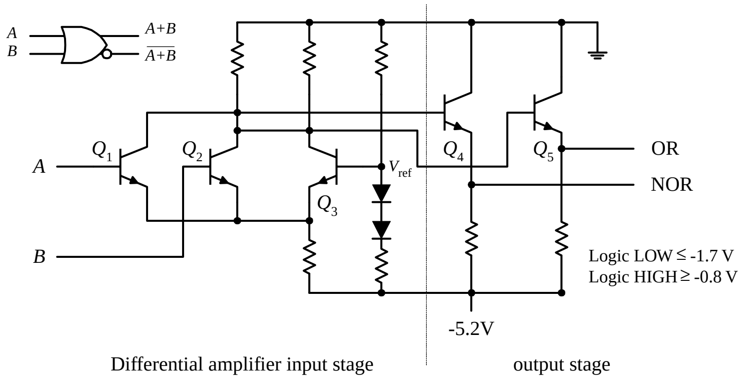


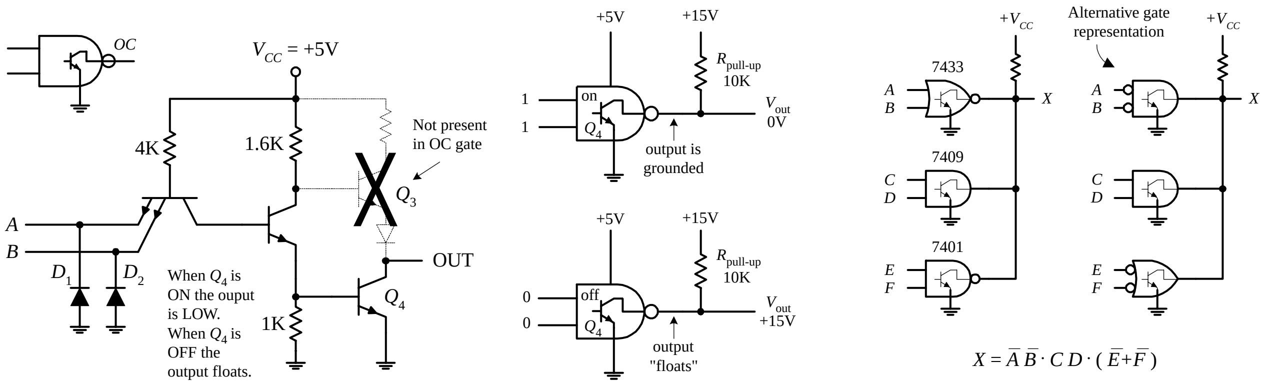
FIGURE 12.61

The OR/NOR gate shown here is composed of a differential amplifier input stage and an output stage. In the differential amplifier stage, a reference voltage is setup at Q_3's base via the voltage divider network (diodes/resistors). This reference voltage determines the threshold between high and low logic levels. When the base of Q_3 is at a more positive potential with respect to the emitter of Q_1 and Q_2, Q_3 conducts. When Q_5 conducts, the OR output goes low. If either input A or B is raised to –0.8 V (high), the base of Q_1 or Q_2 will be at a higher potential than the base of Q_3, and Q_3 will cease conducting, forcing the OR output high. The overall effect of the ECL design prevents transistors from saturating, thereby eliminating charge buildup on the base of the transistors that limits switching speeds.

12.4.7 Logic Gates with Open-Collector Outputs

Among the members of the TTL series there exists a special class of logic gates that have open-collector output stages instead of the traditional totem-pole configuration you saw earlier. (Within the CMOS family, there are similar devices that are said to have open-drain output stages). These devices are not to be confused with the typical logic gates you have seen so far. Logic gates with open-collector outputs have entirely different output characteristics. Figure 12.62 shows a NAND gate with open-collector (OC) output. Notice that the Q_3 transistor is missing in the OC NAND gate. By removing Q_3, the output no longer goes high when A and B logic levels are set to 00, 01, or 10. Instead, the output floats. When A and B logic levels are both high (1), the output is grounded. This means that the OC gate can only sink current, it cannot source current! So how do you get a high output level? You use an external voltage source and a pullup resistor, as shown in the center figure below. Now, when the output floats, the pullup resistor connected to the external voltage source will "pull" the output to the same level as the external voltage source, which in this case is at +15 V. That's right, you don't have to use +5 V. That is one of the primary benefits of using OC gates—you can drive load-requiring voltage levels different from those of the logic circuitry.

FIGURE 12.62

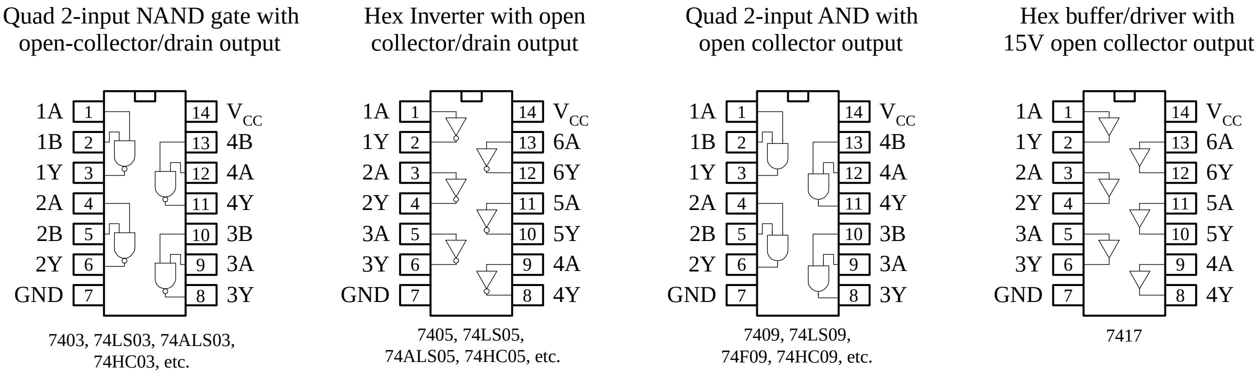


Internal circuitry of an open-collector NAND gate. Note the totem-pole output stage is no longer present.

Using a pull-up resistor with open collector logic gates.

Wired-AND logic: outputs of all three gates must float in order to get a HIGH output at X.

Another important feature of OC gates is their ability to sink large amounts of currents. For example, the 7506 OC inverter buffer / driver IC is capable of sinking 40 mA, which is 10 times the amount of current a standard 7404 inverter can sink. (The 7404 OC buffer / driver has the same sinking ability as the 7406 OC but does not provide any logic function—it simply acts as a buffer stage.) The ability for an OC gate to sink a fairly large current makes it useful for driving relays, motors, LED displays, and other high-current loads. Figure 12.63 shows a number of OC logic gate ICs.



Quad 2-input NAND gate with open-collector/drain output — 7403, 74LS03, 74ALS03, 74HC03, etc.

Hex Inverter with open collector/drain output — 7405, 74LS05, 74ALS05, 74HC05, etc.

Quad 2-input AND with open collector output — 7409, 74LS09, 74F09, 74HC09, etc.

Hex buffer/driver with 15V open collector output — 7417

FIGURE 12.63

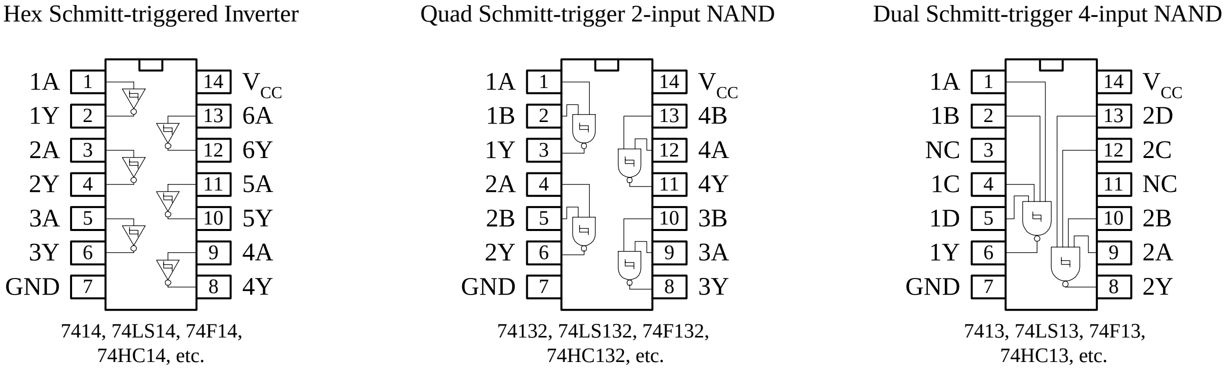
OC gates are also useful in instances where the output from two or more gates or other devices must be tied together. If you were to use standard gates with totem-pole output stages, if one gate were to output a high (+5 V) while another gate were to output a low (0 V), there would be a direct short circuit created, which could cause either or both gates to burn out. By using OC gates, this problem can be avoided.

When working with OC gates, you cannot apply the same Boolean rules you used earlier with the standard gates. Instead, you must use what is called *wired-AND logic*, which amounts to simply ANDing all gates together, as shown in Fig. 12.62. In other words, the outputs of all the gates must float in order to get a high output level.

12.4.8 Schmitt-Triggered Gates

There are special-purpose logic gates that come with Schmitt-triggered inputs. Unlike the conventional logic gates, Schmitt-triggered gates have two input threshold voltages. One threshold voltage is called the *positive threshold voltage* (V_{T^+}), while the other is called the *negative threshold voltage* (V_{T^-}). Example Schmitt-triggered ICs include the quad 7404 inverter, the quad 2-input NAND gate, and the dual 4-input NAND gate shown below.

FIGURE 12.64



Hex Schmitt-triggered Inverter — 7414, 74LS14, 74F14, 74HC14, etc.

Quad Schmitt-trigger 2-input NAND — 74132, 74LS132, 74F132, 74HC132, etc.

Dual Schmitt-trigger 4-input NAND — 7413, 74LS13, 74F13, 74HC13, etc.

To get a sense of how these devices work, let's compare the Schmitt-triggered 7414 inverter gate with a conventional inverter gate, the 7404. With the 7404, to make the output go from high to low or from low to high, the input voltage must fall above or below the single 2.0-V threshold voltage. However, with the 7414, to make the output go from low to high, the input voltage must dip below V_{T^-} (which is +0.9 V for this particular IC); to make the output go from high to low, the input voltage must pop above V_{T^+} (which is +1.7 V for this particular IC). The difference in voltage between V_{T^+} and V_{T^-} is called the *hysteresis voltage* (see Chap. 7 for details). The symbol used to designate a Schmitt trigger is based on the appearance of its transfer function, as shown in the figure below.

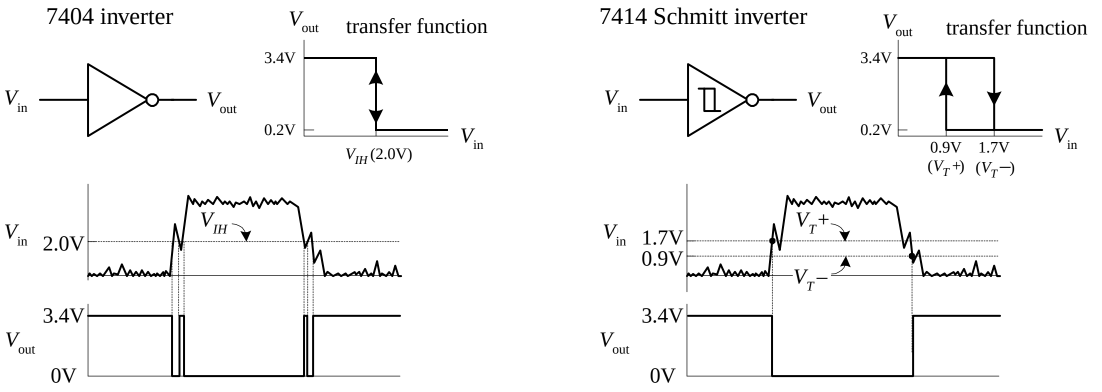


FIGURE 12.65

In terms of applications, Schmitt-triggered devices are quite handy for transforming noisy signals or signals that waver around critical threshold levels into sharply defined, jitter-free output signals. This is illustrated in the lower graphs shown in Fig. 12.65. The conventional 7404 experiences an unwanted output spike resulting from a short-term spike present during low-to-high and high-to-low input voltage transitions. The Schmitt-triggered inverter ignores these spikes because it incorporates hysteresis.

12.4.9 Interfacing Logic Families

Mixing of logic families, in general, should be avoided. Obvious reasons for not mixing include differences in input/output logic levels, supply voltages, and output drive capability that exist among the various families. Another important reason involves differences in speed between the various families; if you mix slow-logic ICs with faster-logic ICs, you can run into timing problems.

There are times, however, when mixing is unavoidable or even desirable. For example, perhaps a desired special-purpose device (e.g., memory, counter, etc.) only exists in CMOS, but the rest of your system consists of TTL. Mixing of families is also common when driving loads. For example, a TTL gate (often with an open-collector output) is frequently used as an interface between a CMOS circuit and an external load, such as a relay or indicator light. A CMOS output, by itself, usually does not provide sufficient drive current to power such loads. I will discuss driving loads in Section 12.10.

Figure 12.66 shows tricks for interfacing various logic families. These tricks take care of input/output incompatibility problems as well as supply voltage incompatibility problems. The tricks, however, do not take care of any timing incompatibility problems that may arise.

Interfacing logic families

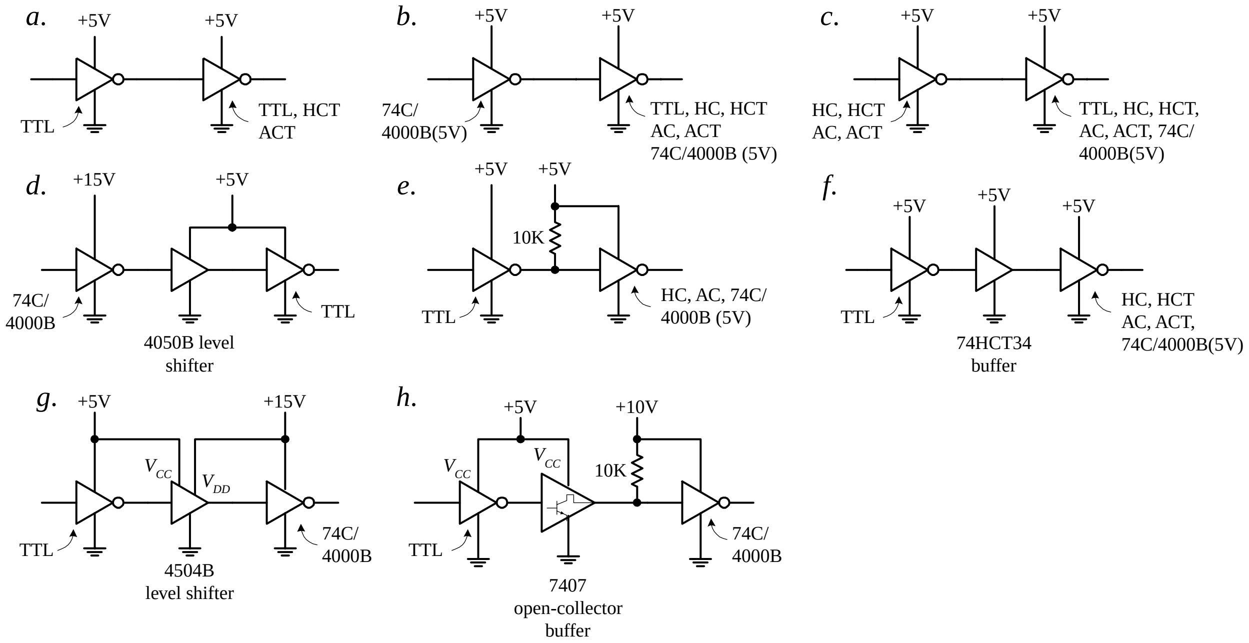


Figure a. TTL can be directly interfaced with itself or with HCT or ACT.

Figure b. CMOS 74C/4000(B) with V_{DD} = +5 V can drive TTL, HC, HCT, AC, or ACT.

Figure c. HC, HCT, AC, and ACT can directly drive TTL, HC, HCT, AC, ACT, and 74C/4000B (5 V).

Figure d. When 74C/4000(B) uses a supply voltage that is higher than +5 V, a level-shifting buffer IC, like the 4050B, can be used. The 4050B is powered by a 5-V supply and can accept 0-V/15-V logic levels at its inputs, while providing corresponding 0-V/5-V logic level outputs. The buffer also provides increased output drive current (4000B has a weak output drive capability when compared to TTL).

Figure e. Recall that the actual high output of a TTL gate is around 3.4 V instead of 5 V. But CMOS (V_{DD} = 5 V) inputs may require from 4.4 (HC) to 4.9 V (4000B) for high input levels. If the CMOS device is of the 74C/4000B series, the actual required high input voltage depends on the supply voltage and is equal to $\frac{2}{3}V_{DD}$. To provide enough voltage to match voltage levels, a pullup resistor is used. The pullup resistor pulls the input to the CMOS gate up to the supply voltage to which the pullup resistor is connected.

Figure f. Another trick for interfacing TTL with CMOS is to simply use a CMOS TTL-compatible gate, like the 74HCT or 74ACT.

Figures g, h. These two figures show different methods for interfacing a TTL gate with a CMOS gate set to a higher supply voltage. In Figure g, a 4504B level-shifting buffer is used. The 4504B requires two supply voltages: a TTL supply (for 0/5 V levels) and a CMOS supply (for 0 to 15 V levels). In figure h, an open-collector buffer and 10-k pullup resistor are used to convert the lower-level TTL output voltages into higher-level CMOS input voltages.

FIGURE 12.66

12.5 Powering and Testing Logic ICs and General Rules of Thumb

12.5.1 Powering Logic ICs

Most TTL and CMOS logic devices will work with 5V ± 0.25V (5 percent) supplies like the ones shown in Fig. 12.67. The battery supplies should be avoided when using certain TTL families like the 74*xx*, 74S, 74AS, and 74F, which dissipate considerably more current than, say, the CMOS 74HC series. Of course, the low-power, low-voltage 74LV, 74LVC, 74LVT, 74ALVC, and 74BCT series, which require from 1.2 to 3.6 V with as low as 2.5 μW / gate power dissipation (for 74BCT), are ideal for small battery-powered applications.

5-V line and battery supplies for digital logic circuits

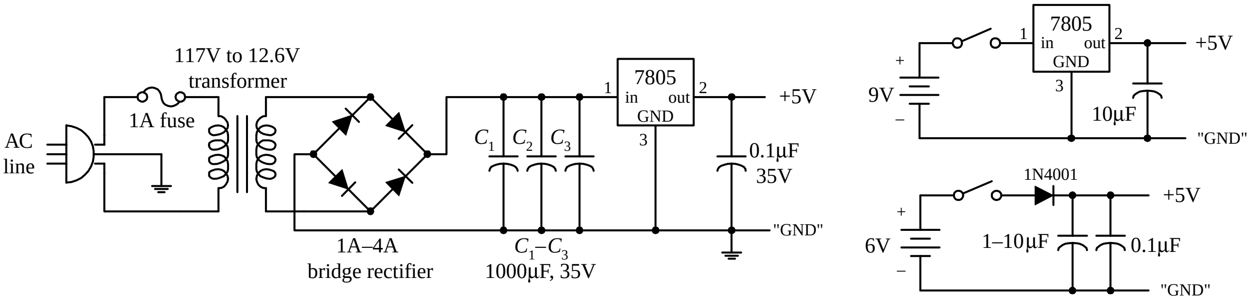


FIGURE 12.67

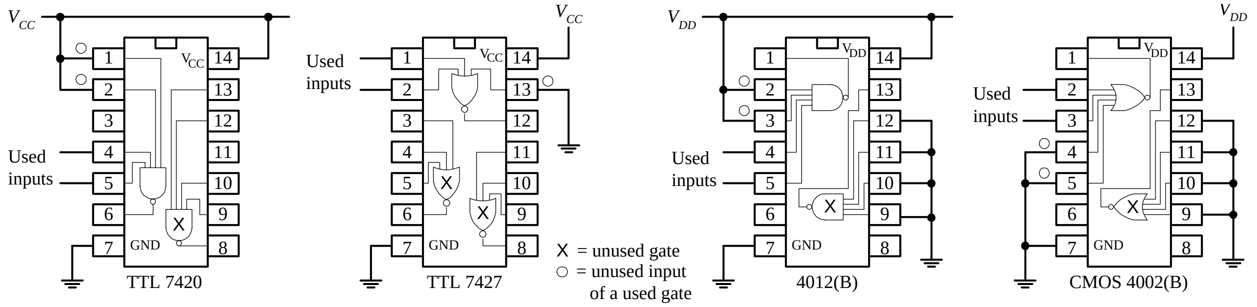
12.5.2 Power Supply Decoupling

When a TTL device makes a low-to-high or a high-to-low level transition, there is an interval of time that the conduction times in the upper and lower totem-pole output transistors overlap. During this interval, a drastic change in power supply current occurs, which results in a sharp, high-frequency current spike within the supply line. If a number of other devices are linked to the same supply, the unwanted spike can cause false triggering of these devices. The spike also can generate unwanted electromagnetic radiation. To avoid unwanted spikes within TTL systems, decoupling capacitors can be used. A decoupling capacitor, typically tantalum, from 0.01 to 1 μF (>5 V), is placed directly across the V_{CC}-to-ground pins of each IC in the system. The capacitors absorb the spikes and keep the V_{CC} level at each IC constant, thus reducing the likelihood of false triggering and generally electromagnetic radiation. Decoupling capacitors should be placed as close to the ICs as possible to keep current spikes local, instead of allowing them to propagate back toward the power supply. You can usually get by with using one decoupling capacitor for every 5 to 10 gates or one for every 5 counter or register ICs.

12.5.3 Unused Inputs

Unused inputs that affect the logical state of a chip should not be allowed to float. Instead, they should be tied high or low, as necessary (floating inputs are liable to pickup external electrical noise, which leads to erratic output behavior). For example, a four-input NAND TTL gate that only uses two inputs should have its two unused inputs held high to maintain proper logic operation. A three-input NOR gate that only uses two inputs should have its unused input held low to maintain proper logic operation. Likewise, the CLEAR and PRESET inputs of a flip-flop should be grounded or tied high, as appropriate.

If there are unused sections within an IC (e.g., unused logic gates within a multigate package), the inputs that link to these sections can be left unconnected for TTL but not for CMOS. When unused inputs are left unconnected in CMOS devices, the inputs may pick up unwanted charge and may reach a voltage level that causes output MOS transistors to conduct simultaneously, resulting in a large internal current spike from the supply (V_{DD}) to ground. The result can lead to excessive supply current drain and IC damage. To avoid this fate, inputs of unused sections of a CMOS IC should be grounded. Figure 12.68 illustrates what to do with unused inputs for TTL and CMOS NAND and NOR ICs.



Connect unused inputs of a used NAND gate HIGH to maintain proper logic function. Don't assume they'll be naturally HIGH for TTL. Connect unused input of a NOR gate LOW to maintain proper logic function. Inputs of unused TTL gates can be left unconnected.

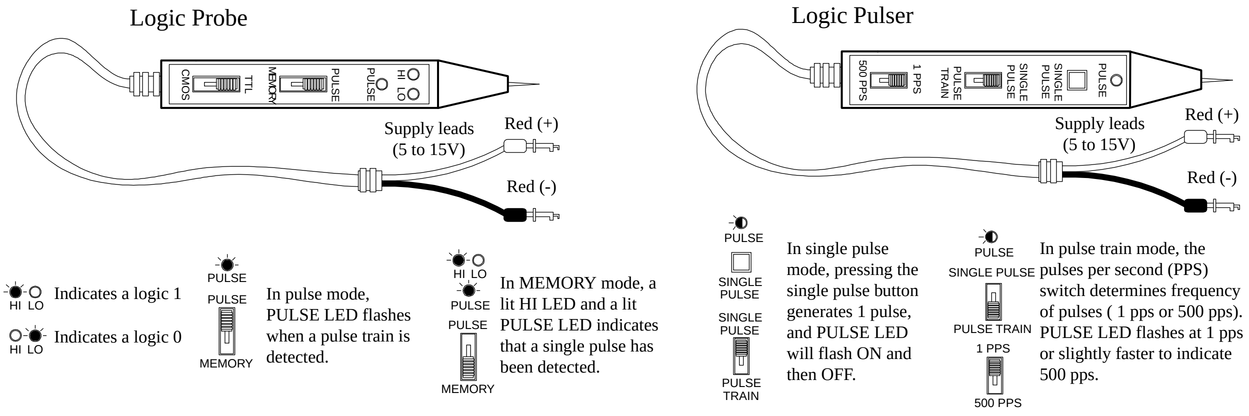
Connect unused inputs of a used NAND gate HIGH to maintain proper logic function. Connect unused inputs of a used NOR gate LOW to maintain proper logic function. Inputs of unused CMOS gates should be grounded.

FIGURE 12.68

As a last note of caution, never drive CMOS inputs when the IC's supply voltage is removed. Doing so can damage the IC's input protection diodes.

12.5.4 Logic Probes and Logic Pulsers

Two simple tools used to test logic ICs and circuits include the test probe and logic pulser, as shown below.



Indicates a logic 1

Indicates a logic 0

In pulse mode, PULSE LED flashes when a pulse train is detected.

In MEMORY mode, a lit HI LED and a lit PULSE LED indicates that a single pulse has been detected.

In single pulse mode, pressing the single pulse button generates 1 pulse, and PULSE LED will flash ON and then OFF.

In pulse train mode, the pulses per second (PPS) switch determines frequency of pulses (1 pps or 500 pps). PULSE LED flashes at 1 pps or slightly faster to indicate 500 pps.

FIGURE 12.69

A typical logic probe comes in a penlike package, with metal probe tip and power supply wires, one red, one black. Red is connected to the positive supply voltage of the digital circuit (V_{CC}), while black is connected to the ground (V_{SS}) of the circuit. To test a logic state within a circuit, the metal tip of the probe is applied. If a high voltage is detected, the probe's high LED lights up; if a low voltage is detected, the probe's low LED turns off. Besides performing simple static tests, logic probes can perform a few simple dynamic tests too, such as detecting a single momentary pulse that is too fast for the human eye to detect or detecting a pulse train, such as a clock signal. To detect a single pulse, the probe's PULSE/MEMORY switch is thrown to the MEMORY position. When a single pulse is detected, the internal memory circuit remembers the single pulse and lights up both the HI LED and

PULSE LED at the same time. To clear the memory to detect a new single-pulse, the PULSE / MEMORY switch is toggled. To detect a pulse train, the PULSE / MEMORY switch is thrown to the PULSE position. When a pulse train is detected, the PULSE LED flashes on and off. Logic probes usually will detect single pulses with widths as narrow as 10 ns and will detect pulse trains with frequencies around 100 MHz. Check the specifications that come with your probe to determine these min / max limits.

A logic pulser allows you to send a single logic pulse or a pulse train through IC and circuits, where the results of the applied pulses can be monitored by a logic probe. Like a logic probe, the pulser comes with similar supply leads. To send a single pulse, the SINGLE-PULSE / PULSE-TRAIN switch is set to SINGLE-PULSE, and then the SINGLE-PULSE button is pressed. To send a pulse train, switch to PULSE-TRAIN mode. With the pulser model shown in Fig. 12.69, you get to select either one pulse per second (1 pps) or 500 pulses per second.

12.6 Sequential Logic

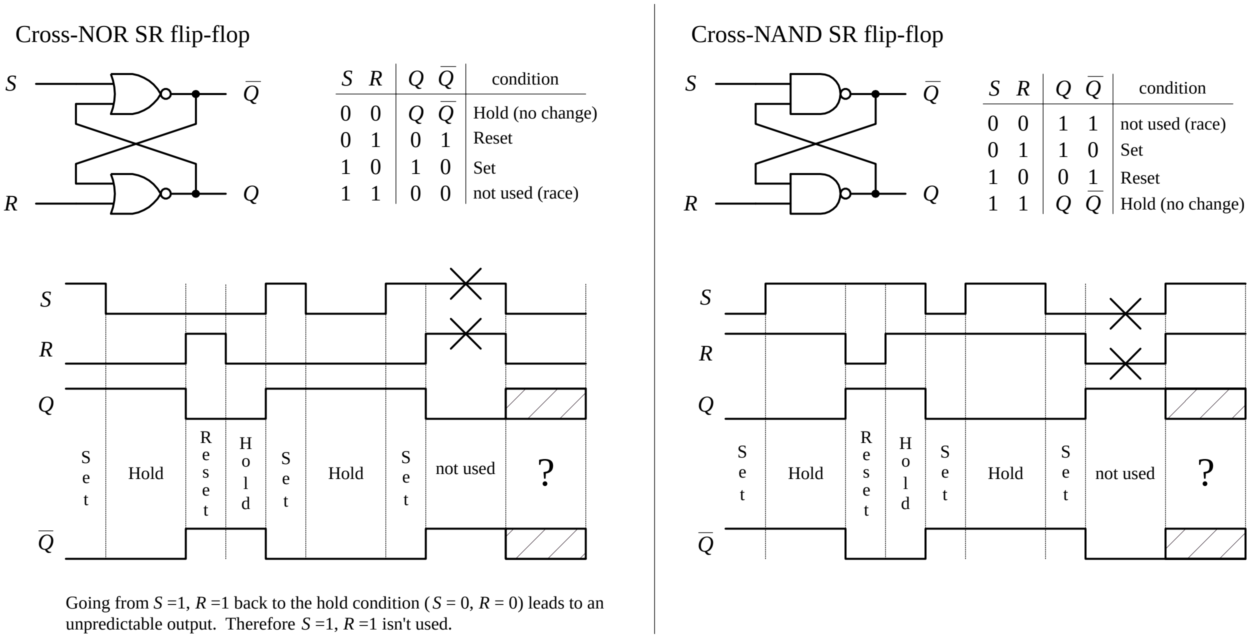
The combinational circuits covered previously (e.g., encoders, decoders, multiplexers, parity generators / checkers, etc.) had the property of input-to-output immediacy. This means that when input data are applied to a combinational circuit, the output responds right away. Now, combinational circuits lack a very important characteristic—they cannot store information. A digital device that cannot store information is not very interesting, practically speaking.

To provide "memory" to circuits, you must create devices that can latch onto data at a desired moment in time. The realm of digital electronics devoted to this subject is referred to as *sequential logic*. This branch of electronics is referred to as *sequential* because for data bits to be stored and retrieved, a series of steps must occur in a particular order. For example, a typical set of steps might involve first sending an enable pulse to a storage device, then loading a group of data bits all at once (parallel load), or perhaps loading a group of data bits in a serial manner—which takes a number of individual steps. At a latter time, the data bits may need to be retrieved by first applying a control pulse to the storage device. A series of other pulses might be required to force the bits out of the storage device.

To push bits through sequential circuits usually requires a clock generator. The clock generator is similar to the human heart. It generates a series of high and low voltages (analogous to a series of high and low pressures as the heart pumps blood) that can set bits into action. The clock also acts as a time base on which all sequential actions can be referenced. Clock generators will be discussed in detail later on. Now, let's take a look at the most elementary of sequential devices, the SR flip-flop.

12.6.1 SR Flip-Flops

The most elementary data-storage circuit is the *SR* (*set-reset*) *flip-flop*, also referred to as a *transparent latch*. There are two basic kinds of SR flip-flops, the cross-NOR SR flip-flop and the cross-AND SR flip-flop.



Cross-NOR SR flip-flop

S	R	Q	$\overline{Q}$	condition
0	0	Q	$\overline{Q}$	Hold (no change)
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	not used (race)

Cross-NAND SR flip-flop

S	R	Q	$\overline{Q}$	condition
0	0	1	1	not used (race)
0	1	1	0	Set
1	0	0	1	Reset
1	1	Q	$\overline{Q}$	Hold (no change)

Going from S =1, R =1 back to the hold condition (S = 0, R = 0) leads to an unpredictable output. Therefore S =1, R =1 isn't used.

FIGURE 12.70

First, let's consider the cross-NOR SR flip-flop shown above. At first it appears that figuring out what the cross-NOR SR flip-flop does given only two input voltages is impossible, since each of the NOR gates' inputs depend on the outputs—and what are the outputs anyway? (For now, pretend that Q and $\overline{Q}$ are not complements but separate variables—you could call them X and Y if you like.) Well, first of all, you know that a NOR gate will output a high (logic 1) only if both inputs are low (logic 0). From this you can deduce that if $S = 1$ and $R = 0$, Q must be 1 and $\overline{Q}$ must be 0, regardless of the outputs—this is called the *set* condition. Likewise, by similar argument, we can deduce that if $S = 0$ and $R = 1$, Q must be 0 and $\overline{Q}$ must be 1—this is called the *reset* condition.

But now, what about $R = 0$, $S = 0$? Can you predict the outputs only given input levels? No! It is impossible to predict the outputs because the outputs are essential for predicting the outputs—it is a "catch-22." However, if you know the states of the outputs beforehand, you can figure things out. For example, if you first set the flip-flop ($S = 1$, $R = 0$, $Q = 1$, $\overline{Q} = 0$) and then apply $S = 0$, $R = 0$, the flip-flop would remain set (upper gate: $S = 0$, $Q = 1 \rightarrow \overline{Q} = 0$; lower gate: $R = 0$, $\overline{Q} = 0 \rightarrow Q = 1$). Likewise, if you start out in reset mode ($S = 0$, $R = 1$, $Q = 0$, $\overline{Q} = 0$) and then apply $S = 0$, $R = 0$, the flip-flop remains in reset mode (upper gate: $S = 0$, $Q = 0 \rightarrow \overline{Q} = 1$; lower gate: $R = 0$, $\overline{Q} = 1 \rightarrow Q = 0$). In other words, the flip-flop remembers, or latches onto, the previous output state even when both inputs go low (0)—this is referred to as the *hold* condition.

The last choice you have is $S = 1$, $R = 1$. Here, it is easy to predict what will happen because you know that as long as there is at least one high (1) applied to the input to the NOR gate, the output will always be 0. Therefore, $Q = 0$ and $\overline{Q} = 0$. Now, there are two fundamental problems with the $S = 1$, $R = 1$ state. First, why would you want to set and reset at the same time? Second, when you return to the hold condition from $S = 1$, $R = 1$, you get an unpredictable result, unless you know which input returned low last. Why? When the inputs are brought back to the hold position ($R = 0$, $S = 0$, $Q = 0$,

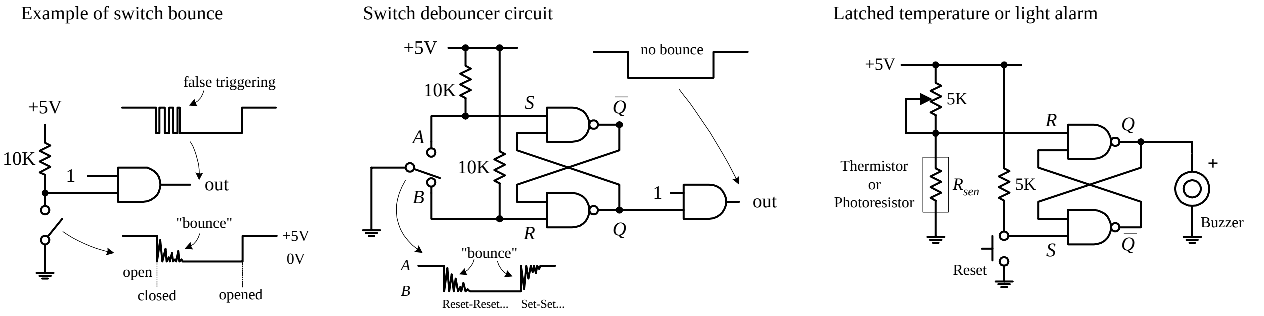
$\overline{Q} = 0$), both NOR gates will want to be 1 (they want to be held). But let's say one of the NOR gate's outputs changes to 1 a fraction of a second before the other. In this case, the slower flip-flop will not output a 1 as planned but will instead output 0. This is a classic example of a race condition, where the slower gate loses. But which flip-flop is the slower one? This unstable, unpredictable state cannot be avoided and is simply not used.

The cross-NAND SR flip-flop provides the same basic function as the NOR SR flip-flop, but there is a fundamental difference. Its hold and indeterminate states are reversed. This occurs because unlike the NOR gate, which only outputs a low when both its inputs are the same, the NAND gate only outputs a high when both its inputs are the same. This means that the hold condition for the cross-NAND SR flip-flop is $S = 1$, $R = 1$, while the indeterminate condition is $S = 0$, $R = 0$.

Here are two simple applications for SR flip-flops.

Switch Debouncer

Say you want to use the far-left switch / pullup resistor circuit (Fig. 12.71) to drive an AND gate's input high or low (the other input is fixed high). When the switch is open, the AND gate should receive a high. When the switch is closed, the gate should receive a low. That's what should happen, but that's not what actually happens. Why? Answer: Switch bounce. When a switch is closed, the metal contacts bounce a number of times before coming to rest due to inherent springlike characteristics of the contacts. Though the bouncing typically lasts no more than 50 ms, the results can lead to unwanted false triggering, as shown in the far left circuit below. A simple way to get rid of switch bounce is to use the switch debouncer circuit, shown at center. This circuit simply uses an SR flip-flop to store the initial switch contact voltage while ignoring all trailing bounces. In this circuit, when the switch is thrown from the *B* to *A* position, the flip-flop is set. As the switch bounces alternately high and low, the *Q* output remains high because when the switch contact bounces away from *A*, the *S* input receives a low (*R* is low too), but that's just a hold condition—the output stays the same. The same debouncing feature occurs when the switch is thrown from position *A* to *B*.


Example of switch bounce

Switch debouncer circuit

Latched temperature or light alarm

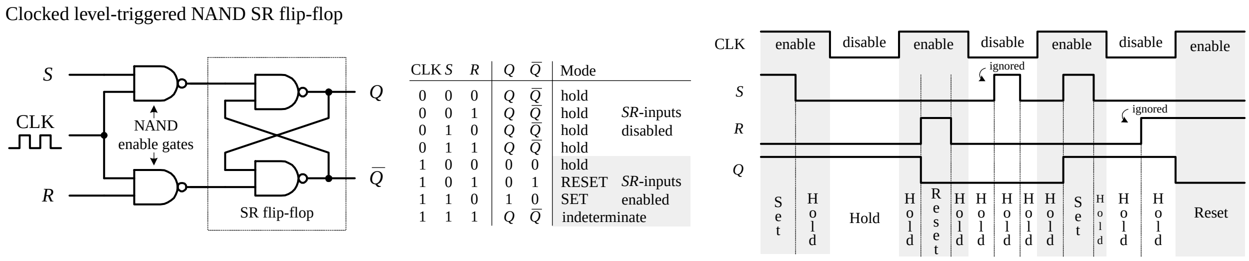
FIGURE 12.71

Latched Temperature or Light Alarm

This simple circuit (Fig. 12.71) uses an SR flip-flop to sound a buzzer alarm when the temperature (when using a thermistor) or the light intensity (when using a photoresistor) reaches a critical level. When the temp / light increases, the resistance of the thermistor / photoresistor decreases, and the *R* input voltage goes down. When the *R* input voltage goes below the high threshold level of the NAND gate, the flip-flop is set, and the alarm is sounded. The alarm will continue to sound until the RESET switch is pressed and the temp / light level has gone below the critical triggering level. The pot is used to adjust this level.

Level-Triggered SR Flip-Flop (The Beginning of Clocked Flip-Flops)

Now it would be nice to make an SR flip-flop synchronous—meaning making the *S* and *R* inputs either enabled or disabled by a control pulse, such as a clock. Only when the clock pulse arrives are the inputs sampled. Flip-flops that respond in this manner are referred to as *synchronous* or *clocked flip-flops* (as opposed to the preceding asynchronous flip-flops). To make the preceding SR flip-flop into a synchronous or clocked device, simply attach enable gates to the inputs of the flip-flop, as shown in Fig. 12.72. (Here, the cross-NAND arrangement is used, though a cross-NOR arrangement also can be used.) Only when the clock is high are the *S* and *R* inputs enabled. When the clock is low, the inputs are disabled, and the flip-flop is placed in hold mode. The truth table and timing diagram below help illustrate how this device works.



Clocked level-triggered NAND SR flip-flop

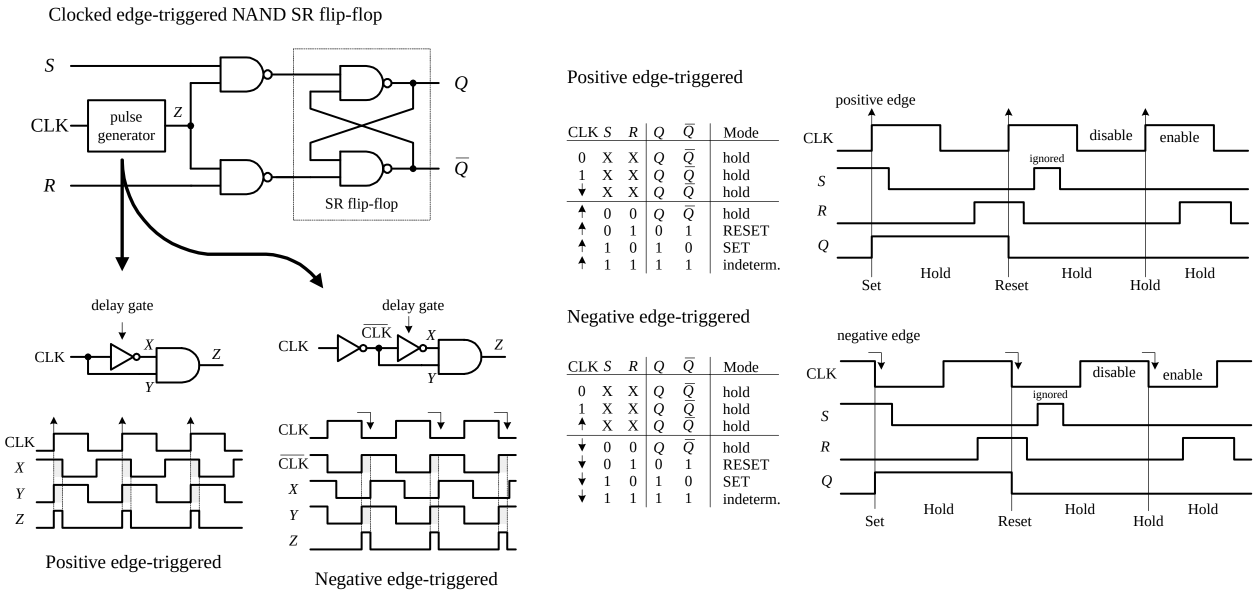
CLK	S	R	Q	$\overline{Q}$	Mode	
0	0	0	Q	$\overline{Q}$	hold	
0	0	1	Q	$\overline{Q}$	hold	*SR*-inputs
0	1	0	Q	$\overline{Q}$	hold	disabled
0	1	1	Q	$\overline{Q}$	hold	
1	0	0	0	0	hold	
1	0	1	0	1	RESET	*SR*-inputs
1	1	0	1	0	SET	enabled
1	1	1	Q	$\overline{Q}$	indeterminate	

FIGURE 12.72

Edge-Triggered SR Flip-Flops

Now there is an annoying feature with the last level-triggered flip-flop; its *S* and *R* inputs have to be held at the desired input condition (set, reset, no change) for the entire time that the clock signal is enabling the flip-flop. With a slight alteration, however, you can make the level-triggered flip-flop more flexible (in terms of timing control) by turning it into an edge-triggered flip-flop. An edge-triggered flip-flop samples the inputs only during either a positive or negative clock edge (↑ = positive edge, ↓ = negative edge). Any changes that occur before or after the clock edge are ignored—the flip-flop will be placed in hold mode. To make an edge-triggered flip-flop, introduce either a positive or a negative level-triggered clock pulse generator network into the previous level-triggered flip-flop, as shown in Fig. 12.73.

In a positive edge-triggered generator circuit, a NOT gate with propagation delay is added. Since the clock signal is delayed through the inverter, the output of the AND gate will not provide a low (as would be the case without a propagation delay) but will provide a pulse that begins at the positive edge of the clock signal and lasts for a duration equal to the propagation delay of the NOT gate. It is this pulse that is used to clock the flip-flop. Within the negative edge-triggered



Positive edge-triggered

CLK	S	R	Q	$\overline{Q}$	Mode
0	X	X	Q	$\overline{Q}$	hold
1	X	X	Q	$\overline{Q}$	hold
↓	X	X	Q	$\overline{Q}$	hold
↑	0	0	Q	$\overline{Q}$	hold
↑	0	1	0	1	RESET
↑	1	0	1	0	SET
↑	1	1	1	1	indeterm.

Negative edge-triggered

CLK	S	R	Q	$\overline{Q}$	Mode
0	X	X	Q	$\overline{Q}$	hold
1	X	X	Q	$\overline{Q}$	hold
↑	X	X	Q	$\overline{Q}$	hold
↓	0	0	Q	$\overline{Q}$	hold
↓	0	1	0	1	RESET
↓	1	0	1	0	SET
↓	1	1	1	1	indeterm.

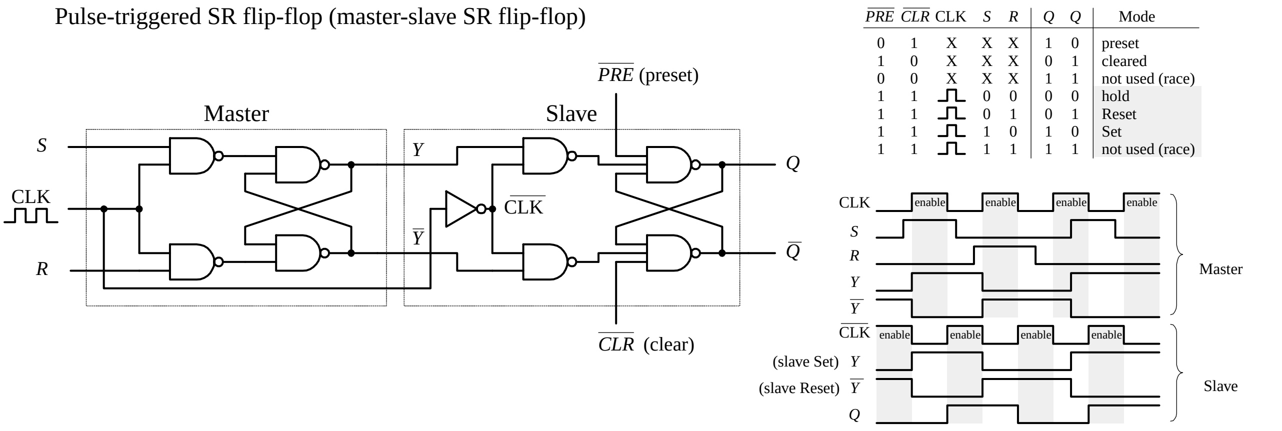
FIGURE 12.73

generator network, the clock signal is first inverted and then applied through the same NOT/AND network. The pulse begins at the negative edge of the clock and lasts for a duration equal to the propagation delay of the NOT gate. The propagation delay is typically so small (in nanoseconds) that the pulse is essentially an "edge."

Pulse-Triggered SR Flip-Flops (Master-Slave Flip-Flops)

A pulse-triggered SR flip-flop is a level-clocked flip-flop; however, for any change in output to occur, both the high and low levels of the clock must rise and fall. Pulse-triggered flip-flops are also called *master-slave flip-flop;* the master accepts the initial inputs and then "whips" the slave with its output when the negative clock edge arrives. Another analogy often used is to say that during the positive edge, the master gets cocked (like a gun), and during the negative clock edge, the slave gets triggered. Figure 12.74 shows a simplified pulse-triggered cross-NAND SR flip-flop.

FIGURE 12.74



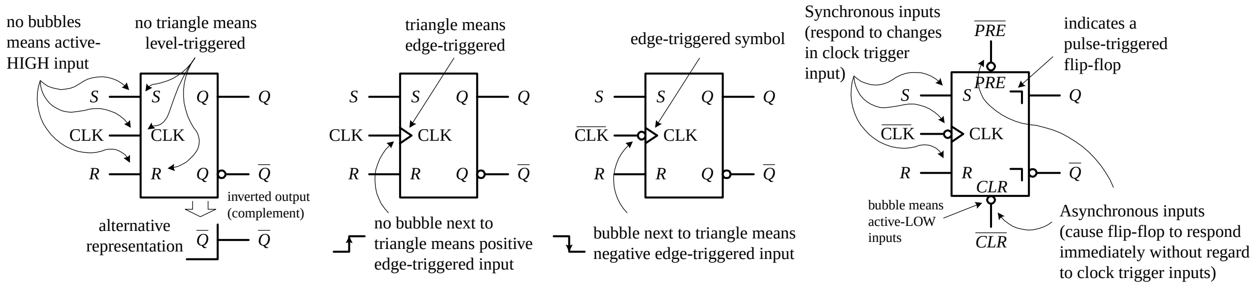
$\overline{PRE}$	$\overline{CLR}$	CLK	S	R	Q	Q	Mode
0	1	X	X	X	1	0	preset
1	0	X	X	X	0	1	cleared
0	0	X	X	X	1	1	not used (race)
1	1	⎍	0	0	0	0	hold
1	1	⎍	0	1	0	1	Reset
1	1	⎍	1	0	1	0	Set
1	1	⎍	1	1	1	1	not used (race)

The master is simply a clocked SR flip-flop that is enabled during the high clock pulse and outputs Y and $\overline{Y}$ (either set, reset, or no change). The slave is similar to the master, but it gets enabled only during the negative clock pulse (due to the inverter). The moment the slave is enabled, it uses the Y and $\overline{Y}$ outputs of the master as inputs and then outputs the final result. Notice the preset ($\overline{PRE}$) and clear ($\overline{CLR}$) inputs. These are called *asynchronous inputs*. Unlike the synchronous inputs, S and R, the asynchronous input disregard the clock and either clear (also called *asynchronous reset*) or preset (also called *asynchronous set*) the flip-flop. When $\overline{CLR}$ is high and $\overline{PRE}$ is low, you get asynchronous reset, $Q = 1$, $\overline{Q} = 0$, regardless of the CLK, S, and R inputs. These active-low inputs are therefore normally pulled high to make them inactive. As you will see later when I discuss flip-flop applications, the ability to apply asynchronous set and resets is often used to clear entire registers that consist of an array of flip-flops.

General Rules for Deciphering Flip-Flop Logic Symbols

Now, typically, you do not have to worry about constructing flip-flops from scratch—instead, you buy flip-flop ICs. Likewise, you do not have to worry about complex logic gate schematics—instead, you use symbolic representations like the ones shown below. Although the symbols below apply to SR flip-flops, the basic rules that are outlined can be applied to the D and JK flip-flops, which will be discussed in following sections.

Symbolic representation of level-triggered, edge-triggered and pulse-triggered flip-flops

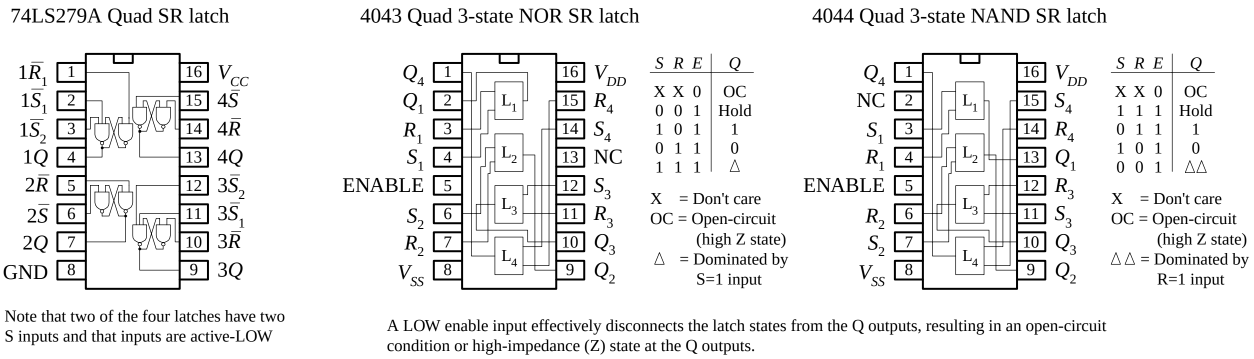


no bubbles means active-HIGH input
no triangle means level-triggered
S
CLK
R
S
CLK
R
Q
Q̄
Q
Q̄
inverted output (complement)
alternative representation
Q̄
Q̄
triangle means edge-triggered
S
CLK
R
S
CLK
R
Q
Q̄
Q
Q̄
no bubble next to triangle means positive edge-triggered input
edge-triggered symbol
S
CLK̄
R
S
CLK
R
Q
Q̄
Q
Q̄
bubble next to triangle means negative edge-triggered input
Synchronous inputs (respond to changes in clock trigger input)
PRE̅
indicates a pulse-triggered flip-flop
S
CLK̄
R
S
PRE
CLK
R
CLR
Q
Q̄
bubble means active-LOW inputs
CLR̄
Asynchronous inputs (cause flip-flop to respond immediately without regard to clock trigger inputs)

FIGURE 12.75

12.6.2 SR Flip-Flop (Latch) ICs

Figure 12.76 shows a few sample SR flip-flop (latch) ICs. The 74LS279A contains four independent SR latches (note that two of the latches have an extra set input). This IC is commonly used in switch debouncers. The 4043 contains four three-state cross-coupled NOR SR latches. Each latch has individual set and reset inputs, as well as separate Q outputs. The three-state feature is an extra bonus, which allows you to effectively disconnect all Q outputs, making it appear that the outputs are open circuits (high impedance, or high Z). This three-state feature is often used in applications where a number of devices must share a common data bus. When the output data from one latch are applied to the bus, the outputs of other latches (or other devices) are disconnected via the high-Z condition. The 4044 is similar to the 4043 but contains four three-state cross-coupled NAND RS latches.



4043:

S	*R*	*E*	*Q*
X	X	0	OC
0	0	1	Hold
1	0	1	1
0	1	1	0
1	1	1	△

X = Don't care
OC = Open-circuit (high Z state)
△ = Dominated by S=1 input

4044:

S	*R*	*E*	*Q*
X	X	0	OC
1	1	1	Hold
0	1	1	1
1	0	1	0
0	0	1	△△

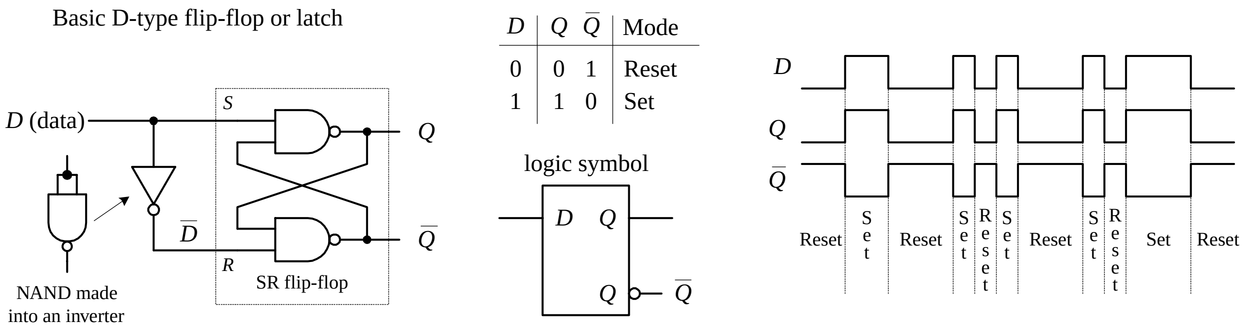
X = Don't care
OC = Open-circuit (high Z state)
△△ = Dominated by R=1 input

A LOW enable input effectively disconnects the latch states from the Q outputs, resulting in an open-circuit condition or high-impedance (Z) state at the Q outputs.

FIGURE 12.76

12.6.3 D Flip-Flops

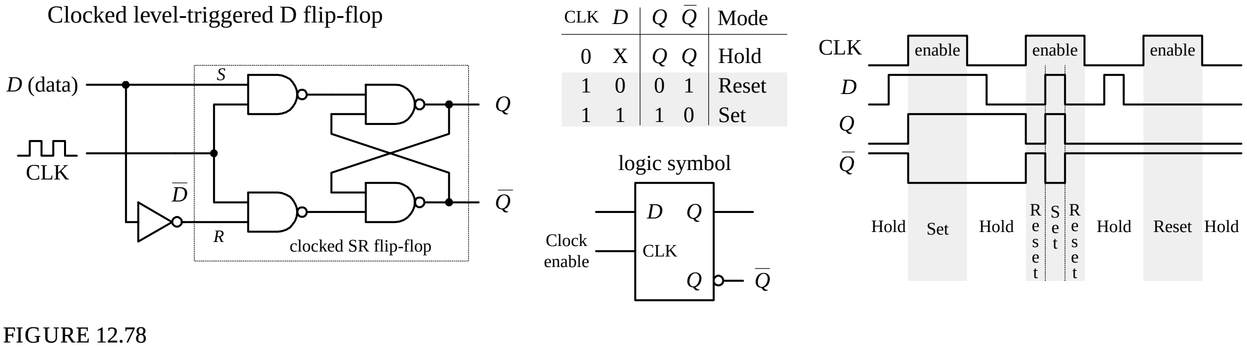
A D-type flip-flop (data flip-flop) is a single input device. It is basically an SR flip-flop, where *S* is replaced with *D* and *R* is replaced $\overline{D}$ (inverted *D*)—the inverted input is tapped from the *D* input through an inverter to the *R* input, as shown below. The inverter ensures that the indeterminate condition (race, or not used state, $S = 1$, $R = 1$) never occurs. At the same time, the inverter eliminates the hold condition so that you are left with only set ($D = 1$) and reset ($D = 0$) conditions. The circuit below represents a level-triggered D-type flip-flop.



D	*Q*	$\overline{Q}$	Mode
0	0	1	Reset
1	1	0	Set

FIGURE 12.77

To create a clocked D-type level-triggered flip-flop, first start with the clocked level-triggered SR flip-flop and throw in the inverter, as shown in Fig. 12.78.

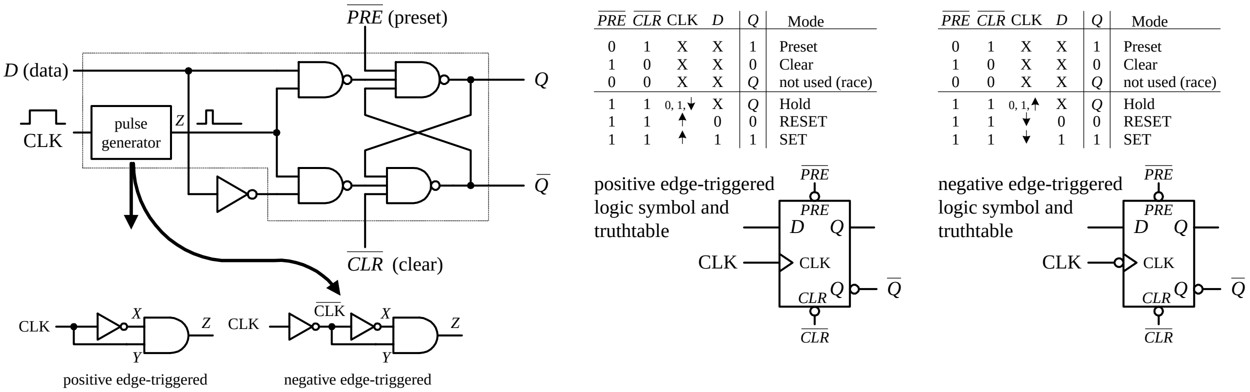


CLK	*D*	*Q*	$\overline{Q}$	Mode
0	X	*Q*	$\overline{Q}$	Hold
1	0	0	1	Reset
1	1	1	0	Set

FIGURE 12.78

To create a clocked, edge-triggered D-type flip-flop, take a clocked edge-triggered SR flip-flop and add an inverter, as shown in Fig. 12.79.

Edge-triggered *D* flip-flop



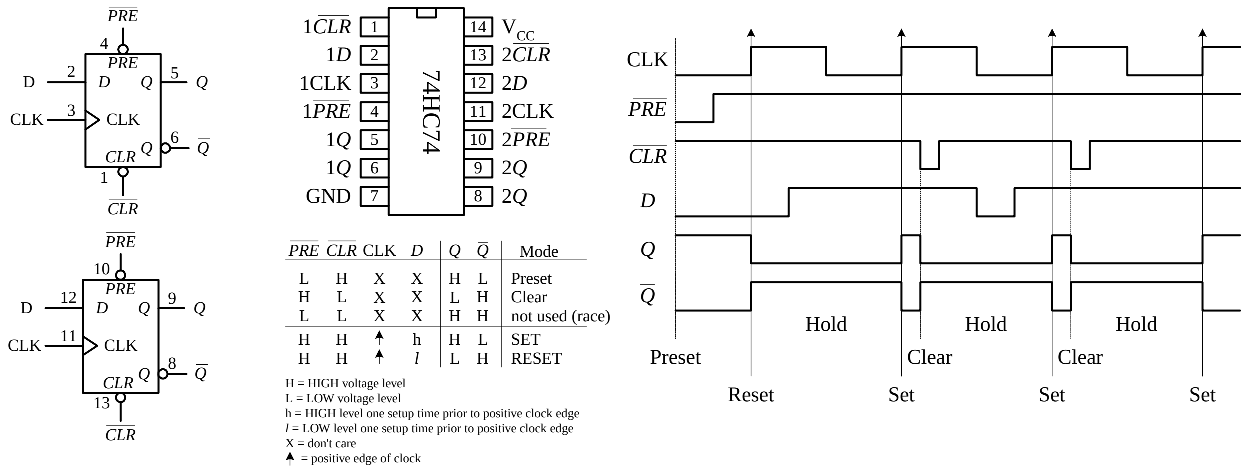
$\overline{PRE}$	$\overline{CLR}$	CLK	*D*	*Q*	Mode
0	1	X	X	1	Preset
1	0	X	X	0	Clear
0	0	X	X	*Q*	not used (race)
1	1	0, 1, ↓	X	*Q*	Hold
1	1	↑	0	0	RESET
1	1	↑	1	1	SET

$\overline{PRE}$	$\overline{CLR}$	CLK	*D*	*Q*	Mode
0	1	X	X	1	Preset
1	0	X	X	0	Clear
0	0	X	X	*Q*	not used (race)
1	1	0, 1, ↑	X	*Q*	Hold
1	1	↓	0	0	RESET
1	1	↓	1	1	SET

FIGURE 12.79

Here's a popular edge-trigger D-type flip-flop IC, the 7474 (e.g., 74HC74, etc.). It contains two D-type positive edge-triggered flip-flops with asynchronous preset and clear inputs.

74HC74 Dual D-type positive edge-triggered flip-flop with preset and clear



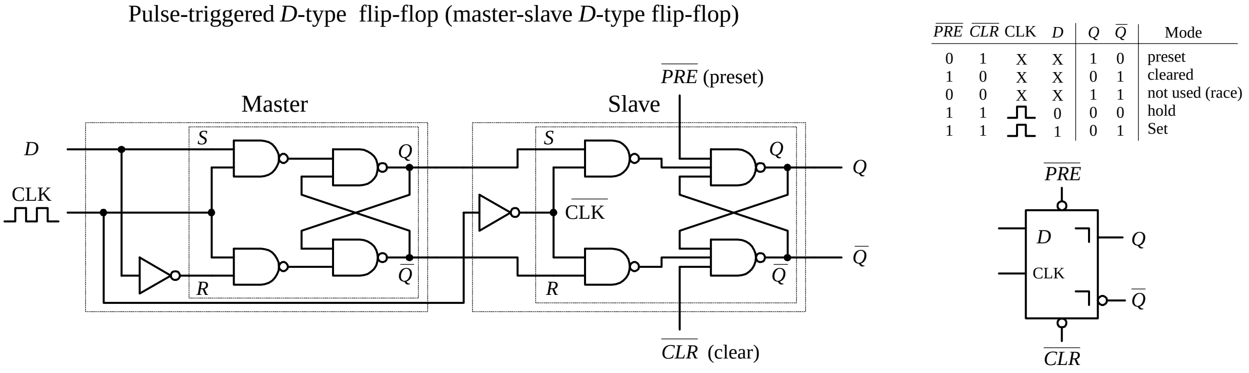
$\overline{PRE}$	$\overline{CLR}$	CLK	*D*	*Q*	$\overline{Q}$	Mode
L	H	X	X	H	L	Preset
H	L	X	X	L	H	Clear
L	L	X	X	H	H	not used (race)
H	H	↑	h	H	L	SET
H	H	↑	*l*	L	H	RESET

H = HIGH voltage level
L = LOW voltage level
h = HIGH level one setup time prior to positive clock edge
l = LOW level one setup time prior to positive clock edge
X = don't care
↑ = positive edge of clock

FIGURE 12.80

Note the lowercase letters *l* and *h* in the truth table in this figure. The *h* is similar to the *H* for a high voltage level, and the *l* is similar to the *L* for low voltage level; however, there is an additional condition that must be met for the flip-flop's output to do what the truth table indicates. The additional condition is that the *D* input must be fixed high (or low) in duration for at least one *setup time* (t_s) before the positive clock edge. This condition stems from the real-life propagation delays present in flip-flop ICs; if you try to make the flip-flop switch states too fast (do not give it time to move electrons around), you can end up with inaccurate output readings. For the 7474, the setup time is 20 ns. Therefore, when using this IC, you must not apply input pulses that are within the 20-ns limit. Other flip-flops will have different setup times, so you will have to check the manufacturer's data sheets. I will discuss setup time and some other flip-flop timing parameters in greater detail at the end of this section.

D-type flip-flops are sometimes found in the pulse-triggered (master-slave) variety. Recall that a pulse-triggered flip-flop requires a complete clock pulse before the outputs will reflect what is applied at the input(s) (in this case the *D* input). The figure below shows the basic structure of a pulse-triggered D flip-flop. It is almost exactly like the pulse-triggered SR flip-flop, except for the inverter addition to the master's input.



$\overline{PRE}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$	Mode
0	1	X	X	1	0	preset
1	0	X	X	0	1	cleared
0	0	X	X	1	1	not used (race)
1	1	⎍	0	0	0	hold
1	1	⎍	1	0	1	Set

FIGURE 12.81

12.6.4 A Few Simple D-Type Flip-Flop Applications

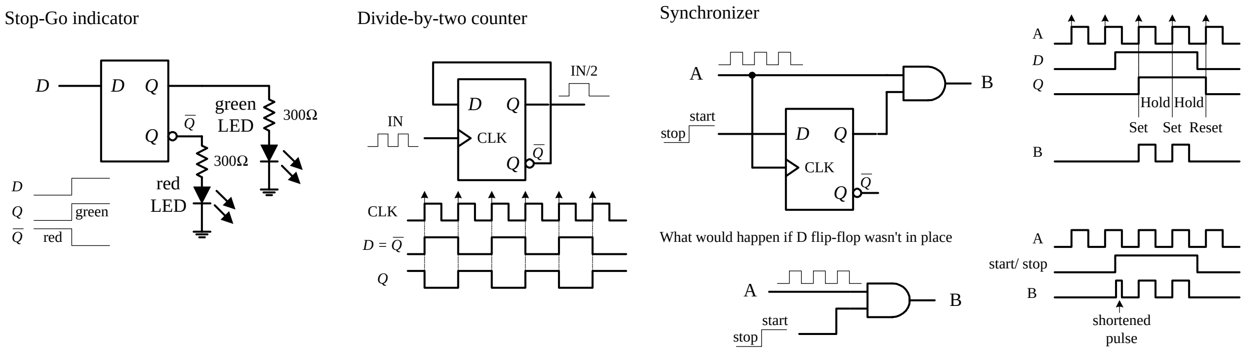


FIGURE 12.82

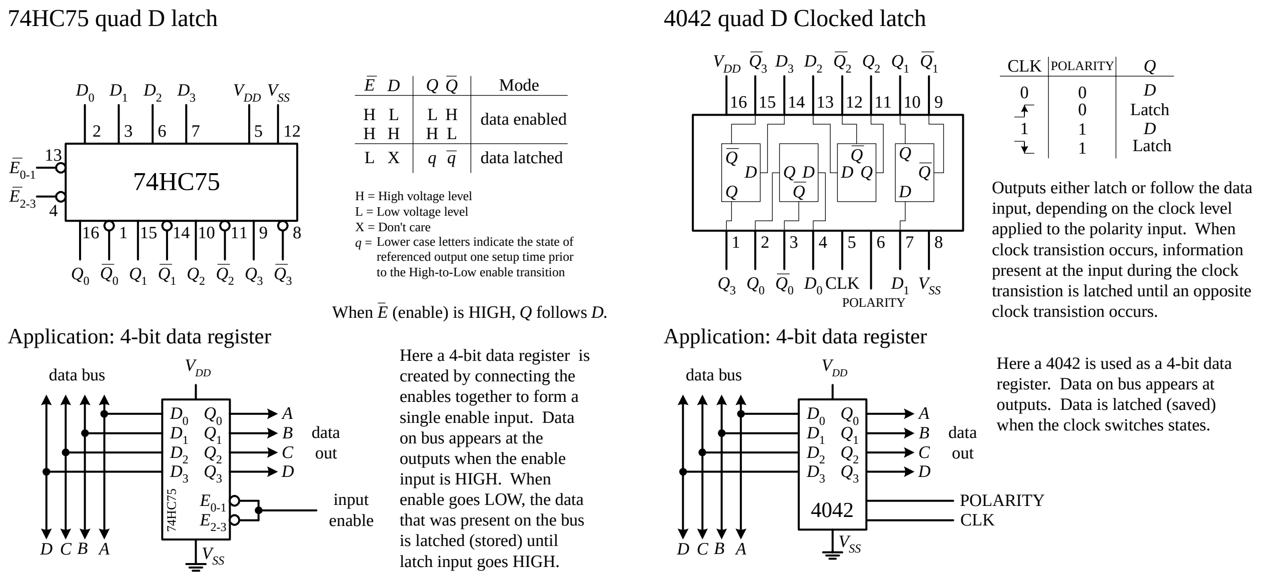
In the stop-go indicator circuit, a simple level-triggered D-type flip-flop is used to turn on a red LED when its *D* input is low (reset) and turn on a green LED when the *D* input is high (set). Only one LED can be turned on at a time.

The divide-by-two counter uses a positive edge-triggered D-type flip-flop to divide an applied signal's frequency by two. The explanation of how this works is simple: The positive edge-triggered feature does not care about negative edges. You can figure out the rest.

A synchronizer is used when you want to use an external asynchronous control signal (perhaps generated by a switch or other input device) to control some action within a synchronous system. The synchronizer provides a means of keeping the phase of the action generated by the control signal in synch with the phase of the synchronous system. For example, say you want an asynchronous control signal to control the number of clock pulses that get from point *A* to point *B* within a synchronous system. You might try using a simple enable gate, as shown below the synchronizer circuit in the figure above. However, because the external control signal is not synchronous (in phase) with the clock, when you apply the external control signal, you may shorten the first or last output pulse, as shown in the lower timing diagram. Certain applications do not like shortened clock pulses and will not function properly. To avoid shortened pulses, throw in an edge-triggered D-type flip-flop to create a synchronizer. The flip-flop's CLK input is tapped off the input clock line, its *D* input receives the external control signal, and its *Q* output is connected to the AND gate's enable input. With this arrangement, there will never be shortened clock pulses because the *Q* output of the flip-flop will not supply enable pulses to the AND gate that are out of phase with the input clock signal. This is due to the fact that after the flip-flop's CLK input receives a positive clock edge, the flip-flop ignores any input changes applied to the *D* input until the next positive clock edge.

12.6.5 Quad and Octal D Flip-Flops

Most frequently you will find a number of D flip-flops or D latches grouped together within a single IC. For example, the 74HC75, shown below, contains four transparent D latches. Latches 0 and 1 share a common active-low enable E_0–E_1, while latches 2 and 3 share a common active-low enable E_2–E_3. From the function table, each *Q* output follows each *D* input as long as the corresponding enable line is high. When the enable line goes low, the *Q* output will become latched to the value that *D* was one setup time prior to the high-to-low enable transition. The 4042 is another quad D-type latch—an explanation of how it works is provided in the figure below. D-type latches are commonly used as data registers in bus-oriented systems; the figure below explains the details.



74HC75 quad D latch

$\bar{E}$	D	Q	$\bar{Q}$	Mode
H	L	L	H	data enabled
H	H	H	L	
L	X	q	$\bar{q}$	data latched

H = High voltage level
L = Low voltage level
X = Don't care
q = Lower case letters indicate the state of referenced output one setup time prior to the High-to-Low enable transition

When $\bar{E}$ (enable) is HIGH, *Q* follows *D*.

Application: 4-bit data register

Here a 4-bit data register is created by connecting the enables together to form a single enable input. Data on bus appears at the outputs when the enable input is HIGH. When enable goes LOW, the data that was present on the bus is latched (stored) until latch input goes HIGH.

4042 quad D Clocked latch

CLK	POLARITY	Q
0	0	D
↑	0	Latch
1	1	D
↓	1	Latch

Outputs either latch or follow the data input, depending on the clock level applied to the polarity input. When clock transistion occurs, information present at the input during the clock transistion is latched until an opposite clock transistion occurs.

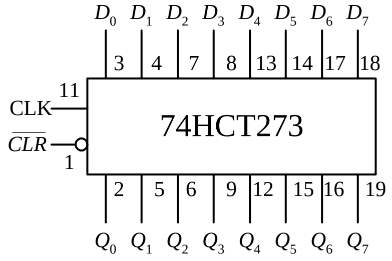
Application: 4-bit data register

Here a 4042 is used as a 4-bit data register. Data on bus appears at outputs. Data is latched (saved) when the clock switches states.

FIGURE 12.83

D flip-flops also come in octal form—eight flip-flops per IC. These devices are frequently used as 8-bit data registers within microprocessor systems, where devices share 8-bit or 2 × 8 = 16-bit data or address buses. An example of an octal D-type flip-flop is the 74HCT273 shown in Fig. 12.84. All D flip-flops within the 74HCT273 share a common positive edge-triggered clock input and a common active-low clear input. When the clock input receives a positive edge, data bits applied to D_0 through D_7 are stored in the eight flip-flops and appear at the outputs Q_0 through Q_7. To clear all flip-flops, the clear input is pulsed low. I will talk more about octal flip-flops and other bus-oriented devices later.

74HCT273 octal edge-triggered D-type flip-flop with Clear


D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
3 4 7 8 13 14 17 18
11
CLK
74HCT273
$\overline{CLR}$
1
2 5 6 9 12 15 16 19
Q_0 Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7

$\overline{CLR}$	CLK	D_n	Q_n	Mode
L	X	X	L	Clear
H	↑	h	H	Set
H	↑	*l*	L	Reset

V_{CC} = pin 20
GND = pin 10

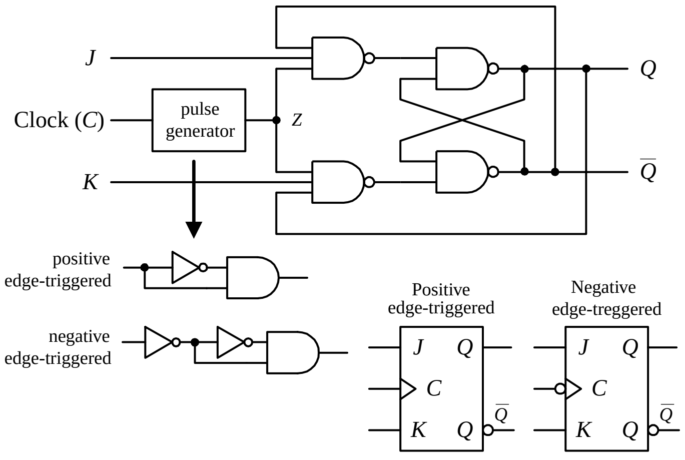
H = High voltage level
L = Low voltage level
h = High voltage level one setup time prior to the low-to-high clock transition
l = Low voltage level one setup time prior to the low-to-high clock transition
X = Don't care
↑ = Low-to-high clock transition

FIGURE 12.84

12.6.6 JK Flip-Flops

Finally, we come to the last of the flip-flops, the JK flip-flop. A JK flip-flop resembles an SR flip-flop, where J acts like S and K acts like R. Likewise, it has a set mode ($J = 1$, $K = 0$), a reset mode ($J = 0$, $K = 1$), and a hold mode ($J = 0$, $K = 0$). However, unlike the SR flip-flop, which has an indeterminate mode when $S = 1$, $R = 1$, the JK flip-flop has a *toggle* mode when $J = 1$, $K = 1$. *Toggle* means that the Q and $\overline{Q}$ outputs switch to their opposite states at each active clock edge. To make a JK flip-flop, modify the SR flip-flop's internal logic circuit to include two cross-coupled feedback lines between the output and input. This modification, however, means that the JK flip-flop cannot be level-triggered; it can only be edge-triggered or pulse-triggered. Figure 12.85 shows how you can create edge-triggered flip-flops based on the cross-NAND SR edge-triggered flip-flop.

Edge-triggered JK flip-flops



Positive edge-triggered

C	J	K	Q	$\overline{Q}$	Mode
0	X	X	Q	$\overline{Q}$	hold
1	X	X	Q	$\overline{Q}$	hold
↓	X	X	Q	$\overline{Q}$	hold
↑	0	0	Q	$\overline{Q}$	hold
↑	0	1	0	0	Reset
↑	1	0	0	0	Set
↑	1	1	$\overline{Q}$	Q	Toggle

Negative edge-triggered

C	J	K	Q	$\overline{Q}$	Mode
0	X	X	Q	$\overline{Q}$	hold
1	X	X	Q	$\overline{Q}$	hold
↑	X	X	Q	$\overline{Q}$	hold
↓	0	0	Q	$\overline{Q}$	hold
↓	0	1	0	0	Reset
↓	1	0	0	0	Set
↓	1	1	$\overline{Q}$	Q	Toggle

Timing diagram for negative-edge triggered JK flip-flop

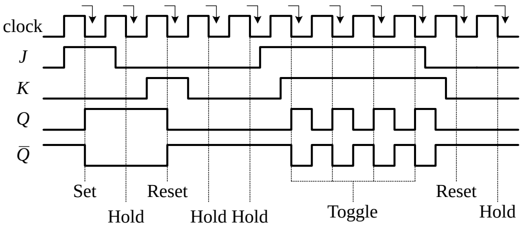
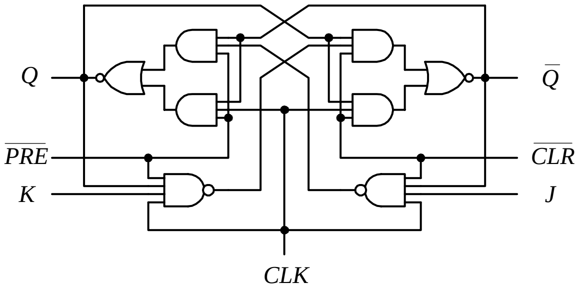


FIGURE 12.85

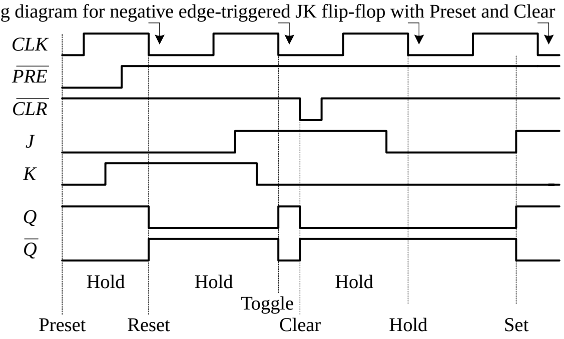
Edge-triggered JK flip-flops also come with preset (asynchronous set) and clear (asynchronous reset) inputs. See Fig. 12.86.

Edge-triggered JK flip-flops with Preset and Clear

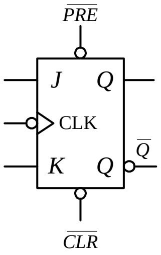
Schematic for negative-triggered JK flip-flop with Preset and Clear



Timing diagram for negative edge-triggered JK flip-flop with Preset and Clear



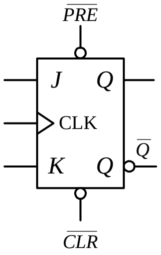
Negative edge-triggered JK flip-flop with Preset and Clear



$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$	Mode
0	1	X	X	X	1	0	Preset
1	0	X	X	X	0	1	Clear
0	0	X	X	X	1	1	not used (race)
1	1	↓	0	0	Q_0	$\overline{Q}_0$	hold
1	1	↓	0	1	0	0	Reset
1	1	↓	1	0	0	0	Set
1	1	↓	1	1	$\overline{Q}_0$	Q_0	Toggle
1	1	↑ 0,1	1	1	Q_0	$\overline{Q}_0$	hold

Q_0 = state of Q before HIGH-to-LOW edge of clock.

Positive edge-triggered JK flip-flop with Preset and Clear



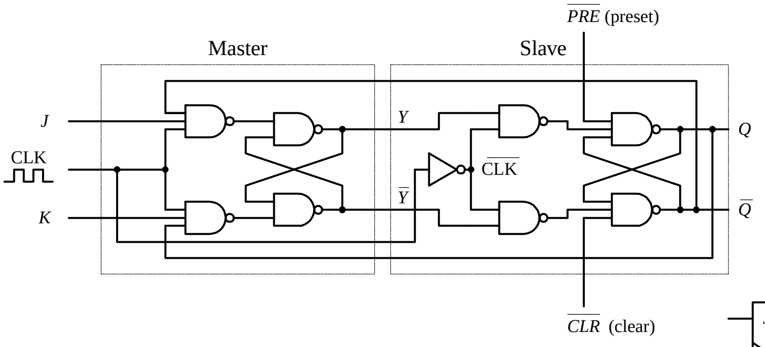
$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$	Mode
0	1	X	X	X	1	0	Preset
1	0	X	X	X	0	1	Clear
0	0	X	X	X	1	1	not used (race)
1	1	↑	0	0	Q_0	$\overline{Q}_0$	hold
1	1	↑	0	1	0	0	Reset
1	1	↑	1	0	0	0	Set
1	1	↑	1	1	$\overline{Q}_0$	Q_0	Toggle
1	1	↓ 0,1	1	1	Q_0	$\overline{Q}_0$	hold

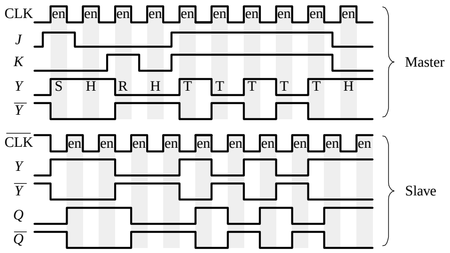
Q_0 = state of Q before LOW-to-HIGH edge of clock.

FIGURE 12.86

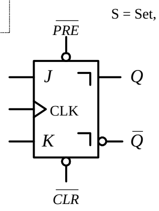
There are pulse-triggered (master-slave) flip-flops too, although they are not as popular as the edge-triggered JK flip-flops for an undesired effect that occurs, which I will talk about in a second. These devices are similar to the pulse-triggered SR flip-flops with the exception of the distinctive JK cross-coupled feedback connections from the slave's Q and $\overline{Q}$ outputs back to the master's input gates. The figure below shows a simple NAND pulse-triggered JK flip-flop.

Pulse-triggered JK flip-flop (master-slave JK flip-flop)





S = Set, R = Reset, H = Hold, T = Toggle, en = enable

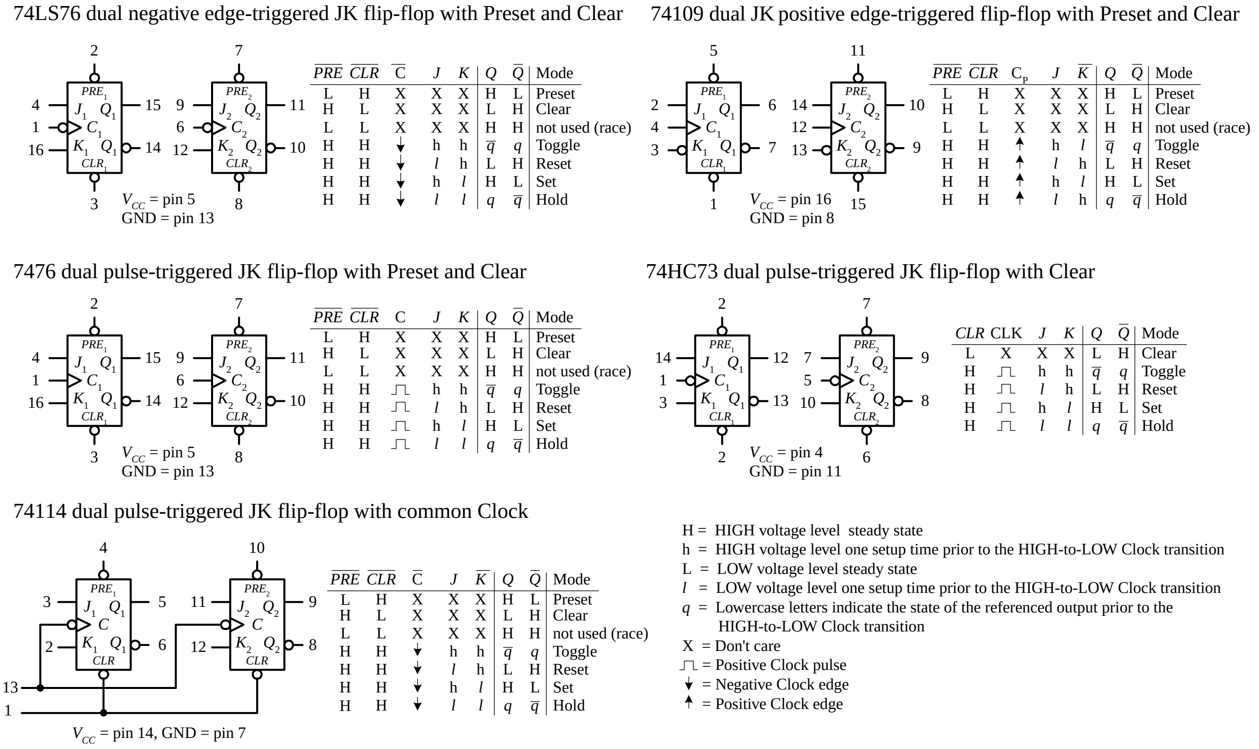


$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$	Mode
0	1	X	X	X	1	0	Preset
1	0	X	X	X	0	1	Clear
0	0	X	X	X	1	1	not used (race)
1	1	⎍	0	0	Q_0	$\overline{Q}_0$	Hold
1	1	⎍	0	1	0	1	Reset
1	1	⎍	1	0	1	0	Set
1	1	⎍	1	1	$\overline{Q}_0$	Q_0	Toggle

FIGURE 12.87

Now there is often a problem with pulse-triggered JK flip-flops. They occasionally experience what is called *ones catching.* In ones catching, unwanted pulses or glitches caused by electrostatic noise appear on *J* and *K* while the clock is high. The flip-flop remembers these glitches and interprets them as true data. Ones catching normally is not a problem when clock pulses are of short duration; it is when the pulses get long that you must watch out. To avoid ones catching all together, stick with edge-triggered JK flip-flops.

A Few JK Flip-Flop ICs



74LS76 dual negative edge-triggered JK flip-flop with Preset and Clear

$\overline{PRE}$	$\overline{CLR}$	$\overline{C}$	J	K	Q	$\overline{Q}$	Mode
L	H	X	X	X	H	L	Preset
H	L	X	X	X	L	H	Clear
L	L	X	X	X	H	H	not used (race)
H	H	↓	h	h	$\overline{q}$	q	Toggle
H	H	↓	l	h	L	H	Reset
H	H	↓	h	l	H	L	Set
H	H	↓	l	l	q	$\overline{q}$	Hold

V_{CC} = pin 5
GND = pin 13

74109 dual JK positive edge-triggered flip-flop with Preset and Clear

$\overline{PRE}$	$\overline{CLR}$	C_P	J	$\overline{K}$	Q	$\overline{Q}$	Mode
L	H	X	X	X	H	L	Preset
H	L	X	X	X	L	H	Clear
L	L	X	X	X	H	H	not used (race)
H	H	↑	h	l	$\overline{q}$	q	Toggle
H	H	↑	l	h	L	H	Reset
H	H	↑	h	l	H	L	Set
H	H	↑	l	h	q	$\overline{q}$	Hold

V_{CC} = pin 16
GND = pin 8

7476 dual pulse-triggered JK flip-flop with Preset and Clear

$\overline{PRE}$	$\overline{CLR}$	C	J	K	Q	$\overline{Q}$	Mode
L	H	X	X	X	H	L	Preset
H	L	X	X	X	L	H	Clear
L	L	X	X	X	H	H	not used (race)
H	H	⎍	h	h	$\overline{q}$	q	Toggle
H	H	⎍	l	h	L	H	Reset
H	H	⎍	h	l	H	L	Set
H	H	⎍	l	l	q	$\overline{q}$	Hold

V_{CC} = pin 5
GND = pin 13

74HC73 dual pulse-triggered JK flip-flop with Clear

$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$	Mode
L	X	X	X	L	H	Clear
H	⎍	h	h	$\overline{q}$	q	Toggle
H	⎍	l	h	L	H	Reset
H	⎍	h	l	H	L	Set
H	⎍	l	l	q	$\overline{q}$	Hold

V_{CC} = pin 4
GND = pin 11

74114 dual pulse-triggered JK flip-flop with common Clock

$\overline{PRE}$	$\overline{CLR}$	$\overline{C}$	J	$\overline{K}$	Q	$\overline{Q}$	Mode
L	H	X	X	X	H	L	Preset
H	L	X	X	X	L	H	Clear
L	L	X	X	X	H	H	not used (race)
H	H	↓	h	h	$\overline{q}$	q	Toggle
H	H	↓	l	h	L	H	Reset
H	H	↓	h	l	H	L	Set
H	H	↓	l	l	q	$\overline{q}$	Hold

V_{CC} = pin 14, GND = pin 7

H = HIGH voltage level steady state
h = HIGH voltage level one setup time prior to the HIGH-to-LOW Clock transition
L = LOW voltage level steady state
l = LOW voltage level one setup time prior to the HIGH-to-LOW Clock transition
q = Lowercase letters indicate the state of the referenced output prior to the HIGH-to-LOW Clock transition
X = Don't care
⎍ = Positive Clock pulse
↓ = Negative Clock edge
↑ = Positive Clock edge

FIGURE 12.88

12.6.7 Applications for JK Flip-Flops

Two major applications for JK flip-flops are found within counter and shift register circuits. For now, I will simply introduce a counter application—I will discuss shift registers and additional counter circuits later on in this chapter.

Ripple Counter (Asynchronous Counter)

A simple counter, called a *MOD-16 ripple counter* (or *asynchronous counter*), can be constructed by joining four JK flop-flops together, as shown in Fig. 12.89. (*MOD-16,* or *modulus 16,* means that the counter has 16 binary states.) This means that it can count from 0 to 15—the zero is one of the counts.

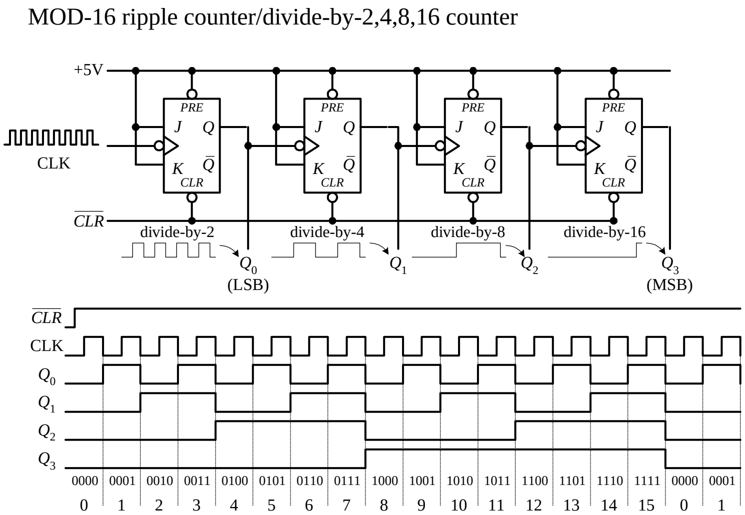
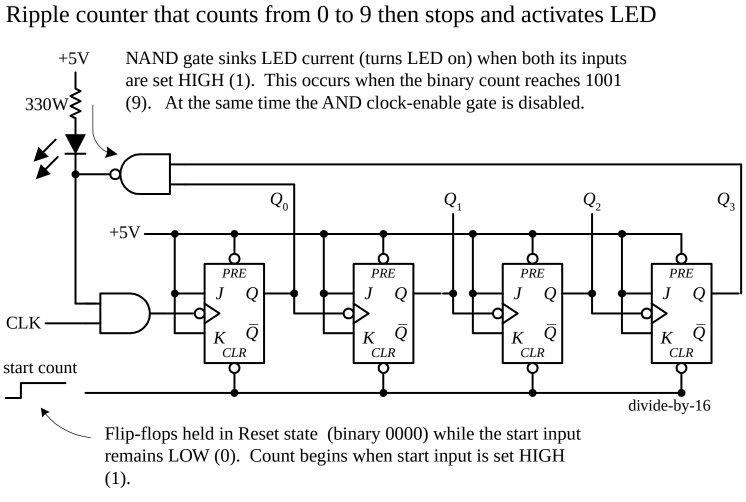

MOD-16 ripple counter/divide-by-2,4,8,16 counter

FIGURE 12.89

Each flip-flop in the ripple counter is fixed in toggle mode (J and K are both held high). The clock signal applied to the first flip-flop causes the flip-flop to divide the clock signal's frequency by 2 at its Q_0 output—a result of the toggle. The second flip-flop receives Q_0's output at its clock input and likewise divides by 2. The process continues down the line. What you get in the end is a binary counter with four digits. The least significant bit (LSB) is Q_0, while the most significant bit (MSB) is Q_4. When the count reaches 1111, the counter recycles back to 0000 and continues from there. To reset the counter at any given time, the active-low clear line is pulsed low. To make the counter count backward from 1111 to 0000, you would simply use the $\overline{Q}$ outputs.

The ripple counter above also can be used as a divide-by-2,4,8,16 counter. Here, you simply replace the clock signal with any desired input signal that you wish to divide in frequency. To get a divide-by-2 counter, you only need the first flip-flop; to get a divide-by-8 counter, you need the first three flip-flops.

Ripple counters with higher MOD values can be constructed by slapping on more flip-flops to the MOD-16 counter. But how do you create a ripple counter with a MOD value other than 2, 4, 8, 16, etc.? For example, say you want to create a MOD-10 (0 to 9) ripple counter. Likewise, what do you do if you want to stop the counter after a particular count has been reached and then trigger some device, such as an LED or buzzer. The figure below shows just such a circuit.


Ripple counter that counts from 0 to 9 then stops and activates LED

NAND gate sinks LED current (turns LED on) when both its inputs are set HIGH (1). This occurs when the binary count reaches 1001 (9). At the same time the AND clock-enable gate is disabled.

Flip-flops held in Reset state (binary 0000) while the start input remains LOW (0). Count begins when start input is set HIGH (1).

FIGURE 12.90

To make a MOD-10 counter, you simply start with the MOD-16 counter and connect the Q_0 and Q_3 outputs to a NAND gate, as shown in the figure. When the counter reaches 9 (1001), Q_0 and Q_3 will both go high, causing the NAND gate's output to go low. The NAND gate then sinks current, turning the LED on, while at the same time disabling the clock-enable gate and stopping the count. (When the NAND gate is high, there is no potential difference across the LED to light it up.) To start a new count, the active-low clear line is momentarily pulsed low. Now, to make a MOD-15 counter, you would apply the same basic approach used to the left, but you would connect Q_1, Q_2, and Q_3 to a three-input NAND gate.

Synchronous Counter

There is a problem with the ripple counter just discussed. The output stages of the flip-flops further down the line (from the first clocked flip-flop) take time to respond to changes that occur due to the initial clock signal. This is a result of the internal propagation delay that occurs within a given flip-flop. A standard TTL flip-flop may have an internal propagation delay of 30 ns. If you join four flip-flops to create a MOD-16 counter, the accumulative propagation delay at the highest-order output will be 120 ns. When used in high-precision synchronous systems, such large delays can lead to timing problems.

To avoid large delays, you can create what is called a *synchronous counter.* Synchronous counters, unlike ripple (asynchronous) counters, contain flip-flops whose clock inputs are driven at the same time by a common clock line. This means that output transitions for each flip-flop will occur at the same time. Now, unlike the ripple counter, you must use some additional logic circuitry placed between various flip-flop inputs and outputs to give the desired count waveform. For example, to create a 4-bit MOD-16 synchronous counter requires adding two additional AND gates, as shown below. The AND gates act to keep a flip-flop in hold mode (if both input of the gate are low) or toggle mode (if both inputs of the gate are high). So, during the 0–1 count, the first flip-flop is in toggle mode (and always is); all the rest are held in hold mode. When it is time for the 2–4 count, the first and second flip-flops are placed in toggle mode; the last two are held in hold mode. When it is time for the 4–8 count, the first AND gate is enabled, allowing the the third flip-flop to toggle. When it is time for the 8–15 count, the second AND gate is enabled, allowing the last flip-flop to toggle. You can work out the details for yourself by studying the circuit and timing waveforms.

MOD-16 synchronous counter

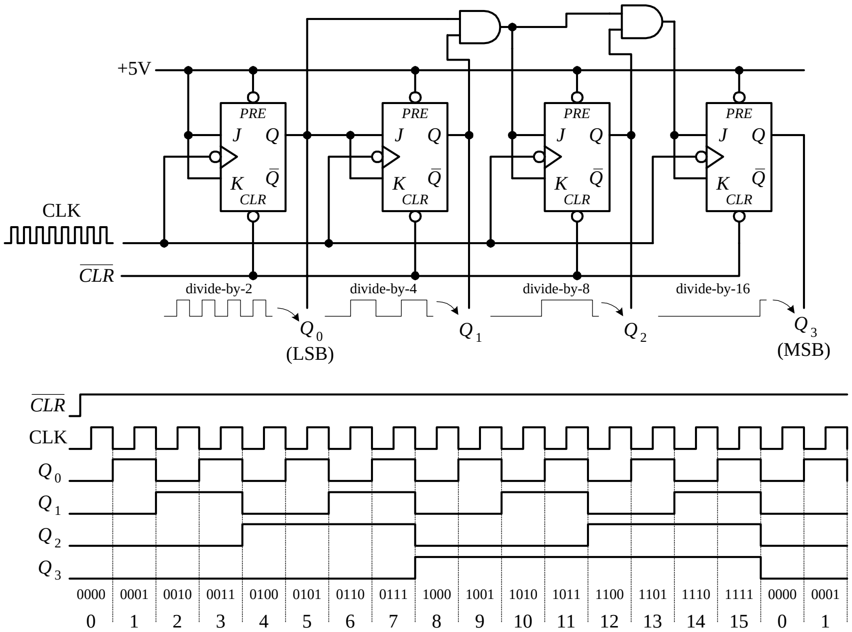


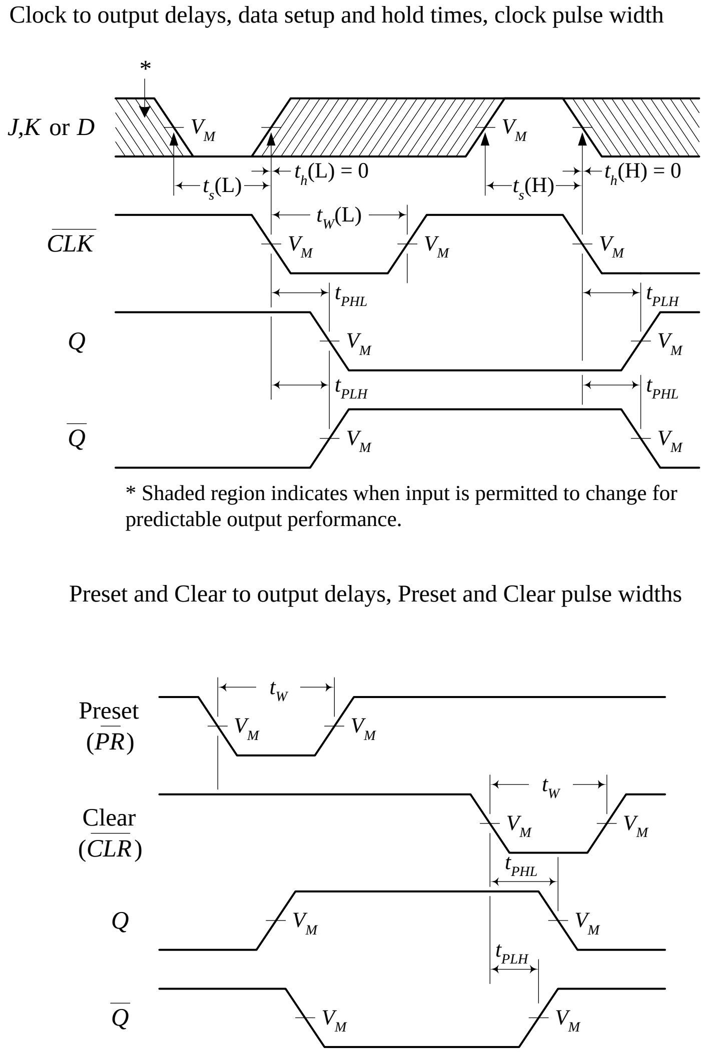
FIGURE 12.91

The ripple (asynchronous) and synchronous counters discussed so far are simple but hardly ever used. In practice, if you need a counter, be it ripple or synchronous, you go out and purchase a counter IC. These ICs are often MOD-16 or MOD-10 counters and usually come with many additional features. For example, many ICs allow you to preset the count to a desired number via parallel input lines. Others allow you to count up or to count down by means of control inputs. I will talk in great detail about counter ICs in a moment.

12.6.8 Practical Timing Considerations with Flip-Flops

When working with flip-flops, it is important to avoid race conditions. For example, a typical race condition would occur if, say, you were to apply an active clock edge at the very moment you apply a high or low pulse to one of the inputs of a JK flip-flop. Since the JK flip-flop uses what is present on the inputs at the moment the clock edge arrives, having a high-to-low input change will cause problems because you cannot determine if the input is high or low at that moment—it is a straight line. To avoid this type of race condition, you must hold the inputs of the flip-flop high or low for at least one setup time t_s before the active clock transition. If the input changes during the t_s to clock edge region, the output levels will be unreliable. To determine the setup time for a given flip-flop, you must look through the manufacturer's data sheets. For example, the minimum setup time for the 74LS76 JK flip-flop is 20 ns. Other timing parameters, such as hold time, propagation delay, etc., are also given by the manufacturers. A description of what these parameters mean is given below.

Flip-Flop Timing Parameters


Clock to output delays, data setup and hold times, clock pulse width

* Shaded region indicates when input is permitted to change for predictable output performance.

Preset and Clear to output delays, Preset and Clear pulse widths

FIGURE 12.92

IMPORTANT TERMS

Setup time t_s—The time required that the input must be held before the active clock edge for proper operation. For a typical flip-flop, t_s is around 20 ns.

Hold time t_h—The time required that the input must be held after the active clock edge for proper operation. For most flip-flops, this is 0 ns—meaning inputs need not be held beyond the active clock signal.

T_{PLH}—Propagation delay from clock trigger point to the low-to-high Q output swing. A typical T_{PLH} for a flip-flop is around 20 ns.

T_{PHL}—Propagation delay from clock trigger point to the high-to-low Q output swing. A typical T_{PLH} for a flip-flop is around 20 ns.

$f_{\max}$—Maximum frequency allowed at the clock input. Any frequency above this limit will result in unreliable performance. Can vary greatly.

$t_W(L)$—Clock pulse width (low), the minimum width (in nanoseconds) that is allowed at the clock input during the low level for reliable operation.

$t_W(H)$—Clock pulse width (high), the minimum width (in nanoseconds) that is allowed at the clock input during the high level for reliable operation.

Preset or clear pulse width—Also given by $t_W(L)$, the minimum width (in nanoseconds) of the low pulse at the preset or clear inputs.

12.6.9 Digital Clock Generator and Single-Pulse Generators

You have already seen the importance of clock and single-pulse control signals. Now let's take a look at some circuits that can generate these signals.

Clocks (Astable Multivibrators)

A clock is simply a squarewave oscillator. In Chap. 9 I discussed ways to generate square waves—so you can refer back there to learn the theory. Here I will simply present some practical circuits. Digital clocks can be constructed from discrete components such as logic gates, capacitors, resistors, and crystals or can be purchased in IC form. Here are some sample clock generators.

FIGURE 12.93

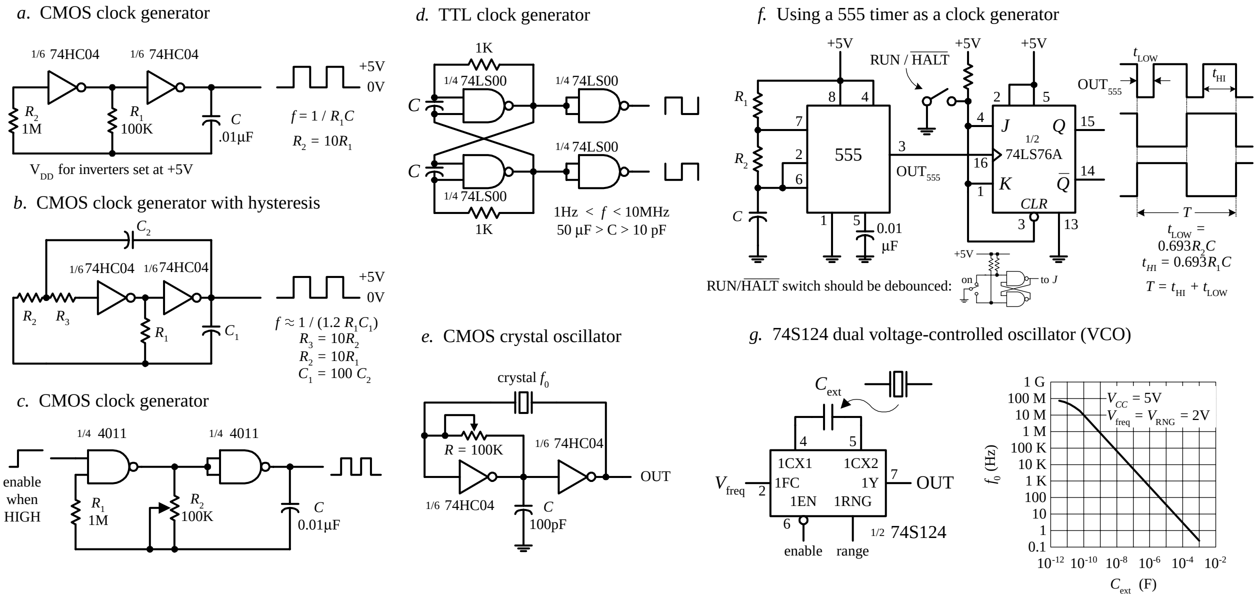


Figure a. Here, two CMOS inverters are connected together to form an *RC* relaxation oscillator with squarewave output. The output frequency is determined by the *RC* time constant, as shown in the figure.

Figure b. The previous oscillator has one problem—it may not oscillate if the transition regions of its two gates differ, or it may oscillate at a slightly lower frequency than the equation predicts due to the finite gain of the leftmost gate. The oscillator shown here resolves these problems by adding hysteresis via the additional *RC* network.

Figure c. This oscillator uses a pair of CMOS NAND gates and *RC* timing network along with a pot to set the frequency. A squarewave output is generated with a maximum frequency of around 2 MHz. The enable lead could be connected to the other input of the first gate, but here it is brought out to be used as an clock enable input (clock is enabled when this lead is high).

Figure d. Here, a TTL SR flip-flop with dual feedback resistors uses an *RC* relaxation-type configuration to generate a square wave. The frequency of the clock is determined by the *R* and *C* values, as shown in the figure. Changing the C_1-to-C_2 ratio changes the duty cycle.

Figure e. When high stability is required, a crystal oscillator is the best choice for a clock generator. Here, a pair of CMOS inverters and a feedback crystal are used (see Chap. 8 for details). The frequency of operation is determined by the crystal (e.g., 2 MHz, 10 MHz, etc.). Adjustment of the pot may be needed to start oscillations.

Figure f. A 555 timer in astable mode can be used to generate square waves. Here, we slap on a JK flip-flop that is in toggle mode to provide a means of keeping the low and high times the same, as well as providing clock-enable control. The timing diagram and the equations provided within the figure will paint the rest of the picture.

Figure g. The 74S124 dual voltage-controlled oscillator (VCO) outputs square waves at a frequency that is dependent on the value of an external capacitor and the voltage levels applied its frequency-range input (V_{RNG}) and its frequency control input (V_{freq}). The graph in this figure shows how the frequency changes with capacitance, while V_{RNG} and V_{freq} are fixed at 2 V. This device also comes with active-low enable input. Other VCOs that are designed for clock generation include the 74LS624, 4024, and 4046 (PLL). You will find many more listed in the catalogs.

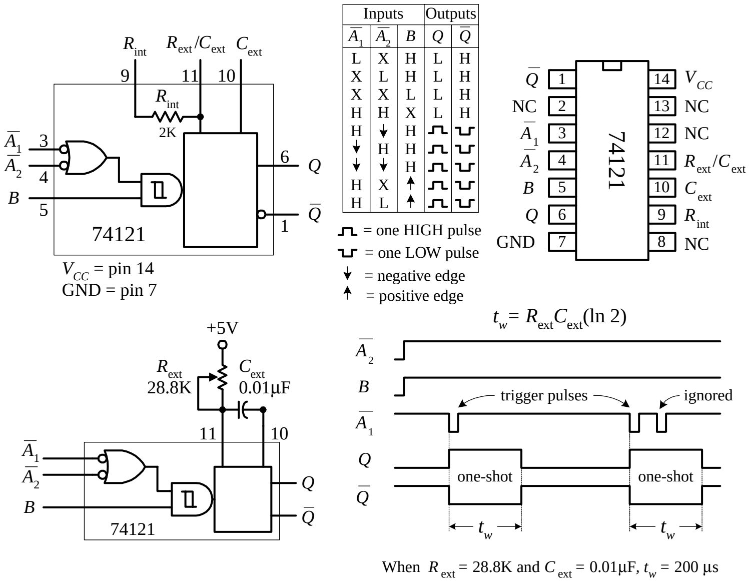
Monostables (One-Shots)

To generate single-pulse signals of a desired width, you can use a discrete device called a *monostable multivibrator*, or *one-shot* for short. A one-shot has only one stable state, high (or low), and can be triggered into its unstable state, low (or high) for a duration of time set by an *RC* network. One-shots can be constructed from simple gates, capacitors, and resistors. These circuits, however, tend to be “finicky” and simply are not worth talking about. If you want a one-shot, you go out and buy a one-shot IC, which is typically around 50 cents.

Two popular one shots, shown below, are the 74121 nonretriggerable monostable multivibrator and the 74123 retriggerable monostable multivibrator.

FIGURE 12.94

74121 non-retriggerable monostable multivibrator (one-shot)



Inputs			Outputs	
$\overline{A}_1$	$\overline{A}_2$	B	Q	$\overline{Q}$
L	X	H	L	H
X	L	H	L	H
X	X	L	L	H
H	H	X	L	H
H	↓	H	⊓	⊔
↓	H	H	⊓	⊔
↓	↓	H	⊓	⊔
H	X	↑	⊓	⊔
H	L	↑	⊓	⊔

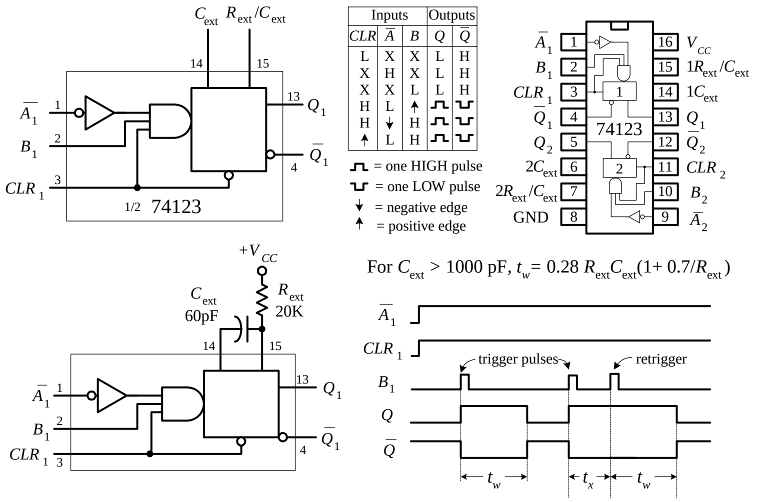
⊓ = one HIGH pulse
⊔ = one LOW pulse
↓ = negative edge
↑ = positive edge

$t_w = R_{ext}C_{ext}(\ln 2)$

When R_{ext} = 28.8K and C_{ext} = 0.01µF, t_w = 200 µs

The 74121 has three trigger inputs ($\overline{A}_1$, $\overline{A}_2$, B), true and complemented outputs (Q, $\overline{Q}$), and timing inputs to which an *RC* network is attached (R_{ext}/C_{ext}, C_{ext}). To trigger a pulse from the 74123, you can choose between five possible trigger combinations, as shown in the truth table in the figure. Bringing the input trigger in on *B*, however, is attractive when dealing with slowly rising or noisy signals, since the signal is directly applied to an internal Schmitt-triggered inverter (recall hysteresis). To set the desired output pulse width (t_w), a resistor/capacitor combination is connected to the R_{ext}/C_{ext} and C_{ext} inputs, as shown. (An internal 2-k resistor is provided, which can be used alone by connecting pin 9 to V_{CC} and placing the capacitor across pins 10 and 11, or which can be used in series with an external resistor attached to pin 9. Here, the internal resistor will not be used.) To determine what values to give to the external resistor and capacitor, use the formula given by the manufacturer, which is shown to the left. The maximum t_w should not exceed 28 s (R = 40 k, C = 1000 µF) for reliable operation. Also, note that with a nonretriggerable one-shot like the 74121, any trigger pulses applied when the device is already in its astable state will be ignored.

74123 retriggerable monostable multivibrator (one-shot)



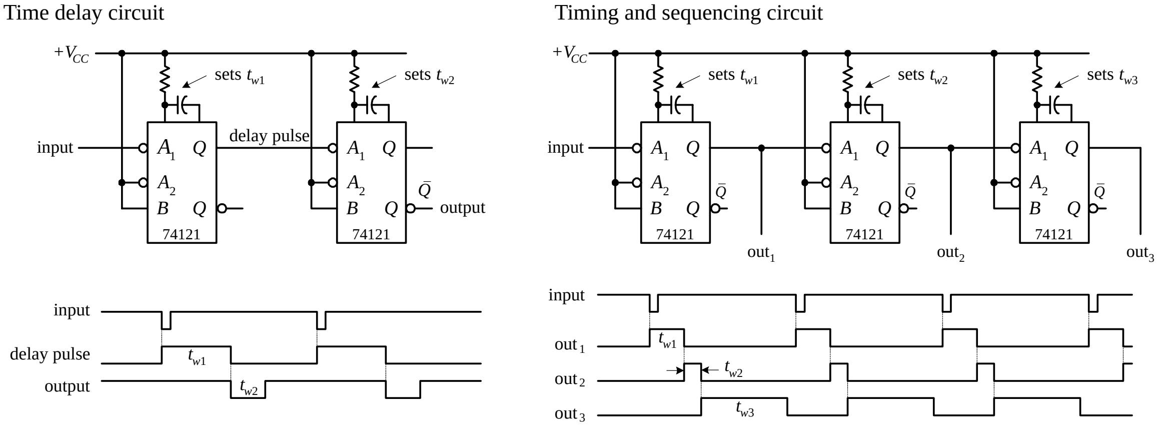
Inputs			Outputs	
CLR	$\overline{A}$	B	Q	$\overline{Q}$
L	X	X	L	H
X	H	X	L	H
X	X	L	L	H
H	L	↑	⊓	⊔
H	↓	H	⊓	⊔
↑	L	H	⊓	⊔

⊓ = one HIGH pulse
⊔ = one LOW pulse
↓ = negative edge
↑ = positive edge

For C_{ext} > 1000 pF, $t_w = 0.28\ R_{ext}C_{ext}(1 + 0.7/R_{ext})$

The 74123 is a dual, retriggerable one-shot. Unlike nonretriggerable one-shots, this device will not ignore trigger pulses that are applied during the astable state. Instead, when a new trigger pulse arrives during an astable state, the astable state will continue to be astable for a time of t_w. In other words, the device is simply retriggered. The 74123 has two trigger inputs ($\overline{A}$, B) and a clear input (CLR). When CLR is low, the one-shot is forced back into its stable state (Q = low). To determine t_w, use the formula given to the left, provided C_{ext} > 1000 pF. If C_{ext} < 1000 pF, use $t_w/C_{ext}/R_{ext}$ graphs provided by the manufacturer to find t_w.

Besides acting as simple pulse generators, one-shots can be combined to make time-delay generators and timing and sequencing circuits. See the figure below.


Time delay circuit

Timing and sequencing circuit

FIGURE 12.95

Now if you do not have a one-shot IC like the 74121, you can use a 555 timer wired in its monostable configuration, as shown below. (I discussed the 555 in Chap. 8—go there if you need the details.)

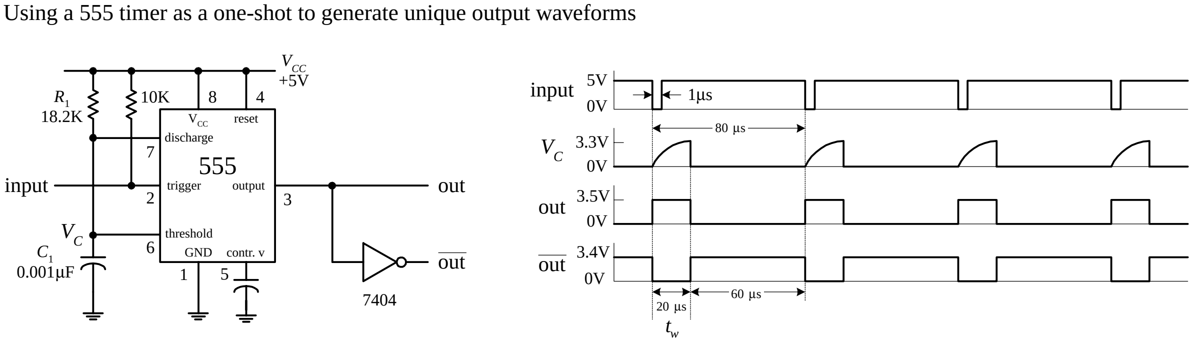
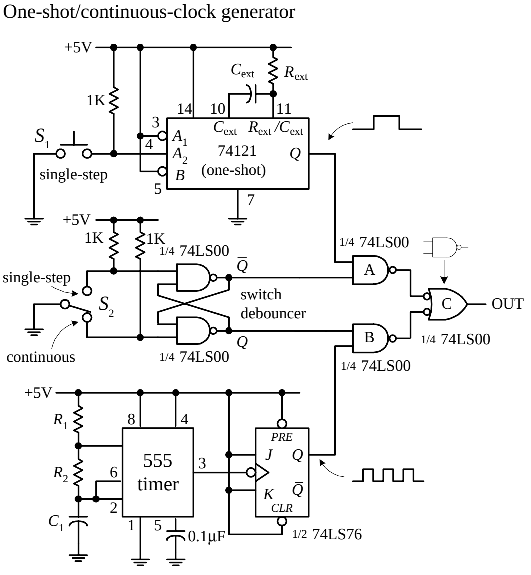

Using a 555 timer as a one-shot to generate unique output waveforms

FIGURE 12.96

One-Shot/Continuous-Clock Generator

The circuit below is a handy one-shot/continuous clock generator that is useful when you start experimenting with logic circuits. The details of how this circuit work are explained below.

FIGURE 12.97


One-shot/continuous-clock generator

In this circuit, switch S_2 is used to select whether a single-step or a continuous-clock input is to be presented to the output. When S_2 is in the single-step position, the cross-NAND SR flip-flop (switch debouncer) is set ($Q = 1$, $\overline{Q} = 0$). This disables NAND gate *B* while enabling NAND gate *A*, which will allow a single pulse from the one-shot to pass through gate *C* to the output. To trigger the one-shot, press switch S_1. When S_2 is thrown to the continuous position, the switch debouncer is reset ($Q = 0$, $\overline{Q} = 1$). This disables NAND gate *A* and enables NAND gate *B*, allowing the clock signal generated by the 555/flip-flop to pass through gate *C* and to the output. (Just as a note to avoid confusion, you need gate *C* to prevent the output from being low and high at the same time.)

12.6.10 Automatic Power-Up Clear (Reset) Circuits

In sequential circuits it is usually a good idea to clear (reset) devices when power is first applied. This ensures that devices, such as flip-flops and other sequential ICs, do not start out in a weird mode (e.g., counter IC does not start counting at, say, 1101 instead of 0000). The following are some techniques used to provide automatic power-up clearing.

Automatic power-up CLEAR circuit

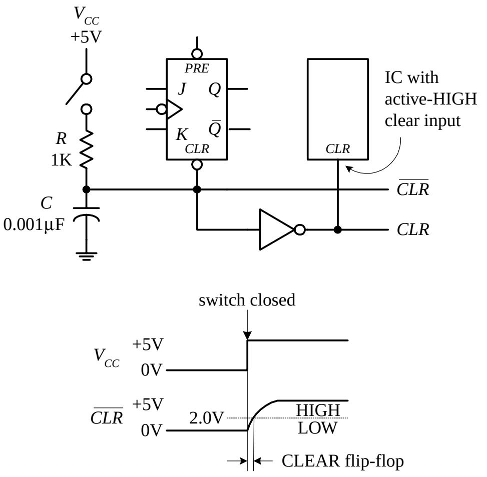


FIGURE 12.98

Let's pretend that one of the devices in a circuit has a JK flip-flop that needs clearing during power-up. In order to clear the flip-flop and then quickly return it to synchronous operations, you would like to apply a low (0) voltage to its active-low clear input; afterwards, you would like the voltage to go high (at least above 2.0 V for a 74LS76 JK flip-flop). A simple way to implement this function is to use an *RC* network like the one shown in the figure. When the power is off (switch open), the capacitor is uncharged (0 V). This means that the $\overline{CLR}$ line is low (0 V). Once the power is turned on (switch closed), the capacitor begins charging up toward V_{CC} (+5 V). However, until the capacitor's voltage reaches 2.0 V, the $\overline{CLR}$ line is considered low to the active-low clear input. After a duration of $t = RC$, the capacitor's voltage will have reached 63 percent of V_{CC}, or 3.15 V; after a duration of $t = 5RC$, its voltage will be nearly equal to +5 V. Since the 74LS76's $\overline{CLR}$ input requires at least 2.0 V to be placed back into synchronous operations, you know that $t = RC$ is long enough. Thus, by rough estimate, if you want the $\overline{CLR}$ line to remain low for 1 µs after power-up, you must set $RC = 1$ µs. Setting $R = 1$ k and $C = 0.001$ µF does the trick.

This automatic resetting scheme can be used within circuits that contain a number of resettable ICs. If an IC requires an active-high reset (not common), simply throw in an inverter and create an active-high clear line, as shown in the figure. Depending on the device being reset, the length of time that the clear line is at a low will be about 1 µs. As more devices are placed on the clear line, the low time duration will decrease due to the additional charging paths. To prevent this from occurring, a larger capacitor can be used.

An improved automatic power-up clear circuit is shown in Fig. 12.99. Here a Schmitt-triggered inverter is used to make the clear signal switch off cleanly. With CMOS Schmitt-triggered inverters, a diode and input resistor (R_2) are necessary to protect the CMOS IC when power is removed.

Improved automatic power-up CLEAR circuit

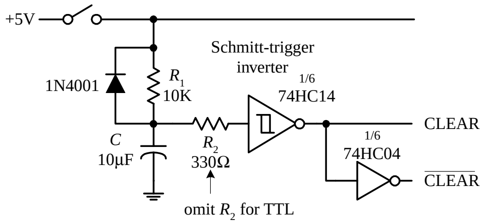


FIGURE 12.99

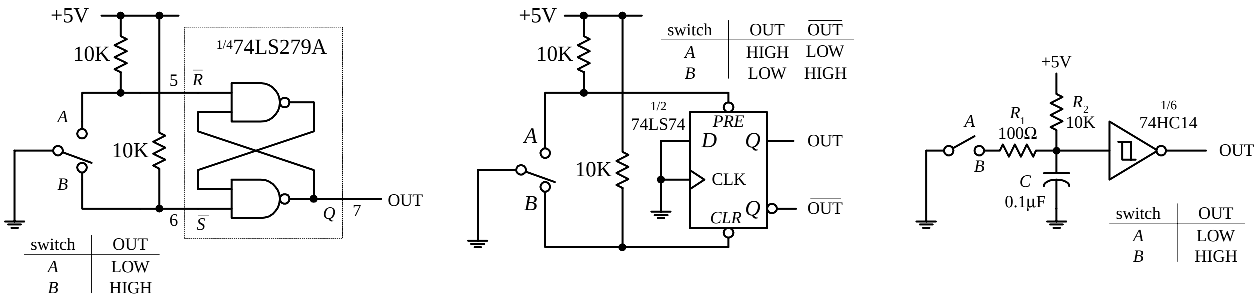
12.6.11 More on Switch Debouncers

The switch debouncer shown to the far left in Fig. 12.100 should look familiar. It is simply a cross-NAND SR-latch-type switch debouncer. Here, a 74LS279A IC is used that contains 4 SR latches—an ideal choice when you need a number of switch debouncers.

Now, a switch debouncer does not have to be constructed from an SR-latch. In fact, most any old flip-flop with preset and clear (reset) inputs can be used. For example, the middle circuit in Fig. 12.100 uses a 74LS74 D-type flip-flop, along with pullup resistor, as a switch debouncer. The *D* input and *CLK* input are tied to ground so that the only two modes that can be enacted are the preset and clear modes. Also, the pullup resistors will always make either the preset input or clear input high, regardless of whether or not the switch is bouncing. From these two facts, you can figure out the rest for yourself, using the truth table for the 74HC74 in Fig. 12.80 as a guide.

Another approach that can be used to debounce an SPST switch is shown in to the far right in Fig. 12.100. This debouncer uses a Schmitt-triggered inverter along with a unique *RC* timing network. When the switch is open, the capacitor is fully charged

(+5 V), and the output is low. When the switch is closed, the capacitor discharges rapidly to ground through the 100-Ω resistor, causing the output to go high. Now, as the switch bounces, the capacitor will repeatedly attempt to charge slowly back to +5 V via the 10-k resistor, and then again will discharge rapidly to zero through the 100-W resistor, making the output high. By making the 10-k pullup resistor larger than the 100-Ω discharge resistor, the voltage across the capacitor or the voltage applied to the inverter's input will not get a chance to exceed the positive threshold voltage (V_T+) of the inverter during a bounce. Therefore, the output remains high, regardless of the bouncing switch. In this example, the charge-up time constant (R_2C = 10 k × 0.1 μF) ensures sufficient leeway. When the switch is reopened, the capacitor charges up toward +5 V. When the capacitor's voltage reaches V_T+, the output switches low.



SPDT switch debouncer using 74LS279 quad SR latch.

SPDT switch debouncer using 74LS74 dual D-type flip-flop.

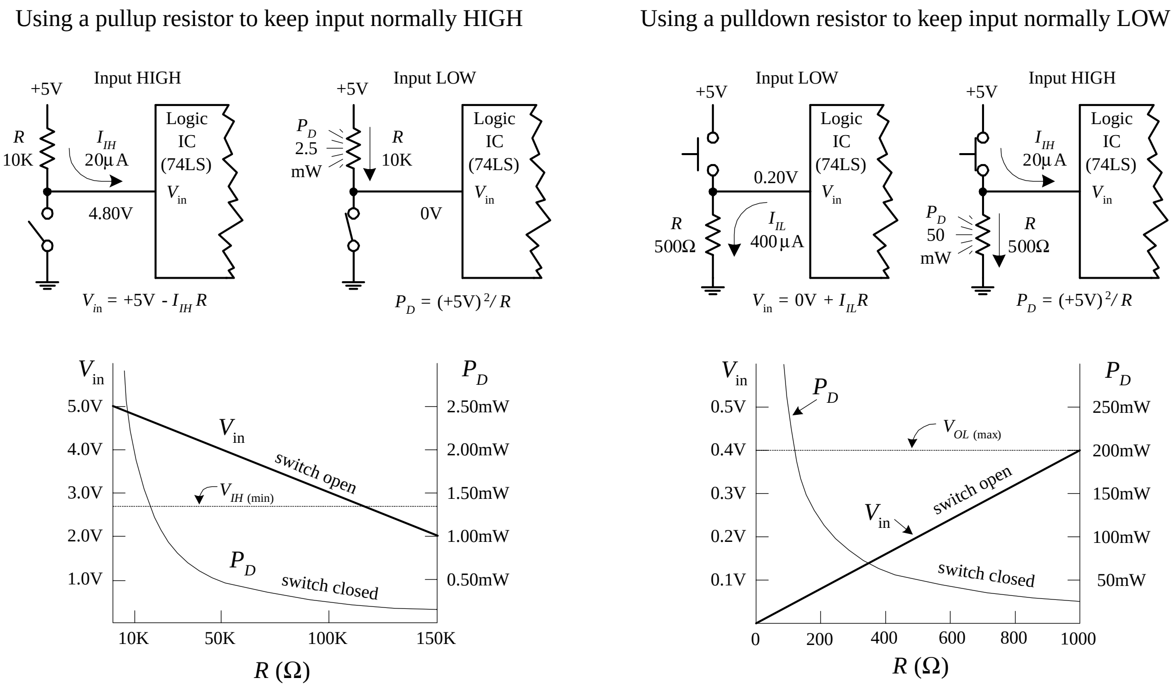
SPST switch debouncer using 74HC14 hex Schmitt-triggered inverter.

FIGURE 12.100

12.6.12 Pullup and Pulldown Resistors

As you learned when dealing with the switch debouncer circuits, a pullup resistor is used to keep an input high that would otherwise float if left unconnected. If you want to set the "pulled up" input low, you can ground the pin, say, via a switch. Now, it is important to get an idea of the size of pullup resistor to use. The key here is to make

FIGURE 12.101


Using a pullup resistor to keep input normally HIGH

Using a pulldown resistor to keep input normally LOW the resistor value small enough so that the voltage drop across it does not weigh down the input voltage below the minimum high threshold voltage ($V_{IH,\text{min}}$) of the IC. At the same time, you do not want to make it too small; otherwise, when you ground the pin, excessive current will be dissipated.

In the left figure on p. 383, a 10-k pullup resistor is used to keep a 74LS device's input high. To make the input low, close the switch. To figure out if the resistor is large enough so as not to weigh down the input, use $V_{\text{in}} = +5\text{ V} - RI_{IH}$, where I_{IH} is the current drawn into the IC during the high input state, when the switch is open. For a typical 74LS device, I_{IH} is around 20 μA. Thus, by applying the simple formula, you find that $V_{\text{in}} = 4.80$ V, which is well above the $V_{IH,\text{min}}$ level for a 74LS device. Now, if you close the switch to force the input low, the power dissipated through the resistor ($P_D = V^2/R$) will be $(5\text{ V})^2/10\text{ k} = 25$ mW. The graph shown in the figure provides V_{in} versus R and P_D versus R curves. As you can see, if R becomes too large, V_{in} drops below the $V_{IH,\text{min}}$ level, and the output will not go high as planned. As R gets smaller, the power dissipation skyrockets. To determine what value of R to use for a specific logic IC, you look up the $V_{IH,\text{min}}$ and $I_{IH,\text{max}}$ values within the data sheets and apply the simple formulas. In most applications, a 10-k pullup resistor will work fine.

Now you will run into situations where a pulldown resistor is used to keep a floating terminal low. Unlike a pullup resistor, the pulldown resistor must be smaller because the input low current I_{IL} (sourced by IC) is usually much larger than I_{IH}. Typically, a pulldown resistor is around 100 to 1 kΩ. A lower resistance ensures that V_{in} is low enough to be interpreted as a low by the logic input. To determine if V_{in} is low enough, use $V_{\text{in}} = 0\text{ V} + I_{IL}R$. As an example, use a 74LS device with an $I_{IL} = 400$ μA and a 500-Ω pulldown resistor. When the switch is open, the input will be 0.20 V—well below the $V_{IL,\text{max}}$ level for the 74LS (~0.8 V). When the switch is closed, the power dissipated by the resistor will be $(5\text{ V})^2/500\ \Omega = 50$ mW. The graph shown in Figure 12.101 provides V_{in} versus R and P_D versus R curves. As you can see by the curves, if R becomes too large, V_{in} surpasses $V_{IL,\text{max}}$ and the output will not be low as planned. As R gets small, the power dissipation skyrockets. If you have to use a pulldown resistor/switch arrangement, be wary of the high power dissipation through the resistor when the switch is closed.

12.7 Counter ICs

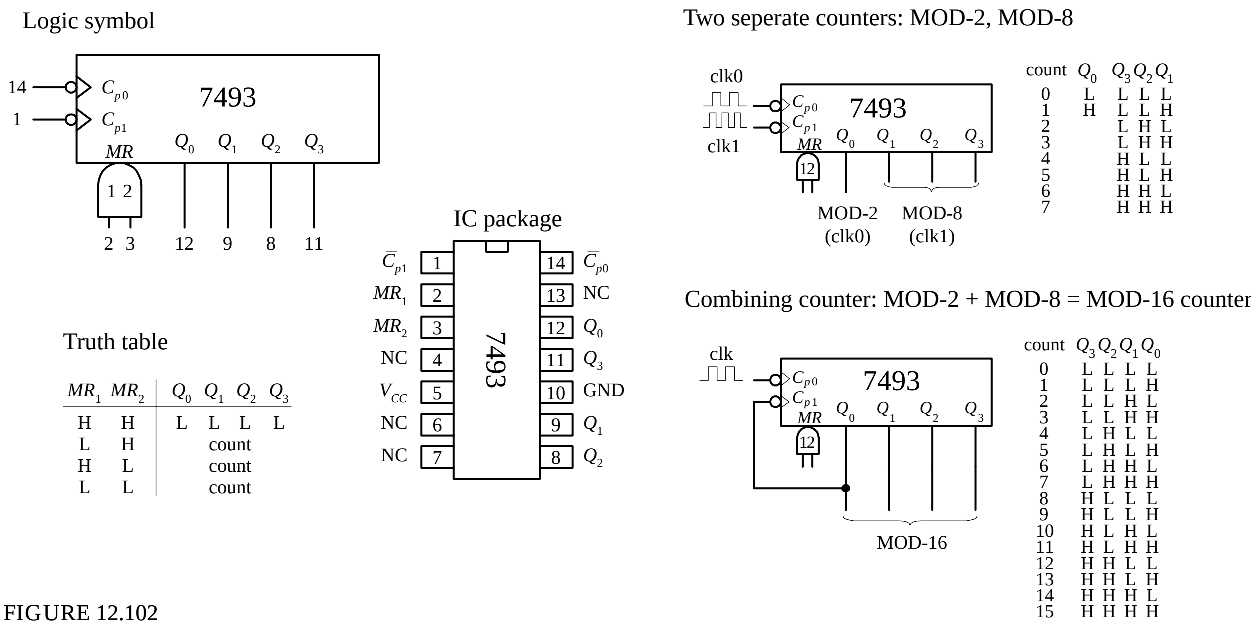
A few pages back you saw how flip-flops could be combined to make both asynchronous (ripple) and synchronous counters. Now, in practice, using discrete flip-flops is to be avoided. Instead, use a prefab counter IC. These ICs cost a buck or two and come with many additional features, like control enable inputs, parallel loading, etc. A number of different kinds of counter ICs are available. They come in either synchronous (ripple) or asynchronous forms and are usually designed to count in binary or binary-coded decimal (BCD).

12.7.1 *Asynchronous Counter (Ripple Counter) ICs*

Asynchronous counters work fine for many noncritical applications, but for high-frequency applications that require precise timing, synchronous counters work better. (Recall that unlike an asynchronous counter, a synchronous counter contains flip-flops that get clocked at the same time, and hence the synchronous counter does not accumulate nearly as many propagation delays as is the case with the asynchronous counter.) Here are a few asynchronous counter ICs you will find in the electronics catalogs.

7493 4-Bit Ripple Counter with Separate MOD-2 and MOD-8 Counter Sections

The 7493's internal structure consists of four JK flip-flops connected to provide separate MOD-2 (0-to-1 counter) and MOD-8 (0-to-7 counter) sections. Both the MOD-2 and MOD-8 sections are clocked by separate clock inputs. The MOD-2 section uses C_{p0} as its clock input, while the MOD-8 section uses C_{p1} as its clock input. Likewise, the two sections have separate outputs—MOD-2's output is Q_0, while MOD-8's outputs consist of Q_1, Q_2, and Q_3. The MOD-2 section can be used as a divide-by-2 counter, while the MOD-8 section can be used as either a divide-by-2 counter (output tapped at Q_1), a divide-by-4 counter (output tapped at Q_2), or a divide-by-8 counter (output tapped at Q_3). Now, if you want to create a MOD-16 counter, simply join the MOD-2 and MOD-8 sections by wiring Q_0 to C_{p1}—while using C_{p0} as the single clock input. The MOD-2, MOD-8, or the MOD-16 counter can be cleared by making both AND-gated master reset inputs (MR_1 and MR_2) high. To begin a count, one or both of the master reset inputs must be made low. When the negative edge of a clock pulse arrives, the count advances one step. After the maximum count is reached (1 for MOD-2, 111 for MOD-8, or 1111 for MOD-16), the outputs jump back to zero, and a new count begins.



Truth table

MR_1	MR_2	Q_0	Q_1	Q_2	Q_3
H	H	L	L	L	L
L	H		count		
H	L		count		
L	L		count		

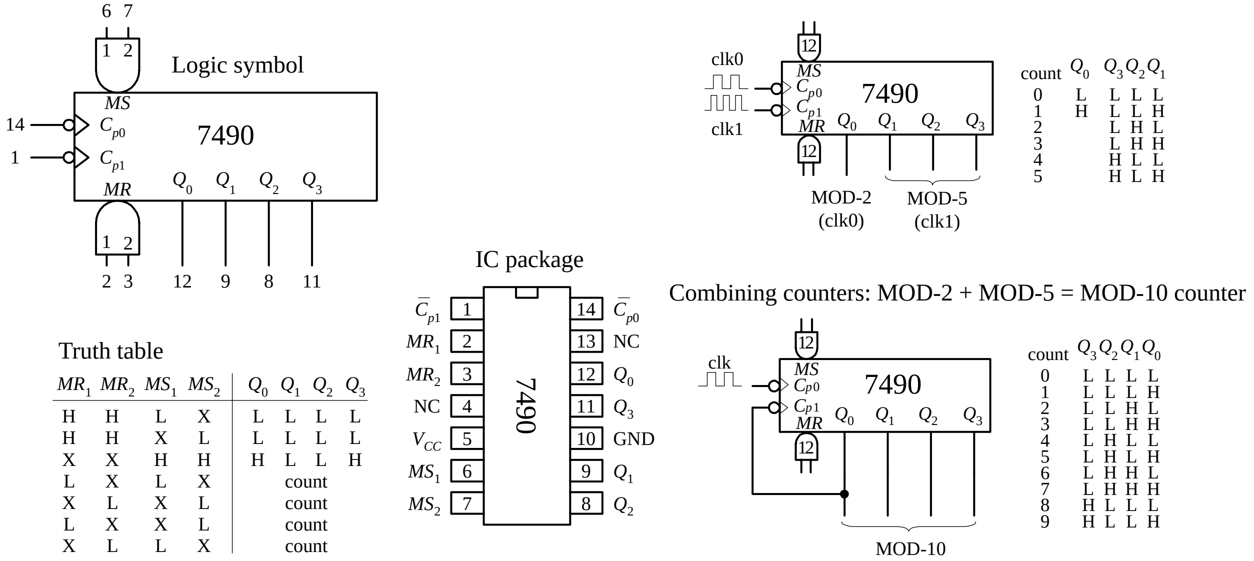
count	Q_0	Q_3	Q_2	Q_1
0	L	L	L	L
1	H	L	L	H
2		L	H	L
3		L	H	H
4		H	L	L
5		H	L	H
6		H	H	L
7		H	H	H

count	Q_3	Q_2	Q_1	Q_0
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

FIGURE 12.102

7490 4-Bit Ripple Counter with MOD-2 and MOD-5 Counter Sections

The 7490, like the 7493, is another 4-bit ripple counter. However, its flip-flops are internally connected to provide MOD-2 (count-to-2) and MOD-5 (count-to-5) counter sections. Again, each section uses a separate clock: C_{p0} for MOD-2 and C_{p1} for MOD-5. By connecting Q_0 to C_{p1} and using C_{p0} as the single clock input, a MOD-10 counter (decade or BCD counter) can be created. When master reset inputs MR_1 and MR_2 are set high, the counter's outputs are reset to 0—provided that master set inputs MS_1 and MS_2 are not both high (the MS inputs override the MR inputs). When MS_1 and MS_2 are high, the outputs are set to $Q_0 = 1$, $Q_1 = 0$, $Q_2 = 0$, and $Q_3 = 1$. In the MOD-10 configuration, this means that the counter is set to 9 (binary 1001). This master set feature comes in handy if you wish to start a count at 0000 after the first clock transition occurs (with master reset, the count starts out at 0001).



Truth table

MR_1	MR_2	MS_1	MS_2	Q_0	Q_1	Q_2	Q_3
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
L	X	L	X	count			
X	L	X	L	count			
L	X	X	L	count			
X	L	L	X	count			

count	Q_0	Q_3	Q_2	Q_1
0	L	L	L	L
1	H	L	L	H
2		L	H	L
3		L	H	H
4		H	L	L
5		H	L	H

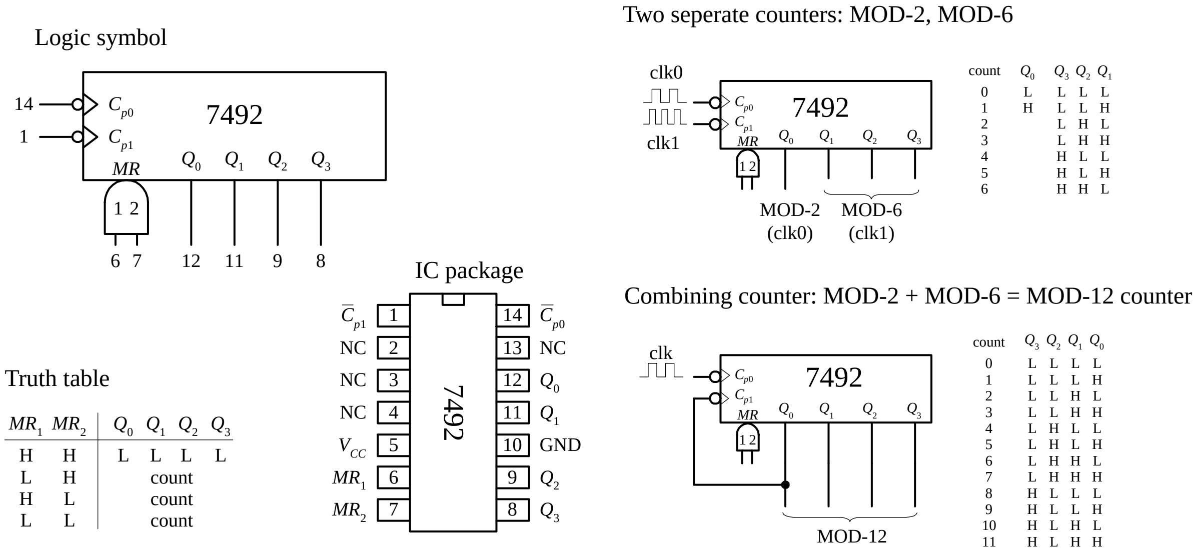
count	Q_3	Q_2	Q_1	Q_0
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

FIGURE 12.103

7492 Divide-by-12 Ripple Counter with MOD-2 and MOD-6 Counter Sections

The 7492 is another 4-bit ripple counter that is similar to 7490. However, it has a MOD-2 and a MOD-6 section, with corresponding clock inputs C_{p0} (MOD-2) and C_{p1} (MOD-8). By joining Q_0 to C_{p1}, you get a MOD-12 counter, where C_{p0} acts as the single clock input. To clear the counter, high levels are applied to master reset inputs MR_1 and MR_2.

FIGURE 12.104



Truth table

MR_1	MR_2	Q_0	Q_1	Q_2	Q_3
H	H	L	L	L	L
L	H	count			
H	L	count			
L	L	count			

count	Q_0	Q_3	Q_2	Q_1
0	L	L	L	L
1	H	L	L	H
2		L	H	L
3		L	H	H
4		H	L	L
5		H	L	H
6		H	H	L

count	Q_3	Q_2	Q_1	Q_0
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H

12.7.2 Synchronous Counter ICs

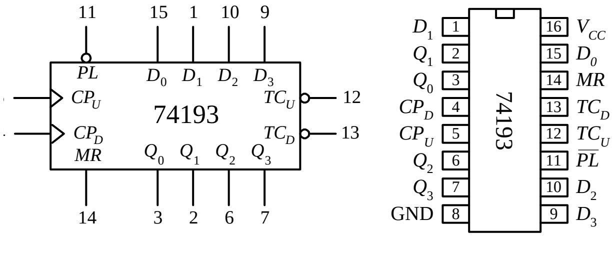
Like the asynchronous counter ICs, synchronous counter ICs come in various MOD arrangements. These devices usually come with extra goodies, such as controls for up or down counting and parallel load inputs used to preset the counter to a desired start count. Synchronous counter ICs are more popular than the asynchronous ICs not only because of these additional features but also because they do not have such long propagation delays as asynchronous counters. Let's take a look at a few popular IC synchronous counters.

74193 Presettable 4-Bit (MOD-16) Synchronous Up/Down Counter

The 74193 is a versatile 4-bit synchronous counter that can count up or count down and can be preset to any count desired—at least a number between 0 and 15. There are two separate clock inputs, C_{pU} and C_{pD}. C_{pU} is used to count up, while C_{pD} is used to count down. One of these clock inputs must be held high in order for the other input to count. The binary output count is taken from Q_0 (2^0), Q_1 (2^1), Q_2 (2^2), and Q_3 (2^3). To preset the counter to any desired count, a corresponding binary number is applied to the parallel inputs D_0 to D_3. When the parallel load input ($\overline{PL}$) is pulsed low, the binary number is loaded into the counter, and the count, either up or down, will start from that number. The terminal count up ($\overline{TC_U}$) and terminal count down ($\overline{TC_D}$) outputs are normally high. The $\overline{TC_U}$ output is used to indicate when the maximum count has been reached and the counter is about to recycle to the minimum count (0000)—carry condition. Specially, this means that $\overline{TC_U}$ goes low when the count reaches 15 (1111) and the input clock (C_{pU}) goes from high to low. $\overline{TC_U}$ remains low until C_{pU} returns high. This low pulse at $\overline{TC_U}$ can be used as an input to the next high-order stage of a multistage counter. The terminal count down ($\overline{TC_D}$) output is used to indicate that the minimum count has been reached (0000) and the counter is about to recycle to the maximum count 15(1111)—borrow condition. Specifically, this means that $\overline{TC_D}$ goes low when the down count reaches 0000 and the input clock (C_{pD}) goes low. The figure below provides a truth table for the 74193, along with a sample load, up-count, and down-count sequence.

FIGURE 12.105

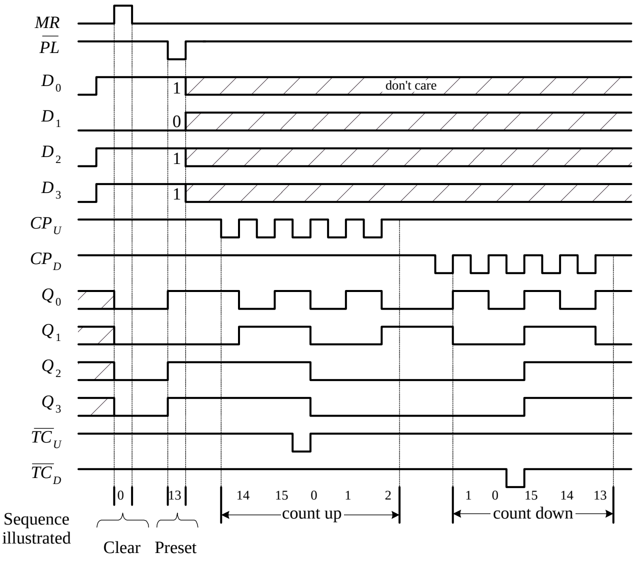
74193 presettable 4-bit binary up/down counter



Mode	Inputs								Ouputs					
	MR_1	$\overline{PL}$	C_{pU}	C_{pD}	D_0	D_1	D_2	D_3	Q_0	Q_1	Q_2	Q_3	$\overline{TC_U}$	$\overline{TC_D}$
Reset	H	X	X	L	X	X	X	X	L	L	L	L	H	L
	H	X	X	H	X	X	X	X	L	L	L	L	H	H
Parallel load	L	L	X	L	L	L	L	L	L	L	L	L	H	L
	L	L	L	X	H	H	H	H	L	L	L	L	H	H
	L	L	H	X	H	H	H	H	H	H	H	H	L	H
	H	L	H	H	L	L	L	L	H	H	H	H	H	H
count up	H	H	↑	H	X	X	X	X	Count up				H	H
count down	L	H	H	↑	X	X	X	X	Count down				H	H

H = HIGH voltage level; L = LOW voltage level; X = don't care; ↑ = LOW-to-HIGH clock transistion

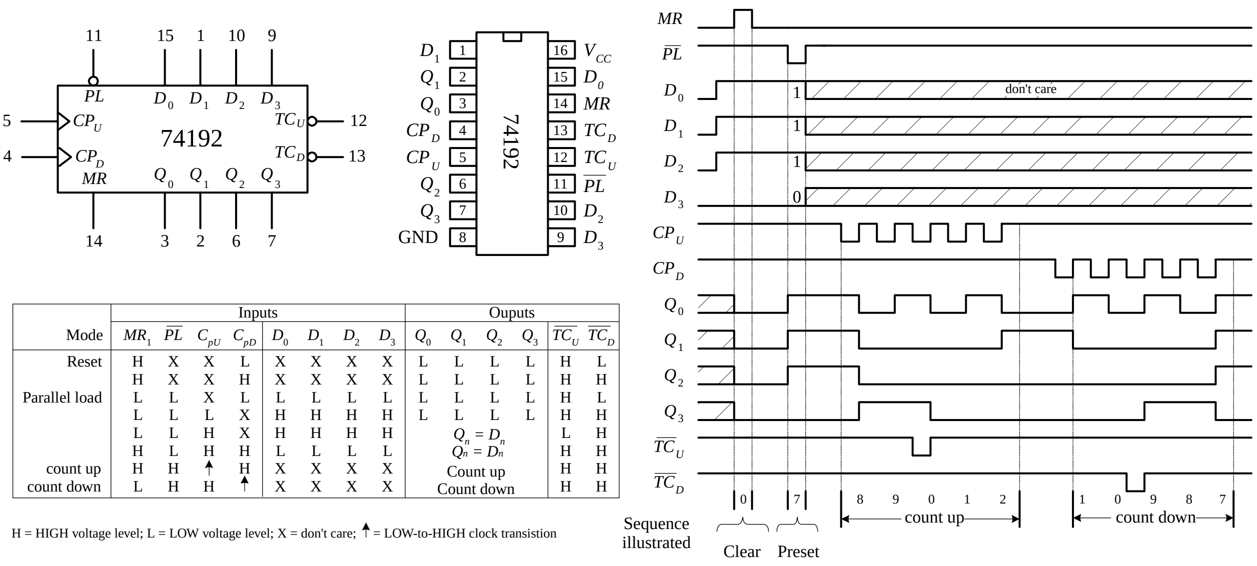
Example load, count-up, count-down sequence



74192 Presettable Decade (BCD or MOD-10) Synchronous Up/Down Counter

The 74193, shown in Fig. 12.106, is essentially the same device as the 74193, except it counts up from 0 to 9 and repeats or counts down from 9 to 0 and repeats. When counting up, the terminal count up ($\overline{TC}_U$) output goes low to indicate when the maximum count is reached (9 or 1001) and the C_{pU} clock input goes from high to low. $\overline{TC}_U$ remains low until C_{pU} returns high. When counting down, the terminal count down output ($\overline{TC}_D$) goes low when the minimum count is reached (0 or 0000) and the input clock C_{pD} goes low. The truth table and example load, count-up, count-down sequence provided in Fig. 12.106 explains how the 74192 works in greater detail.

74192 presettable decade (BCD) up/down counter



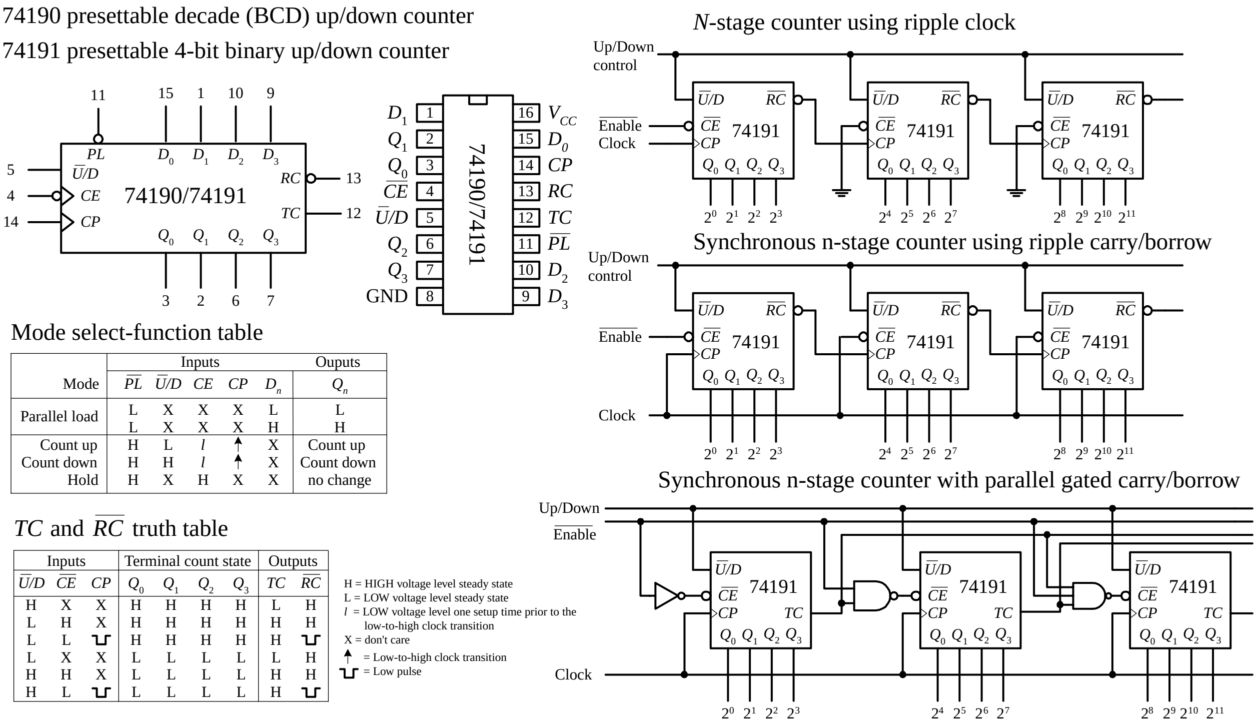
Mode	Inputs								Ouputs					
	MR_1	$\overline{PL}$	C_{pU}	C_{pD}	D_0	D_1	D_2	D_3	Q_0	Q_1	Q_2	Q_3	$\overline{TC}_U$	$\overline{TC}_D$
Reset	H	X	X	L	X	X	X	X	L	L	L	L	H	L
	H	X	X	H	X	X	X	X	L	L	L	L	H	H
Parallel load	L	L	X	L	L	L	L	L	L	L	L	L	H	L
	L	L	L	X	H	H	H	H	L	L	L	L	H	H
	L	L	H	X	H	H	H	H	$Q_n = D_n$				L	H
	H	L	H	H	L	L	L	L	$Q_n = D_n$				H	H
count up	H	H	↑	H	X	X	X	X	Count up				H	H
count down	L	H	H	↑	X	X	X	X	Count down				H	H

H = HIGH voltage level; L = LOW voltage level; X = don't care; ↑ = LOW-to-HIGH clock transistion

FIGURE 12.106

74190 Presettable Decade (BCD or MOD-10) Synchronous Up/Down Counter and the 74191 Presettable 4-Bit (MOD-16) Synchronous Up/Down Counter

The 74190 and the 74191 do basically the same things as the 74192 and 74193, but the input and output pins, as well as the operating modes, are a bit different. (The 74190 and the 74191 have the same pinouts and operating modes—the only difference is the maximum count.) Like the previous synchronous counters, these counters can be preset to any count by using the parallel load ($\overline{PL}$) operation. However, unlike the previous synchronous counters, to count up or down requires using a single input, $\overline{U}/D$. When $\overline{U}/D$ is set low, the counter counts up; when $\overline{U}/D$ is high, the counter counts down. A clock enable input ($\overline{CE}$) acts to enable or disable the counter. When $\overline{CE}$ is low, the counter is enabled. When $\overline{CE}$ is high, counting stops, and the current count is held fixed at the Q_0 to Q_3 outputs. Unlike the previous synchronous counters, the 74190 and the 74191 use a single terminal count output (TC) to indicate when the maximum or minimum count has occurred and the counter is about to recycle. In count-down mode, TC is normally low but goes high when the counter reaches zero (for both the 74190 and 74191). In count-up mode, TC is normally low but goes high when the counter reaches 9 (for the 74190) or reaches 15 (for the 74191). The ripple-clock output ($\overline{RC}$) follows the input clock (CP) whenever TC is high. This means, for example, that in count-down mode, when the count reaches zero, $\overline{RC}$ will go low when CP goes low. The $\overline{RC}$ output can be used as a clock input to the next higher stage of a multistage counter. This, however, leads to a multistage counter that is not truly synchronous because of the small propagation delay from CP to $\overline{RC}$ of each counter. To make a multistage counter that is truly synchronous, you must tie each IC's clock to a common clock input line. You use the TC output to inhibit each successive stage from counting until the previous stage is at its terminal count. The figure below shows various asynchronous (ripple-like) and synchronous multistage counters build from 74191 ICs.


74190 presettable decade (BCD) up/down counter
74191 presettable 4-bit binary up/down counter
11 15 1 10 9
PL D0 D1 D2 D3
5 U/D
4 CE
14 CP
74190/74191
RC 13
TC 12
Q0 Q1 Q2 Q3
3 2 6 7
D1 1, Q1 2, Q0 3, CE 4, U/D 5, Q2 6, Q3 7, GND 8
16 VCC, 15 D0, 14 CP, 13 RC, 12 TC, 11 PL, 10 D2, 9 D3
74190/74191
N-stage counter using ripple clock
Up/Down control
Enable
Clock
U/D RC CE CP 74191 Q0 Q1 Q2 Q3
2^0 2^1 2^2 2^3
2^4 2^5 2^6 2^7
2^8 2^9 2^10 2^11
Synchronous n-stage counter using ripple carry/borrow
Up/Down control
Enable
Clock
Synchronous n-stage counter with parallel gated carry/borrow
Up/Down
Enable
Clock
U/D CE CP TC 74191 Q0 Q1 Q2 Q3

Mode select-function table

Mode	Inputs					Ouputs
	$\overline{PL}$	$\overline{U}/D$	$\overline{CE}$	CP	D_n	Q_n
Parallel load	L	X	X	X	L	L
	L	X	X	X	H	H
Count up	H	L	*l*	↑	X	Count up
Count down	H	H	*l*	↑	X	Count down
Hold	H	X	H	X	X	no change

TC and $\overline{RC}$ truth table

Inputs			Terminal count state				Outputs	
$\overline{U}/D$	$\overline{CE}$	CP	Q_0	Q_1	Q_2	Q_3	TC	$\overline{RC}$
H	X	X	H	H	H	H	L	H
L	H	X	H	H	H	H	H	H
L	L	⊔	H	H	H	H	H	⊔
L	X	X	L	L	L	L	L	H
H	H	X	L	L	L	L	H	H
H	L	⊔	L	L	L	L	H	⊔

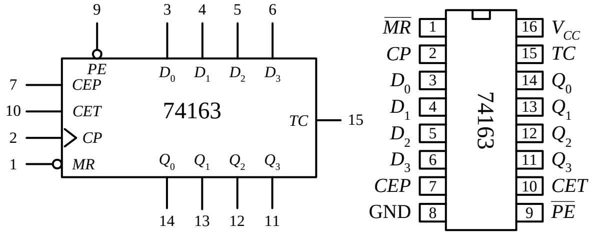
H = HIGH voltage level steady state
L = LOW voltage level steady state
l = LOW voltage level one setup time prior to the low-to-high clock transition
X = don't care
↑ = Low-to-high clock transition
⊔ = Low pulse

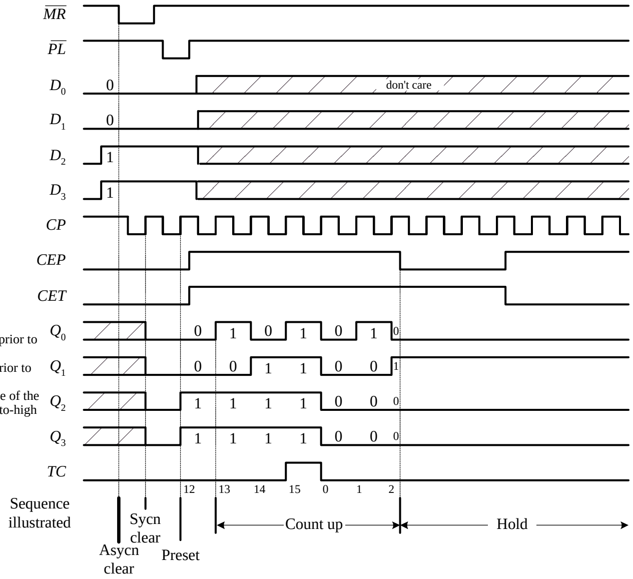
FIGURE 12.107

74163 Presettable 4-Bit (MOD-16) Synchronous Up/Down Counter

The 74160 and 74163 resemble the 74190 and 74191 but require no external gates when used in multistage counter configurations. Instead, you simply cascade counter ICs together, as shown in the last figure below. For both devices, a count can be preset by applying the desired count to the D_0 to D_3 inputs and then applying a low to the parallel enable input ($\overline{PE}$)—the input number is loaded into the counter on the next low-to-high clock transition. The master reset ($\overline{MR}$) is used to force all Q output low, regardless of the other input signals. The two clock enable inputs (CEP and CET) must be high for counting to begin. The terminal count output (TC) is forced high when the maximum count is reached but will be forced low if CET goes low. This is an important feature that makes the multistage configuration synchronous, while avoiding the need for external gating. The truth tables along with the example load, count-up, count-down timing sequences below should help you better understand how these two devices work.

74163 Synchronous 4-bit binary (MOD-16) up counter



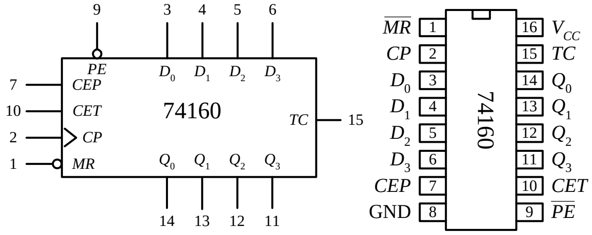


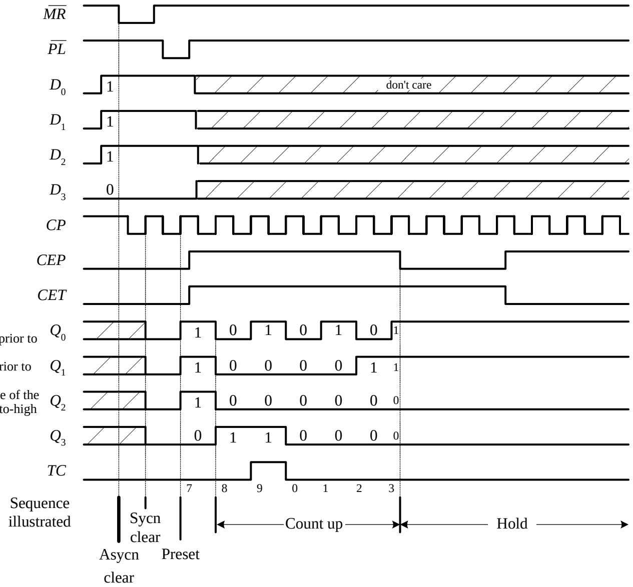
Mode	Inputs						Outputs	
	$\overline{MR}$	CP	CEP	CET	$\overline{PE}$	D_n	Q_n	TC
Reset (Clear)	*l*	↑	X	X	X	X	L	L
Parallel load	h(d) h(d)	↑ ↑	X X	X X	*l* *l*	*l* h	L H	L (b)
Count up	h(d)	↑	h	h	h(d)	X	count	(b)
Hold	h(d) h(d)	X X	*l* (c) X	X *l* (c)	h(d) h(d)	X X	q_n q_n	(b) L

H = High voltage level steady state
L = Low voltage level steady state
h = High voltage level one setup time prior to the low-to-high clock transition
l = Low voltage level one setup time prior to the low-to-high clock transition
q = Lower case letters indicate the state of the referenced output prior to the low-to-high clock transition
↑ = Low-to-high clock transition

Notes
(b) *TC* output is high when *CET* is high and the counter is at terminal count (HHHH).
(c) The high-to-low transition of $\overline{CEP}$ or $\overline{CET}$ should only occur while *CP* is high for conventional operation.
(d) The low-to-high transition of $\overline{PE}$ or $\overline{MR}$ should only occur while *CP* is high for conventional operation.

74160 Synchronous decade (BCD) up counter



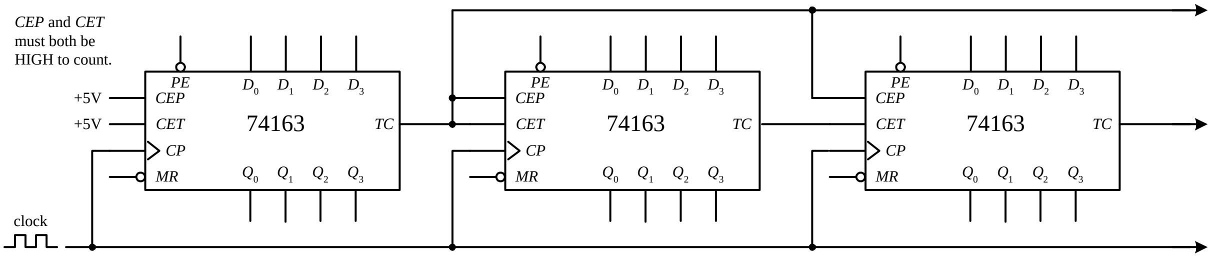


Mode	Inputs						Outputs	
	$\overline{MR}$	CP	CEP	CET	$\overline{PE}$	D_n	Q_n	TC
Reset (Clear)	L	X	X	X	X	X	L	L
Parallel load	H H	↑ ↑	X X	X X	*l* *l*	*l* h	L H	L (b)
Count up	H	↑	h	h	h(d)	X	count	(b)
Hold	H H	X X	*l* (c) X	X *l* (c)	h(d) h(d)	X X	q_n q_n	(b) L

H = High voltage level steady state
L = Low voltage level steady state
h = High voltage level one setup time prior to the low-to-high clock transition
l = Low voltage level one setup time prior to the low-to-high clock transition
q = Lower case letters indicate the state of the referenced output prior to the low-to-high clock transition
↑ = Low-to-high clock transition

Notes
(b) *TC* output is high when *CET* is high and the counter is at terminal count (HLLH).
(c) The high-to-low transition of $\overline{CEP}$ or $\overline{CET}$ should only occur while *CP* is high for conventional operation.
(d) The low-to-high transition of $\overline{PE}$ or $\overline{MR}$ should only occur while *CP* is high for conventional operation.

Synchronous multistage counter using the 74163


CEP and *CET* must both be HIGH to count.

74160 synchronous decade counters can also be cascaded together in this multistage configuration.

FIGURE 12.108

Asynchronous Counter Applications

74LS90: Divide-by-*n* frequency counters

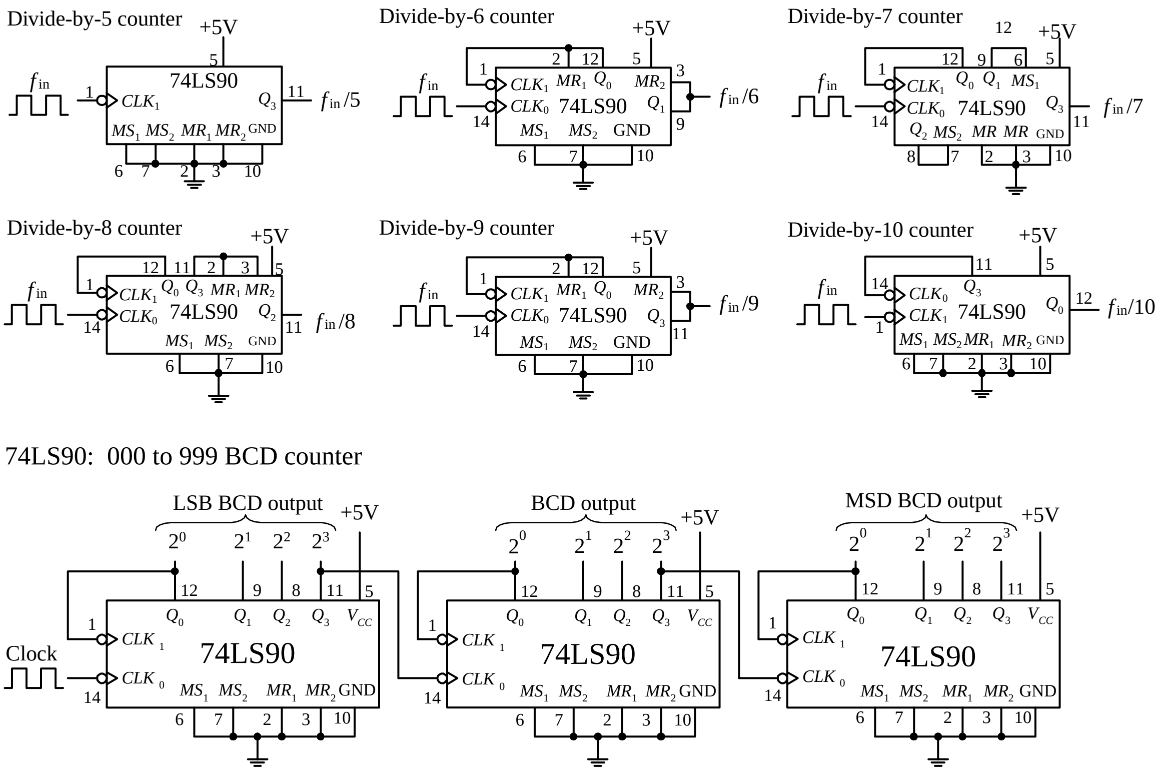
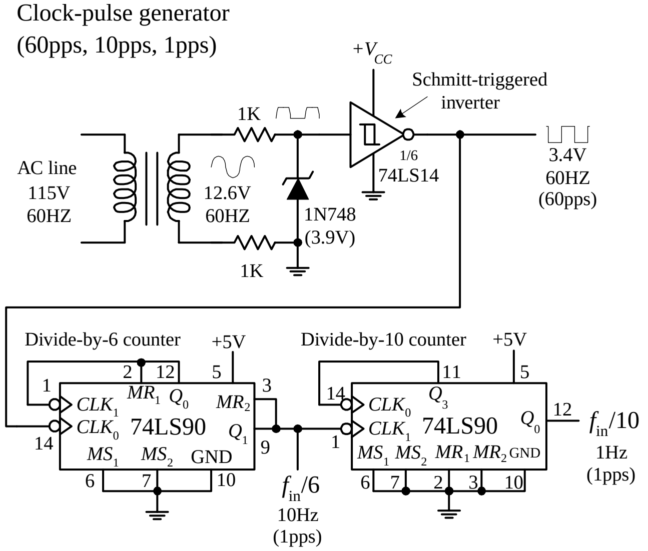


FIGURE 12.109

60-Hz, 10-Hz, 1-Hz Clock-Pulse Generator

FIGURE 12.110



This simple clock-pulse generator provides a unique way to generate 60-, 10-, and 1-Hz clock signals that can be used in applications that require real-time counting. The basic idea is to take the characteristic 60-Hz ac line voltage (from the wall socket) and convert it into a lower-voltage square wave of the same frequency. (Note that countries other than the United States typically use 50 Hz instead of 60 Hz. This circuit, therefore, will not operate as planned if used overseas.) First, the ac line voltage is stepped down to 12.6 V by the transformer. The negative-going portion of the 12.6-V ac voltage is removed by the zener diode (acts as a half-wave rectifier). At the same time, the zener diode clips the positive-going signal to a level equal to its reverse breakdown voltage (3.9 V). This prevents the Schmitt-triggered inverter from receiving an input level that exceeds its maximum input rating. The Schmitt-triggered inverter takes the rectified/chipped sine wave and converts it into a true square wave. The Schmitt trigger's output goes low (~0.2 V) when the input voltage exceeds its positive threshold voltage V_{T^+} (~1.7 V) and goes high (~3.4 V) when its input falls below its negative threshold voltage V_{T^-} (~0.9 V). From the inverter's output, you get a 60-Hz square wave (or a clock signal beating out 60 pulses per second). To get a 10-Hz clock signal, you slap on a divide-by-6 counter. To get a 1-Hz signal, you slap a divide-by-10 counter onto the output of the divide-by-6 counter.

3-digit decimal timer/relay-driver setup to count to 600 and then stop

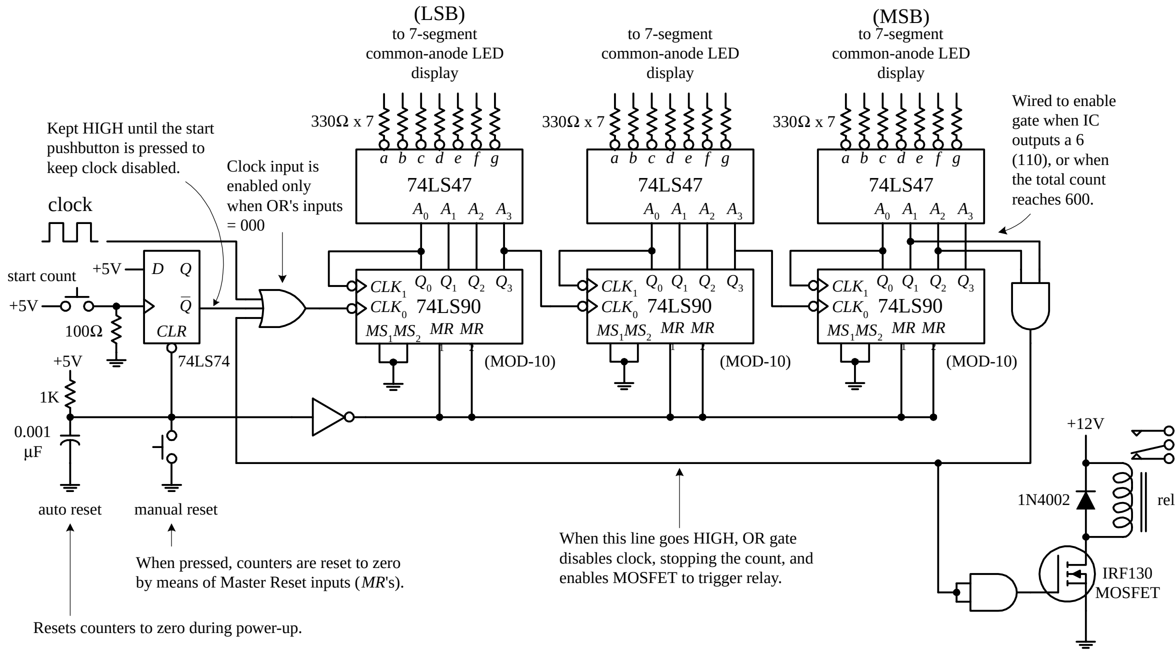

(LSB)
to 7-segment common-anode LED display
to 7-segment common-anode LED display
(MSB)
to 7-segment common-anode LED display
330Ω x 7
330Ω x 7
330Ω x 7
a b c d e f g
74LS47
A_0 A_1 A_2 A_3
Wired to enable gate when IC outputs a 6 (110), or when the total count reaches 600.
Kept HIGH until the start pushbutton is pressed to keep clock disabled.
Clock input is enabled only when OR's inputs = 000
clock
start count
+5V
+5V
D Q
$\overline{Q}$
CLR
100Ω
74LS74
CLK_1 Q_0 Q_1 Q_2 Q_3
CLK_0 74LS90
MS_1 MS_2 MR MR
(MOD-10)
1K
0.001 µF
auto reset
manual reset
+12V
1N4002
relay
IRF130 MOSFET
When this line goes HIGH, OR gate disables clock, stopping the count, and enables MOSFET to trigger relay.
When pressed, counters are reset to zero by means of Master Reset inputs (MR's).
Resets counters to zero during power-up.

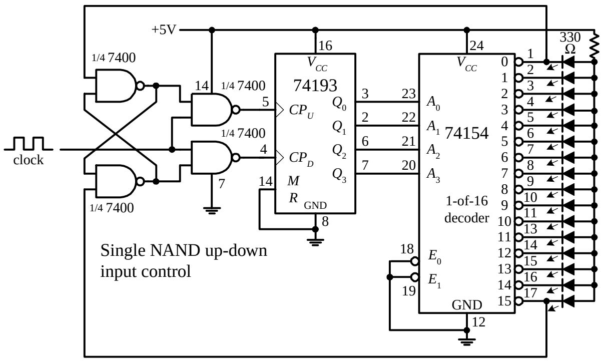
FIGURE 12.111

In the circuit above, three 74LS90 MOD-10 counter ICs are used to create a three-digit (decimal) counter. Features to note here includes an autoreset *RC* network that acts to reset counters during power-up via the master reset inputs. Before the count begins, the D flip-flop's $\overline{Q}$ output is held high, disabling the clock from reaching the first counter's clock input. When the pushbutton switch is closed, the flip-flop's $\overline{Q}$ output goes low, enabling the first counter to count. The BCD outputs of each counter are feed through separate BCD-to-seven-segment decoder/driver ICs, which in turn drive the LED displays. The far-left counter's output represents the count's LSB, while the far-right counter's output represents the count's MSB. As shown, the last counter's output is wired so that when a count of 600 is reached, an AND gate is enabled, causing the three-input OR gate to disable the clock (stop count) while also triggering a relay. To reset the counter, the manual reset switch is momentarily closed.

Synchronous Counter Applications

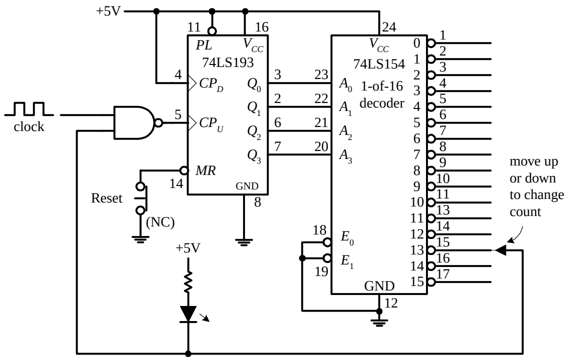
FIGURE 12.112

74193: Up-down counter/flasher circuit

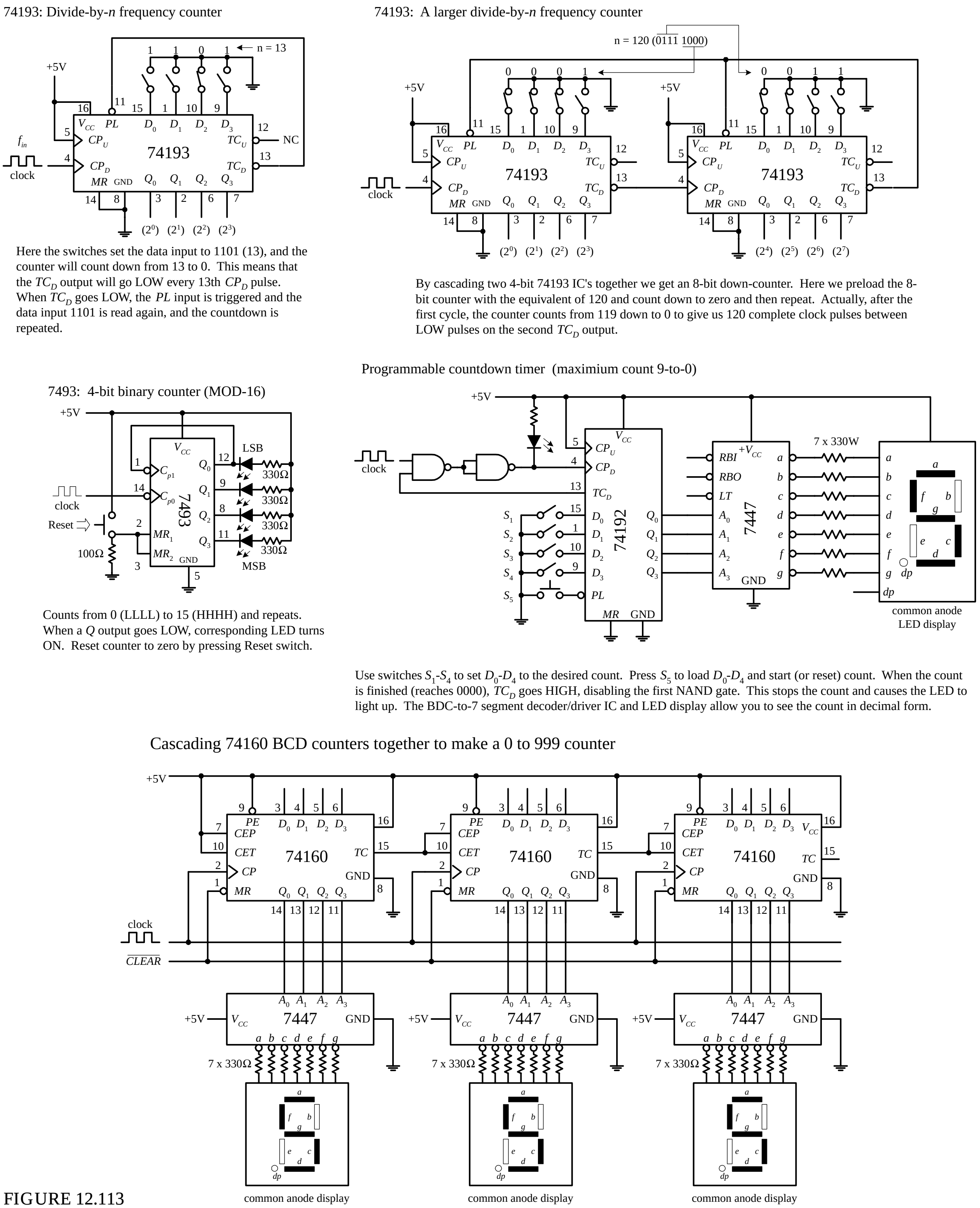


The 74193 is made to count up from 0000 to 1111 and is then swtiched to count down from 1111 to 0000, and then switched to count up again, etc. The NAND gates network provide the count-up and count-down control, while the 74154 1-of-16 decoder accepts the binary count at its address inputs, and then forces an output that corresponds to the address input LOW, thus turning on corresponding LED.

74193: Count up to any number between 0 and 15 and halt



This circuit counts up to a desired number from 0 to 15 and then halts, lighting an LED in the process. A 1-of-16 decoder is used to convert the 4-bit binary output from the counter IC into a single output corresponding to the 4-bit binary number. Here, the circuit is wired to count up to 13 . When the count reaches 13 (all outputs of the decoder are HIGH except the "13" output) the NAND gate is disabled, preventing the clock signal from reaching CP_U.



74193: Divide-by-*n* frequency counter

Here the switches set the data input to 1101 (13), and the counter will count down from 13 to 0. This means that the TC_D output will go LOW every 13th CP_D pulse. When TC_D goes LOW, the PL input is triggered and the data input 1101 is read again, and the countdown is repeated.

74193: A larger divide-by-*n* frequency counter

By cascading two 4-bit 74193 IC's together we get an 8-bit down-counter. Here we preload the 8-bit counter with the equivalent of 120 and count down to zero and then repeat. Actually, after the first cycle, the counter counts from 119 down to 0 to give us 120 complete clock pulses between LOW pulses on the second TC_D output.

7493: 4-bit binary counter (MOD-16)

Counts from 0 (LLLL) to 15 (HHHH) and repeats. When a Q output goes LOW, corresponding LED turns ON. Reset counter to zero by pressing Reset switch.

Programmable countdown timer (maximium count 9-to-0)

Use switches S_1-S_4 to set D_0-D_4 to the desired count. Press S_5 to load D_0-D_4 and start (or reset) count. When the count is finished (reaches 0000), TC_D goes HIGH, disabling the first NAND gate. This stops the count and causes the LED to light up. The BDC-to-7 segment decoder/driver IC and LED display allow you to see the count in decimal form.

Cascading 74160 BCD counters together to make a 0 to 999 counter

FIGURE 12.113

12.7.3 A Note on Counters with Displays

If you want to build a fairly sophisticated counter that can display many digits, the previous techniques are not worth pursuing—there are simply too many discrete components to work with (e.g., separate seven-segment decoder/driver for each digit). A common alternative approach is to use a microcontroller that functions both as a counter and a display driver. What microcontrollers can do that discrete circuits have a hard time doing is multiplexing a display. In a multiplexed system, corresponding segments of each digit of a multidigit display are linked together, while the common lines for each digit are brought out separately. Right away you can see that the number of lines is significantly reduced; a nonmultiplexed 7-segment 4-digit display has 28 segment lines and 4 common lines, while the 4-digit multiplexed display only has 7 + 4 = 11 lines. The trick to multiplexing involves flashing each digit, one after the other (and recycling), in a fast enough manner to make it appear that the display is continuously lit. In order to multiplex, the microcontroller's program must supply the correct data to the segment lines at the same time that it enables a given digit via a control signal sent to the common lead of that digit. I will talk about multiplexing in greater detail in Appendix H.

Another approach used to create multidigit counters is to use a multidigit counter / display driver IC. One such IC is the ICM7217A four-digit LED display programmable up / down counter made by Intersil. This device is typically used in hardwired applications where thumbwheel switches are used to load data and SPDT switches are used to control the chip. The ICM7217A provides multiplexed seven-segment LED display outputs that are used to drive common-cathode displays.

ICM7217A (Intersil) 4-Digit LED Display, Programmable Up/Down Counter

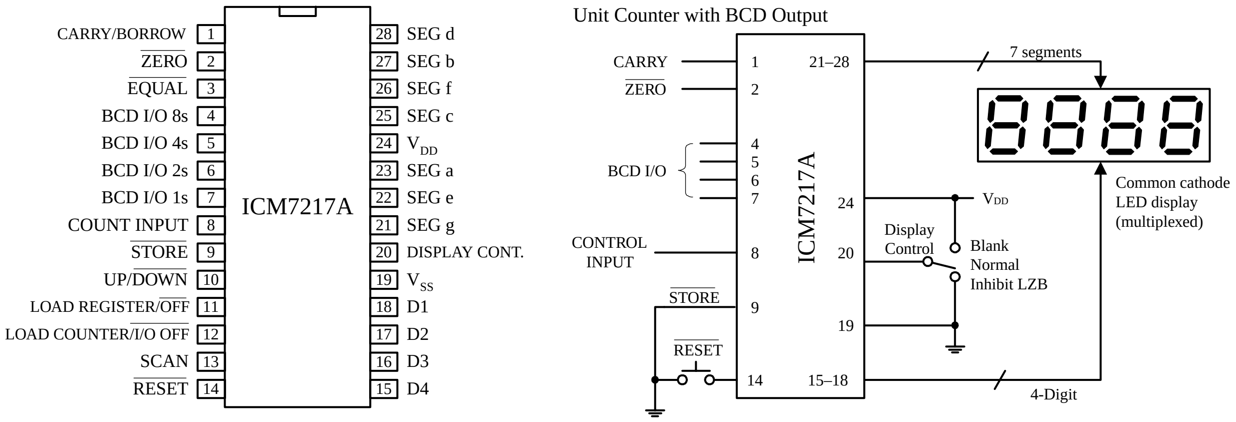


FIGURE 12.114

A simple application of the ICM7217A is a four-digit unit counter shown in the figure above. If you are interested in knowing all the specifics of how this counter works, along with learning about other applications for this device, check out Intersil's data sheets via the Internet (*www.intersil.com*). It is better to learn from the maker in this case. Also, check out the other counter / display driver ICs Intersil has to offer. Other manufacturers produce similar devices, so check out their Web sites as well.

12.8 Shift Registers

Data words traveling through a digital system frequently must be temporarily held, copied, and bit-shifted to the left or to the right. A device that can be used for such applications is the *shift register.* A shift register is constructed from a row of flip-flops connected so that digital data can be shifted down the row either in a left or right direction. Most shift registers can handle parallel movement of data bits as well as serial movement and also can be used to covert from parallel to serial or from serial to parallel. The figure below shows the three types of shift register arrangements: serial-in / serial-out, parallel-in / parallel-out, parallel-in / serial-out, and serial-in / parallel out.

Block diagrams of the serial-in/serial-out, parallel-in/serial-out and serial-in/parallel-out shift registers

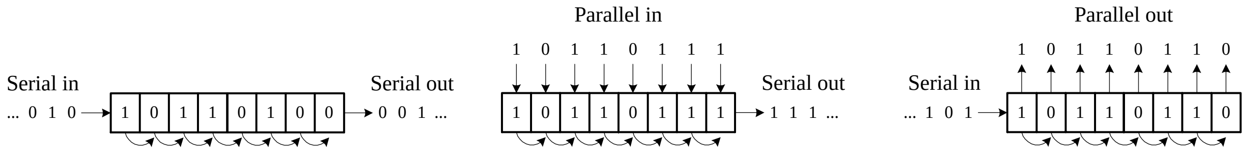


FIGURE 12.115

12.8.1 Serial-In/Serial-Out Shifter Registers

The figure below shows a simple 4-bit serial-in / serial-out shift register made from D flip-flops. Serial data are applied to the *D* input of flip-flop 0. When the clock line receives a positive clock edge, the serial data are shifted to the right from flip-flop 0 to flip-flop 1. Whatever bits of data were present at flip-flop 2s, 3s, and 4s outputs are shifted to the right during the same clock pulse. To store a 4-bit word into this register requires four clock pulses. The rightmost circuit shows how you can rewire the flip-flops to make a shift-left register. To make larger bit-shift registers, more flip-flops are added (e.g., an 8-bit shift register would require 8 flip-flops cascaded together).

Simple 4-bit serial-in/serial-out shift registers

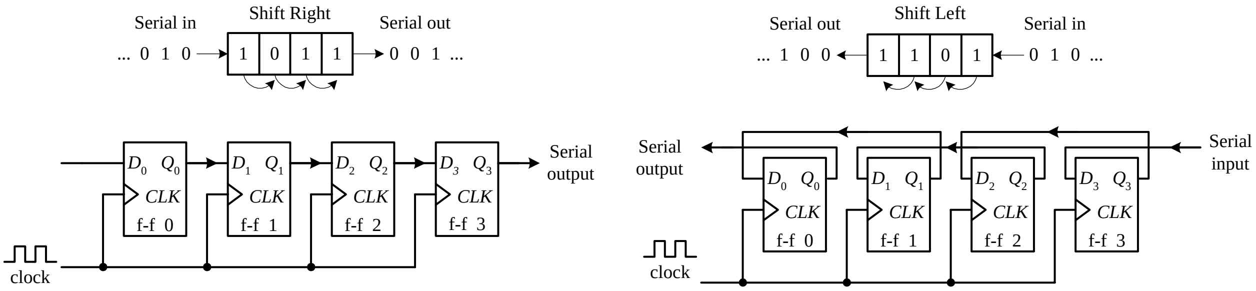


FIGURE 12.116

12.8.2 Serial-In/Parallel-Out Shift Registers

Figure 12.117 shows a 4-bit serial-in / parallel-out shift register constructed from D flip-flops. This circuit is essentially the same as the previous serial-in / serial-out shift register, except now you attach parallel output lines to the outputs of each flip-flop as shown. Note that this shift register circuit also comes with an active-low clear input ($\overline{CLR}$) and a strobe input that acts as a clock enable control. The timing diagram below shows a sample serial-to-parallel shifting sequence.

4-bit serial-in/parallel-out shift register

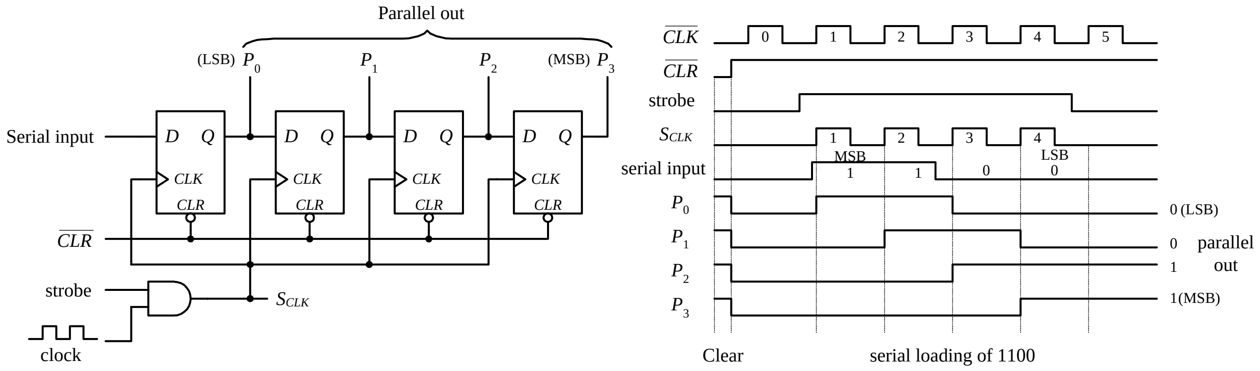


FIGURE 12.117

12.8.3 Parallel-In/Serial-Out Shift Register

Constructing a 4-bit parallel-to-serial shift register from D flip-flops requires some additional control logic, as shown in the circuit below. Parallel data must first be loaded into the *D* inputs of all four flip-flops. To load data, the SHIFT/$\overline{\text{LOAD}}$ is made low. This enables the AND gates with "X" marks, allowing the 4-bit parallel input word to enter the D_0–D_3 inputs of the flip-flops. When a clock pulse is applied during this load mode, the 4-bit parallel word is latched simultaneously into the four flip-flops and appears at the Q_0–Q_3 outputs. To shift the latched data out through the serial output, the SHIFT/$\overline{\text{LOAD}}$ line is made high. This enables all unmarked AND gates, allowing the latched data bit at the *Q* output of a flip-flop to pass (shift) to the *D* input of the flip-flop to the right. In this shift mode, four clock pulses are required to shift the parallel word out the serial output.

Parallel-to-serial shift register

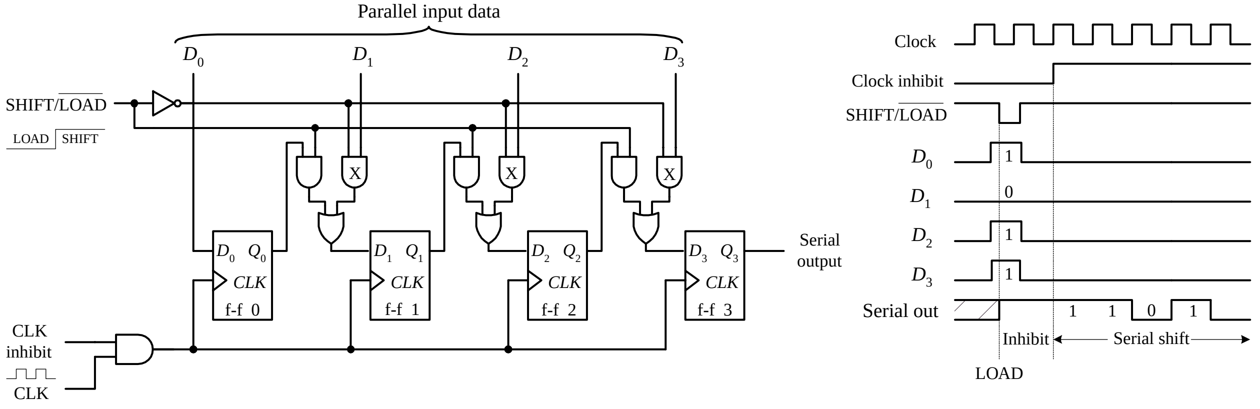


FIGURE 12.118

12.8.4 Ring Counter (Shift Register Sequencer)

The ring counter (shift register sequencer) is a unique type of shift register that incorporates feedback from the output of the last flip-flop to the input of the first flip-flop. Figure 12.119 shows a 4-bit ring counter made from D-type flip-flops. In this circuit, when the $\overline{\text{START}}$ input is set low, Q_0 is forced high by the active-low preset, while Q_1, Q_2, and Q_3 are forced low (cleared) by the active-low clear. This causes the binary word 1000 to be stored within the register. When the $\overline{\text{START}}$ line is brought low, the data bits stored in the flip-flops are shifted right with each positive clock edge. The data bit from the last flip-flop is sent to the D input of the first flip-flop. The shifting cycle will continue to recirculate while the clock is applied. To start fresh cycle, the $\overline{\text{START}}$ line is momentarily brought low.

Ring counter using positive edge-triggered D flip-flops

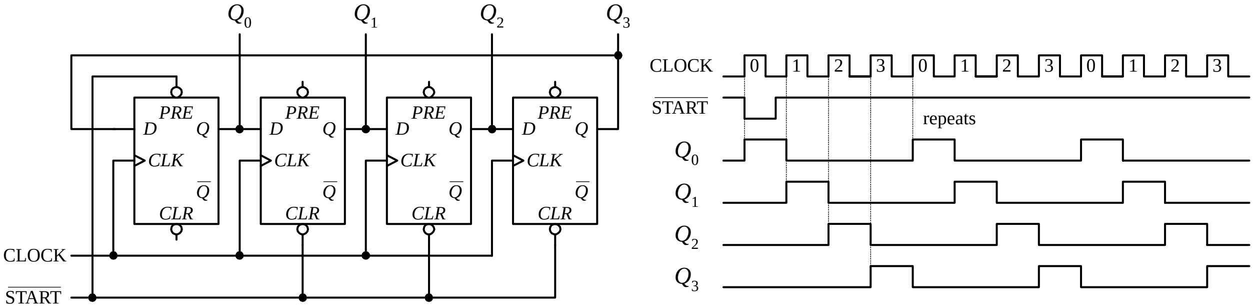


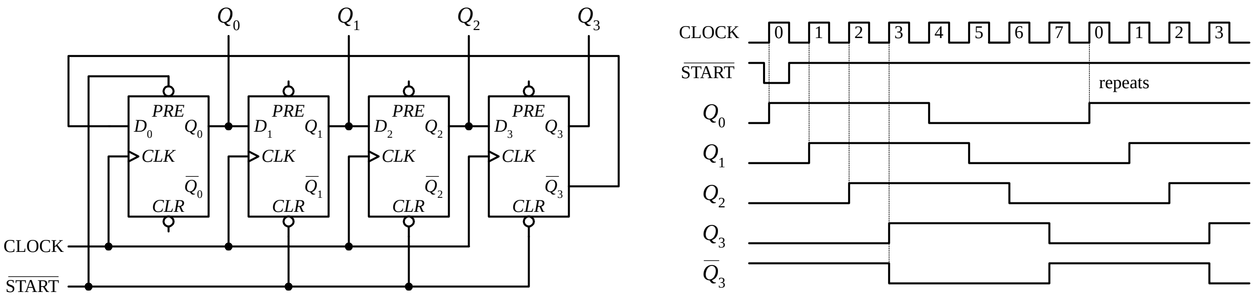
FIGURE 12.119

12.8.5 Johnson Shift Counter

The Johnson shift counter is similar to the ring counter except that its last flip-flop feeds data back to the first flip-flop from its inverted output ($\overline{Q}$). In the simple 4-bit Johnson shift counter shown below, you start out by applying a low to the $\overline{\text{START}}$ line, which sets presets Q_0 high and Q_1, Q_2, and Q_3 low—$\overline{Q}_3$ high. In other words, you load the register with the binary word 1000, as you did with the ring counter. Now, when you bring $\overline{\text{START}}$ line low, data will shift through the register. However, unlike the ring counter, the first bit sent back to the D_0 input of the first flip-flop will be high because feedback is from $\overline{Q}_3$ not Q_3. At the next clock edge, another high is fed back to D_0; at the next clock edge, another high is fed back; at the next edge, another high is fed back. Only after the fourth clock edge does a low get fed back (the 1 has shifted down to the last flip-flop and $\overline{Q}_3$ goes high). At this point, the shift register is full of 1s. As more clock pulses arrive, the feedback loop supplies lows to D_0 for the next four clock pulses. After that, the Q outputs of all the flip-flops are low while $\overline{Q}_3$ goes high. This high from $\overline{Q}_3$ is fed back to $\overline{D}_0$ during the next positive clock edge, and the cycle repeats. As you can see, the 4-bit Johnson shift counter has 8 output stages (which require 8 clock pulses to recycle), not 4, as was the case with the ring counter.

FIGURE 12.120

Johnson counter using positive edge-triggered D flip-flops



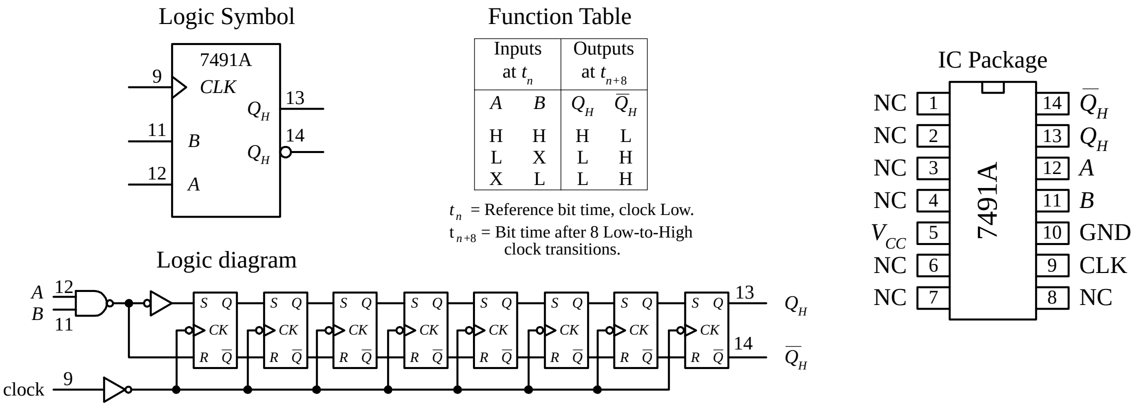
12.8.6 Shift Register ICs

I have just covered the basic theory of shift registers. Now let's take a look at practical shift register ICs that contain all the necessary logic circuitry inside.

7491A 8-Bit Serial-In/Serial-Out Shift Register

The 7491A is an 8-bit serial-in / serial-out shift register that consists of eight internally linked SR flip-flops. This device has positive edge-triggered inputs and a pair of data inputs (*A* and *B*) that are internally ANDed together, as shown in the logic diagram below. This type of data input means that for a binary 1 to be shifted into the register, both data inputs must be high. For a binary 0 to be shifted into the register, either input can be low. Data are shifted to the right at each positive clock edge.

7491A 8-bit serial-in/serial-out shift register IC



Inputs at t_n		Outputs at t_{n+8}	
A	*B*	Q_H	$\overline{Q}_H$
H	H	H	L
L	X	L	H
X	L	L	H

t_n = Reference bit time, clock Low.
t_{n+8} = Bit time after 8 Low-to-High clock transitions.

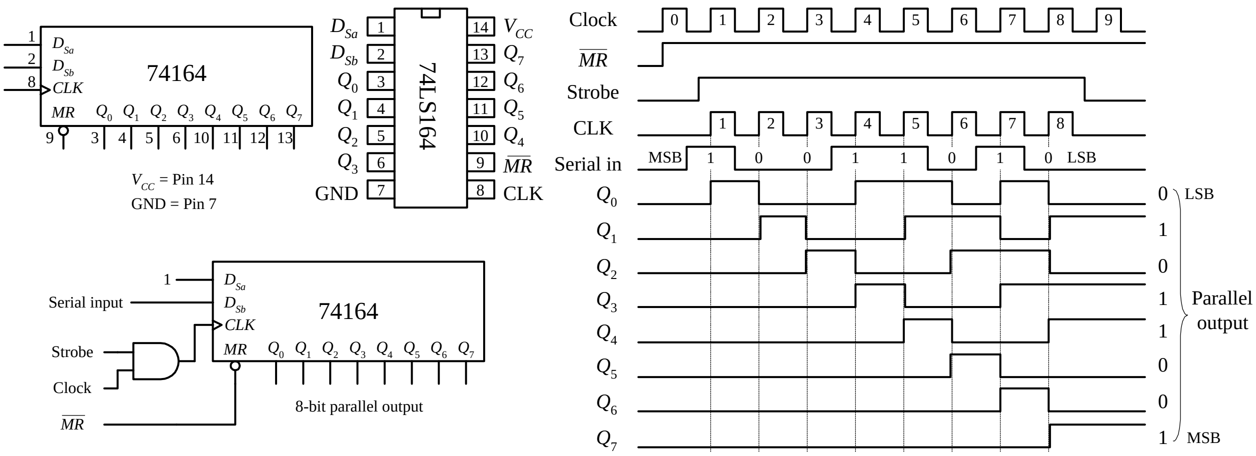
FIGURE 12.121

74164 8-Bit Serial-In/Parallel-Out Shift Register IC

The 74164 is an 8-bit serial-in / parallel-out shift register. It contains eight internally linked flip-flops and has two serial inputs D_{sa} and D_{sb} that are ANDed together. Like the 7491A, the unused serial input acts as an enable / disable control for the other serial input. For example, if you use D_{sa} as the serial input, you must keep D_{sb} high to allow data to enter the register, or you can keep it low to prevent data from entering

FIGURE 12.122

The 74164 8-bit serial-in/parallel-out shift register IC

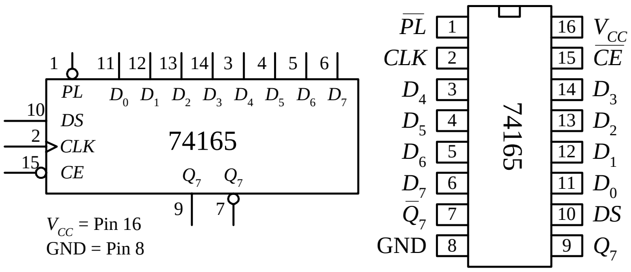


the register. Data bits are shifted one position to the right at each positive clock edge. The first data bit entered will end up at the Q_7 parallel output after the eighth clock pulse. The master reset ($\overline{MR}$) resets all internal flip-flops and forces the Q outputs low when it is pulsed low. In the sample circuit shown below, a serial binary number 10011010 (154_{10}) is converted into its parallel counterpart. Note the AND gate and strobe input used in this circuit. The strobe input acts as a clock enable input; when it is set high, the clock is enabled. The timing diagram paints the rest of the picture.

75165 8-Bit (Serial-In or Parallel-In)/Serial-Out Shift Register

The 75165 is a unique 8-bit device that can act as either a serial-to-serial shift register or as a parallel-to-serial shift register. When used as a parallel-to-serial shift register, parallel data are applied to the D_0–D_7 inputs and then loaded into the register when the parallel load input ($\overline{PL}$) is pulsed low. To begin shifting the loaded data out the serial output Q_7 (or $\overline{Q}_7$ if you want inverted bits), the clock enable input ($\overline{CE}$) must be set low to allow the clock signal to reach the clock inputs of the internal D-type flip-flops. When used as a serial-to-serial shift register, serial data are applied to the serial data input DS. A sample shift, load, and inhibit timing sequence is shown below.

74165 8-Bit (serial-in or parallel-in)/serial-out shift register



Operating Modes	Inputs					Q_n Register		Outputs	
	$\overline{PL}$	$\overline{CE}$	CLK	DS	D_0 - D_7	Q_0	Q_1 - Q_6	Q_7	$\overline{Q}_7$
Parallel load	L L	X X	X X	X X	L H	L H	L - L H - H	L H	H L
Serial shift	H H	L L	↑ ↑	l h	X X	L H	q_0 - q_5 q_0 - q_5	q_6 q_6	$\overline{q}_6$ $\overline{q}_6$
Hold ("do nothing")	H	H	X	X	X	q_0	q_1 - q_6	q_7	$\overline{q}_7$

H = High voltage level; h = High voltage level one setup time prior to the low-to-high clock transition; L = Low voltage level; l = Low voltage level one setup time prior to the low-to-high clock transiton; q_n = Lower case letters indicate the state of the referenced output one setup time prior to the low-to-high clock transition; X = Don't care; ↑ = Low-to-high clock transition.

Sample shift, load, and inhibit sequence for parallel load case

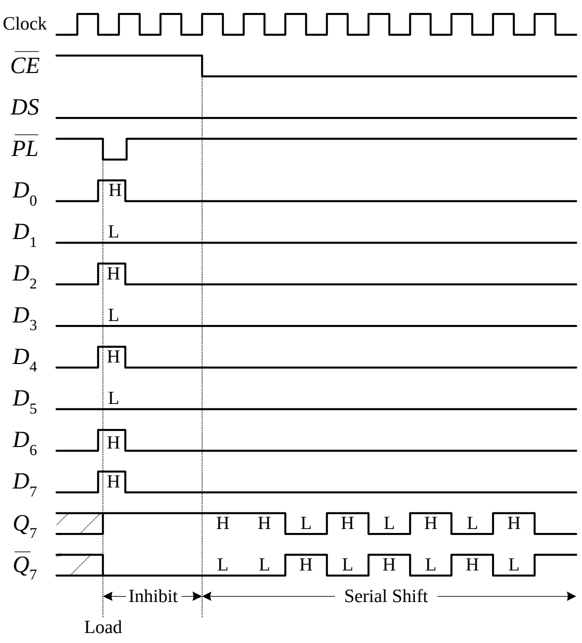


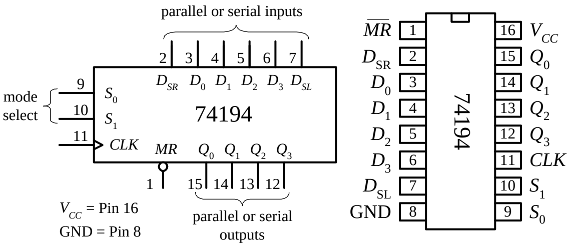
FIGURE 12.123

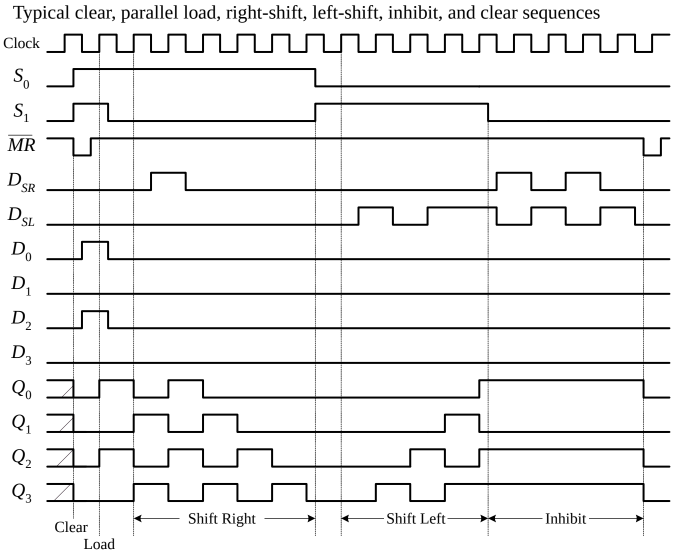
74194 Universal Shift Register IC

Figure 12.124 shows the 74194 4-bit bidirectional universal shift register. This device can accept either serial or parallel inputs, provide serial or parallel outputs, and shift left or shift right based on input signals applied to select controls S_0 and S_1. Serial data can be entered into either the serial shift-right input (D_{SR}) or the serial shift-left input (D_{SL}). Select control S_0 and S_1 are used to initiate either a hold (S_0 = low, S_1 = low), shift left (S_0 = low, S_1 = high), shift-right (S_0 = high, S_1 = low), or to parallel load (S_0 = high, S_1 = high) mode—a clock pulse must then be applied to shift or parallel load the data.

In parallel load mode (S_0 and S_1 are high), parallel input data are entered via the D_0 through D_3 inputs and are transferred to the Q_0 to Q_3 outputs following the next low-to-high clock transition. The 74194 also has an asynchronous master reset ($\overline{MR}$) input that forces all Q outputs low when pulsed low. To make a shift-right recirculating register, the Q_3 output is wired back to the D_{SR} input, while making S_0 = high and S_1 = low. To make a shift-left recirculating register, the Q_0 output is connected back to the D_{SL} input, while making S_0 = low and S_1 = high. The timing diagram below shows a typical parallel load and shifting sequence.

74194 4-bit bidirectional universal shift register





Operating Modes	Inputs							Outputs			
	CP	$\overline{MR}$	S_1	S_0	D_{SR}	D_{SL}	D_n	Q_0	Q_1	Q_2	Q_3
Reset (clear)	X	L	X	X	X	X	X	L	L	L	L
Hold (do nothing)	X	H	1^b	1^b	X	X	X	q_0	q_1	q_2	q_3
Shift Left	↑	H	h	1^b	X	l	X	q_1	q_2	q_3	L
	↑	H	h	1^b	X	h	X	q_1	q_2	q_3	H
Shift Right	↑	H	1^b	h	l	X	X	L	q_0	q_1	q_2
	↑	H	1^b	h	h	X	X	H	q_0	q_1	q_2
Parallel Load	↑	H	h	h	X	X	d_n	d_0	d_1	d_2	d_3

H = High voltage level; h = High voltage level one setup time prior to the low-to-high clock transition; L = Low voltage level; l = Low voltage level one setup time prior to the low-to-high clock transiton; d_n (q_n) = Lower case letters indicate the state of the referenced input (or output) one setup time prior to the low-to-high clock transition; X = Don't care; ↑ = Low-to-high clock transition.

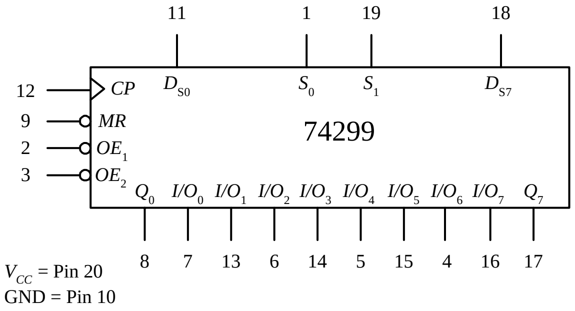
b = The high-to-low transition of S0 and S1 input should only take place while the clock is HIGH for convential operation.

FIGURE 12.124

74299 8-Bit Universal Shift/Storage Register with Three-State Interface

There are a number of shift registers that have three-state outputs—outputs that can assume either a high, low, or high impedance state (open-circuit or float state). These devices are commonly used as storage registers in three-state bus interface applications. An example 8-bit universal shift / storage register with three-state outputs is the 74299, shown in Fig. 12.125. This device has four synchronous operating modes that are selected via two select inputs, S_0 and S_1. Like the previous 74194 universal shift register, the 74299's select modes include shifting right, shifting left, holding, and parallel loading (see function table in Fig. 12.125). The mode-select inputs, serial data inputs (D_{S0} and D_{S7}) and the parallel-data inputs (I/O_0 through I/O_7) inputs are positive edge triggered. The master reset ($\overline{MR}$) input is an asynchronous active-low input that clears the register when pulsed low. The three-state bidirectional I/O port has three modes of operation: read register, load register, and disable I/O. The read-register mode allows data within the register to be available at the I/O outputs. This mode is selected by making both output-enable inputs ($\overline{OE}_1$ and $\overline{OE}_2$) low and making one or both select inputs low. The load-register mode sets up the register for a parallel load during the next low-to-high clock transition. This mode is selected by setting both select inputs high. Finally, the disable-I/O mode acts to disable the outputs (set to a high impedance state) when a high is applied to one or both of the output-enable inputs. This effectively isolates the register from the bus to which it is attached.

74299 8-bit universal shift/storage register with 3-state outputs



Operating Modes	Inputs							Outputs		
	$\overline{MR}$	CP	S_0	S_1	D_{S0}	D_{S7}	I/O_n	Q_0	Q_1 - Q_6	Q_7
Reset (clear)	L	H	X	X	X	X	X	L	L - L	L
Shift right	H	↑	h	*l*	*l*	X	X	L	q_0 - q_5	q_6
	H	↑	h	*l*	h	X	X	H	q_0 - q_5	q_6
Shift left	H	↑	*l*	h	X	*l*	X	q_1	q_0 - q_5	L
	H	↑	*l*	h	X	h	X	q_1	q_0 - q_5	H
Hold ("do nothing")	H	↑	*l*	*l*	X	X	X	q_0	q_1 - q_6	q_7
Parallel load	H	↑	h	h	X	X	*l*	L	L - L	L
	H	↑	h	h	X	X	h	H	H - H	H

3-state I/O port operating mode	Inputs					Inputs/Outputs
	$\overline{OE}_1$	$\overline{OE}_2$	S_0	S_1	Q_n (register)	I/O_0 -- I/O_7
Read register	L	L	L	X	L	L
	L	L	L	X	H	H
	L	L	X	L	L	L
	L	L	X	L	H	H
Load register	X	X	H	H	Q_n = I/O_n	I/O_n = inputs
Disable I/O	H	X	X	X	X	High Z
	X	H	X	X	X	High Z

H = High voltage level; *h* = High voltage level one setup time prior to the low-to-high clock transition; L = Low voltage level; *l* = Low voltage level one setup time prior to the low-to-high clock transiton; q_n = Lowercase letters indicate the state of the referenced output one setup time prior to the low-to-high clock transition; X = Don't care; ↑ = Low-to-high clock transition.

FIGURE 12.125

12.8.7 Simple Shift Register Applications

16-Bit Serial-to-Parallel Converter

A simple way to create a 16-bit serial-to-parallel converter is to join two 74164 8-bit serial-in/parallel-out shift registers, as shown below. To join the two ICs, simply wire the Q_7 output from the first register to one of the serial inputs of the second register. (Recall that the serial input that is not used for serial input data acts as an active-high enable control for the other serial input.) In terms of operation, when data are shifted out Q_7 of the first register (or data output D_7), they enter the serial input of the second (I have chosen D_{Sa} as the serial input) and will be presented to the Q_0 output of the second register (or data output D_8). For an input data bit to reach the Q_7 output of the second register (or data output D_{15}), 16 clock pulses must be applied.

Using two 74164s to create a 16-bit serial-to-parallel converter

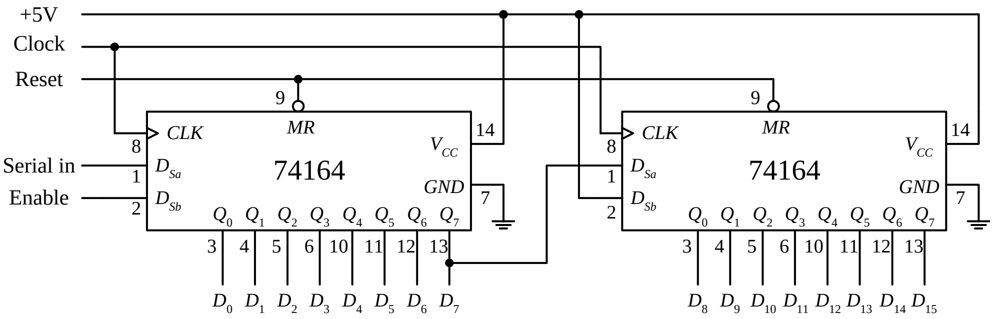

+5V
Clock
Reset
Serial in
Enable
9
MR
CLK
8
D_{Sa}
1
D_{Sb}
2
74164
V_{CC}
14
GND
7
Q_0 Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7
3 4 5 6 10 11 12 13
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
9
MR
CLK
8
D_{Sa}
1
D_{Sb}
2
74164
V_{CC}
14
GND
7
Q_0 Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7
3 4 5 6 10 11 12 13
D_8 D_9 D_{10} D_{11} D_{12} D_{13} D_{14} D_{15}

FIGURE 12.126

8-Bit Serial-to-Parallel Converter with Simultaneous Data Transfer

Figure 12.127 shows a circuit that acts as a serial-to-parallel converter that only outputs the converted 8-bit word when all 8-bits have been entered into the register. Here, a 74164 8-bit serial-in / parallel-out shift register is used, along with a 74HCT273 octal D-type flip-flop and a divide-by-8 counter. At each positive clock edge, the serial data are loaded into the 74164. After eight clock pulses, the first serial bit entered is shifted down to the 74164's Q_7 output, while the last serial bit entered resides at the 74164's Q_0 output. At the negative edge of the eighth clock pulse, the negative-edge triggered divide-by-8 circuit's output goes high. During this high transition, the data present on the inputs of the 74HCT273 (which hold the same data present at the 74164's Q outputs) are passed to the 74HCT273's outputs at the same time. (Think of the 74HCT273 as a temporary storage register that dumps its contents after every eighth clock pulse.)

8-Bit Serial-to-Parallel Data Converter

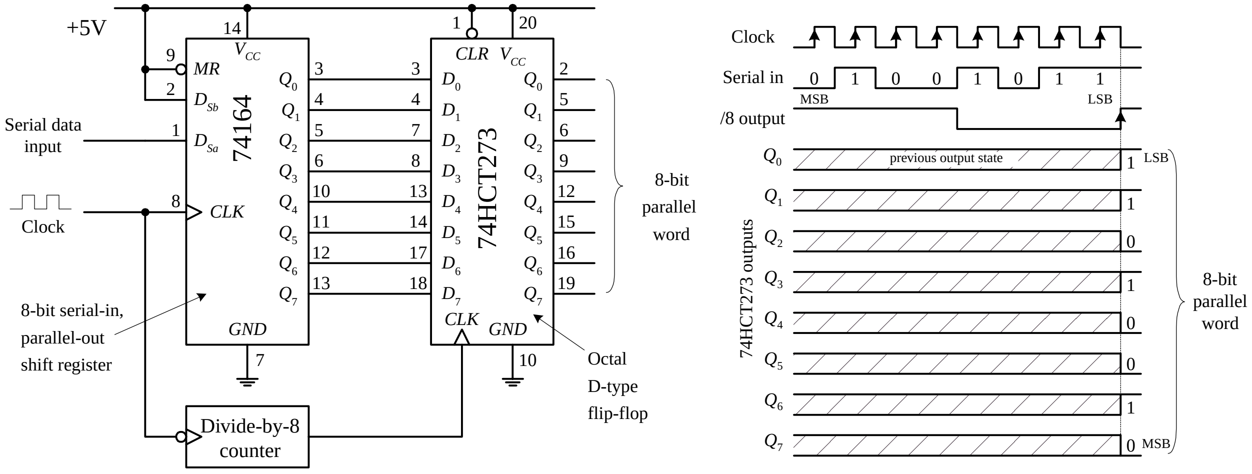


FIGURE 12.127

8-Bit Parallel-to-Serial Interface

Here, a 74165 8-bit parallel-to-serial shift register is used to accept a parallel ASCII word and convert it into a serial ASCII word that can be sent to a serial device. Recall that ASCII codes are only 7 bits long (e.g., the binary code for & is 010 0110). How do you account for the missing bit? As it turns out, most 8-bit devices communicating via serial ASCII will use an additional eighth bit for a special purpose, perhaps to act as a parity bit, or as a special function bit to enact a special set of characters. Often the extra bit is simply set low and ignored by the serial device receiving it. To keep things simple, let's set the extra bit low and assume that that is how the serial device likes things done. This means that you will set the D_0 input of the 74165 low. The MSB of the ASCII code will be applied to the D_1 input, while the LSB of the ASCII code will be applied to the D_7 input. Now, with the parallel ASCII word applied to the inputs of the register, when you pulse the parallel load line ($\overline{PL}$) low, the ASCII word, along with the "ignored bit," is loaded into the register. Next, you must enable the clock to allow the loaded data to be shifted out, serially, by setting the clock enable input ($\overline{CE}$) low for the duration it takes for the clock pulses to shift out the parallel word. After the eighth clock pulse (0 to 7), the serial device will have received all 8 serial data bits. Practically speaking, a microprocessor or microcontroller is necessary to provide the $\overline{CE}$ and $\overline{PL}$ lines with the necessary control signals to ensure that the register and serial device communicate properly.

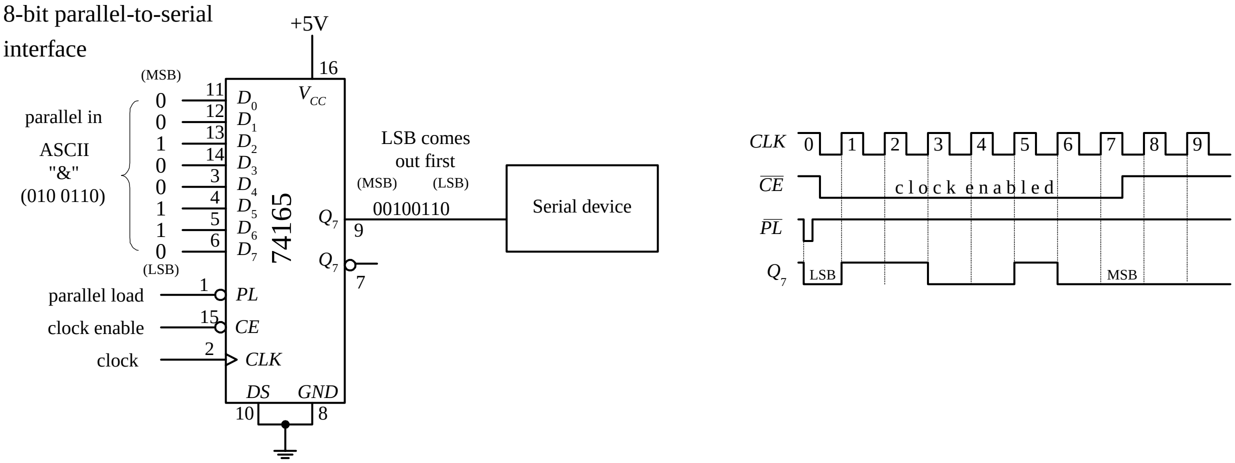


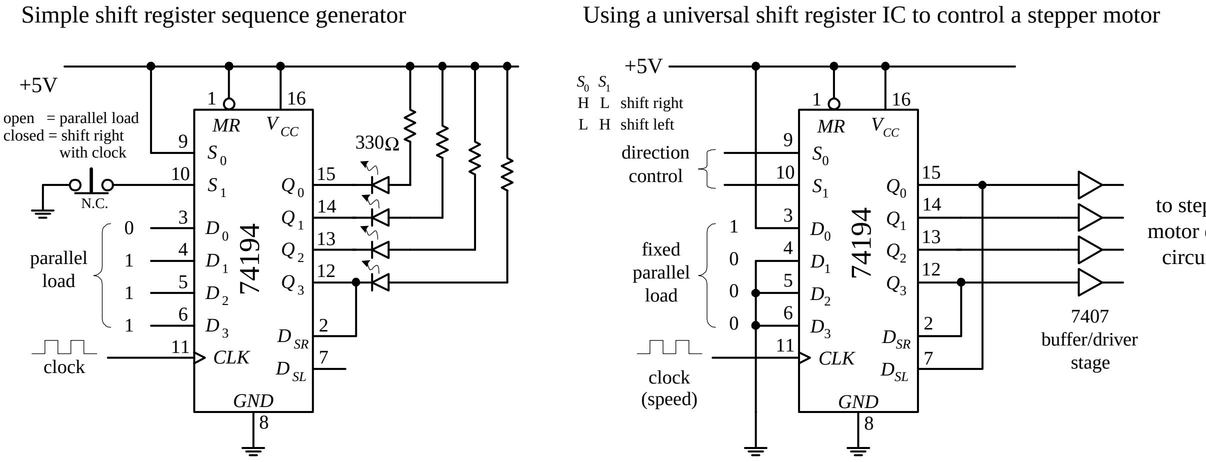
FIGURE 12.128

Recirculating Memory Registers

A *recirculating memory register* is a shift register that is preloaded with a binary word that is serially recirculated through the register via a feedback connection from the output to the input. Recirculating registers can be used for a number of things, from supplying a specific repetitive waveform used to drive IC inputs to driving output drivers used to control stepper motors.

In the leftmost circuit in Fig. 12.129, a parallel 4-bit binary word is applied to the D_0 to D_3 inputs of a 74194 universal shift register. When the S_1 select input is brought high (switch opened), the 4-bit word is loaded into the register. When the S_1 input is then brought low (switch closed), the 4-bit word is shifted in a serial fashion through the register, out Q_3, and back to Q_0 via the D_{SR} input (serial shift-right input) as positive clock edges arrive. Here, the shift register is loaded with 0111. As you begin shifting the bits through the register, a single low output will propagate down through high outputs, which in turn causes the LED attached to the corresponding low output to turn on. In other words, you have made a simple Christmas tree flasher.

FIGURE 12.129


Simple shift register sequence generator

Using a universal shift register IC to control a stepper motor

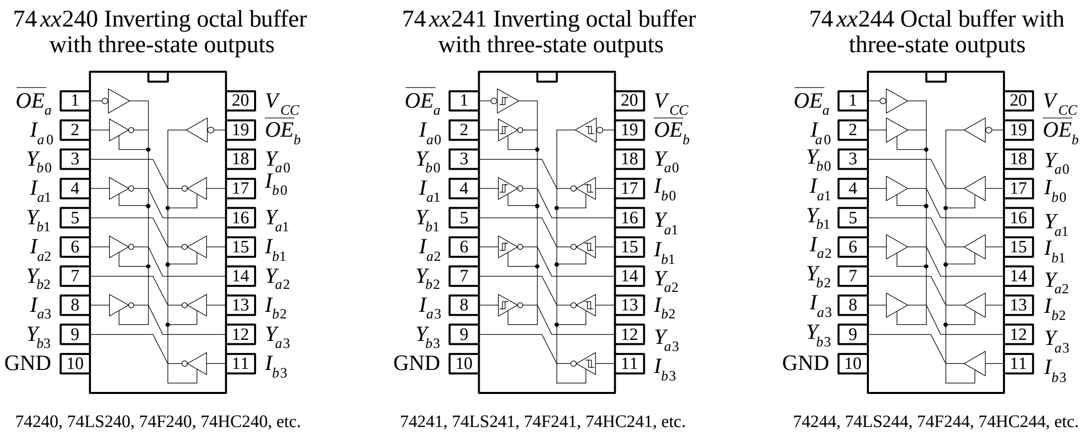
The rightmost circuit in Fig. 12.129 is basically the same thing as the last circuit. However, now the circuit is used to drive a stepper motor. Typically, a stepper motor has four stator coils that must be energized in sequence to make the motor turn a given angle. For example, to make a simple stepper motor turn clockwise, you must energize its stator coils 1, 2, 3, and 4 in the following sequence: 1000, 0100, 0010, 0001, 1000, etc. To make the motor go counterclockwise, apply the following sequence: 1000, 0001, 0010, 0100, 1000, etc. You can generate these simple firing sequences with the 74194 by parallel loading the D_0 to D_3 inputs with the binary word 1000. To output the clockwise firing sequence, simply shift bits to the right by setting S_0 = high and S_1 = low. As clock pulses arrive, the 1000 present at the outputs will then become 0100, then 0010, 0001, 1000, etc. The speed of rotation of the motor is determined by the clock frequency. To output the counterclockwise firing sequence, simply shift bits to the left by setting S_0 = low and S_1 = high. To drive steppers, it is typically necessary to use a buffer/driver interface like the 7407 shown below, as well a number of output transistors, not shown. Also, different types of stepper motors may require different firing sequences than the one shown here. Stepper motors and the various circuits used to drive them are discussed in detail in Chap. 13.

12.9 Three-State Buffers, Latches, and Transceivers

As you will see in a moment, digital systems that use microprocessors require that a number of different devices (e.g., RAM, ROM, I/O devices, etc.) share a common bus of some sort. For simple microprocessor systems, the data bus is often 8 bits wide (eight separate conductors). In order for devices to share the bus, only one device can be transmitting data at a time—the microprocessor decides which devices get access to the bus and which devices do not. In order for the microprocessor to control the flow of data, it needs help from an external register-type device. This device accepts a control signal issued by the microprocessor and responds by either allowing parallel data to pass or prohibiting parallel data to pass. Three popular devices used for such applications are the octal three-state buffer, octal latch/flip-flop, and the transceiver.

12.9.1 Three-State Octal Buffers

A three-state octal buffer is a device that when enabled, passes data present on its eight inputs to its outputs unchanged. When disabled, input data are prevented from reaching the outputs—the outputs are placed in a high-impedance state. This high-impedance state makes data bus sharing between various devices possible. The octal buffer also can provide additional sink or source current required to drive output devices. Three popular three-state octal buffers are shown below. The 74*xx*240 is a three-state inverting octal buffer, the 74*xx*241 is a three-state Schmitt-triggered inverting octal buffer, and the 74244 is a conventional three-state octal buffer. The enable/disable control for all three devices is the same. To enable all eight outputs (allow data to pass from I inputs to Y outputs), both output enable inputs, $\overline{OE}_a$ and $\overline{OE}_b$ must be set low. If you wish to enable only four outputs, make one output enable high while setting the other low (refer to ICs below to see which output enable controls which group of inverters). To disable all eight outputs, both output enables are set high. To disable only four outputs, set only one output enable input high.



74*xx*240 Inverting octal buffer with three-state outputs

74*xx*241 Inverting octal buffer with three-state outputs

74*xx*244 Octal buffer with three-state outputs

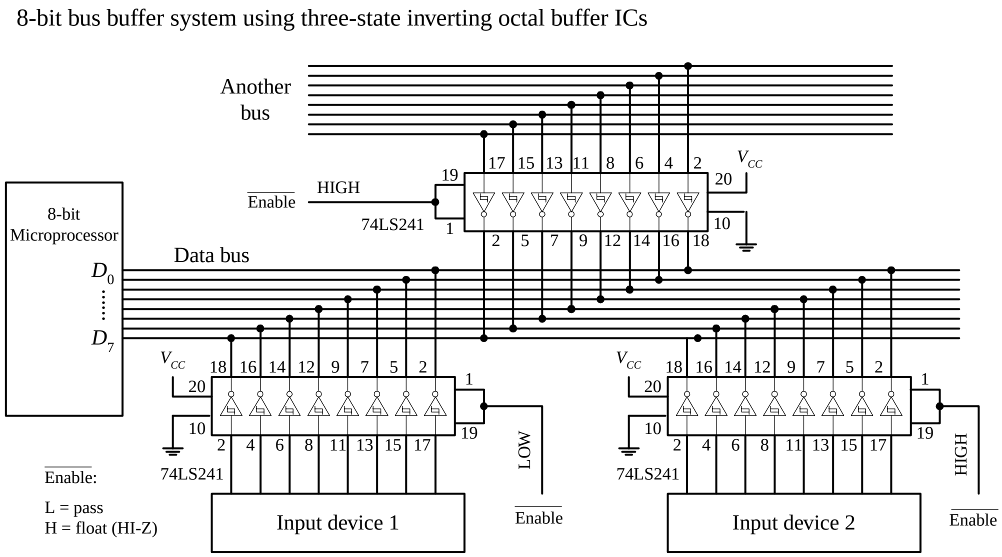
74240, 74LS240, 74F240, 74HC240, etc.

74241, 74LS241, 74F241, 74HC241, etc.

74244, 74LS244, 74F244, 74HC244, etc.

FIGURE 12.130

Here's an example of how three-state inverting octal buffers can be used in an 8-bit microprocessor system. The upper buffer links one bus to a common data bus. The two lower buffers are used to link input devices to the common data bus. With programming and the help of an additional control bus, the microprocessor can select which buffer gets enabled and which buffers get disabled.



8-bit bus buffer system using three-state inverting octal buffer ICs

FIGURE 12.131

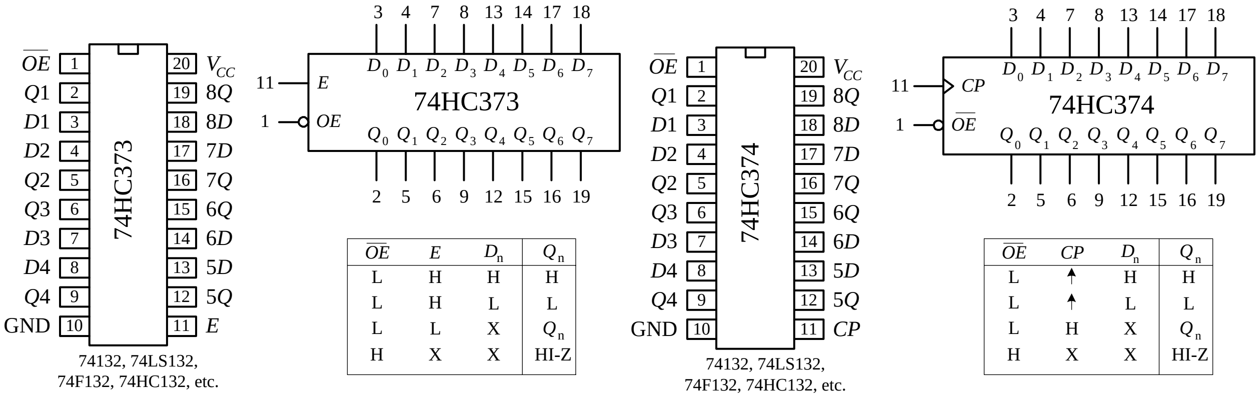
Data from an input device or another bus pass to the data bus only when the corresponding octal buffer is enabled (input enables made low). Only one input device or bus is allowed to pass data to the data bus at a time. Here, input device 1 is the only device allowed to pass data to the data bus because its enable inputs are set low. Note that data are inverted when passed through the inverting octal buffer.

12.9.2 Three-State Octal Latches and Flip-Flops

A three-state octal latch or octal flip-flop, unlike a three-state octal buffer, has the ability to hold onto data present at its data inputs before transmitting the data to its outputs. In microprocessor applications, where a number of devices share a common data bus, this memory feature is handy because it allows the processor to store data, go onto other operations that require the data bus, and come back to the stored data if necessary. This feature also allows output devices to sample held bus data at leisure while the current state of the data bus is changing. To understand how three-state octal latches and flip-flops work, let's first consider the 73*xx*373 three-state octal latch and the 74*xx*374 three-state octal flip-flop shown in Fig. 12.132.

74*xx*373 Octal latch with three-state outputs

74*xx*374 Octal D flip-flop with three-state outputs



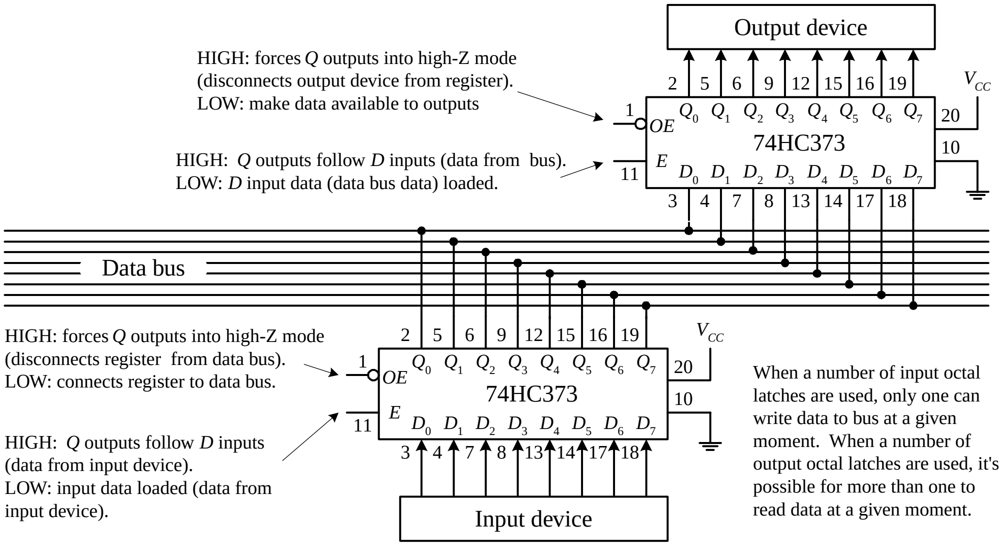
$\overline{OE}$	E	D_n	Q_n
L	H	H	H
L	H	L	L
L	L	X	Q_n
H	X	X	HI-Z

$\overline{OE}$	CP	D_n	Q_n
L	↑	H	H
L	↑	L	L
L	H	X	Q_n
H	X	X	HI-Z

FIGURE 12.132

The 73*xx*373 octal latch contains eight D-type "transparent" latches. When its enable input (*E*) is high, the outputs (Q_0–Q_7) follow the inputs (D_0–D_7). When *E* is low, data present at the inputs are loaded into the latch. To place the output in a high-impedance state, the output enable input ($\overline{OE}$) is set high. Figure 12.133 shows a simple bus-oriented system that uses two 73HC373s to communicate with an input device and output device. Again, like the octal buffers, control signals are typically supplied by a microprocessor.

Using three-state octal latch IC's as data bus registers



HIGH: forces *Q* outputs into high-Z mode (disconnects output device from register). LOW: make data available to outputs

HIGH: *Q* outputs follow *D* inputs (data from bus). LOW: *D* input data (data bus data) loaded.

HIGH: forces *Q* outputs into high-Z mode (disconnects register from data bus). LOW: connects register to data bus.

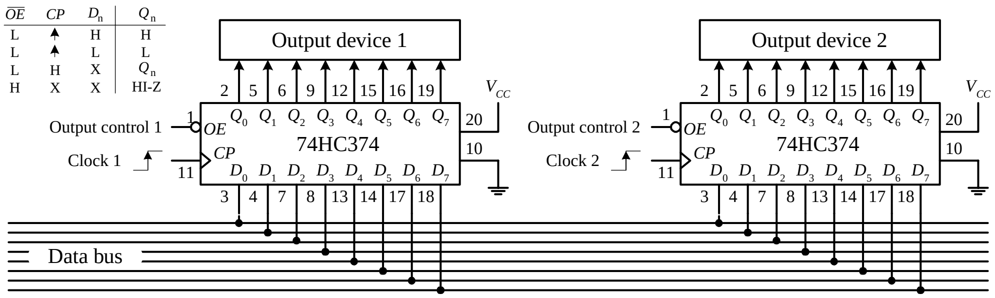
HIGH: *Q* outputs follow *D* inputs (data from input device). LOW: input data loaded (data from input device).

When a number of input octal latches are used, only one can write data to bus at a given moment. When a number of output octal latches are used, it's possible for more than one to read data at a given moment.

FIGURE 12.133

The 74*xx*374 octal flip-flop comes with eight edge-triggered flip-flops. Unlike the octal latch, the 74*xx*374's outputs are not "transparent"—they do not follow the inputs. Instead, a positive clock edge at clock input *CP* must be applied to load the device before data are presented at the *Q* outputs. To place the output in a high-impedance state, the output enable ($\overline{OE}$) input is set high. Figure 12.134 shows a simple bus-oriented system that uses two 73HC374s to communicate with two output devices.

Using octal D flip-flops as clocked, three-state data bus registers



$\overline{OE}$	CP	D_n	Q_n
L	↑	H	H
L	↑	L	L
L	H	X	Q_n
H	X	X	HI-Z

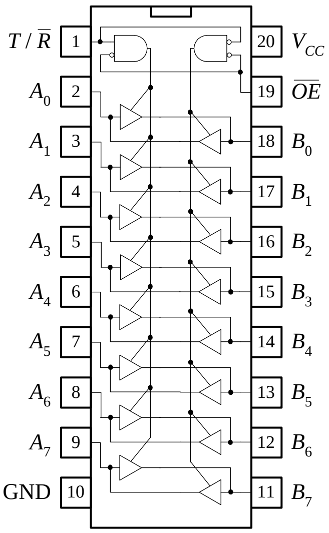
FIGURE 12.134

12.9.3 Transceivers

Another method for connecting devices that share a common bus is to use a transceiver. Unlike the three-state octal buffer, octal latch, and octal flip-flop, the transceiver is a bidirectional device. This means that when used in a bus-oriented system, external devices can read or write from the data bus. The figure below shows the 74*xx*245 octal transceiver, along with sample application. In the application circuit, a 74LS245 is used as a bidirectional interface between two data buses. To send data from bus *A* to bus *B*, the 74LS245's transmit/receive input ($T/\overline{R}$) is set high, while the output enable input ($\overline{OE}$) is set low. To send data from bus *B* to bus *A*, $T/\overline{R}$ is set low. To disable the transceiver's outputs (place output in a high impedance state) a high is applied to $\overline{OE}$.

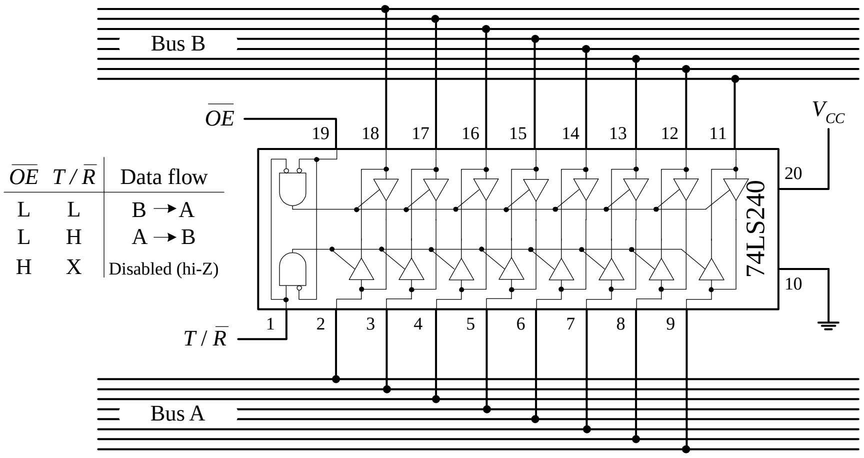
FIGURE 12.135

74*xx*245 octal transceiver



74LS240, 74F240, 74HC240, etc.

Using a 74LS245 as 8-bit bus transceiver



$\overline{OE}$	$T/\overline{R}$	Data flow
L	L	B → A
L	H	A → B
H	X	Disabled (hi-Z)

12.10 Additional Digital Topics

Appendices H to K contain important digital topics regarding digital-to-analog and analog-to-digital conversion, digital display, memory, and microprocessors and microcontrollers.

This page intentionally left blank.