

## BINARY ADDITION CIRCUIT

James T. Smith  
San Francisco State University

When you add two  $n$ -bit unsigned binary integers  $x$  and  $y$  you add  $n$  columns, each with an  $x$  and a  $y$  bit, and keep track of carries. Figure 1 shows two examples with  $n = 4$ :

$x$ bits	1 0 1 1	1 0 1 1
+ $y$ bits	0 0 1 0	0 1 1 0
carry bits	1	1 1
$z = x + y$ bits	1 1 0 1	1 0 0 0 1
		↑ Overflow!

**Figure 1** Two examples

The sums  $z = x + y$  are  $n$ - or  $n + 1$ -bit unsigned integers. In the latter case the leftmost bit is the carry from adding the leftmost column, and the sum would *overflow* the capacity of an  $n$ -bit storage device. If you write zeroes in the blanks where there were no carries, you see that to add  $n$ -bit unsigned binary integers  $x$  and  $y$  you perform  $n$  identical *column additions*. A column addition operation has three inputs and two outputs. Label the columns right to left with subscripts  $i = 0$  to  $n - 1$ . The three inputs to the  $i$ th column addition are the  $i$ th  $x$  and  $y$  bits  $x_i$  and  $y_i$  and the carry bit  $c_i$  from the previous column. (Carry bit  $c_0$  is 0.) The two outputs from the  $i$ th column addition are the  $i$ th sum bit  $z_i$  and the carry bit  $c_{i+1}$  that you input to the  $i + 1$ st column addition. (Carry bit  $c_n$  signals overflow!) Figure 2 shows the output bits that correspond to each possible combination of input bits.

$c_i$	0 0 0 0 1 1 1 1
$x_i$	0 0 1 1 0 0 1 1
$y_i$	0 1 0 1 0 1 0 1
$z_i$	0 1 1 0 1 0 0 1
$c_{i+1}$	0 0 0 1 0 1 1 1

**Figure 2** Corresponding input and output bits

For each input or output bit use the corresponding upper case letter to represent the proposition that the bit is 1. For example,  $X_i$  stands for the equation  $x_i = 1$ . You can use Figure 2 to represent the output equations as disjunctive normal forms:

$$Z_i = (\neg C_i \ \& \ \neg X_i \ \& \ Y_i) \vee (\neg C_i \ \& \ X_i \ \& \ \neg Y_i) \vee (C_i \ \& \ \neg X_i \ \& \ \neg Y_i) \vee (C_i \ \& \ X_i \ \& \ Y_i)$$

$$C_{i+1} = (\neg C_i \ \& \ X_i \ \& \ Y_i) \vee (C_i \ \& \ \neg X_i \ \& \ Y_i) \vee (C_i \ \& \ X_i \ \& \ \neg Y_i) \vee (C_i \ \& \ X_i \ \& \ Y_i).$$

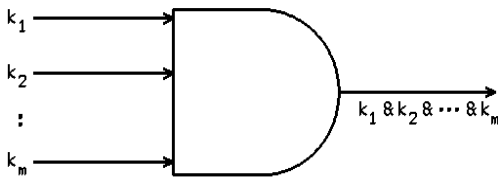
To help you interpret Figure 9 later, label the disjuncts in these normal forms as follows

$$Z_i = \#1 \vee \#2 \vee \#3 \vee \#4$$

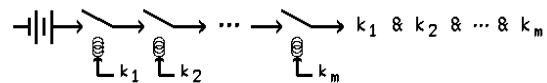
$$C_{i+1} = \#7 \vee \#6 \vee \#5 \vee \#4 .$$

These notes will design a switching network for column addition. It has three input circuits corresponding to  $x_i$ ,  $y_i$  and  $c_i$ , and two output circuits corresponding to  $z_i$  and  $c_{i+1}$ . In each circuit, current is on or off: set the corresponding input or output bit equal to 1 or 0 to indicate that condition. The switches will be arranged so that for each possible combination of input currents, the output currents reflect the entries in Figure 2.

An *and-gate* is an electrical device with  $m$  input circuits  $k_1$  to  $k_m$  and one output circuit; current flows in the output just when it flows in *all* inputs. The output circuit is called  $k_1 \ \& \ \dots \ \& \ k_m$ . A diagram like Figure 3 represents an and-gate. Figure 4 shows how to construct an and-gate by connecting relay switches in series. Each switch remains open unless the coil below is magnetized by the corresponding input current.



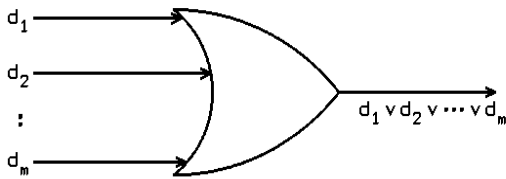
**Figure 3** And-gate



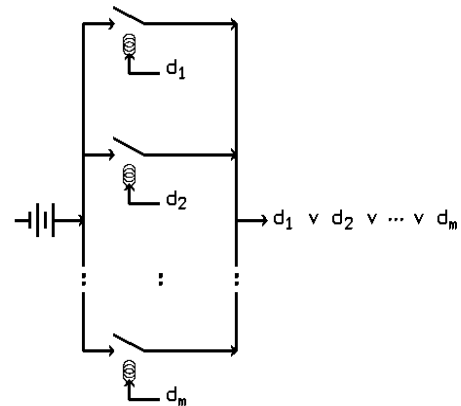
**Figure 4** And-gate implemented by relays in series

An *or-gate* is an electrical device with  $m$  input circuits  $d_1$  to  $d_m$  and one output circuit; current flows in the output just when it flows in *at least one* input. The output circuit is called  $d_1 \vee \dots \vee d_m$ . A diagram like Figure 5 represents an or-gate. Figure 6 shows how to construct an or-gate by connecting relay switches in parallel.

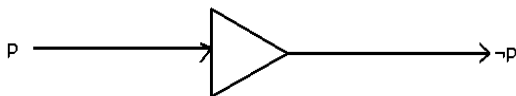
A *not-gate* is an electrical device with one input circuit  $p$  and one output circuit; current flows in the output just when it does *not* in the input. The output circuit is called  $\neg p$ . A diagram like Figure 7 represents a not-gate. Figure 8 shows how to construct a not-gate from a relay switch. The switch remains closed unless the coil below is magnetized by the corresponding input current.



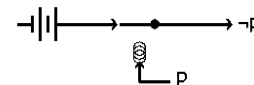
**Figure 5** Or-gate



**Figure 6** Or-gate implemented by relays in parallel



**Figure 7** Not-gate



**Figure 8** Not-gate implemented by a relay

## Binary addition circuit

Figure 9 shows how to construct the switching network for column addition. Just connect and-, or- and not-gates as directed by the disjunctive normal forms of the output equations. You need seven three-input and-gates, two four-input or-gates, and nine not-gates. Altogether, you need

$$(7 \times 3) + (2 \times 4) + 9 = 38$$

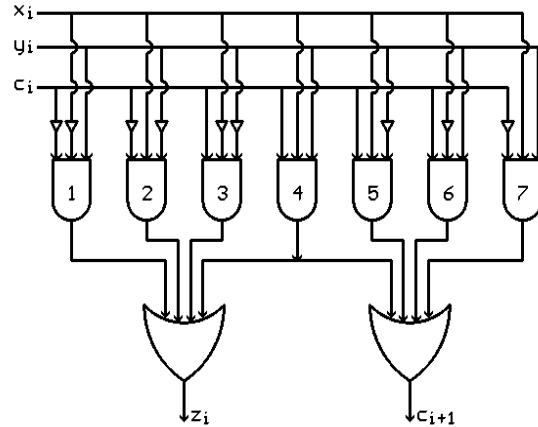
relay switches.

Early computers, telephone networks, and other control devices were constructed from relays. They were bulky, slow, and costly. Modern equipment uses solid state electronic switches, which are far smaller, faster, and cheaper.

Intel Pentium computers add 32-bit unsigned binary integers. If implemented this way, their addition circuits would require  $32 \times 38 = 1216$  switches. Even with solid-state electronics, it's useful to minimize the number of switches required.

### Project

Find equivalent Boolean forms for the output equations with as few connectives as possible. Redesign the network. How few switches can you use?



**Figure 9** Switching network for column addition

## References

Claude E. Shannon, “A symbolic analysis of relay and switching circuits,” *Transactions of the American Institute of Electrical Engineers*, LVII (1938), 713-723. This is essentially Shannon’s MIT PhD dissertation, the origin of the application of Boolean logic to circuit design.

Jean E. Rubin, *Mathematical logic: Applications and theory*. Philadelphia: Saunders College Publishing, 1990. This comprehensive elementary text is readable, if a bit pedestrian. Sections 4.4 and 4.5 cover the material here. Section 4.3 introduces the most popular technique for minimizing the number of connectives required to express a Boolean compound sentence: Karnaugh maps. It contains references to relevant engineering texts.

Maurice Karnaugh, “The map method for synthesis of combinational logic circuits,” *Transactions of the American Institute of Electrical Engineers*, LXXII (1953), 593-599. This is the origin of the most popular technique for minimizing the number of connectives required to express a Boolean compound sentence: Karnaugh maps.