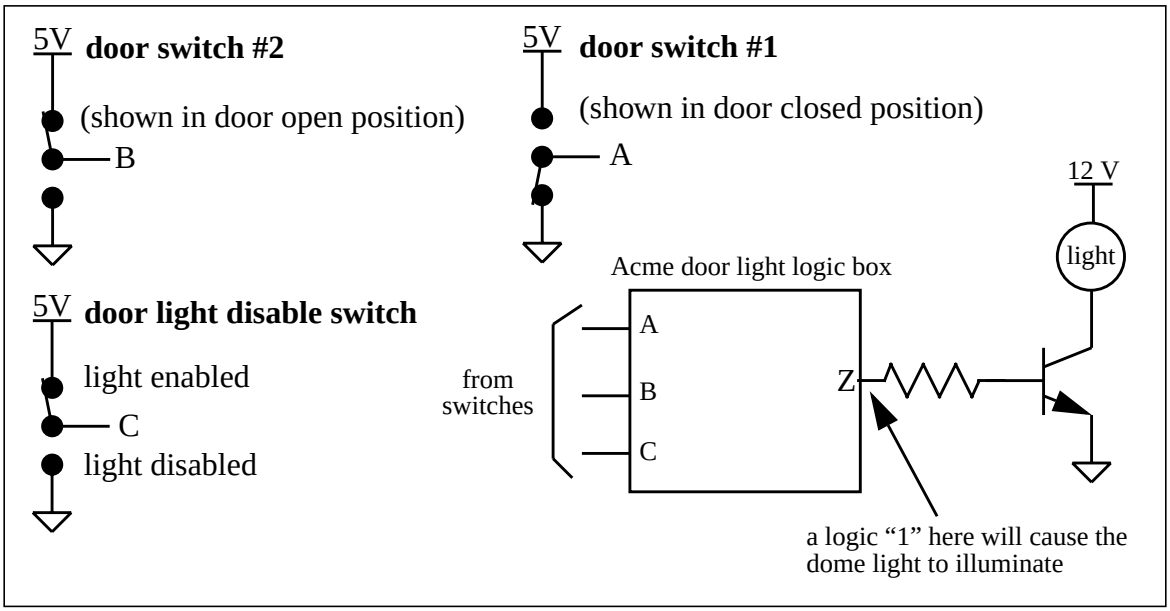
Ad hoc design of digital circuits

There are many ways to design circuits with digital logic. But to develop our intuitive understanding for the use of these gates, let's design a system ad hoc. Here is the problem:

A two-door car has a switch for each door so that when a door is open, the dome light is illuminated. Inside the car there is also another switch used to disable this feature. Design the digital logic necessary to implement this system. Your logic design will go in the "Acme door light logic box".



The inputs and outputs to our logic box are at the normal logic levels for our logic family; in this case, 5 volts. To arrive at a logic circuit, we simply think out loud about the operation to be performed. For example, "If door 1 *OR* door 2 is open, *AND* the disable switch is off, the done light should illuminate." Writing this out in equation form:

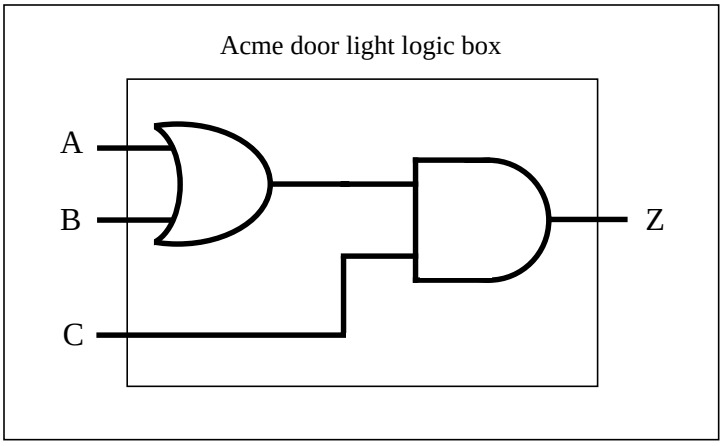
[(door 1 open) OR (door 2 open)] AND (disable switch is off) = light illuminated

Replacing the conditions with logic values:

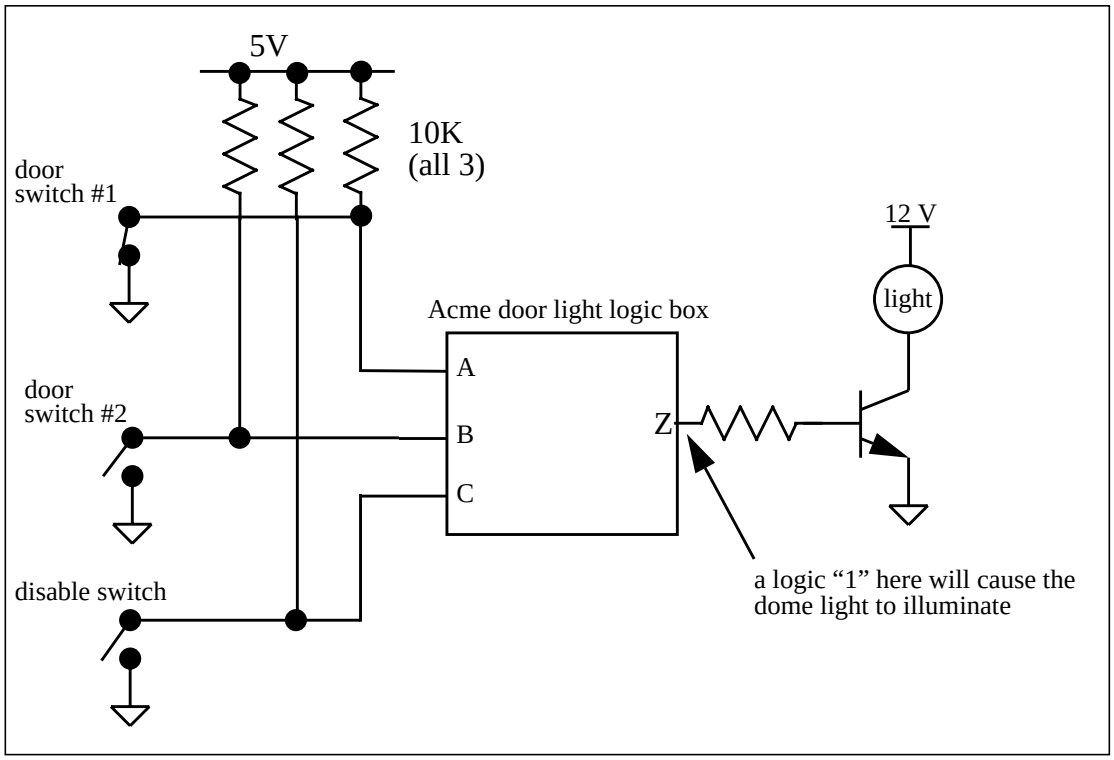
[(door 1 = 1) OR (door 2 = 1)] AND (disable switch = 1) then Z = 1

Thus if either door opens, inputs A or B will become logic one. In that case, if input C is also one, indicating that the disable switch is off, then the output Z will become logic one. This output drives the base of the transistor, turning on the dome light.

With this equation, its easy to see what logic realization would be required:

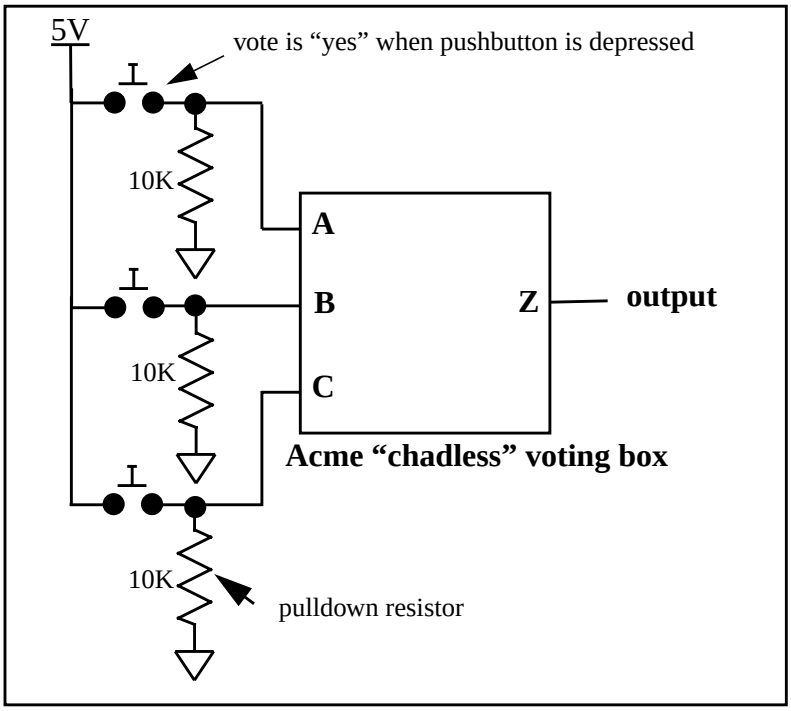


Instead of using the single-pole-double-throw switches as shown above, we could use a single-pole-single-throw switch with a *pullup* resistor. See the new schematic below. The circuit behaves exactly as before.



The purpose of the “pullup resistor” is to apply a logic “high” voltage to the input to the logic gates when the switches are open. If logic gates do not have their inputs connected to valid logic levels, their outputs are not defined.

Let's try another problem. We need to design a voting box. We want a logic one output from the voting box when at least two out of three votes are "yes". A yes vote is detected by a logic one and a no vote by logic zero. Note that this circuit has pulldown resistors to terminate the logic gate inputs to a valid logic low level when the switches are not depressed. Large value resistors are suitable because the input currents to the logic gates is very small.:



Let's use a different approach. We used a tabular approach to see the function of basic gates. We can extend the tabular approach to the function of more complicated circuits. Below we the majority function expressed as a truth table.

inputs			output
A	**B**	**C**	**Z**
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	**1**	**1**	**1**

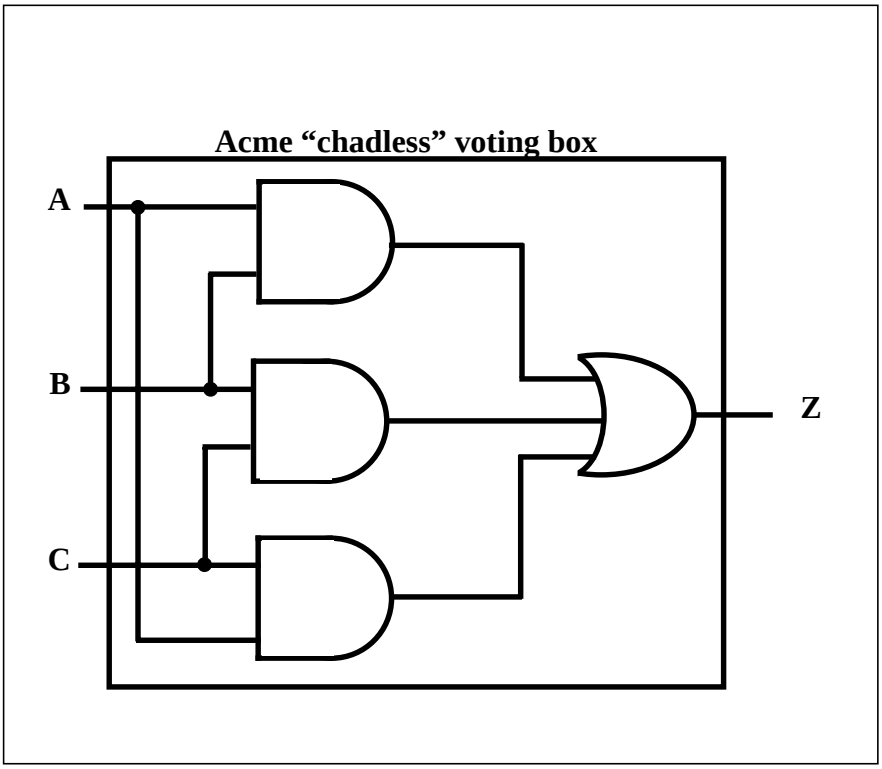
We see that whenever two or more inputs are asserted, the output becomes asserted. We can also read the table in such a way that suggests the implementation. For example, in the fourth row, if B AND C are logic one, the output is logic one. For row six, if A and C are true, the output is logic one. We see that these AND terms are all part of the output. To get the combined effects of all the terms, we simply OR them.

If we write this behavior as equation we would have:

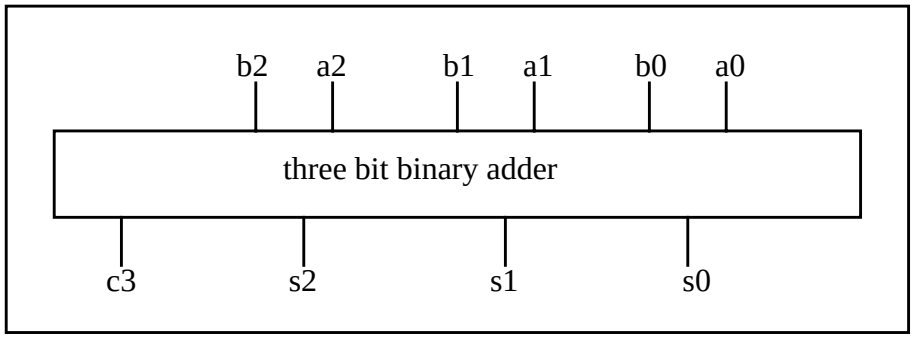
Z = (B * C) + (A * C) + (A * B) + (A * B * C); where * represents AND, + represents OR

(Can you see that the last term is redundant and can be omitted from our realization?)

Describing this as a logic diagram, we obtain:



As a final example of ad hoc design, lets design a three bit binary adder. This adder will have two, three bit inputs and will output four bits, three sum bits and one carry out. As as block diagram, it would look like this:



This adder can add two numbers as follows:

In general:

$$\begin{array}{r} a2\ a1\ a0 \\ +b2\ b1\ b0 \\ \hline c2\ s2\ \ s1\ s0 \end{array}$$

For this example:

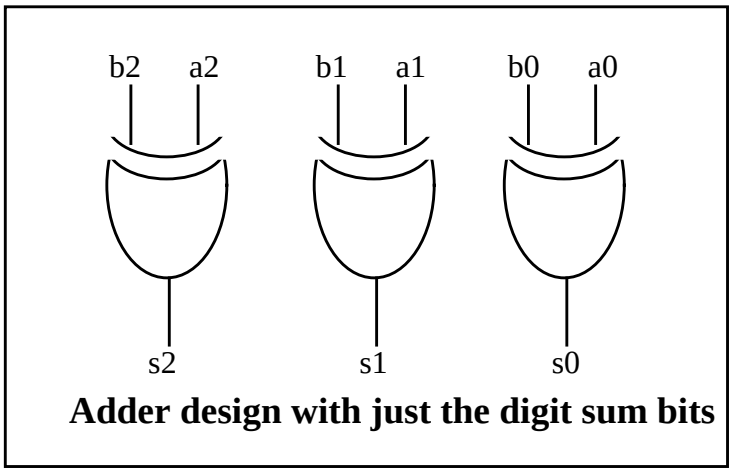
$$\begin{array}{r} 1\ 0\ 1_2 \\ +1\ 1\ 0_2 \\ \hline 1\ 0\ 1\ 1_2 \end{array}$$

To create the sum for each set of two digits, we use the XOR gate. The inputs to the XOR gate are the pair a_n and b_n. This application of the XOR gate gives the correct algebraic result for two, one bit numbers as shown below.

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ +0 & +0 & +1 & +1 \\ \hline 0 & 1 & 1 & 0 \end{array}$$

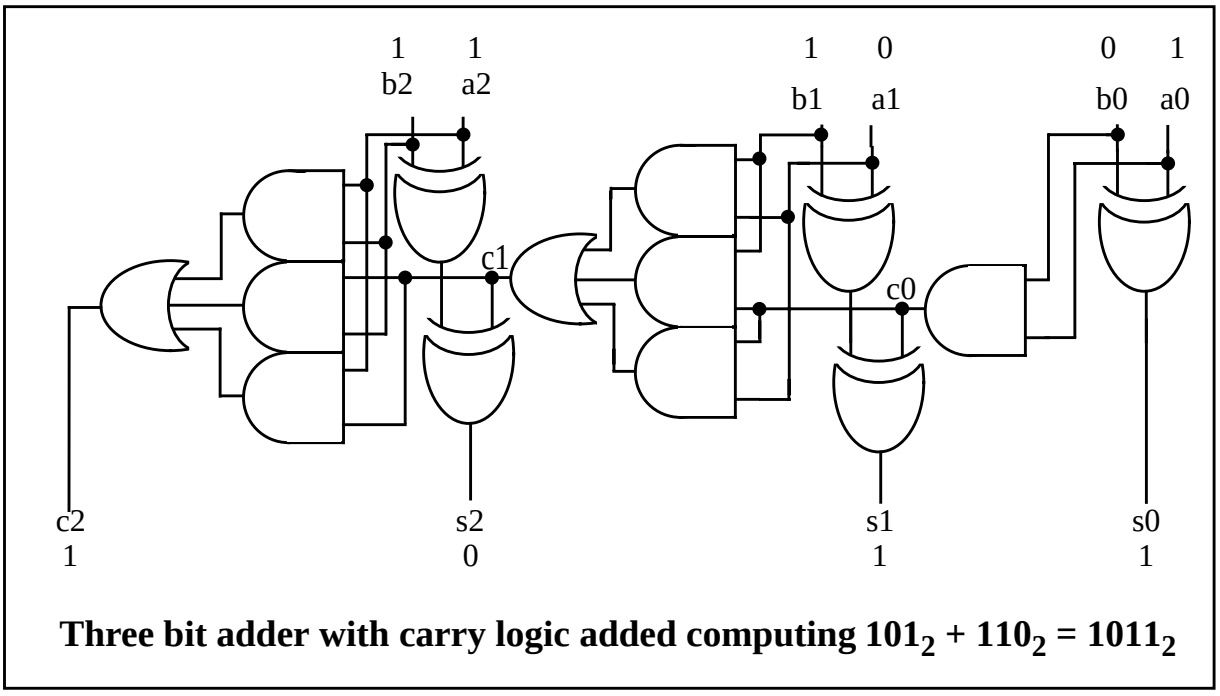
generating a carry also

Therefore to generate the partial sums for each pair of digits we create the following network.



Adder design with just the digit sum bits

We have correctly created the sum s0 but s1 and s2 and c3 are not yet generated. We need to devise a way to add possible carry outs to each sum digit from each preceeding pair of input bits. For s0, there is no carry in signal. But for s1, the possibility of a carry from a0,b0 must be added to the sum of a1,b1. For a carry to occur from a0,b0, both a0 AND b0 must be a "1". The solution is obviously to detect the 1,1 condition at a0,b0 and if it exists to add a "1" to the sum of a1,b1. Thus we will need another XOR gate to do the addition. Now our adder looks like this.



Three bit adder with carry logic added computing $101_2 + 110_2 = 1011_2$