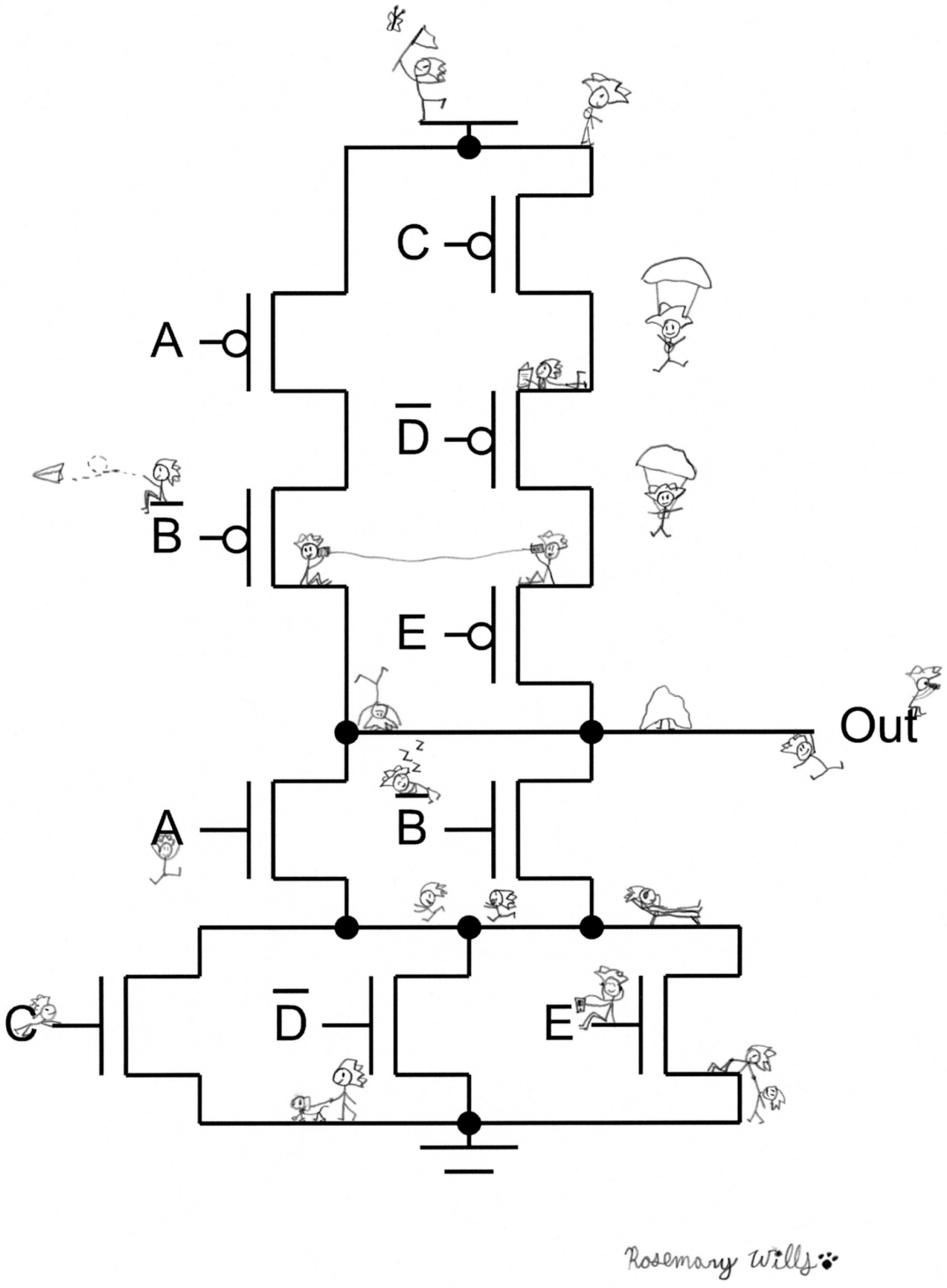
Designing Computer Systems

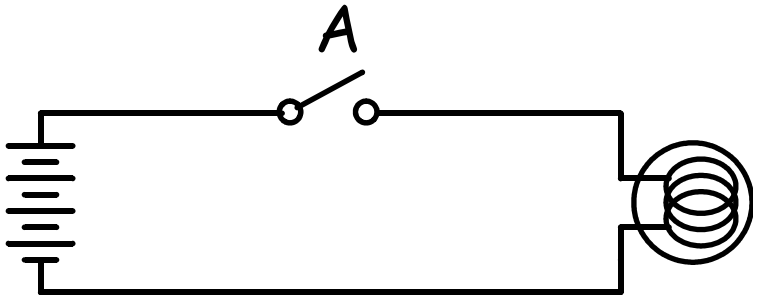
Switches and Wire



Designing Computer Systems

Switches and Wire

Despite their apparent complexity, digital computers are built from simple elements, namely *switches* and *wire*. To see how switches and wire can perform computations, consider the circuit below. The battery on the left is connected to the bulb on the right through the switch labeled **A**.

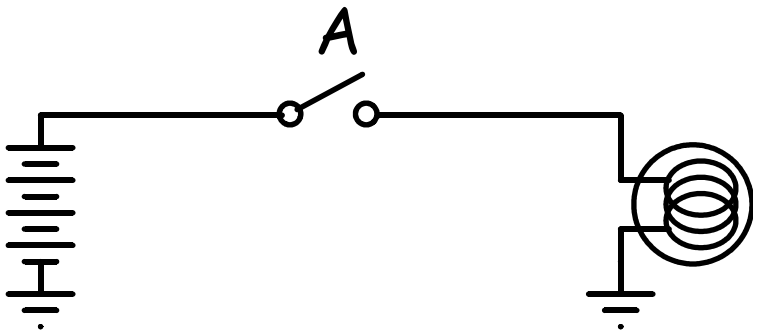


The battery will light the bulb if there is a complete path for current to flow from one side of the battery to the other. If the switch is open, no current can flow so the light is off. If the switch is closed, current flows and the light is on. The behavior of this simple circuit can be expressed using a table.

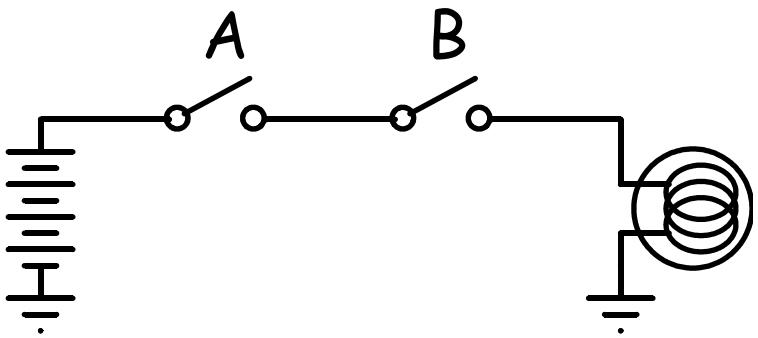
switch	light
open	off
closed	on

This type of table has been given the lofty name *Truth Table*. A more meaningful name would be *behavior table* since it describes the behavior of the circuit. Truth tables list all possible inputs to a system on the left and resulting outputs on the right. A truth table specifies how a system should behave. It does not specify how it should be implemented; this can be done in many ways.

Sometimes an icon is used to show connected nodes without drawing a wire. In the circuit below, the triangular symbols below the battery and bulb represent *ground*. We can imagine that all points attached to ground icons are connected together. So this circuit behaves identically to the circuit above.



Here's a system with two switches in series.



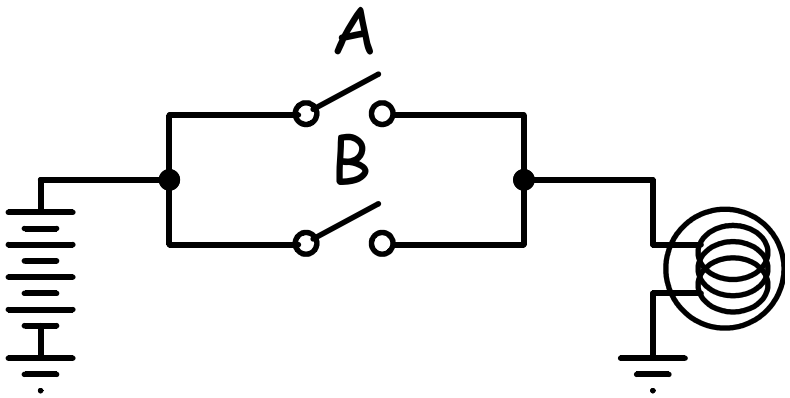
Because each switch can be in one of two states (open or closed) and there are two switches, the truth table has four rows. It's not so important how we list the input combinations so long as all cases are included exactly once.

switch A	switch B	light
open	open	off
closed	open	off
open	closed	off
closed	closed	on

In this circuit, the light is on when switch A is closed *AND* switch B is closed. This illustrates an important point; **series switches produce AND behavior**. Using words like open/closed and on/off to describe system behavior is verbose. We can assign the value **0** to an open switch and **1** to a closed switch. Further we can assign the value **0** to an off (dark) bulb and **1** to an on (lit) bulb. Sometimes we'll refer to 1 as *true* and 0 as *false*. Now the truth table becomes more compact.

A	B	Out
0	0	0
1	0	0
0	1	0
1	1	1

The next system has two switches in parallel.

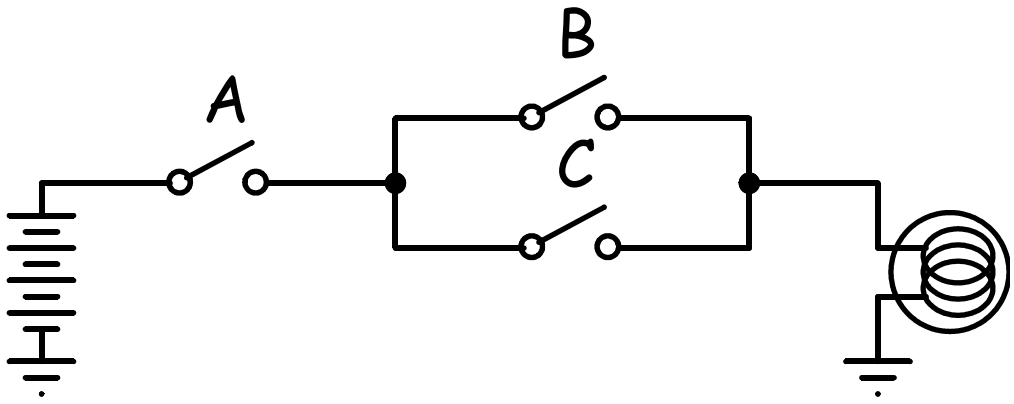


Here the output is true if switch A is closed OR switch B is closed. This illustrates another important point; **parallel switches produce OR behavior**. Here's the truth table.

A	B	Out
0	0	0
1	0	1
0	1	1
1	1	1

We might call this an "OR circuit" (in contrast to the previous "AND circuit") because its output is true when either A is true OR B is true. Defining a system's behavior by when its output is true is call *positive logic*. This contrasts with *negative logic* where a system's behavior is defined when its output is false. We have to pick one convention; positive logic seems more intuitive.

The last circuit is more complex.

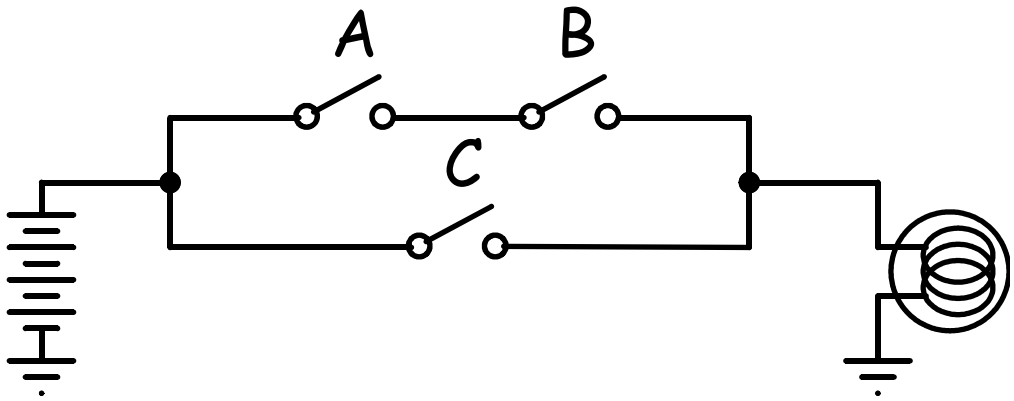


The truth table has eight rows to capture all input combinations of the switches. In general, if a system has *N* binary inputs (i.e., each input can be in one of two states), there are 2^N entries in the truth table. This circuit behavior is accurately, if not clearly, described in the truth table.

A	B	C	Out
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

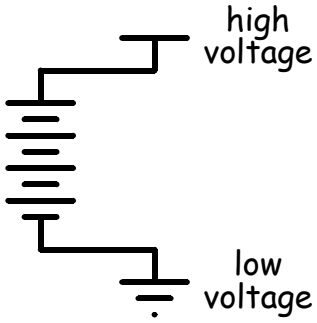
A more concise description of the behavior is derived from the series and parallel arrangement of the switches. By starting at the outer-most connections, switch A is in **series** with a second switch combination, switch B in **parallel** with switch C. Therefore, the output is true when **A AND (B OR C)**.

The parentheses are significant since AND has higher precedence than OR, just as multiplication has higher precedence than addition. The arithmetic expression $A \cdot (B + C)$ differs from $(A \cdot B) + C$ that results if the parentheses are removed. Similarly, A AND (B OR C) differs from (A AND B) OR C. The second expression would be implemented as the following, non-equivalent circuit.



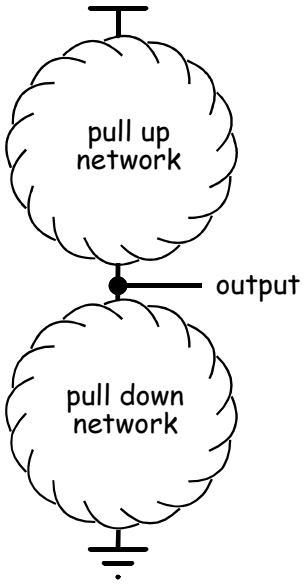
Designing systems with these switches has one major limitation. The input to the switch is a mechanical activator (your finger). But the output of the system is optical (light). This prevents composing systems since the output of one system cannot control the input of another. To remedy this problem, we'll consider a different kind of switch.

A voltage controlled switch fits our need since (A) most systems have a handy voltage source from a power supply or batteries, (B) switches can easily connect to either a high or low voltage to produce an output, and (C) controlling switches with a voltage does not implicitly dissipate a lot of energy. The sources of high (1) and low (0) voltages are show below.

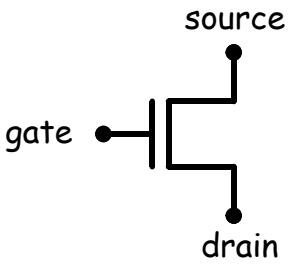


A battery is connected to the ground symbol to represent the low voltage source. The other side of the battery is connected to a new symbol that resembles a "T". It is used to represent the high voltage source. These high and low supply symbols are used throughout a design to provide necessary voltages needed to activate voltage controlled switches. But only one battery is needed to generate them!

In order to build composable circuits (i.e., where the output of one circuit can control the input of another), voltage controlled switches must connect an output to either the high or low supply, as shown below. We'll use networks of voltage controlled switches to provide correct output values for each combination of inputs.

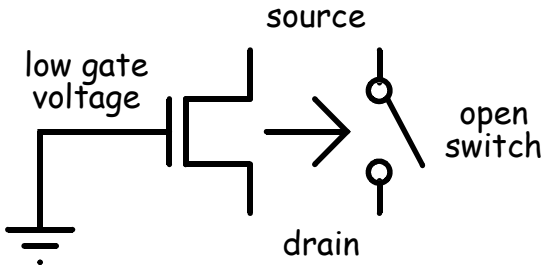


N-Type Switch: The voltage controlled switch is called an **N-type** switch. It has three places for wires to connect called terminals. The source and drain terminals represent the ends of a switch that are connected when the switch is closed. The gate terminal is the voltage controlled input that opens or closes the switch.

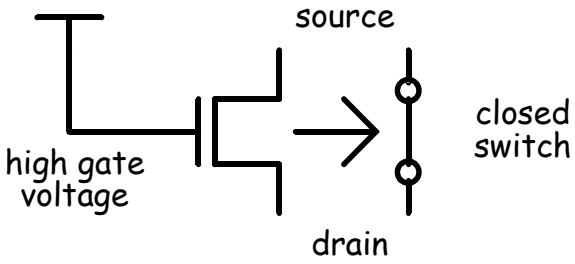


gate	switch
low (0)	open
high (1)	closed

If the voltage connected to the gate is low, the switch is open. If the voltage on the gate is high, the switch is closed. N-type switches are called an **active high**.

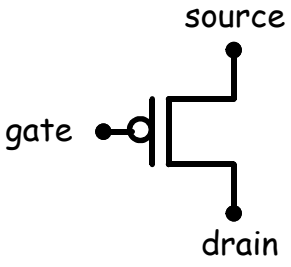


A low gate voltage opens the switch



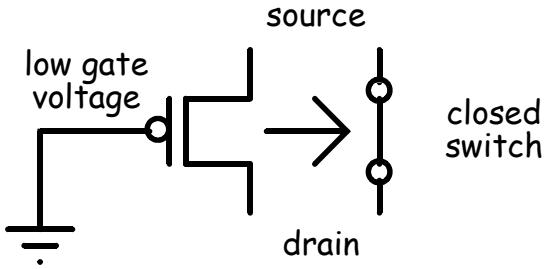
A high gate voltage closes the switch

P-Type Switch: As one might expect in a binary world, there is also a **P-type** switch (note the bubble drawn on the gate terminal). It behaves just like an N-type switch, except that the source-drain switch closes when the gate voltage is low.

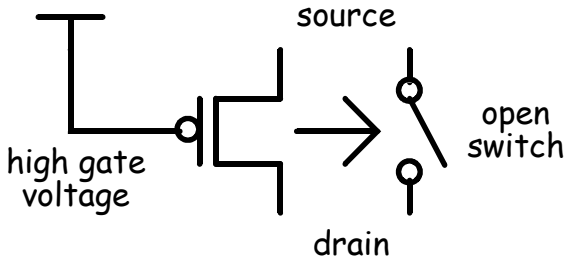


gate	switch
low (0)	closes
high (1)	open

Since the switch closes when the gate voltage is low, a P-type switch is called **active low**.



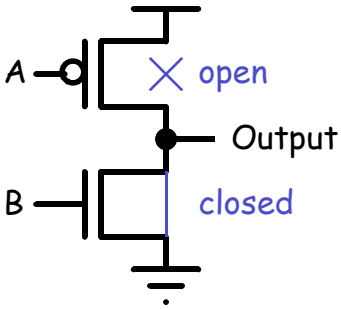
A low gate voltage closes the switch



A high gate voltage opens the switch

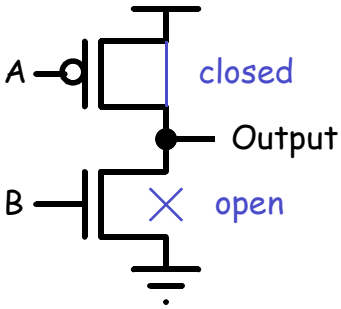
Designing Logic: Building logical circuits with voltage controlled switches begins like the battery and light designs. Only now a complimentary circuit must be

created to connect the output to the high voltage sometimes and the low voltage other times. Errors in the design process can lead to unfortunate consequences. Consider this simple two input circuit.



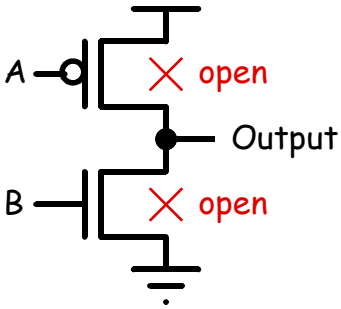
A	B	Out
0	0	
1	0	
0	1	
1	1	0

When A and B are both high, the N-type switch is closed connecting the output to the low voltage. Since the P-type switch is open, it does not participate.



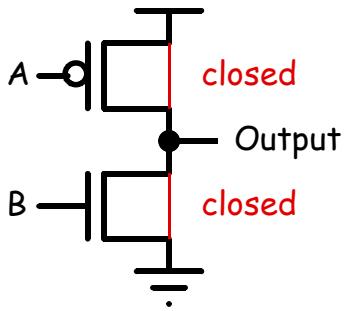
A	B	Out
0	0	1
1	0	
0	1	
1	1	0

When A and B are both low, the P-type switch is closed connecting the output to the high voltage. Since the N-type switch is open, it does not participate.



A	B	Out
0	0	1
1	0	float
0	1	
1	1	0

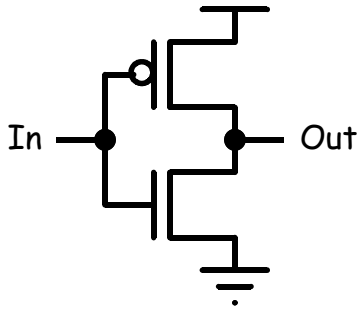
When A is high and B is low, both the N-type and P-type switch are open. The output is not connected to the high voltage or the low voltage. This undefined state, often called a *floating node*, does not provide a valid output for controlling other switches. For this reason, this condition should be avoided.



A	B	Out
0	0	1
1	0	float
0	1	short
1	1	0

When A is low and B is high, both the N-type and P-type switch are closed. This condition is more serious than a floating output in that the high voltage is connected to the low voltage. This is called a *short*. It is particularly bad since, in addition to having an undefined output value, a lot of current flows, generating heat that can damage the switches. Shorts should be eliminated in all designs!

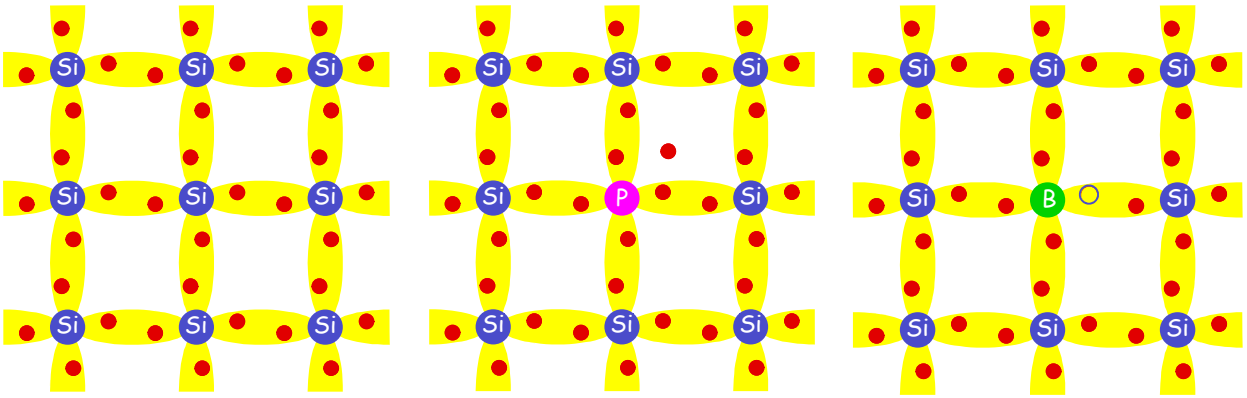
NOT Gate Implementation: A NOT gate can be implemented using a variation of this circuit. The two undesirable states (float and short) occur when A and B have different values. If the two inputs are connected together as a single input (In), unwanted output conditions are eliminated. This circuit implements a NOT gate.



In	Out
0	1
1	0

Note that this circuit is controlled by a high or low voltage, and it produces a high or low voltage output. These switches can be used to implement complex expressions of AND and OR functions. But first we need to understand a few things about switch technology.

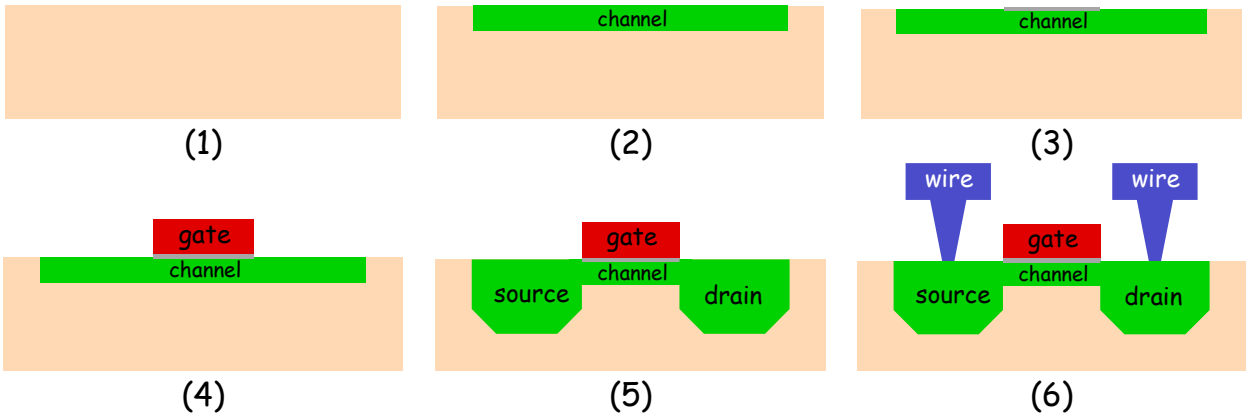
See MOS Switches: These voltage-controlled switches are built out of silicon (Si). This is a surprise since pure silicon and silicon dioxide (silica) are insulators. In fact, most high voltage insulators are ceramics made from silicon. Si atoms have four valance (outer-shell) electrons. When arranged in a crystal lattice, all of these electrons are bound so charge carriers are not available for conduction (A). In order to change things, atoms of other elements are embedded in the lattice through ion implantation or diffusion. These elements, called dopants, have either one more or one less valance electron. Phosphorus has an extra electron (five) (B); Boron has one less (three) (C).



(A) (B) (C)

When these elements find themselves in the lattice, their extra electron contributes a negative charge carrier (an electron) while their lacking electron contributes a positive charge carrier (a hole). Introducing these charge carries produces an interesting *semiconductor* material that can be controlled by an electrical field.

Through clever processing of a silicon wafer, doped regions can form the semiconducting channels of a switch. The basic structure is shown below in cross-section. First a wafer of pure silicon is cut and polished to a smooth surface (1). Then dopants are selectively implanted to form a channel (2). A small isolative layer is grown above the channel (3). Then a layer of conductive polysilicon is selectively formed over the channel to act as the gate (4). Additional dopant implantation deepens the source and drain regions (5). Then wires (typically aluminum or copper) are formed to contact and connect source, drain, and gate terminals of the formed devices (6). The completed device is a Metal Oxide Semiconductor Field Effect Transistor (MOSFET).

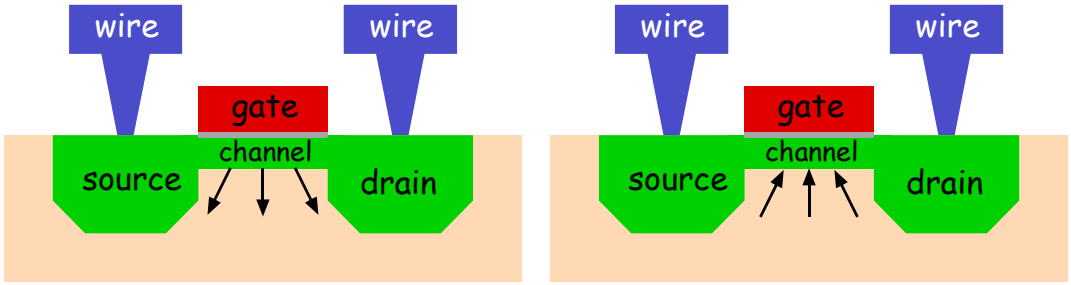


(1) (2) (3) (4) (5) (6)

This selective processing is accomplished using photolithography. In this process, a light sensitive material is used to pattern the layered structures on the silicon surface. Because this fabrication process is performed on the entire wafer surface, a couple hundred chips, each containing hundreds of millions of transistors

can be fabricated at the same time. This dramatically reduces the cost for each chip.

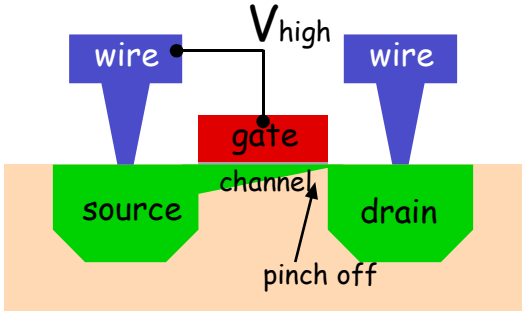
This semiconductor device can operate as a voltage controlled switch. When a voltage is placed on the gate terminal, relative to the silicon around the device (know as the substrate), a vertical electrical field is generated. This field can push charge carries out of the channel, opening the switch (A). Or the field can attract charge carries into the channel, closing the switch (B).



(A) (B)

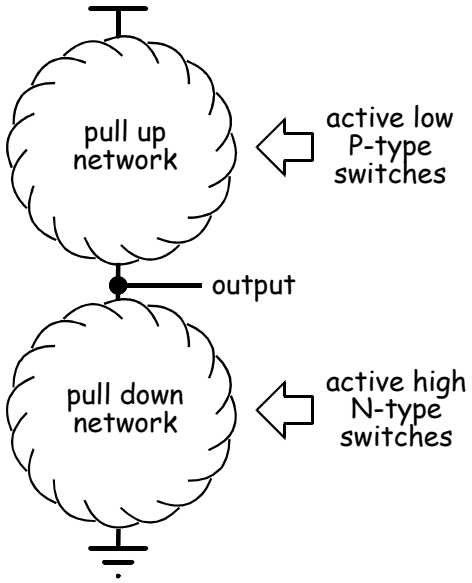
These semiconducting switches come in two types. An N-type Field Effect Transistor (NFET) *closes* (conducts) when a high voltage is placed on its gate. A P-type Field Effect Transistor (PFET) *opens* (isolates) when a high voltage is placed on its gate. The two switch types differ only in the dopant used and some early preparation of the surrounding silicon so the substrate can be properly biased to create the field. When these two types are used to implement a Boolean expression, the resulting design is called Complementary Metal Oxide Semiconductor or CMOS.

These switches have a critical limitation. When the voltage applied to the gate to close the switch is also present at the source or drain, the generated horizontal field will deplete the charge carriers at the opposite side of the channel, *pinching off* channel conduction. As the source-drain voltage approaches a technology specific *threshold voltage*, the opposite terminal will no longer be pulled towards the supply voltage. Here's an N-type switch connected to the high voltage.



This means N-type switches cannot be used to pull the output high. Nor can a P-type switch be used to pull the output low. Our switch design strategy, using

networks of voltage controlled switches to produce an output voltage must incorporate this technology limitation.



Time to Design: We're now ready to implement Boolean expressions using N and P type switches. We need to use ideas we've covered so far:

- Series switches produce the AND function.
- Parallel switches produce the OR function.
- P-type switches are active low and can pull high.
- N-type switches are active high and can pull low.
- An output should always be pulled either high or low.

Plus we must add a small but significant relationship called DeMorgan's Theorem that states:

- X AND Y = NOT(NOT X OR NOT Y)
- X OR Y = NOT(NOT X AND NOT Y)

Boolean expressions quickly become awkward when written this way. We can use symbols from arithmetic · and + to represent AND and OR functions. We can also use bars to represent the NOT operation. So DeMorgan's Theorem becomes:

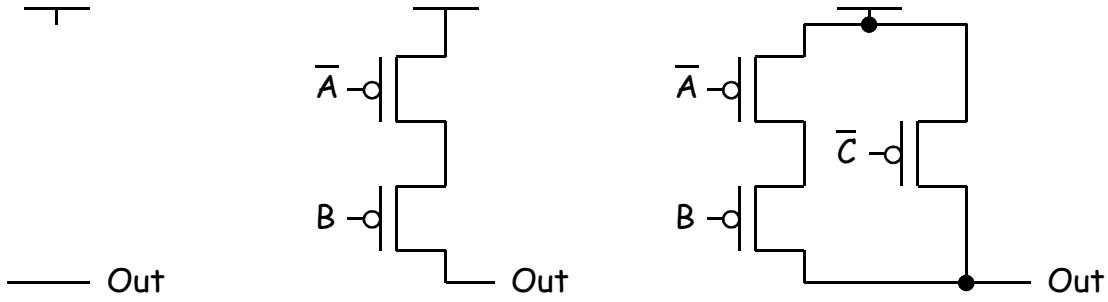
- $X \cdot Y = \overline{\overline{X} + \overline{Y}}$
- $X + Y = \overline{\overline{X} \cdot \overline{Y}}$

This means an AND and OR operations can be exchanged by complementing their inputs and output. This will come in handy in switch design.

Suppose we want to implement a Boolean expression composed of AND and OR operations applied to binary inputs (in their true and complemented form). Here's an example.

$$Out = A \cdot \overline{B} + C$$

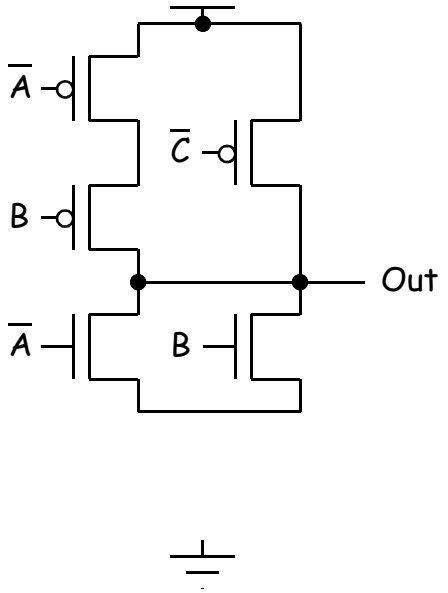
We'll need to design a pull high network of P-type switches, and a pull low network of N-type switches (1). The output should be high when A is high AND B is low OR when C is high. The first part of this OR expression should connect the output to the high supply when A is high AND B is low. This is accomplished by a series combination of switches. But with active low P-type switches, we must complement the inputs (2). So when A is high, $\overline{A}$ is low closing the active low P-type switch. If B is also low ($\overline{B}$ is high), then this low input will close the other P-type switch, completing the connection between the output and the high supply. The second part of the OR pulls the output high if C is high. OR is implemented by a parallel connection of switches. Again, C must be complemented so that when C is high, the active low P-type switch will close pulling the output high (3).



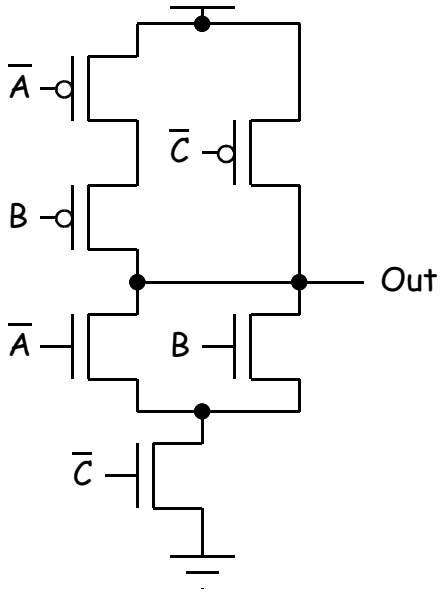
(1) (2) (3)

Now we must design a pull low network that connects the output to the low supply whenever it is not being pulled high. DeMorgan's Theorem shows that ANDs and ORs can be swapped if inputs and outputs are complemented. Using N-type rather than P-type switches complements all inputs. Pulling low rather than high complements the output. So we can exchange AND and OR by exchanging series and parallel switch arrangements. We must begin with the outermost operation, in this case the OR. In the first part of the OR, A and are in series in the pull up network. So they are in parallel in the pull down network (4). Since $A \cdot \overline{B}$ are in

parallel with C on in the pull up network, they will be in series in the pull down network (5).

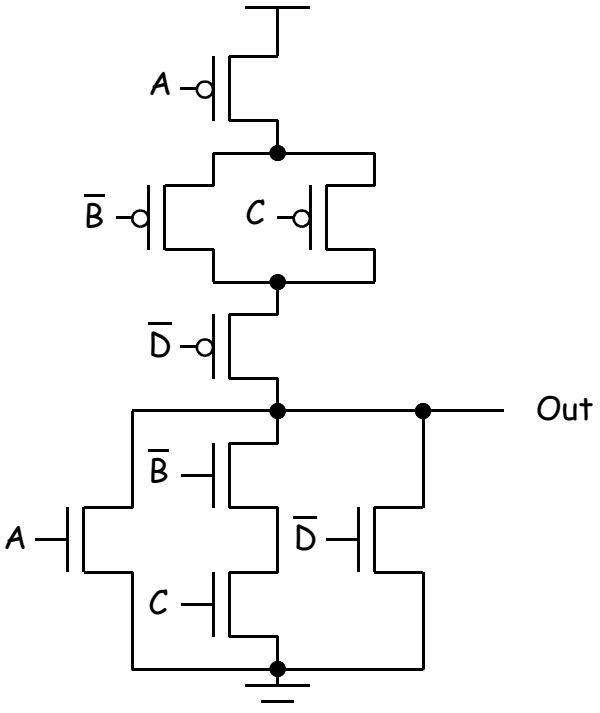


(4)

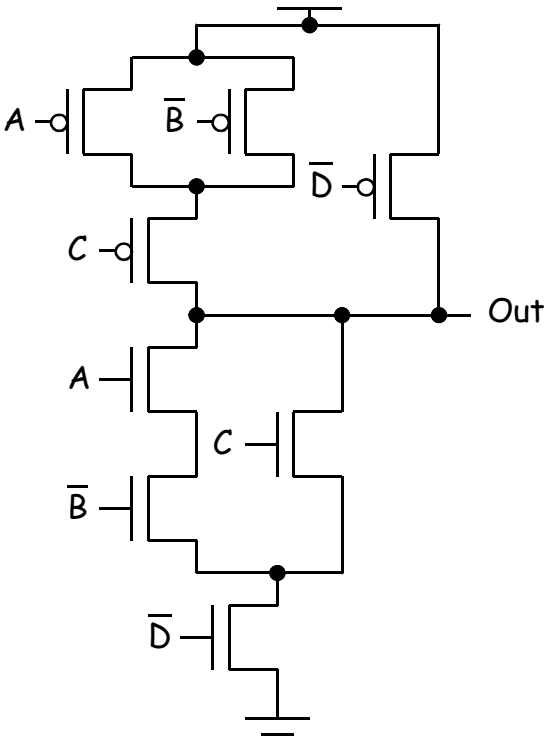


(5)

Here are a few more examples:



$\overline{A} \cdot (B + \overline{C}) \cdot D$



$(\overline{A} + B) \cdot \overline{C} + D$

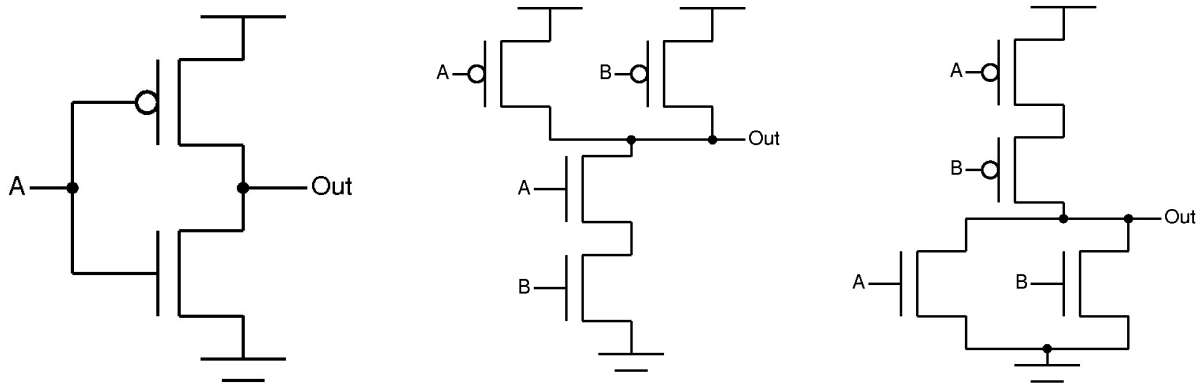
Not that parallel switches in the pull up network are in series in the pull down network, and vise versa. Care must be exercised to work on the operations from

the outside in. That is, the last evaluated operation in the expression is the outermost combination of switching circuits.

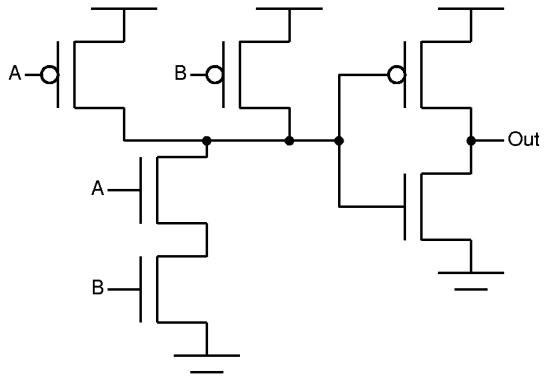
Building Abstractions: Switch design allows direct implementation of a behavior described in Boolean expression. It is often yields the fastest (in terms of delay) and most efficient (in terms of switches required) solution. But sometimes a little convenience is worth a slightly higher cost. People don't prepare all their food from scratch even though it would be more healthy and less expensive. Engineers don't write programs in the machine language of computers, even though the executable file would be smaller and it would run faster. And when we are designing a digital system with a couple hundred million transistors, we may prefer not to implement all functions with switches.

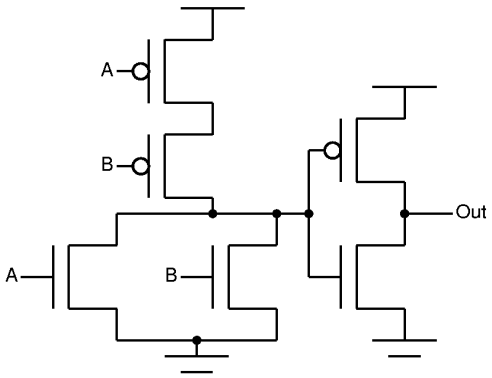
So we do what all engineers do. We create larger, more complex functional abstractions and then design with them. An automobile is a complex system. Fortunately automobile designers combine already understood subsystems for power, steering, braking, etc. and then adapt as necessary. Computer designers do the same thing, but with different building blocks.

Basic Gates: Since we already express our designs using logical functions, a natural choice for new, more complex abstractions would be logical gates. Here are the basic gates used in digital design: NOT, NAND, NOR, AND, and OR. Consider a gate with **i** inputs. inverting gates (gates that begin with "N") require **2i** switches for each input. Non-inverting gates (AND and OR) require **2i+2** switches.



NOT NAND NOR





AND OR

In the chapter on Gate Design, we'll see how designing with gates compares to the switch design. But first we will visit the mathematics of digital design in Boolean Algebra.

Summary: Switch design is at the heart of nearly every computer technology we use today. Here are the key points.

- In contrast with human experience, computation is largely performed on binary values (zeros and ones).
- The computing world is built with digital switches.
- These switches are voltage controlled, and are assembled in networks to produce high or low voltage outputs.
- Series switches implement the AND function; Parallel switches implement OR.
- NFETs are active high switches, and are preferred for pulling an output low.
- PFETs are active low switches, and are preferred for pulling an output high.
- Outputs that are not connected to the high or low voltage are floating (undefined), and this is not good.
- Outputs are are connected to both the high and the low voltages result in a short, and this is bad.
- Switches are actually MOSFETs, made from silicon with other dopant elements that create free charge carriers.
- High integration of MOSFETs on a single chip provides many connected switches for digital computation, at a low cost.
- Boolean expressions can be efficiently implemented using MOSFETs.

