CS 251: Computer Organization and Design

Charles Shen

Spring 2016, University of Waterloo

Notes written from Safaa Bedawi's lectures.

Feel free to email feedback to me at echen902@gmail.com.

Contents

1 From Zero To One, Abstractions

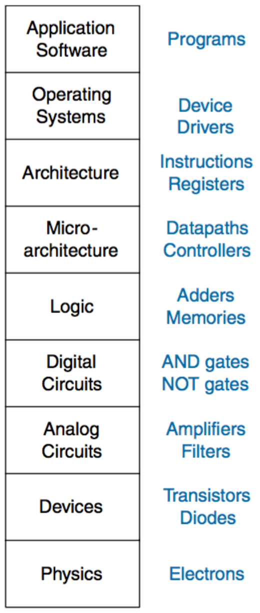


Figure 1: Levels of abstraction for electronic computing system

1.1 The Three -Y's

Hierarchy involves dividing a system into modules, then further subdividing each of these modules until the pieces are easy to understand

Modularity states that the modules have well-defined functions and interfaces, so that they connect together easily without unanticipated side effects

Regularity seeks uniformity among the modules. Common modules are reused many times, reducing the number of distinct modules that must be designed

1.2 Hexadecimal Number System

Hexadecimal Digit	Decimal Digit	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

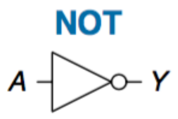
1.3 Bytes, Nibbles, and All That Jazz

A group of eight bits is called a byte. Represents $256 = 2^8$ possibilities.
A group of four bits is called a nibble. Represents $16 = 2^4$ possibilities.
A microprocessor is a processor built on a single chip.
Microprocessors handle data in chunks called words. The size of a word depends on the architecture of the microprocessor.
Within a group of bits, the bit in the 1's column is called the least significant bit (lsb), and the bit at the other end is called the most significant bit (msb).
Within a word, the bytes are identified as least significant byte (LSB) through most significant byte (MSB).

1.4 Logic Gates

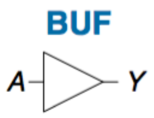
A circuit element that performs a basic logic function

Single Input



$Y = \overline{A}$

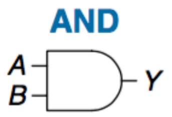
A	Y
0	1
1	0



$Y = A$

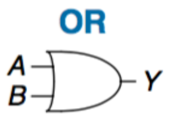
A	Y
0	0
1	1

Double Output



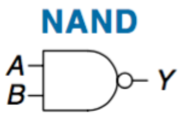
$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



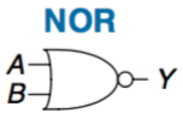
$Y = A + B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



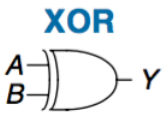
$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



$Y = \overline{A + B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

1.5 Beneath the Digital Abstraction

A digital system uses discrete-valued variables.
Variables are represented by continuous physical quantities such as the voltage on a wire, the position of a gear, or the level of fluid in a cylinder.

For example:
Consider representing a binary signal A with a voltage on a wire.
Let 0 volts (V) indicate A = 0 and 5V indicate A = 1.
Any real system must tolerate some noise, so 4.97V probably ought to be interpreted as A = 1 as well.
But what about 4.3V? Or 2.8V? Or 2.5000000V?

1.5.1 Supply Voltage

Suppose the lowest voltage in the system is 0V, also called ground or GND
The highest voltage in the system comes from the power supply and is usually called V_{DD}

1.5.2 Logic Levels

Mapping of a continuous variable onto a discrete binary variable is done by defining logic levels. First gate is called the driver and the second gate is called the receiver. Output of the driver is connected to the input of the receiver.
The driver produces a LOW (0) output in the range of 0 to V_{OL} or a HIGH (1) output in the range of V_{OH} to V_{DD}

1.5.3 Noise Margins

If the output of the driver is to be correctly interpreted at the input of the receiver, we must choose $V_{OL} < V_{IL}$ and $V_{OH} > V_{IH}$
Even if the output of the driver is contaminated by some noise, the input of the receiver will still detect the correct logic level. Noise margin is the amount of noise that could be added to a worst-case output such that the signal can still be interpreted as a valid input.

1.5.4 DC Transfer Characteristics

The DC transfer characteristics of a gate describe the output voltage as a function of the input voltage when the input is changed slowly enough that the output can keep up.
Called transfer characteristics because they describe the relationship between input and output voltages.

2 CMOS Transistors

Transistors are electrically controlled switches that turn ON or OFF when a voltage or current is applied to a control terminal.
Two main types of transistors are bipolar transistors and metal-oxide-semiconductor field effect transistors (MOSFETs or MOS transistors).

2.1 Semiconductors

MOS transistors are built from silicon.
Silicon has four electrons in its valence shell and forms bonds with four adjacent atoms, resulting in a crystalline lattice.
By itself, silicon is a poor conductor due to all the electrons are tied up in covalent bonds.
Becomes a better conductor when small amounts of impurities, called dopant

atoms, are added.
The conductivity of silicon changes over many orders and of magnitude depending on the concentration of dopants, silicon is called a semiconductor.
Add Arsenic
If arsenic (As), a group V dopant, is added, the dopant atoms have an extra electron that is not involved in the bonds.
Electron can easily move about the lattice, leaving an ionized dopant atom (As+) behind.
Electron carries a negative charge, so arsenic is an n-type dopant.
Add Boron
If boron (B), a group III dopant, is added, the dopant atoms are missing an electron.
The missing electron is called a hole.
An electron from a neighboring silicon atom may move over to fill the missing bond, forming an ionized dopant atom (B-) and leaving a hole at the neighboring silicon atom.
The hole can migrate around the lattice.
Hole is a lack of negative charge, so it acts like a positively charged particle.
So boron is a p-type dopant.

2.2 Diodes

Junction between p-type and n-type silicon is called a diode. The p-type region is called the anode and the n-type region is called the cathode.
When the voltage on the anode rises above the voltage on the cathode, the diode is forward biased, and current flows through the diode from the anode to the cathode.
When the anode voltage is lower than the voltage on the cathode, the diode is reverse biased, and no current flows.
The diode symbol intuitively shows that current only flows in one direction.

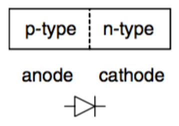


Figure 1.27 **The p-n junction diode structure and symbol**

2.3 Capacitors

A *capacitor* consists of two conductors separated by an insulator.
When a voltage V is applied to one of the conductors, the conductor accumulates electric *charge* Q and the other conductor accumulates the opposite charge $-Q$.
The *capacitance* C of the capacitor is the ratio of charge to voltage: $C = \frac{Q}{V}$.
The capacitance is proportional to the size of the conductors and inversely

proportional the distance between them.
Capacitance is important because charing or discharging a conductor takes time and energy. More capacitance means that a circuit will be slower and require more energy to operate.

2.4 nMOS and pMOS Transistors

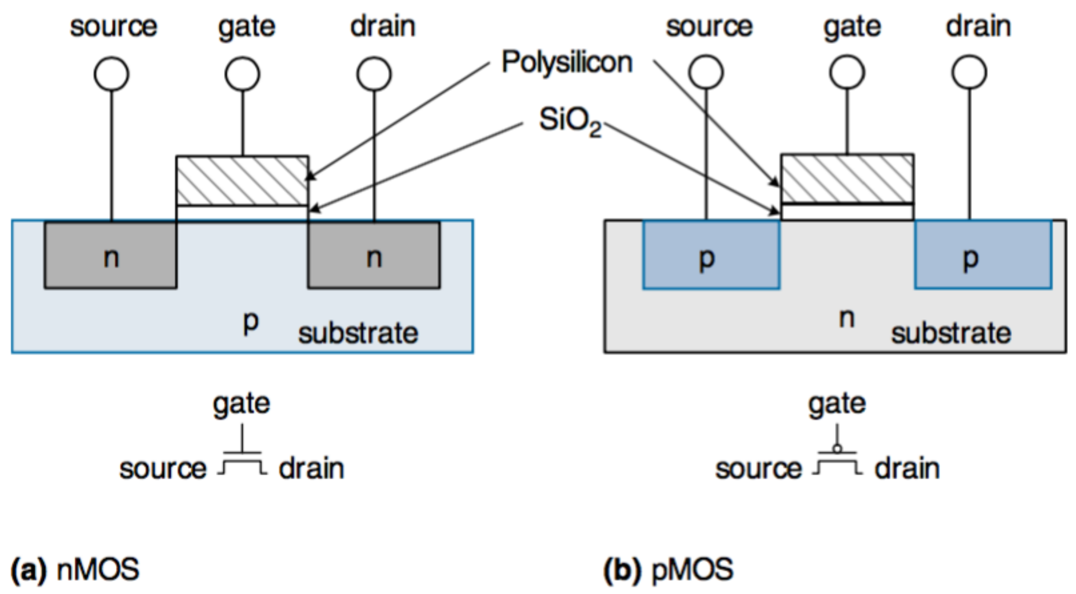
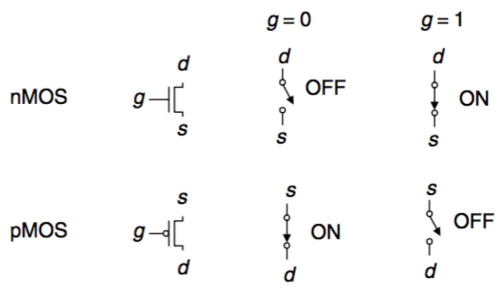


Figure 1.29 nMOS and pMOS transistors

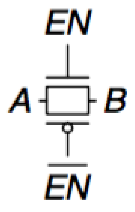
A MOSFET is a sandwich of several layers of conducting and insulating materials.
There are two flavors of MOSFETs: nMOS and pMOS.
The n-type transistors, called *nMOS*, have regions of n-type dopants adjacent tot he gate called the *source* and the *drain* and are built on a p-type semiconductor substrate.
The *pMOS* transistors are just the opposite, consisting of p-type source and drain regions in an n-type *substrate*.
A MOSFET behaves as a voltage-controlled switch in which the gate voltage creates an electric field that turns ON or OFF a connection between the source and drain.
nMOS transistors pass 0's well but passes 1's poorly. Similarly, pMOS transistors pass 1's well but 0's poorly.
nMOS transistors need a p-type substrate, and pMOS transistors need an

n-type substrate. To build both flavors of transistors on the same chip, manufacturing processes typically start with a p-type wafer, then implant n-type region called wells where the pMOS transistors should go. These processes that provide both flavors of transistors are called Complementary MOS or CMOS.
CMOS processes are used to build the vast majority of all transistors fabricated today.
CMOS processes provide two types of electrically controlled switches. nMOS transistors are OFF when the gate is 0 and ON when the gate is 1. pMOS transistors are just the opposite: ON when the gate is 0 and OFF when the gate is 1.



2.5 Transmission Gates

At times, designers find it convenient to use an ideal switch that can pass both 0 and 1 well.
nMOS transistors are good at passing 0 and pMOS transistors are good at passing 1, so the parallel combination of the two passes both values well.
Image to the right shows such a circuit, called *transmission gate* or *pass gate*.



The two sides of the switch are called A and B because a switch is bidirectional and has no preferred input or output side. The control signals are called *enables*, EN and $\overline{EN}$.
When $EN = 0$ and $\overline{EN} = 1$, both transistors are OFF. Hence, the transmission gate is OFF or disabled, so A and B are not connected.
When $EN = 1$ and $\overline{EN} = 0$, the transmission is ON or enabled, and any logic value can flow between A and B.

2.6 Pseudo-nMOS Logic

An N-input CMOS NOR gate uses N nMOS transistors in parallel and N pMOS transistors in series. Transistors in series are slower than transistors in parallel, just as resistors in series have more resistance than resistors in parallel.
Moreover, pMOS transistors are slower than nMOS transistors because holes cannot move around the silicon lattice as fast as electrons. Therefore parallel

nMOS transistors are fast and the series pMOS transistors are slow, especially when many are in series.

Pseudo-nMOS logic replaces the slow stack of pMOS transistors with a single weak pMOS transistor that is always ON. This pMOS transistors is often called a *weak pull-up*. The physical dimensions of the pMOS transistor are selected so that the pMOS transistor will pull the output, Y, HIGH weakly—that is, only if none of the nMOS transistors are ON. But if any nMOS transistor is ON, it overpowers the weak pull-up and pulls Y down close enough to GND to produce a logic 0.

The advantage of pseudo-nMOS logic is that it can be used to build fast NOR gates with many inputs.
The disadvantage is that a short circuit exists between V_{DD} and GND when the output is LOW; the weak pMOS and nMOS transistors are both ON. The short circuit draws continuous power, so pseudo-nMOS logic must be used sparingly.

3 Summary So Far

There are 10 kinds of people in this world: those who can count in binary and those who can't.
The real world is analog, though digital designers discipline themselves to use a discrete subset of possible signals. In particular, binary variables have just two states: 0 and 1, also called FALSE and TRUE or LOW and HIGH.

Logic gates compute a binary output from one or more binary inputs. Some of the common logic gates are:

NOT: TRUE when all input is FALSE

AND: TRUE when all input are TRUE

OR: TRUE when any inputs are TRUE

XOR: TRUE when an odd number of inputs are TRUE

Logic gates are commonly built from CMOS transistors, which behave as electrically controlled switches.
nMOS transistors turn ON when the gate is 1.
pMOS transistors turn ON when the gate is 0.

4 Introduction to Combinational Logic Design

In digital electronics, a *circuit* is a network that processes discrete-valued variables.
A circuit can be viewed as a black box, with

- one or more discrete-valued *input terminals*
- one or more discrete-valued *output terminals*
- a *functional specification* describing the relationship between inputs and outputs
- a *timing specification* describing the delay between inputs changing and output responding

Peeing inside the black box, circuits are composed of nodes and elements.
An *element* is itself a circuit with inputs, outputs, and a specification.
A *node* is a wire, whose voltage coveys a discrete-valued variable. Nodes are classified as *input, output,* or *internal.*
Input receive values from the external world.
Output deliver values to the external world.
Wires that are not inputs or outputs are called internal nodes.

Digital circuits are classified as *combinational* or *sequential.*
A combinational circuit's outputs depend only on the current values of the inputs; in other words, it combines the current input values to compute the output. For example, a logic gate is a combinational circuit.
A sequential circuit's outputs depend on both current and previous values of the inputs; in other words, it depends on the input sequence.
A combinational circuit is *memoryless*, but a sequential circuit has *memory.*

The functional specification of a combination circuit expresses the output values in terms of the current input values.
The timing specification of a combinational circuit consists of lower and upper bounds on the delay from input to put.

To simply drawings, a single line with a slash through it and a number next to it is often used to indicate a *bus*, a bundle of of multiple signals.

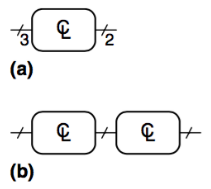


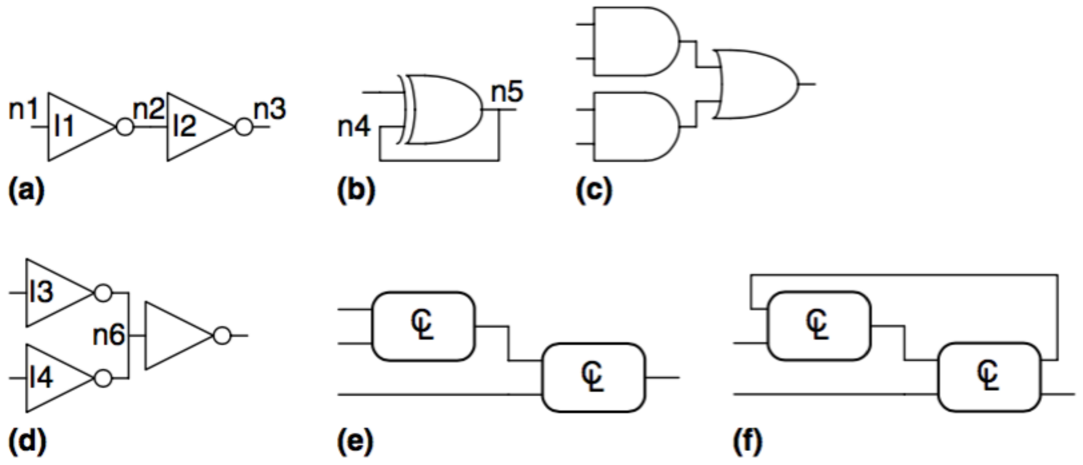
Figure 2.6 Slash notation for multiple signals

In Figure 2.6(a), it represents a block of combination logic with three inputs and output outputs.
If the number of bits is unimportant or obvious from the context, the slash may be shown without a number.
Figure 2.6(b) indicates two blocks of combinational logic with an arbitrary number of outputs from one block serving as inputs to the second block.

The rules of *combinational composition* tell us how we can build a large combinational circuit from smaller combinational circuit elements.
A circuit is combinational if it consists of interconnected circuit elements such that:

- Every circuit element is itself combinational
- Every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element
- The circuit contains no cyclic paths: every path through the circuit visits each circuit node at most once

Examples:



(a) is combinational. It is constructed from two combinational circuit elements (inverters $I1$ and $I2$). It has three nodes: $n1$, $n2$, and $n3$. $n1$ is an input to the circuit and to $I1$; $n2$ is an internal node, which is the output of $I1$ and the input to $I2$; $n3$ is the output of the circuit and of $I2$.
(b) is *not* combinational, because there is a cyclic path: the output of the

XOR feeds back to one of its input. Hence, a cyclic path starting at $n4$ passes through the XOR to $n5$, which returns to $n4$.
(c) is combinational.
(d) is not combinational, because node $n6$ connects to the output terminals of both $I3$ and $I4$.
(e) is combinational, illustrating two combinational circuits connecting to from a larger combinational circuit.
(f) does not obey the rules of combinational composition because it has a cyclic path through the two elements. Depending on the functions of the elements, it may or may not be a combinational circuit.

The functional specification of a combinational circuit is usually expressed as a truth table or a Boolean equation.

5 Boolean Equations

Boolean equations deal with variables that are either TRUE or FALSE, so they are perfect for describing digital logic.

5.1 Terminology

The *complement* of a variable, A, is its inverse, $\overline{A}$.
The variables or its complement is called a *literal*. For example, A, $\overline{A}$, B, and $\overline{B}$ are literals.
A is called the *true form* of the variable and $\overline{A}$ the complementary form; "true form" does not mean that A is TRUE, but merely that A does not have a line over it.

The AND of one or more literals is called a *product* or an *implicant*.
$\overline{A}B$, $A\overline{BC}$, and B are all implicants for a function of three variables.
A *minterm* is a product involving all of the inputs to the function. $A\overline{BC}$ is a minterm for a function of the three variables A, B, and C, but $\overline{A}B$ is not, because it does not involve C.

Similarly, the OR of one or more literals is called a *sum*. A *maxterm* is a sum involving all of the inputs to the function. $A + \overline{B} + C$ is a maxterm for a function of the three variables A, B, and C.

5.2 Sum-of-Products Form

A truth table of N inputs contains 2^N rows, one for each possible value of the inputs.
Each row in a truth value is associated with a minterm that is TRUE for that row.
A Boolean equation can be written for any truth table by summing each of the minterms for which the output, Y, is TRUE. This is called the *sum-of-products canonical form* of a function because it is the sum (OR) of products (ANDs forming minterms).

The sum-of-products form provides a Boolean equation for any truth table with any number of variables. Unfortunately, it does not necessarily generate the simplest equation.

6 Boolean Algebra

6.1 Axioms

	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1′	$B = 1$ if $B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2′	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3′	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4′	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5′	$1 + 0 = 0 + 1 = 1$	AND/OR

6.2 Theorems of One Variable

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

6.3 Theorems of Several Variables

	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + (B \bullet D) = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots}$ $= (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots}$ $= (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan's Theorem

7 From Logic to Gates

A *schematic* is a diagram of a digital circuit showing the elements and the wires that connect them together.

Good Style in Circuit Drawing:

- Assume all literals (variables and their negations) are available
- Rectilinear wires, dots when wires split
- Do not draw spaghetti wires for inputs; instead, write each literal as needed

8 Multilevel Combinational Logic

Logic in sum-of-products form is called *two-level logic* because it consists of literals connected to a level of AND gates connected to a level of OR gates.
Designers often build circuits with more than two levels of logic gates. These multilevel combinational circuits may use less hardware than their two-level counterparts.
Bubble pushing is especially helpful in analyzing and designing multilevel circuits.

8.1 Hardware Reduction

Some logic functions require an enormous amount of hardware when built using two-level logic. A notable example is the XOR function of multiple variables. A three-input XOR can be built out of a cascade of two-input XORs.
Similarly, an eight-input XOR would require 128 eight-input AND gates and one 128-input OR gate for a two level-sum-of-products implementation. A much better option is to use a tree of two-input XOR gates.

"Best" has many meanings: fewest gates, fastest, shortest design time, least cost, least power consumption.
"Best" circuit in one technology is not necessarily the best in another. ANDs and ORs are used often, but in CMOS, NANDs and NORs are more efficient.

8.2 Bubble Pushing

CMOS circuits prefer NANDs and NORs over ANDs and ORs. However, reading the equation by inspection can be difficult.
Bubble pushing is a helpful way to redraw these circuits so that the bubbles cancel out and the function can be more easily determined.
Guidelines for bubble pushing:

- Begin at the output of the circuit and work toward the inputs.
- Push any bubbles on the final output back toward the inputs so that you can read an equation in terms of the output.
- Working backward, draw each gate in a form so that bubbles cancel. If the current gate has an input bubble, draw the preceding gate with an output bubble. If the current gate does not have an input bubble, draw the preceding gate without an output bubble.

9 Combinational Building Blocks

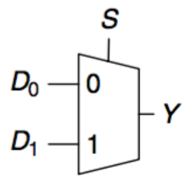
Combinational logic is often grouped into larger building blocks to build more complex systems. This is an application of the principle of abstraction, hiding the unnecessary gate-level details to emphasize the function of the building block.

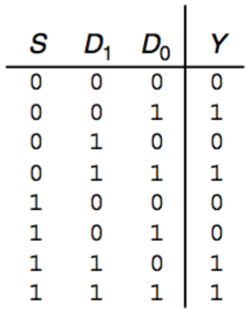
9.1 Multiplexers

Multiplexers, also known as *mux*, are among the most commonly used combinational circuits. They choose an output from among several possible inputs based on the value of a *select* signal.

9.1.1 2:1 Multiplexer

Image shows the schematic and truth table for a 2:1 multiplexer with two data inputs, D_0 and D_1, as select input, S, and one output, Y.
The multiplexer chooses between the two data inputs based on the select: if $S = 0$, $Y = D_0$, and if $S = 1$, $Y = D_1$.
S is also called a control signal because it controls what the multiplexer does.





S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Multiplexers can be built from tristate buffers.
The tristate enables are arranged such that, at all times, exactly one tristate buffer is active.
When $S = 0$, tristate T_0 is enabled, allowing D_0 to flow to Y.
When $S = 1$, tristate T_1 is enabled, allowing D_1 to flow to Y.

9.1.2 Wider Multiplexers

A 4:1 multiplexer has four data inputs and one output. Two select signals are needed to choose among the four data inputs. The 4:1 mux can be built using sum-of-products logic, tristates, or multiple 2:1 mux.

The product terms enabling the tristates can be formed using AND gates and inverters. They can also be formed using a decoder.

In general, an $N : 1$ multiplexer needs $log_2 N$ select lines.

9.1.3 Multiplexer Logic

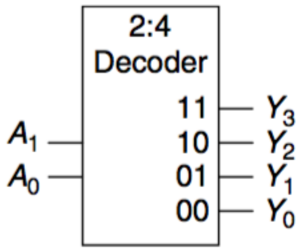
Multiplexers can be used as *lookup tables* to perform logic functions.
In general, a 2^N-input multiplexer can be programmed to perform any N-input logic functions by applying 0's and 1's to the appropriate data inputs.

By changing the data inputs, the multiplexer can be reprogrammed to perform a different function.

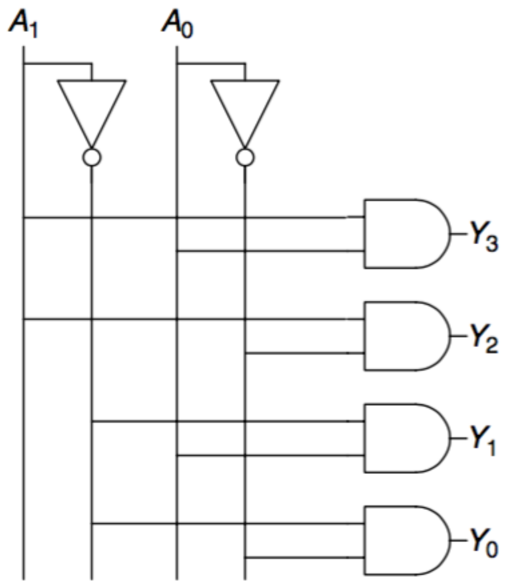
9.2 Decoder

A decoder has N inputs and 2^N outputs.
It asserts exactly one of its outputs depending on the input combination. The outputs are called *one-hot*, because exactly one is "hot" (HIGH) at a given time.

Example of 2:4 decoder and its implementation:



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



9.2.1 Decoder Logic

Decoders can be combined with OR gates to build logic functions.
When using decoders to build logic, it is easiest to express functions as a truth table or in canonical sum-of-products form. A N-input function with M 1's in the truth table can be built with an $N : 2^N$ decoder and an M-input or gate attached to all of the minterms containing 1's in the truth table.

10 Summary So Far

A digital circuit is a module with discrete-valued inputs and outputs and a specification describing the function and timing of the module.

The function of a combinational circuit can be given by a truth table or a Boolean equation. The Boolean equation for any truth table can be obtained systematically using sum-of-products or product-of-sums form.
In sum-of-products form, the function is written as the sum (OR) of one or more implicants. Implicants are the product (AND) of literals. Literals are the true or complementary forms of the input variables.

Boolean equation can be simplified using the rules of Boolean algebra. In particular, they can be simplified into minimal sum-of-products form by combining implicants that differ only in the true and complementary forms of one of the literals: $PA + P\overline{A} = P$.

Logic gates are connected to create combinational circuits that perform the desired function.
Any functions in sum-of-products form can be built using two-level logic with the literals as inputs: NOT gates form the complementary literals, AND gates form the products, and OR gates form the sum.
CMOS circuits favor NAND and NOR gates because these gates can be built directly from CMOS transistors without requiring extra NOT gates.
When using NAND and NOR gates, bubble pushing is helpful to keep track of the inversions.

Logic gates are combined to produce larger circuits such as multiplexers, decoders, and priority circuits.
A multiplexer chooses one of the data inputs based on the select input.
A decoder sets one of the outputs HIGH according to the input.
A priority circuit produces an output indicating the highest priority input.

The timing specification of a combinational circuit consists of the propagation and contamination delays through the circuit.
These indicate the longest and shortest times between an input change and the consequent output change.
Calculating the propagation delay of a circuit involves identifying the critical path through the circuit, then adding up the propagation delays of each element along that path.

11 Introduction to Sequential Logic Design

The output of sequential logic depend on both current and prior input values. Hence, sequential logic has memory. Sequential logic might explicitly remember certain previous inputs, or it might distill the prior inputs into a smaller amount of information called the *state* of the system.
The state of a digital sequential circuit is a set of bits called *state variables* that contain all the information about the past necessary to explain the future behavior of the circuit.

12 Latches and Flip-Flops

The fundamental building block of memory is a *bistable* element, an element with two stable states.
An element with N stable states conveys $\log_2 N$ bits of information, so a bistable element stores one bit.
The state of the cross-coupled inverters is contained in one binary state variable, Q. The value of Q tells us everything about the past that is necessary to explain the future behavior of the circuit.
Specifically, if $Q = 0$, it will remain 0 forever, and if $Q = 1$, it will remain 1 forever.
The circuit does have another node, $\overline{Q}$, but $\overline{Q}$ does not contain any additional information because if Q is known, $\overline{Q}$ is also known. $\overline{Q}$ is also an acceptable choice for the state variable.

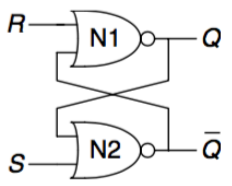
When power is first applied to a sequential circuit, the initial state is unknown and usually unpredictable. It may differ each time the circuit is turned on.

Although the cross-coupled inverters can store a bit of information, they are not practical because the user has no inputs to control the states.
However, other bistable elements, such as *latches* and *flip-flops*, provide inputs to control the value of the state variable.

Just a Y is commonly used for the output of combinational logic, Q is commonly used for the output of sequential logic.

12.1 SR Latch

SR latch is one of the simplest sequential circuit as it is composed of two cross-coupled NOR gates.
The latch has two inputs, S and R, and two outputs, Q and $\overline{Q}$.
The SR latch is similar to the cross-coupled inverters, but its state can be controlled through the S and R inputs, which *set* and *reset* the output Q.



Consider the four possible combinations of R and S:

Case I: $R = 1, S = 0$
N1 sees at least one TRUE input, R, so it produces a FALSE output on Q. N2 sees both Q and S FALSE, so it produces a TRUE output on $\overline{Q}$.

Case II: $R = 0, S = 1$
N1 receives inputs of 0 and $\overline{Q}$. Because we don't yet know $\overline{Q}$, we can't determine the output Q. N2 receives at least one TRUE input, S, so it produces a FALSE output on $\overline{Q}$. Now we can revisit N1, knowing that both inputs are FALSE, so the output Q is TRUE.

Case III: $R = 1, S = 1$
N1 and N2 both set at least one TRUE input (R or S), so each produces a FALSE output. Hence Q and $\overline{Q}$ are both FALSE.

Case IV: $R = 0, S = 0$
N1 receives inputs of 0 and $\overline{Q}$. Because we don't yet know $\overline{Q}$, we can't determine the output. N2 receives inputs of 0 and Q. Because we don't yet know Q, we can't determine the output. Now we are stuck. This is reminiscent of the cross-coupled inverters. But we know that Q must either be 0 or 1. So we can solve the problem by checking what happens in each of these sub-cases.

Case IVa: $Q = 0$
Because S and Q are FALSE, N2 produces a TRUE output on $\overline{Q}$. Now N1 receives one TRUE input, $\overline{Q}$, so its output, Q, is FALSE.

Case IVb: $Q = 1$
Because Q is TRUE, N2 produces a FALSE output on $\overline{Q}$. Now N1 receives two FALSE inputs, R and $\overline{Q}$, so its output, Q, is TRUE.

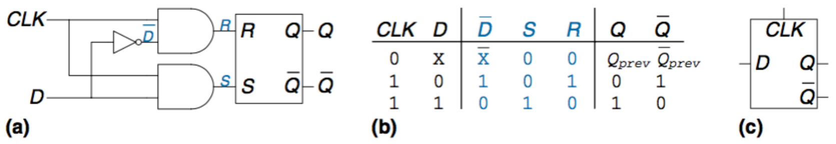
Suppose Q has some known prior value, Q_{prev}, before we enter Case IV. Q_{prev} is either 0 or 1, and represents the state of the system. When R and S are 0, Q will remember this old value, Q_{prev}, and $\overline{Q}$ will be its complement, $\overline{Q}_{prev}$. This circuit has memory.

Like the cross-coupled inverters, the SR latch is a bistable element with one bit of state stored in Q. However, the state can be controlled through the S and R inputs.
When R is asserted, the state is reset to 0.
When S is asserted, the state is set to 1.
When neither is asserted, the state retains its old value.

12.2 D Latch

The SR latch is awkward because it behaves strangely when both S and R are simultaneously asserted. Moreover, the S and R inputs conflate the issues of *what* and *when*. Asserting one of the inputs determines not only *what* the state should be but also *when* it should change.

The D latch has two inputs. The *data* input, D, controls what the next state should be. The *clock* input, CLK, controls when the state should change.



CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Figure 3.7 D latch: (a) schematic, (b) truth table, (c) symbol

The clock controls when data flows through the latch.
When $CLK = 1$, the latch is *transparent*. The data at D flows through to Q as if the latch were just a buffer.
When $CLK = 0$, the latch is *opaque*. It blocks the new data from flowing through to Q, and Q retains the old value.
The D latch updates its state continuously $CLK = 1$.

12.3 D Flip-Flop