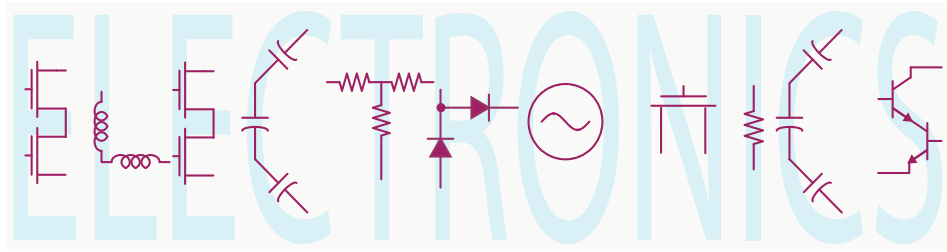
EE 42/100
Lecture 21: Digital Gates and Combinatorial Logic



Rev B 4/9/2012 (9:39 PM)

Prof. Ali M. Niknejad

University of California, Berkeley

Digital versus Analog

- Real signals are continuous (at least we perceive them that way) whereas binary signals can only take on two values.
- For instance the current temperature in this room is a certain number of degrees and the resolution of this number is in principle infinite (noise limited).
- But for many applications, the precise value is not important. A finite representation (certain number of significant figures) is all that we need. We may therefore sample the signal periodically (not continuously) and also round off the actual amplitude of the signal. Note that we are making two approximations: the discrete time sampling of the signal and the discretization process.
- Intuitively, as long as we sample the signal faster than the rate at which it changes (can be made precise using Nyquist's Sampling Theorem), then no information is lost.

Quantization Noise

- On the other hand, the quantization of the signal introduces round-off errors in our signal. If we subtract out the ideal signal $s(t)$ from the quantized signal $\hat{s}(t)$, the difference is the error signal, which varies randomly from sample to sample.

$$n(t) = s(t) - \hat{s}(t)$$

- We can therefore think of the process as introducing “quantization noise” in the signal.

$$\hat{s}(t) = s(t) - n(t)$$

- In other words, if we are insensitive to changes of the signal smaller than the quantization noise, then the approximation is okay.

Analog-to-Digital Converter

- The important specifications for an ADC are the sampling rate (clock rate) of the ADC and its resolution (number of bits).
- Analog signals need to be sampled at a rate of twice the signal bandwidth (audio bandwidth is roughly 5 kHz).
- The resolution determines the smallest discernible signal since the full-scale input voltage (say 5V) is divided by 2^N where N is the number of bits. Any signal smaller than this level is lost in the "quantization noise" (round-off error).

Binary Numbers

- Binary numbers can only take on two values $\{0, 1\}$. To specify a bigger number, you have to use more digits. For instance:

$$1010_b = (1)2^3 + (0)2^2 + (1)2^1 + 0(2^0) = 8 + 2 = 10$$

- Some aliens with only two fingers may naturally use binary numbers but we use them because our computers can only store two states (note: digit means finger).
- To convert a decimal number into binary, you can repeatedly divide by 2 and take the remainder as the next digit.
- Example: 50 in binary is:
  $50/2 = 25r0, 25/2 = 12r1, 12/2 = 6r0, 6/2 = 3r0, 3/2 = 1, r1, 1/2 = 0r1$. This implies the number is given by 110010_b (read it backwards). Verify that
  $2^4 + 2^1 = 8 + 4 = 12$. Or

$$32 + 16 + 2 = 2^5 + 2^4 + 2^1 = 110010_b$$

Binary Signals

- Binary signals are designed to only take on two values, say $\{0V, 5V\}$ to represent $\{0, 1\}$ (positive or negative logic possible).
- As signals propagate in wires, inevitably noise, distortion, and cross-talk occurs, which corrupts the signal. An analog signal is corrupted directly whereas a digital signal is only corrupted if the noise exceeds the noise margin of the circuit.
- Digital signals are therefore more robust and can be easily regenerated whereas analog signals get more noisy as we process them.

Hexadecimal

- A long string of binary numbers if hard to read. We therefore group 4 bits into a *nibble* and group eight bits into a *byte* (early CS people had a good sense of humor!).
- A nibble can take on 16 different distinct values, so a base 16 number system is used to describe it: $\{0, 1, 2, \cdots, A, B, C, D, E, F\}$. Note that we use letters to represent numbers. This takes some getting used to!
- A byte is described by two hexadecimal digitals. For instance, we converted 50 in binary as 110010_b. Let's extend it to 8 bits by zero padding, 00110010_b, or two nibbles of 0011 and 0010, which in hex are 3_{hex} and 2_{hex}, which we write as 32_{hex}. Verify

$$32_{hex} = 3 \cdot 16^1 + 2 \cdot 16^0 = 48 + 2 = 50$$

Binary Arithmetic

- Binary arithmetic is exactly the same as what you learned in grade school, except there are only two digits! For adding single bits

$$0 + 0 = 0$$

$$1 + 0 = 0 + 1 = 1$$

$$1 + 1 = 0 + \text{carry}$$

- Let's add two nibbles

```
 0101    (5)
+0011    (3)
-----
 1000    (8)
```

Signed Binary and Two's Complement

- To represent negative numbers, we could add a "sign bit" in front of the number but it turns out that addition no longer works (between positive and negative numbers). A more elegant system that preserves addition (and hence supports subtraction) is the two's complement system.
- In the two's complement system, the first half of the 2^N numbers are assigned to zero and the positive integers and the second half is used for the negative integers. Because addition is modulo 2^N (wraps around after 2^N), when you add a number m and it's negative counterpart, it should sum up to wrap back to zero, which would happen if m were equal to $2^N - m$.
- Take the number 0001. It's 2's complement is $2^4 - 1 = 15$ or 1111. If we add these two numbers in binary, we get

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

  If we only preserve 4 bits of the result, then we're back to zero.
- Another way to calculate the 2's complement: Invert all the bits and add 1. So for 1 we have $0001 \rightarrow 1110 \rightarrow 1111$, which is the same as above.

Two's Complement Example

- Example: Add $5 + (-4)$. $5 = 0101$ and -4 is formed by first forming 4, $4 = 0100$, inverting, 1011, and then adding 1, $0001 + 1011 = 1100$. Now if we add $5 + (-4) = 0101 + 1100 = 0001$, which shows that everything works!
- Note that the most significant bit is still 1 for all negative numbers, which is also a kind of sign bit.

What's a logic gate?

- First let's consider the *Digital Abstraction* in which we build circuits where the inputs and outputs are only interpreted as "high" or "low". Since the circuit only operates on two inputs, it's a binary circuit. In the binary circuit, the two inputs are binary digits "0" and "1".
- A logic gate performs a logic function. It has several inputs and it produces an output. The inputs and outputs can only take on values of {0,1}.
- By convention, all the inputs are put on the left and the output(s) are on the right
- The function is completely described by a truth table

AND, OR, and XOR

- The AND function is very common. The output is 1 only if both inputs are one. We write this relation as a product

$$y = AB$$

- The OR function produces a 1 as long as one input is 1. We write this relation as

$$y = A + B$$

- Exclusive OR, or XOR, only produce a one if one of its inputs is 1. If both are 1 or 0, the output is 0. This is written as

$$y = A \oplus B$$

NOT Gate

- The NOT gate inverts its input and it's not surprising that it's also known as an inverter. As we'll see, this is a very crucial function. The notation for NOT is

$$y = A'$$

$$y = \sim A$$

$$y = !A$$

$$y = \overline{A}$$

- Notice that the OR operation is the same as the AND operation if we NOT its inputs.

Buffer is more than a wire

- A buffer has the same output as its input. It seems rather trivial!
- From the perspective of logic, the buffer is the same as a wire, but in a real electrical circuit, a buffer is an amplifier that can be used to reduce the capacitive loading of a stage and help drive large capacitance. Thinks of the voltage follower.

NAND and NOR

- The NAND and NOR gates are the same as AND and OR except the outputs are inverted. In real circuit implementation, they are often easier to realize than AND and OR.

Multi Input Logic

- So far we have only dealt with two input logic gates. The concept generalizes to any number of inputs and outputs.
- Multi-input logic gates can be synthesized from two-input gates.

Combinatorial versus Sequential

- The kinds of gates that we have been describing can be interconnected to build a combinatorial logic circuit. This is as opposed to a “sequential” logic circuit. The difference is that a combinatorial logic circuit is only a function of the current inputs. In other words, it has no memory. A sequential logic circuit, on the other hand, has memory and so the current output is a function of the current input and past inputs/outputs.

Boolean Equations

- The algebra of binary digits is known as Boolean algebra. It's very simple because there are only two possible inputs. Typical expressions are of the following form:

$$y = A\overline{B}\,\overline{C}$$

- When an expression is a product of all of it's inputs, it's known as a minterm. When a term is a sum of all of it's inputs, it's called a maxterm.

Order of Precedence

- How do you interpret an expression such as $y = A + BC$?

$$y = A + BC \stackrel{?}{=} (A\ OR\ B)\ AND\ C \stackrel{?}{=} A\ OR\ (B\ AND\ C)$$

- The rules of precedence determine the order of evaluation. The NOT operator has the highest precedence, followed by AND, and then OR.

Sum of Products Form

- Any function can be written in a sum of products form. Simply examine the truth table and for each '1' output, write the corresponding minterm:

$$y = \overline{A}\,\overline{B}\,C + A\overline{B}C + AB\overline{C}$$

Product of Sums

- Any function can also be written in the product of sums form. Simply examine the truth table and for each '0' input, write the corresponding maxterm:

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

- When is the product of sums form better? When it produces a shorter expression. This happens of course when there are fewer zeros than ones in the truth table output.

Boolean Algebra Axioms

A1	$B = 0$ if $B \neq 1$	A1'	$B = 1$ if $B \neq 0$
A2	$\overline{0} = 1$	A2'	$\overline{1} = 0$
A3	$0 \cdot 0 = 0$	A3'	$1 + 1 = 1$
A4	$1 \cdot 1 = 1$	A4'	$0 + 0 = 0$
A5	$0 \cdot 1 = 1 \cdot 0 = 0$	A5'	$1 + 0 = 0 + 1 = 1$

- By definition, axioms cannot be proven. We use them as self-consistent definitions that can be used as givens when deriving theorems.

Boolean Algebra Theorems

T1	$B \cdot 1 = B$	T1'	$B + 0 = B$
T2	$B \cdot 0 = 0$	T2'	$B + 1 = 1$
T3	$B \cdot B = B$	T3'	$B + B = B$
T4	$\overline{\overline{B}} = B$	T4'	$\overline{\overline{B}} = B$
T5	$B \cdot \overline{B} = 0$	T5'	$B + \overline{B} = 1$

- Theorems can be proven from Axioms. Most of these are pretty obvious but you can also always check by putting in all possible inputs and verifying that both sides are equal.
- These theorems are very useful for simplifying results.

More Theorems

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

- The theorems are the usual things we're familiar with from algebra, including commutativity, associativity, and distributivity. There's also covering, combining, and consensus, which are new.
- De Morgan's Theorem is extremely useful and should be studied carefully.

Equation Minimization

- Using the Axioms and Theorems of Boolean Algebra, we can simplify expressions. For example

$$y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$$

- Can be simplified using distribution in the first two terms

$$y = \overline{B}\,\overline{C}(A + \overline{A}) + A\overline{B}C$$

- But $A + \overline{A} = 1$ and anything times 1 is itself

$$y = \overline{B}\,\overline{C}(1) + A\overline{B}C = \overline{B}\,\overline{C} + A\overline{B}C$$

- Can we do better? Notice the simplification resulted because the first two terms only differed by 1 bit, A. But the last two terms also differ in only 1 bit, C. We could have combined these two terms as well, giving us the same complexity in the minimized term.
- But we can also be clever and duplicate the middle term! That's because $A + A = A$. Show that we then can get down to

$$y = \overline{B}\,\overline{C} + A\overline{B}$$

From Logic to Gates

- It's now very easy to see how we can synthesize any Boolean expression using logic gates. For instance, to realize an expression in sum of products terms, we first form the miterms using AND gates and then we OR the outputs.

Other Possible Ouputs

- When an illegal output occurs in a logic gate, we denote that as ‘X’. Suppose we have a circuit where there is a contention between two logic gates. The output will then be an illegal output (not a well defined ‘0’ or ‘1’) and the circuit may not function correctly or as expected.
- The same notation is used for a “don’t care” output, in other words if we don’t care about a particular output, so be careful.
- Many logic gates have an “enable” input that can put the output into a third state, known as a floating or high impedance (high-‘Z’) state. This is known as a ‘Z’ state for this reason.
- A common block with this functionality is a tristate buffer. This is very useful in busses and other circuits where multiple gates drive the same output. Now a contention is not longer present as long as only one gate is enabled.

MUX

- A MUX or multiplexer simply selects one of its inputs based on the “select” input. You can think of it as a three input logic gate with the truth table shown below.
- A MUX can be implemented using a two-level logic, or using tri-state buffers.

4:1 MUX

- Higher order MUX circuits are also handy and can be realized in different ways.
- Notice that any truth table with 2 inputs can be realized by connecting the inputs of the 4:1 MUX to either supply or gnd !

Decoders

- A decoder is very handy when building a memory. There are N inputs and 2^N outputs. Only one output is selected at a time.

Logic Levels

- In a real circuit, the actual voltage waveforms take on a continuous range of values. In order for our digital abstraction to work, we have to define the valid range of logic levels that we will interpret as a zero or one.
- Consider one gate driving another gate. For the driver gate, we call the V_{OH} the smallest valid "high" output. Likewise, we call V_{OL} the largest valid "low" output.
- For the receiver gate, we call the smallest valid "high" input level V_{IH} and the largest valid "low" input level V_{IL}. Any input outside this range are in the forbidden zone and the output is indeterminate if the inputs fall in this range.

Noise Margin

- Notice that if $V_{OL} < V_{IL}$ and $V_{OH} > V_{IH}$ we have some *noise margin* in our circuit. In other words, the output of one gate can vary by as much as $NM_H = V_{OH} - V_{IH}$ and the data will still be valid on the high side. Likewise, the low side noise margin is $NM_L = V_{IL} - V_{OL}$.
- This is an important concept because it means that digital logic gates "clean up" their inputs and allow cascading of gates to occur without ever worrying about the NM decreasing. In fact, quite the opposite happens.

DC Transfer Characteristic

- The transfer characteristics for an ideal logic inverter and a real inverter are shown below. Note that for an ideal inverter there's a threshold voltage at the midpoint. If the signal is below the midpoint, the output saturates to the supply. If it's above, it saturates to the ground potential.
- A real curve as a transition region in which the output voltage is not well defined (high or low). This is the forbidden region for the gate.
- The V_{OH} and V_{OL} are defined by the points where dV_o/dV_i is equal to -1. This is done to maximize the noise margin of the circuit.

CMOS Transistors

- The vast majority of logic gates today are built from silicon CMOS transistors. CMOS transistors have been the driving force of electronic miniaturization and power scaling for the past few decades.
- CMOS transistors are used to make logic gates, memory cells, and buffers and amplifiers.
- The level of integration has been growing exponentially for the past four decades and today you build a chip with 1 billion transistors!

Fabrication

- CMOS stands for Complementary MOS, and MOS stands for Metal-Oxide-Silicon. A MOS sandwich is the basic structure which forms a capacitor between the gate and the body of the transistor. The structure is fabricated by lithographically defining an opening in the silicon where the native SiO_2 oxide is grown. A p-type silicon is usually used for the body.
- To complete the transistor, two new terminals are added, the source and drain. The source and drain are grown by doping openings in the silicon adjacent to the gate. Under normal operating conditions, the body biased at the most negative potential (ground), and that means that the source/drain junctions are reverse biased and isolated from the body and from each other.

nMOS Operation

- Ignoring the gate terminal for now, since the source/drain form two back to back diodes, no current can flow through the device.
- If a positive voltage is applied at the gate terminal, it begins to attract negative charge carriers to the surface. It's a capacitor after all and in equilibrium we would expect the positive charge on the gate to be balanced by negative charge at the surface of silicon. Thus we can say that a "channel" begins to form due to the field-effect of the gate terminal.
- If the voltage is made sufficiently large, the channel becomes conductive enough that current can flow from the source to the drain. Thus the gate voltage modulates the conductivity of the channel.
- The device is called a FET, or field-effect transistor.

pMOS Operation

- The pMOS device has exactly the opposite or complementary behavior. It's grown in an n-type body and the source and drain are p-type. To get current to flow through the device, therefore, requires positive charges to accumulate at the surface. To do this a negative voltage is applied to the gate terminal.
- The voltage for which conduction begins is called the threshold voltage, V_{TH}.

Switch Model

- The easiest way to describe a FET transistor is to model it as a switch. In the nMOS device, when a voltage $V_G > V_{TH}$ is applied to the gate terminal, current can flow through the device. It does have "on-resistance" but we can usually ignore that. The gate terminal is insulated from the transistor (it's like a capacitor) and so no DC current flows into the gate.
- When the gate voltage is too low to form a channel, $V_G < V_{TH}$, virtually no current flows and we model the device as an open circuit.

Limit to Switch Operation

- In the nMOS device, if the drain voltage is made too positive, then the field between the gate and the channel near the drain gets too low and the channel begins to disappear near the drain end. Under these conditions the switch model breaks down.
- To keep this from happening, we like to keep the drain voltage low

$$V_G - V_D > V_{TH}$$

$$V_D < V_G - V_{TH}$$

- Exactly the opposite is true for the pMOS. Therefore to operate the devices as good switches, we should employ nMOS devices when the voltage on the drain is low and pMOS devices when the voltage at the drain is high.

CMOS Not Gate – Inverter

- The CMOS inverter is one of the most important circuit building blocks. The gates and drains of two transistor are connected together. The nMOS is on the bottom and the pMOS is on top. The sources are connected to ground and the supply voltage.
- When a low input is applied, the nMOS is off (open circuit) but the pMOS is on and shorts the output node (drain node) to the supply. The output is therefore high, or inverse of the input.
- When a high input is applied, the pMOS is off (open circuit) but the nMOS is on and shorts the output node (drain node) to ground. The output is therefore low, or inverse of the input.

CMOS NAND Gate

- Consider the circuit below. There are two nMOS transistors in series at the bottom and two parallel pMOS transistors on top.
- When both inputs are high, the nMOS transistors both turn on and short the output to ground.
- When only one input is high, though, there is no conducting path to ground. But then one of the pMOS transistors turns on and shorts the output to the supply. The same is true when both inputs are low.

Series and Parallel Connections

- We can generalize the results of the previous examine and note that whenever we put transistors in series, they function as an AND. When in parallel, they function as an OR.

Pull Up and Pull Down Networks

- CMOS logic gates work by having a pull-up and pull-down network. These networks should always complement each other, otherwise a contention would occur and the output would go into an undefined state. When the output is low, the pull-down network should turn on (and pull-up should turn off) and provide a conducting path to ground.
- When the output is high, the pull-up network should turn on and the pull-down network should turn off and a conducting path should be present to the supply.

CMOS Power Consumption

- We can estimate the power consumption of CMOS by noting that under ideal conditions no DC power consumption occurs. That's because of the complementary nature of the pull-up and pull-down networks.
- The output node of a CMOS logic gate has capacitance, though, which is from the gates of all the devices loading the output (the load capacitance) as well as the internal capacitance of the output itself (the drain nodes).

CMOS Power Consumption

- We know that to charge the capacitance to a voltage V_{DD} requires charge $q = C_L V_{DD}$ and an energy $E = qV_{DD} = C_L V_{DD}^2$ is drawn from the supply.
- Since this capacitance is charged and discharged during an active cycle, the net energy drawn from supply and then return to ground. The total power consumption is

$$P = E/T = C_L V_{DD}^2 f \alpha$$

- In the above equation, α is the activity factor (switching rate) and it's a statistical quantity because it depends on how often the inputs transition.
- The above equation has been the guiding light for CMOS transistor design for the past several decades. Voltage levels have scaled from tens of volts down to 1V.
- As the devices have gotten smaller and smaller, the C_L term has scaled but the sheer number of transistors has gone up and switching speeds have gone up $f \sim 1$ GHz today. Just imagine 10 million transistors switching, each with 3fF of capacitance, at a rate of 1 GHz. That's a whopping 30W of power!
- Today instead of making transistors run faster, it's better to use parallelism to run multiple cores at lower speeds but yet try to get the same amount of work done.