

# *Notes for ECE 29595PD - Principles of Digital System Design*

*Shubham Saluja Kumar Agarwal*

*January 19, 2024*

These are lecture notes for spring 2024 ECE 29595PD at Purdue. Modify, use, and distribute as you please.

## *Contents*

<i>Course Introduction</i>	1
<i>Introduction</i>	2
<i>Digital Logic Signals</i>	2
<i>Logic Circuits</i>	3
<i>Gates</i>	4
<i>Timing</i>	4
<i>Software Aspects of Digital Design</i>	5
<i>Integrated Circuits</i>	6
<i>Logic Families</i>	6
<i>CMOS Logic Circuits</i>	6
<i>Preview of future topics</i>	8
<i>Number Systems and Codes</i>	9
<i>Adding and Subtracting</i>	10
<i>Using the complement to conduct operations</i>	11

## *Course Introduction*

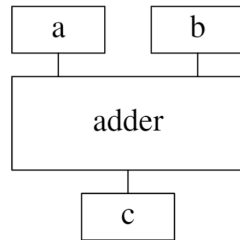
This course serves as an introduction to digital system design, with an emphasis on principles of digital hardware and embedded system design. It is an alternate class to ECE 27000.

Learning Outcomes:

1. Ability to analyze and design combinational logic circuits.
2. Ability to analyze and design sequential logic circuits.
3. Ability to analyze and design computer logic circuits.
4. Ability to realize, test, and debug practical digital circuits.

## Introduction

Digital design entails creating hardware that can conduct an operation or set of operations within a computer system. For example, adding two numbers.



This hardware can add two numbers, that is, conduct the operation  $c = a + b$ . It could also perform  $f = d + e$  or  $i = g - h = g + (-h)$  as it is not restricted to the sole values of  $a$  and  $b$  as inputs. This process fits into the logic design and switching algebra portions of chip manufacturing.

The creation of systems like these is based on the fact that voltage and current are time-varying and can assume any value in a continuous range of real numbers, but are mapped to only two values.

## Digital Logic Signals

A digital signal is modeled assuming that at anytime, it can have only one of two discrete values, which represent:

0	1
LOW	HIGH
FALSE	TRUE

This is called positive logic. It maps the infinite values of voltage and current to these two values. An example of this is CMOS 2-Volt logic:

0	1
0-0.5V	1.5-2.0V

These completely separated ranges of values allow for 0 and 1 to be completely separate, with noise and other possible errors being ignored. In this way, all physical values can be partitioned into the two values, though, for intents and purposes of digital system design, voltage is the most relevant.

Additionally, circuits known as buffer circuits can be used to restore logic values. That is, if a voltage is not sufficiently close to the values

of 0 or 2 (in the case of the CMOS), they can push these values far closer to the desired values, reducing the possibility of error.

Digital circuits have replaced analog circuits because they are far easier to design. They can be made using Hardware Description Languages (HDLs), softwares that are similar to programming languages. A logic value is called a binary digit, or bit. A set of  $n$  bits, represent  $2^n$  values. For example:

	$b_0$	$b_1$
$b_0$	0	0
0	1	0
1	0	1
	1	1

### Logic Circuits

At the highest level, a logic circuit is a *black box* with  $n$  inputs and  $m$  outputs. Only zeros and ones are required to represent inputs and outputs.

Combinational circuits are circuits that have outputs that solely depend on the inputs. It can be represented by a truth table.

An adder can be represented by a truth table in this manner.

$x_1$	$x_2$	$x_3$	$z_2$	$z_1$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

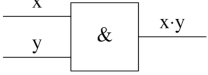
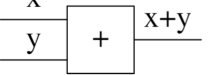
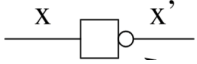
On the other hand, a sequential circuit has memory. That is, the output is dependent on both the inputs and the current state of the circuit itself. This is representable through a state table.

input	present state	next state	output
0	00	00	0
1	00	01	0
0	01	01	0
1	01	10	0
0	10	10	0
1	10	00	1

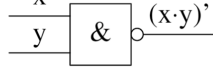
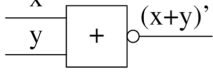
## Gates

Basic digital devices are called gates. These implement basic logic functions. These are AND, OR, NOT. However, AND, NOT and OR, NOT are sufficient to make any combination, as the third gate can be formed as a combination of the other two.

Each of the gates is represented symbolically like the following, and have their truth tables shown below:

AND			OR			NOT	
							
x	y	$x \wedge y$	x	y	$x \vee y$	x	$\neg x$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

This can be done for more inputs, by combining multiple gates. There are two more often used gates that can be created through a combination of these two.

NAND			NOR		
					
x	y	$\neg(x \wedge y)$	x	y	$\neg(x \vee y)$
0	0	1	0	0	1
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0

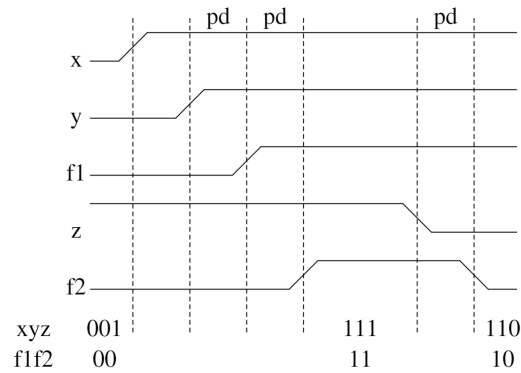
It is a good exercise to see how two of the basic gates (AND, NOT or OR, NOT) can be used to create the rest, as it is possible to connect gates to form complex circuits and thus, complex functions.

*Note: By convention, signal flows from left to right. Thus, output comes out the right and input goes in at left.*

## Timing

When the values of an input changes, it takes time for the outputs to change as well, which is known as *propagation delay*, which can be represented in a timing diagram. However, for most cases, it is possible to ignore these, as the behavior is well defined regardless of delay.

The following is a timing diagram:



Propagation delay, represented by *pd* in the timing diagram, is irrelevant if there are no changes, regardless of the fact that there is a new set of inputs. The values at the bottom of the diagram are the values that each of the elements hold after all the necessary transitions have happened for that set of inputs.

Transition time is the time it takes for a signal to change from 0 to 1, and is represented by "*rt*".

### *Software Aspects of Digital Design*

Software is not technically essential to digital system design. People used to draw symbol by hand, and could make technically equivalent systems. However, the availability, utility, and ease of use of HDLs has made the use of software prevalent in the current technological landscape.

Electronic Design Automation tools are useful in improving designer productivity. The following are examples of these:

- Schematic entry: Allows for fully detailed digital diagrams to be created digitally.
- HDLs: Can be used to design anything from individual function modules to multichip digital systems. This course will involve extensive use of VHDL.
- HDL text editors, compilers, synthesizers: text editor to define, compiler to debug syntax, synthesizer to create corresponding circuit or chip.
- Simulator: It is virtually impossible to debug a synthesized chip, so simulators are used before synthesis. They allow for verification of functionality prior to the actual tedious process of synthesis. Among simulators, there are also PCB simulators, and FPGAs, which are like programmable chips. They are very important.

- Test benches: Designs are simulated and tested using these. They run a series of checks to ensure that nothing stopped working due to changes.
- Timing analyzers and verifiers: Correct timing of inputs and reactions are paramount in digital systems. This facet of simulation allows for the automation of timing functionality verification.
- Word Processors: Allow for pretty documentation to be created.

Software is important, but the understanding of what it actually does is even more so.

### *Integrated Circuits*

A collection of gates on a single silicon chip is called an integrated circuit or IC.

An IC originates from a wafer, which is segmented into many identical ICs. The wafer is divided into rectangles, called *dies* which in turn have *pads* which allow for wire connections. Microscopic probes are used to debug the wafers, and defects are discarded, and the successes, cut out.

For this class, an IC, is a packaged die.

ICs were divided by size, small SSIs with 1 to 20 gates, medium MSIs with 20 to 200, large LSIs with upto 1000, and very large VLSIs from everything above that. The largest VLSIs now have over 10 billion transistors.

An IC process is the steps taken to create an IC, and are categorized by the transistor density within the chip.

### *Logic Families*

There are many different ways of implementing logic circuits. Connectable logic circuits are of the same logic family. So, the technology controlling them is different. Chips need to be from the same family to be connected to each other. If they are from different families, they cannot be connected to each other.

For example TTL (Transistor-Transistor Logic) is a logic family based on bipolar junction transistors.

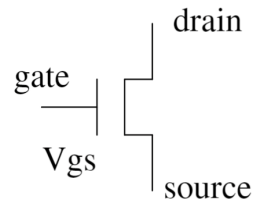
CMOS (Complementary Metal-Oxide Semiconductor) is based on MOSFETs (MOS Field-Effect Transistors). These are more commonly used, and the ones we will be analyzing in this class.

### *CMOS Logic Circuits*

A MOS transistor is modelable as a voltage controlled resistor. This means that the input voltage controls the resistance of the MOSFET,

allowing it to act as a kind of switch. This is because it has only two effective states: very high resistance and very low resistance.

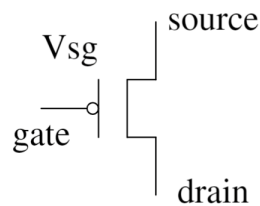
This is an n-channel (NMOS) transistor:



When  $V_{gs} = 0$ , the resistance is very high.

When  $V_{gs}$  is increased, the resistance is very low.

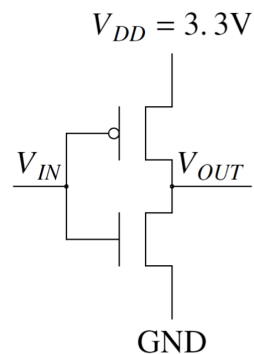
On the other hand, this is the PMOS transistor:



When  $V_{sg} = 0$ , the resistance is very high.

When  $V_{sg}$  is increased, the resistance is very low.

NMOS and PMOS together allow us to form CMOS logic. For example, this is a CMOS inverter, in which the output will be the opposite of the input.

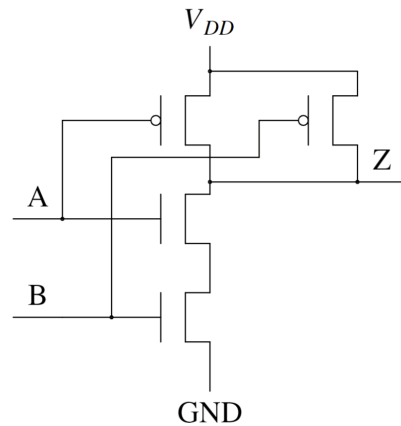


If  $V_{in} = 3.3V$ , or high, the PMOS transistor be off, and the NMOS would be on. This would mean that the output and ground are shorted, causing the output to be low.

If  $V_{in} = 0V$ , or low, the PMOS transistor be on, and the NMOS would be off. This would mean that the output and  $V_{DD}$  are shorted, causing the output to be high.

This allows this to be act as an inverter.

Now we will observe the CMOS arrangement of a NAND gate.



If A and B are both HIGH, both PMOS are HIGH, and thus  $V_{DD}$  is shorted to Z.

If they are both LOW, the exact opposite happens, with Z shorting to ground.

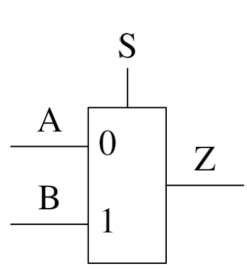
If A or B are HIGH, than one of the PMOS will be high, and one of the NMOS will be LOW, causing the output to be HIGH.

The NOR gate can be constructed easily from the same thought process, but NMOS in parallel, and PMOS in series.

These can be expanded to more inputs by adding more transistors.

### *Preview of future topics*

The following is a multiplexer:



It outputs A when S is 0, and B when S is 1. It allows us to select between different functions, such as adding and subtracting.

Truth tables are another form of representation of a digital system. We will later learn how to analyze and derive equations from truth tables, as well as the gate implementation.

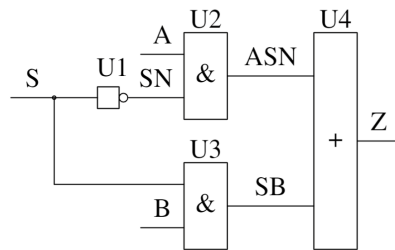
*Note: CMOS logic doesn't have AND, OR gates, so the actual implementation of these two is NAND, NOR gates with an inverter.*



Gates require 4 transistors, and inverters need two transistors. Verilog Structural model:

```
module Ch1mux_s(A,B,S,Z);
    input A,B,S;
    output Z;
    wire SN,ASN,SB;
    not U1(SN,S);
    and U2(ASN,A,SN);
    and U3(SB,B,S);
    or U4(Z,ASN,SB);
endmodule
```

Which is equivalent to:



*Note: the order of these statements is irrelevant, they will result in the same digital system.* Verilog Behavioral Model:

```
module Ch1mux_b(A,B,S,Z);
    input A,B,S; \\input declaration
    output reg Z; \\output declaration
    always @(A,B,S) \\if input changes, recompute
    if (S==1) Z=B;
    else Z=A; \\this is saying what the multiplexer above did
endmodule
```

## Number Systems and Codes

A digital circuit processes binary digits in the form of bits.

It is important to see how these relate to real life values.

Numbers in the decimal system:

$$abcd = 1000a + 100b + 10c + d = 10^3a + 10^2b + 10^1c + 10^0d$$

The base, or radix is 10. For a general radix  $r$ , it would be:

$$abcd = r^3a + r^2b + r^1c + r^0d$$

For this course, we care about binary, or radix 2.

base 2	base 8	base 16
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	10	8
1001	11	9
1010	12	A
1011	13	B
1100	14	C
1101	15	D
1110	16	E
1111	17	F

Occasionally, base 8 (octal) and base 16 (hexadecimal) are also relevant. To convert from octal to binary, replace every octal with three bits starting from the least significant bit. For hexadecimal, replace each digit with 4 bits.

The least significant bit (LSB) is the last bit, and the first is the most significant bit (MSB).

To convert from decimal to binary by dividing the number by 2, and assigning the remainder to the least significant unassigned bit. Do this continuously to the quotient until done.

### *Adding and Subtracting*

To add and subtract in binary:

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

To add signs to numbers, it has been standardized that the first bit defines the sign. If the leftmost bit is 0, the number is positive. and if it is 1, the number is negative.

This allows an  $n$ -bit signed integer to be from the range  $-2^{n-1} + 1$  to  $2^{n-1} - 1$ .

To actually add and subtract numbers, we first check if the sign is the same, if it is, we add directly, and append the sign bit. If they have opposing signs, we find the larger number, subtract the smaller, and use the sign of the larger.

*Using the complement to conduct operations*

In the complement number system, we can add or subtract directly, the operations can be done directly. The complementation process is more complicated than sign checking, but it simplifies addition.

The 2-complement system is the difference between  $2^n$  and the  $n$ -bit integer. To compute  $-B$ , we first need to calculate  $2^n - B$ , which we can compute as  $(2^n - 1) - B + 1$ , which is essentially inverting each bit, and adding 1 to the result. (Carries outside the bit length are ignored.)

The most significant bit of the complement acts as a sign bit. This can be seen in the following table: The magnitude of a number can be

number	binary	complement
0	0000	-0
1	0001	-1
2	0010	-2
3	0011	-3
4	0100	-4
5	0101	-5
6	0110	-6
7	0111	-7
-	-	-8

computed as for an unsigned number, except that the weight of the MSB is  $-2^{n-1}$  instead of  $2^{n-1}$ . So, to subtract  $2^n$ , replace the weight of the MSB with  $-2^{n-1}$ .

To add a bit to a complement number, we can simply duplicate the MSB at the beginning of the MSB. As proof of this actually working, we can calculate the complement of both the complement and the modified complement as uncomplemented numbers, and we can observe that they will be the same.

That is, the complement of 11001 and 111001 is the same number, the same happens for 011 and 0011.

We can also reverse this process if the digits corresponding to the removed bits of the given complement are all the same.

Now we will analyze how the complement can be used to add and subtract.

+6	0110	+
-3	1101	
<hr/>		
3	1	0011

This process may not work if overflow occurs. That is, if the number of necessary bits to represent the result of an operation are more than the available bits. To subtract two numbers, we instead use the

following property:  $X - Y = X + (-Y) = X + Y' + 1$  where  $Y'$  is the complement of  $Y$ .