Lecture 1: Digital Systems and Number Systems

Matthew Shuman

Contents

1 The Digital Abstraction — 1.3 in Text

1.1 Analog Systems

Analog systems are *continuous*. Look at the analog clock in figure 1. The second hand on the clock rotates continuously around on the clock. Notice how the fraction of a second can be estimated by the distance of the hand between the second divisions. The fractional value is an indicator that the system is analog. What is the limiting factor of how precise an analog system can be read?

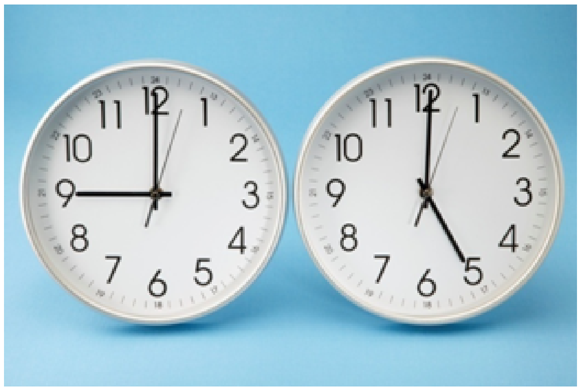


Figure 1: This is a typical analog clock.

1.2 Digital Systems

Digital systems are *discrete*. Look at the digital clock in figure 2. The second digit moves instantly from a 1 to a 2. There is no partial value for seconds and no way to zoom in to gain more precision. Precision could be increased by adding fractional digits that measure smaller amounts of time (tenths or hundredths of seconds).

1.3 Examples

Are for following items digital or analog?

1. The clock in the classroom?
2. The number of people in a room?
3. The voltage of a AA battery?
4. The DMM (Digital Multi Meter) measured voltage of a AA battery?
5. The time it takes to read this question?
6. The measured time it takes to read the previous question?

Figure 2: This is a typical digital clock.

These examples should show that the physical properties (distance, time, voltage, and many others) are analog in nature, but after being measured they become digital information. The precision of this digital information depends directly on the quality of the measurement device.

2 Number Systems — 1.4 in Text

2.1 Decimal Numbers — 1.4.1 in Text

The standard numbers used in the US are base ten, this is the decimal number system. This system uses ten different symbols to represent numbers. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Each digit in a decimal number has a specific value. The number 1982 is represented by 1 thousand, 9 hundreds, 8 tens, and 2 ones.

2.2 Binary Numbers — 1.4.2 in Text

The number system typically used in digital logic will be base 2, this is the binary number system. This system uses two different symbols to represent numbers, 0 and 1.

The table below shows how the binary system progresses to more than three digits to represent even one decimal digit.

Number System	*Decimal*	*Binary*
Zero	0	0
One	1	1
Two	2	10
Three	3	11
Four	4	100
Five	5	101
Six	6	110
Seven	7	111
Eight	8	1000
Nine	9	1001
Ten	10	1010

The standard notation to avoid confusion of 10 in decimal with 10 in binary is to add the base of the system being used as a subscript.

$(10)_2$ indicates two, while $(10)_{10}$ indicates ten.

If no subscript is included then the number is assumed to be decimal.

3 Octal and Hexadecimal Number Systems — 1.4.3 in Text

Other useful number systems used in digital logic are octal and hexadecimal.

Octal uses 8 symbols to represent numbers. These symbols are 0, 1, 2, 3, 4, 5, 6, and 7.

Hexadecimal uses 16 symbols to represent numbers. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Octal is not used very frequently, but hexadecimal is very useful. It can be hard to read 8 bits of binary. The 1's and 0's tend to blend together and it can cause mistakes. The main use of hexadecimal is to condense long binary strings into shorter hexadecimal values. The table below shows the progression used for octal and hexadecimal.

Number System	*Decimal*	*Binary*	*Octal*	*Hexidecimal*
Zero	0	0	0	0
One	1	1	1	1
Two	2	10	2	2
Three	3	11	3	3
Four	4	100	4	4
Five	5	101	5	5
Six	6	110	6	6
Seven	7	111	7	7
Eight	8	1000	10	8
Nine	9	1001	11	9
Ten	10	1010	12	*A*
Eleven	11	1011	13	*B*
Twelve	12	1100	14	*C*
Thirteen	13	1101	15	*D*
Fourteen	14	1110	16	*E*
Fifteen	15	1111	17	*F*
Sixteen	16	10000	20	10

The standard notation of using the subscript to indicate the base of the number still applies.

$(10)_8$ indicates eight, while $(10)_{16}$ indicates sixteen.

4 Converting Between Bases — Not in Text

Converting between bases is initially easy, but it can quickly become confusing when using uncommon bases. The main source of confusion is when the conversion requires math in different bases an example of this is below.
$(10)_7 + (6)_8 = (1101)_2$

This demonstrates the difficulty of even adding two small numbers when using different bases.

A useful method is to convert the entire problem to base 10, and after the answer has been found, convert the answer to the desired base.

4.1 Converting Any Base to Decimal

A useful word to describe the base of a number system is *radix*. The radix of a decimal number is ten, and the radix of a binary number is two.

In order to understand how to convert numbers to base ten is it helpful to understand how decimal numbers are really organized. The number 365.24 is elaborated in the equation below.

$$365.24_{10} = 3 * 100 + 6 * 10 + 5 * 1 + 2 * .1 + 4 * .01$$

This progression leads to the more general equation listed below.

$$365.24_{10} = 3 * 10^2 + 6 * 10^1 + 5 * 10^0 + 2 * 10^{-1} + 4 * 10^{-2}$$

This form shows each digit has a different value. The 3 is in the 100's place and the 6 is in the 10's place.
The following equation converts a binary number into a decimal number.

$$101.101_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = 4 + 1 + .5 + .125 = 5.625$$

The final generalized form for this equation is shown below.

$$d_2 d_1 d_0 . d_{-1_r} = d_2 * r^2 + d_1 * r^1 * d_0 * r^0 + d_{-1} * r^{-1} = DecimalValue$$

4.2 Converting Decimal to Any Base

4.2.1 Successive Quotients

The method used to convert a decimal integer number to any other radix is called *Successive Quotients*. This method uses a recursive algorithm. Consider the example of conversion to binary below.

Integer Quotient	*Remainder*
$(13)_{10}$	–
$(6)_{10}$	1
$(3)_{10}$	0
$(1)_{10}$	1
$(0)_{10}$	1
	$13_{10} \rightarrow 1101_2$

The integer column begins with the decimal number being converted. The second row is the first row divided by the radix, the quotient on the left and the remainder on the right. This pattern is continued until the quotient returns 0. The conversion is read *least significant bit, LSB* on the top. The algorithm for this process is itemized below.

1. Divide the initial decimal number by the radix.
2. Place the remainder into the LSB (Least Significant Bit) digit of the converted result.
3. Divide the current quotient by the radix.
4. Place the remainder into the next LSB digit of the converted result.
5. Repeat Step 3 and 4 until the quotient is 0.

4.2.2 Successive Products

The process of converting decimal fractions is a similar recursive algorithm process. Consider the example of converting .375 into binary.

Fractional Product	$IntegerComponent$
$(.375)_{10}$	–
$(.750)_{10}$	0
$(.5)_{10}$	1
$(0)_{10}$	1
	$.375_{10} \rightarrow .011_2$

Below is the summary of this recursive process.

1. Multiply the initial decimal fraction by the radix.
2. Place the integer into the MSB (Most Significant Bit) digit of the converted result.
3. Multiply the current fraction by the radix.
4. Place the integer into the next MSB digit of the converted result.
5. Repeat Step 3 and 4 until the fraction is 0 or the required number of bits have been converted.

It is important to note that some common decimal numbers can not be represented in binary. Look at the example converting .6 to 8 bits below.

Fractional Product	$IntegerComponent$
$(.6)_{10}$	–
$(.2)_{10}$	1
$(.4)_{10}$	0
$(.8)_{10}$	0
$(.6)_{10}$	1
$(.2)_{10}$	0
$(.4)_{10}$	0
$(.8)_{10}$	0
$(.6)_{10}$	1
$(.2)_{10}$	0
	$.6_{10} \rightarrow .100100010_2$

4.3 Quickly Converting Between Binary and Hexadecimal — 1.4.4 in Text

Binary and hexadecimal numbers are often used in digital logic or in computer programming. It would be possible to convert between these two bases by using a decimal number as an intermediate step, but there is a faster and easier method. This method uses nibbles to divide a larger binary or hexadecimal into smaller conversions. There are some examples below:

Binary	$\rightarrow$	Hexidecimal
$(1001\ 1100)_2$	$\rightarrow$	$9\ C_{16}$
$(0101\ 1010)_2$	$\rightarrow$	$5\ A_{16}$
$(1010\ 1011\ 0111)_2$	$\rightarrow$	$A\ B\ 7_{16}$

Each group of 4 binary digits is called a nibble and can be represented by one hexadecimal digit. This method only works when the bases in the conversion are a power of each other, in this instance $2^4 = 16$ so it is valid to use the nibble short cut. It also works for binary to octal, $2^3 = 8$.

5 Complements — Not really in Text

The meaning of complement is something required to make a thing complete. For example, salsa complements tortilla chips, beer complements pizza, an ice cream cone complements a hot summer day, and apple sauce complements pork chops. A key concept to explore is how two things complement each other. For example, when a piece of pizza is removed from a whole pizza the piece complements what is left behind and vice versa. Each of the 4 following complements use the same concept except in different bases and what is considered a complete number in that base.

5.1 Diminished Radix Complement

The diminished radix complements are called by the $radix - 1$. The diminished complement for a decimal number is the 9's complement and 1's complement for a binary number.

5.1.1 9's Complement

The 9's complement finds whatever is needed to make an entire set of 9's. This is shown in the example below.

$$\begin{array}{lr}
\text{Finding the 5 digit 9's complement of 1357} & \\
All\ 9's & 99999 \\
Initial\ Value & -01357 \\
 & ----\\
9's\ Complement & 98642 \\
\text{The 5 digit 9's complement of 1357 is 98642} &
\end{array}$$

5.1.2 1's Complement

The 1's complement finds whatever is needed to make an entire set of 1's. This is shown in the example below.

$$\begin{array}{lr}
\text{Finding the 8 digit 1's complement of 01101100} & \\
All\ 1's & 11111111 \\
Initial\ Value & -01101100 \\
 & --------\\
9's\ Complement & 10010011 \\
\text{The 8 digit 1's complement of 01101100 is 10010011} &
\end{array}$$

5.2 Radix Complement — 1.4.6 in Text

The radix complements are called by their *radix*. The radix complement for a decimal number is the 10's complement and 2's complement for a binary number. The value that is considered the whole part is $radix^{digit}$.

5.2.1 10's Complement

Finding the 5 digit 10's complement of 1357	
radix^5	100000
Initial Value	-01357
	$----$
$10's$ *Complement*	98643
The 5 digit 10's complement of 1357 is 98643	

5.2.2 2's Complement

Finding the 8 digit 2's complement of 01101100	
radix^8	100000000
Initial Value	-01101100
	$------$
$2's$ *Complement*	10010100
The 8 digit 2's complement of 01101100 is 10010100	

6 Subtracting by Adding — 1.4.6 in Text

A key use of complements is to do subtraction. Building an adder in hardware is fairly easy, but a subtracter is much more difficult. Using the following mathematical property subtraction can be avoided. $A - B = A + (-B)$ The following example shows how adding the radix complement can give an identical result as subtraction.

Showing how to do 72532 - 3250	Normal Way	Using Complements
Initial Value	72532	72532
Adding a 10's complement	-3250	$+96750$
	$------$	$------$
Difference	69282	169282

Note the answer has a positive carry out. This means that the difference is positive. If the carry out was 0, then the difference would be a negative number. Taking the radix complement of this negative number indicates the magnitude of the negative number.

7 Logic Gates — 1.5 in Text

This section covers the background information necessary to understand how binary values and functions are represented and some information about the analog traits of a digital signal.

7.1 Logic Gate Symbols

Figure 4 shows more logic gates. Circles on the inputs or outputs represent inverters attached to the gates. Gates can also be built with multiple inputs, up to 8.

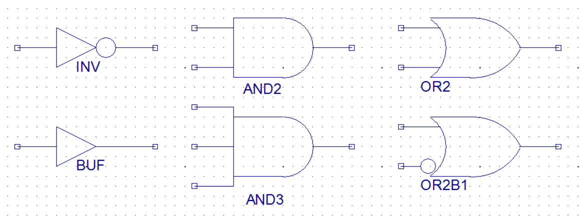


Figure 3: Here are the three fundamental logic gates, with slight variations below them.

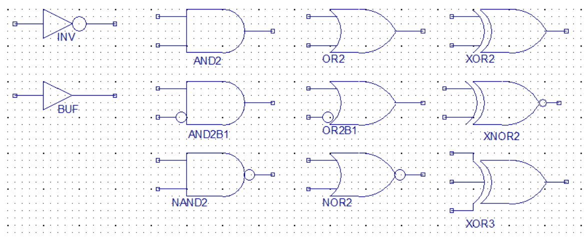


Figure 4: Examples of commonly used logic gates.

8 Beneath the Digital Abstraction — 1.6 in Text

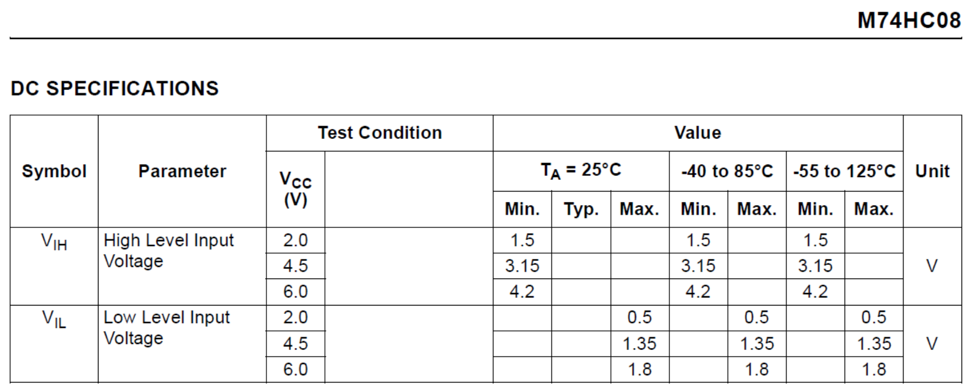
8.1 Threshold Voltages

The voltage being read into a gate is an analog signal, but the gate acts in a digital manner. This is accomplished by using thresholds to compare the incoming signal. Higher than a threshold is considered a high voltage and is assigned to a 1_2. Lower than a voltage is considered a low voltage and assigned to a 0_2. If there is a voltage between the two thresholds then it is considered a *metastable* input and the uncertain input is assigned an 'X'. The images in figure 5 are all from the 74HC08 Quad 2-Input And Gate.

9 Building Gates — 1.7 in Text

9.1 NMOS, PMOS, and CMOS

The PMOS and NMOS transistor are the two transitors that will be further explored in ECE 322, ECE 323, and other advanced courses. In ECE 271 they are simplified to be either a short or open, as shown in figure 6 and figure 7. Inputting a logic zero into a PMOS will connect (short) the source to the drain, but inputting a logic zero into the gate of an NMOS will disconnect (open) the source from the drain.



M74HC08

DC SPECIFICATIONS

Symbol	Parameter	Test Condition		Value							Unit
		V_{CC} (V)		T_A = 25°C			-40 to 85°C		-55 to 125°C		
				Min.	Typ.	Max.	Min.	Max.	Min.	Max.	
V_{IH}	High Level Input Voltage	2.0		1.5			1.5		1.5		V
		4.5		3.15			3.15		3.15		
		6.0		4.2			4.2		4.2		
V_{IL}	Low Level Input Voltage	2.0				0.5		0.5		0.5	V
		4.5				1.35		1.35		1.35	
		6.0				1.8		1.8		1.8	

Figure 5: This table shows threshold voltages for the a discrete quad AND gate IC.

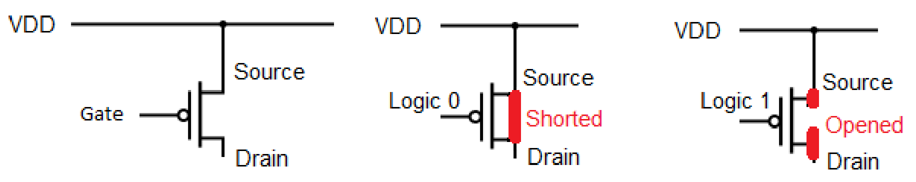


Figure 6: PMOS transistor and 2 modes of operation.

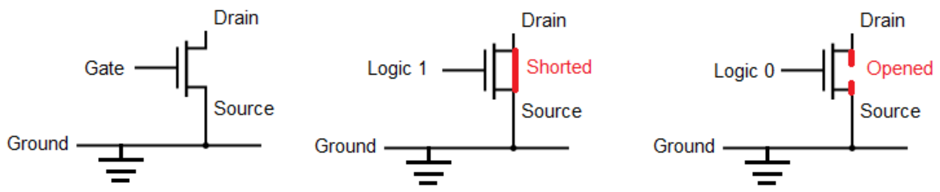


Figure 7: NMOS transistor and 2 modes of operation.

CMOS is a technique used to build logic gates from PMOS and NMOS transistors. PMOS transistors are always located between VDD and the output, while NMOS transistors are always located between the output and ground. A NOT gate is built using CMOS in figure 8 where A is the input and Z is the output.
Figure 9 and 10 show how gates can be constructed in integrated circuits.
RTL (Resistor Transistor Logic) is another method of making logic gates where the PMOS is replace with a resistor. This resistor always conducts, but can be overpowered by the NMOS gates. When the NMOS turn on the output is grounded to zero, but when the NMOS turn off the resistor pulls the output to VDD. Figure 9 shows the RTL schematic of a NOT gate that is driving a capacitor.

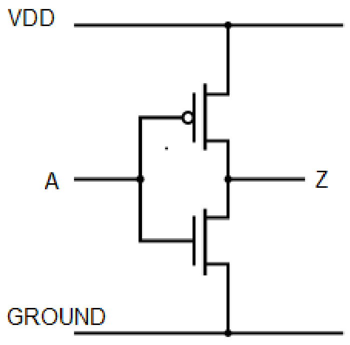


Figure 8: Building a NOT gate using CMOS.

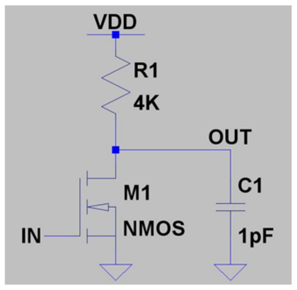


Figure 9: Example RTL schematic of a NOT gate. The 1 pf capacitor is not shown in the schematic.

Figure 10 shows how a logic gate is formed inside of an IC. The tutorial that describes the layout tool is located here:
http://web.engr.oregonstate.edu/~moon/ece423/cadence/example2.html

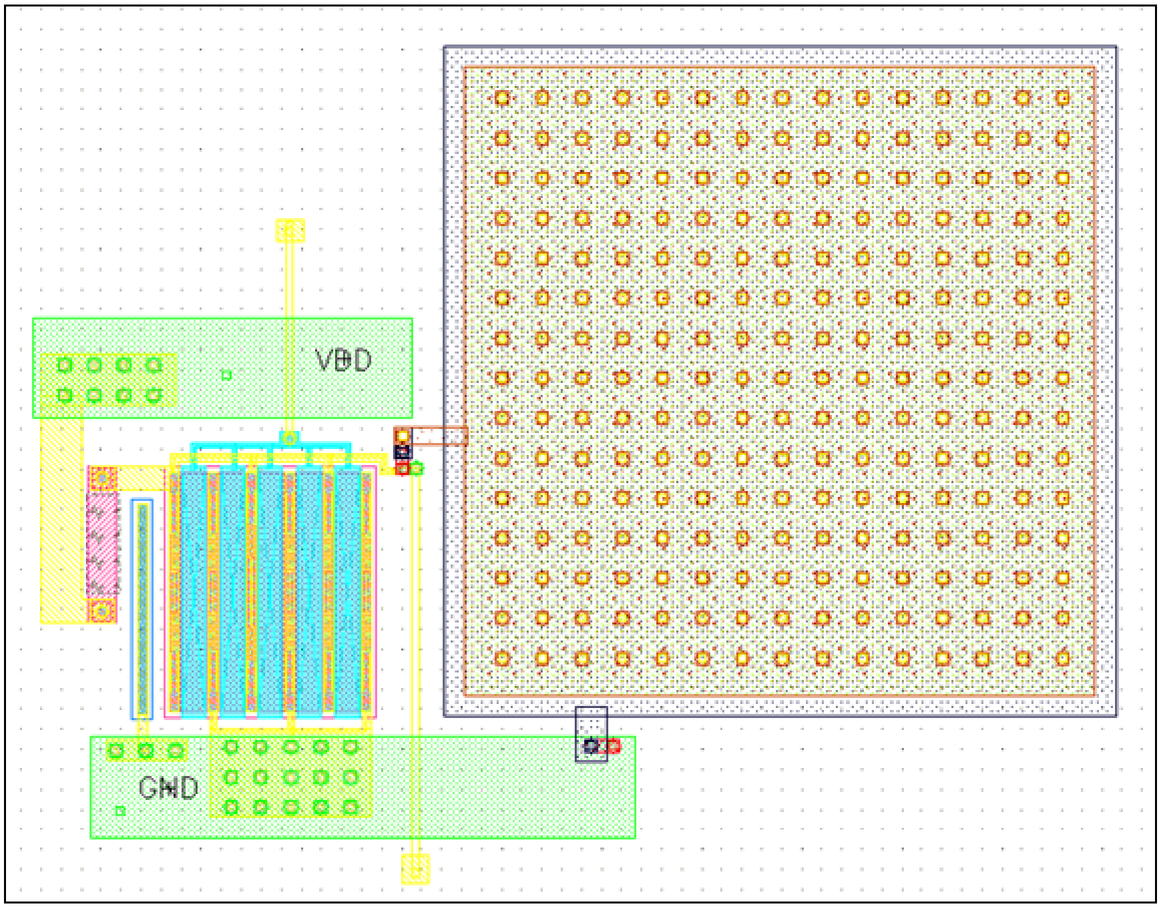


Figure 10: Example layout of a NOT gate.

10 Boolean Equations — 2.2 in Text

10.1 Order of Operations

The order of operations for standard algebra is listed below:

1. Parenthesis
2. Exponents
3. Multiplication
4. Division
5. Addition

6. Subtraction

A well know mnemonic for this list is ”Please Excuse My Dear Aunt Sally”
The order of operations for Boolean algebra is listed below:

1. Parenthesis
2. Not
3. And
4. Or

What is a mnemonic for this list?

Draw the gates for the following expressions:

1. $Z = AB'C + A' * C$
2. $Y = A + B' + C * (A' + C)$
3. $X = \overline{A\overline{B}C} + A' * C$
4. $W = \overline{A + B + C} + A' * C$

10.2 Canonical and Standard Forms

The word canonical can be defined as being *reduced to the simplest and most significant form possible without loss of generality.* The Boolean expression $Z = A * B$ is shown in the truth table below. Both of these forms are complete solutions, but they are not the simplest form.

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

A simpler form to describe Z is the canonical form. There are two canonical forms for Z. The first form, uses Σ , is called the Sum of Products. The second form, uses Π , is called the Product of Sums.

1. $Z(A, B) = \Sigma(3)$
2. $Z(A, B) = \Pi(0, 1, 2)$.

Canonical forms are useful for quickly communicating how a block of digital logic operates, but there are also standard forms for how digital logic blocks can be constructed using logic gates. Each canonical form has a standard form. $Y = CDE' + E$ is used for the following example.

1. Truth Table

C	D	E	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2. Canonical Forms

$$Y(C, D, E) = \Sigma(1, 3, 5, 6, 7)$$

$$Y(C, D, E) = \Pi(0, 2, 4)$$

3. Standard Forms

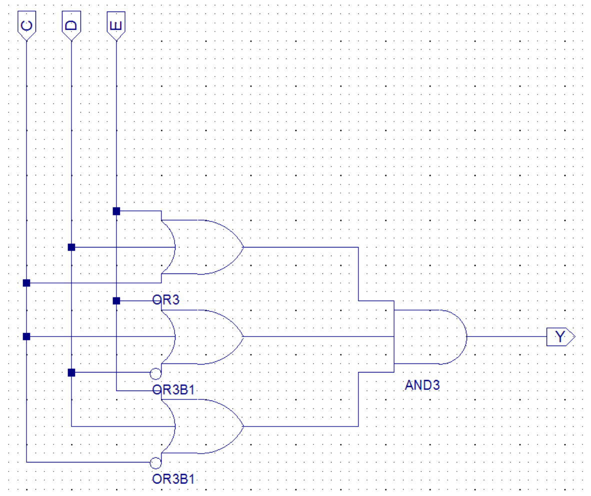


Figure 11: Product of Sums standard form for Y.

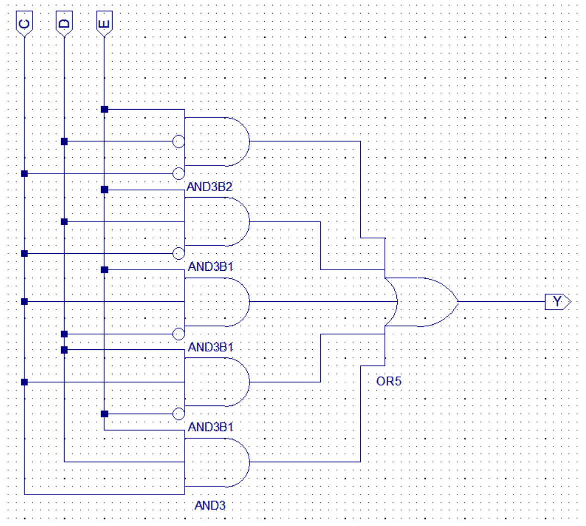


Figure 12: Sum of Products standard form for Y.

11 Boolean Algebra — 2.3 in Text

Below is a summary of the simplifications that can be done through Boolean algebra. These should be intuitive or memorized by the midterm.

OR Operator	AND Operator
$a + 0 = a$	$a * 1 = x$
$a + a' = 1$	$a * a' = 0$
$a + a = a$	$a * a = a$
$a + 1 = 1$	$a * 0 = 0$
$(a')' = a$	
$a + b = b + a$	$a * b = b * a$
$a + (b + c) = (a + b) + c$	$a * (b * c) = (a * b) * c$
$a * (b + c) = (a * b) + (a * c)$	$a + (b * c) = (a + b) * (a + c)$

12 Multilevel Combinational Logic — 2.5 in Text

12.1 Nand And Nor Implementation

Gates can be fabricated using many different technologies. A common technology is CMOS (Complementary Metal Oxide Semiconductor). This technology uses two transistors to make an inverter, and four transistors to make a two-input nand or nor gate (two transistors per input). A two input and gate is then built using a nand gate and an inverter. It makes sense to learn how to build logic gates using only nand or nor gates (as well as inverters). The sequence of figures 13, 14, and 15.

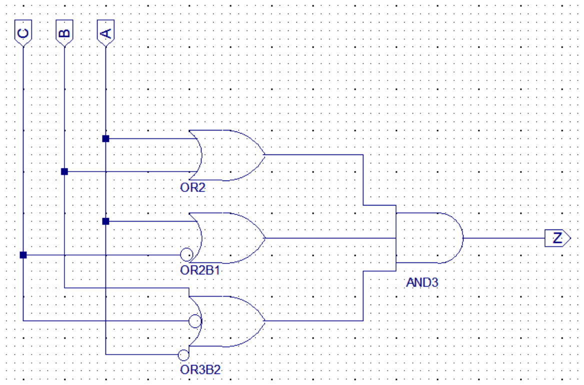


Figure 13: Traditional product of sums implementation using both or and and gates.

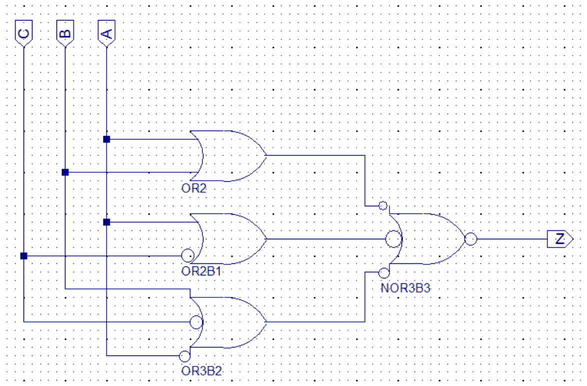


Figure 14: DeMorgan transformation of and gate into NorB3 gate.

13 Logic Minimization — 2.7 in Text

13.1 Karnaugh Maps

$Y = CDE' + E$ is used for the following example, which uses standard forms to represent the digital logic expression Y. Take special notice in how many gates are used in figure 26 and 27.

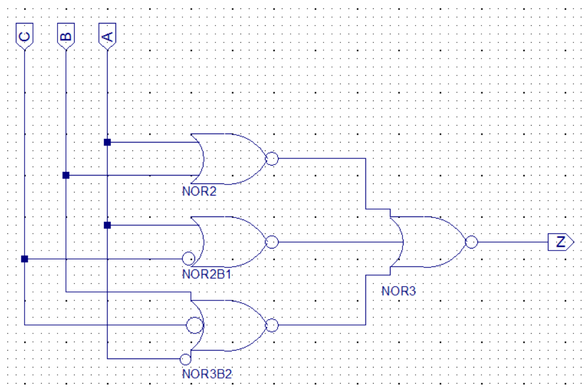


Figure 15: New logic implementation using only nor gates.

1. Truth Table

C	D	E	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2. Canonical Forms

$$Y(C, D, E) = \Pi(0, 2, 4)$$
$$Y(C, D, E) = \Sigma(1, 3, 5, 6, 7)$$

3. Standard Forms

The Boolean expression for the example only uses two gates, but figure 26 uses four gates, and figure 27 uses six gates. A rule of thumb is that every input to an *and* gate or an *or* gate makes the gate bigger and slower. Every gate in the standard form has three inputs. Standard forms are not efficient. It would be possible to use material from lecture three or four to minimize logic using Boolean algebra. Boolean algebra minimizations are tough, and there is an easier method. Karnaugh maps minimize digital logic using a simple visual table. Figure 18 shows a basic two variable Karnaugh map.
There are four main rules for making minimization selections within a Karnaugh map.

1. Selection dimensions must be a power of 2 (i.e. 1,2,4,8).
2. Selections may wrap around any edge.
3. Make the largest selections possible.

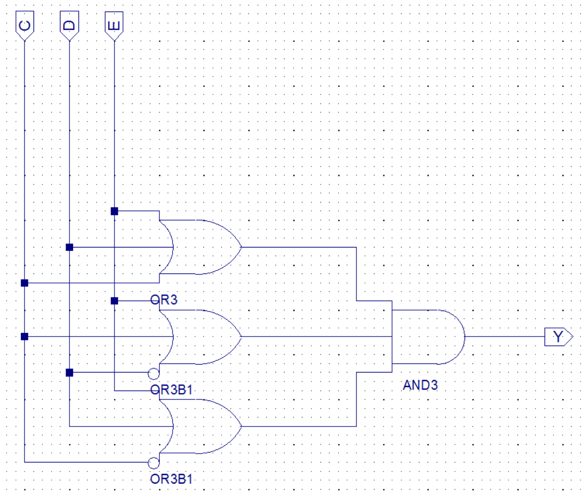


Figure 16: Product of Sums standard form for Y.

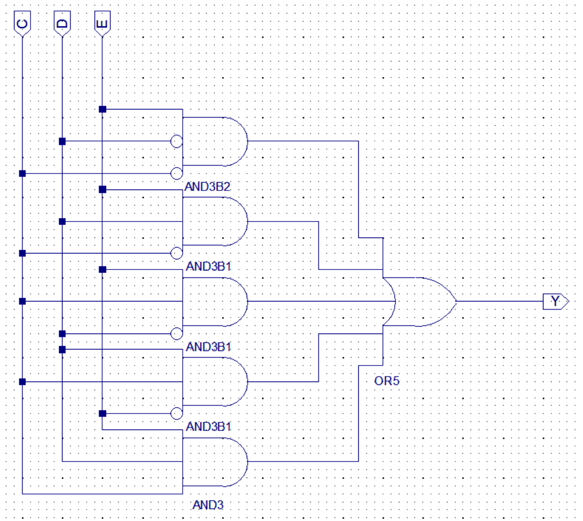


Figure 17: Sum of Products standard form for Y.

4. Use the fewest number of selections.

The selections in figure 19 visually indicate how to make an optimally minimized logic block.

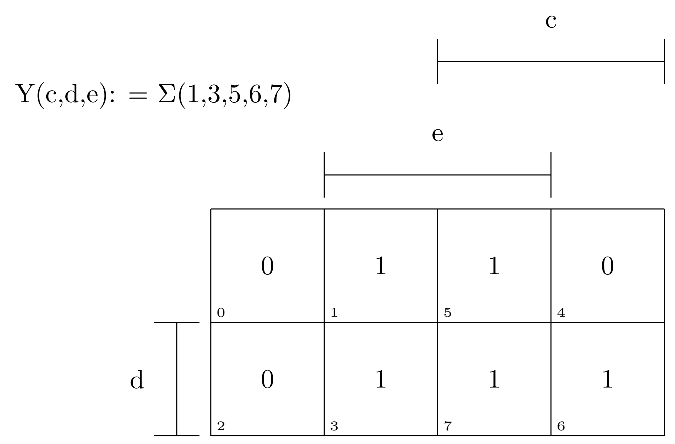


Figure 18: 3 variable Karnaugh map for Y(c,d,e).

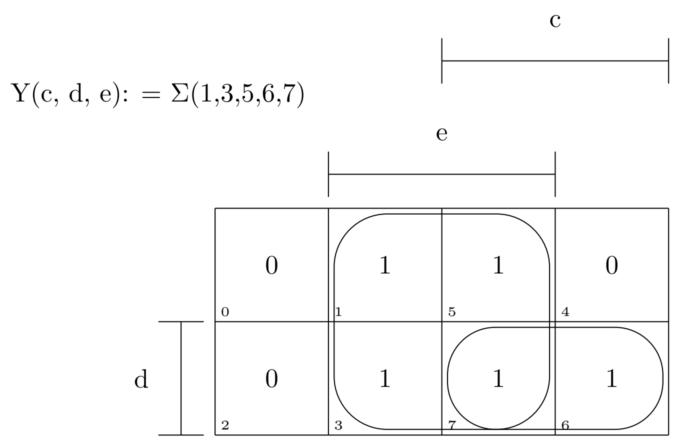


Figure 19: 3 variable Karnaugh map with simplifications circled. The minimization is that $Y = E + CD$. The term for the selection is a *minterm.*

14 Combinational Logic Blocks — 2.8 in Text

14.1 Decoders

A binary decoder takes in a binary value and uses that value to turn on a bit. The 1st output would be turned on when 0b001 is input, and the 5th output would be turned on when 0b101 is input into a three bit decoder.

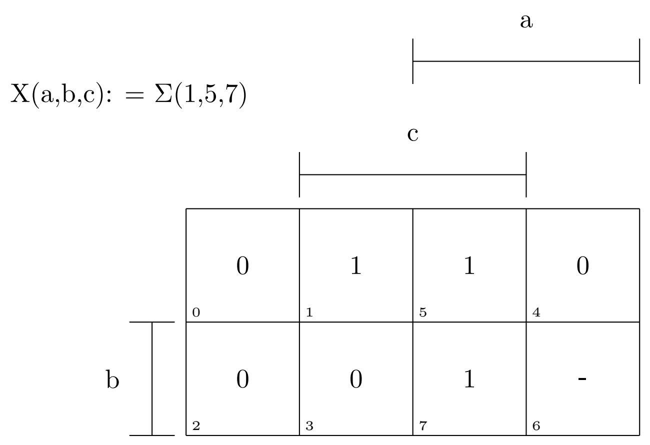


Figure 20: 3 variable Karnaugh map for X(a,b,c).

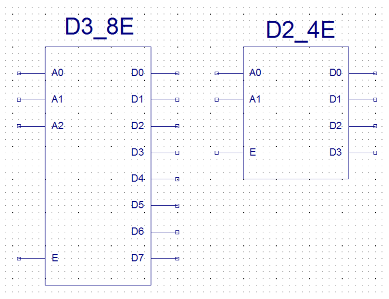


Figure 21: Symbol for two decoders.

14.2 Encoders

A binary encoder has many inputs and outputs a binary value based on which input is high. The output would be 0b001 when the 1st input is high, and the output value would be 0b101 when the 5th input is high into the three bit encoder.

14.3 Multiplexers

A multiplexer acts like a digital switch. When the select bits are 0b01 the 1st input is forwarded to the output and when the select bits are 0b11 the 3rd input is forwarded to the output.

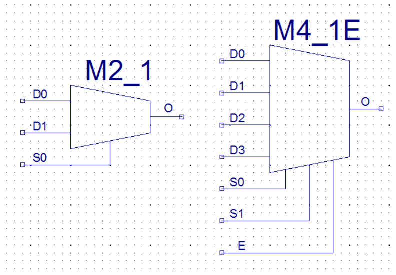


Figure 22: A basic two input multiplexer is on the left, and a more complex 4 input multiplexer with enable is on the right.

15 Logic Gate Timing — 2.9 in Text

The contamination delay, t_{cd}, is the fastest that the logic gate will change output after an input changes. The propagation delay, t_{pd}, is the slowest that the logic gate will change output after an input changes. For example, in the figure below the output of a NOT gate will change between 10 ps and 15 ps after the input changes.

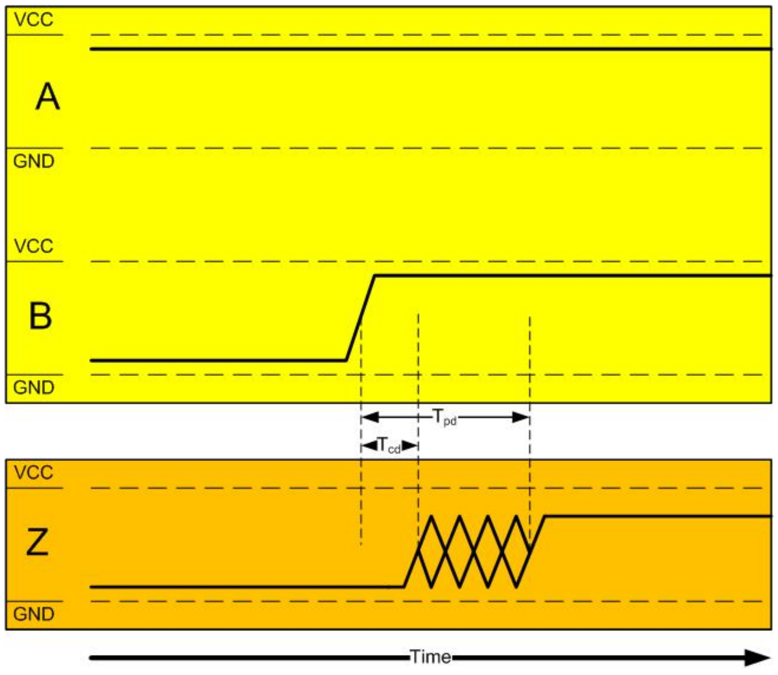
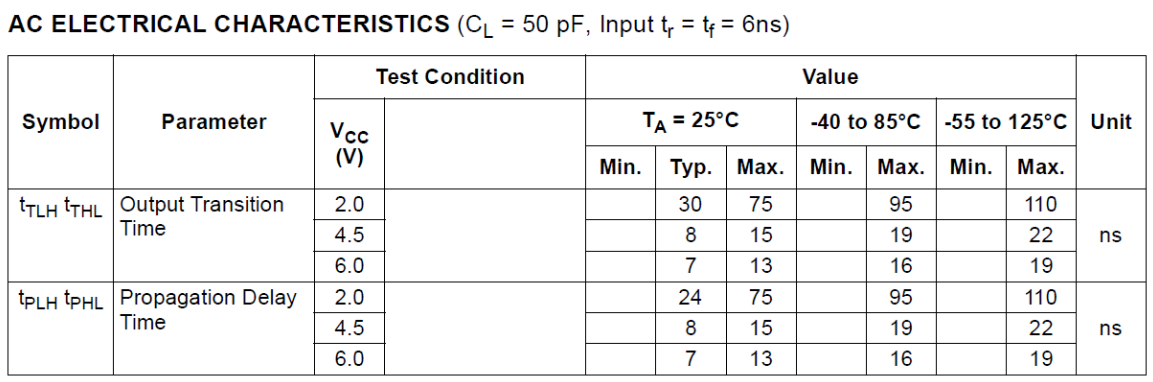


Figure 23: Timing diagram for a logic gate.

Logic Gate	$t_{pd}(picoseconds)$	$t_{cd}(picoseconds)$
NOT	15	10
2-input NAND	20	15
3-input NAND	30	25
2-input NOR	30	25
3-input NOR	45	35
2-input AND	30	25
3-input AND	40	30
2-input OR	40	30
3-input OR	55	45
2-input XOR	60	40

The images in figure 24 and 25 give examples of the propagation delays from the 74HC08 Quad 2-Input AND gate used in ECE 272. Notice that the delays depend on VCC and temperature.



AC ELECTRICAL CHARACTERISTICS (C_L = 50 pF, Input $t_r = t_f$ = 6ns)

Symbol	Parameter	Test Condition		Value							Unit
		V_{CC} (V)		T_A = 25°C			-40 to 85°C		-55 to 125°C		
				Min.	Typ.	Max.	Min.	Max.	Min.	Max.	
t_{TLH} t_{THL}	Output Transition Time	2.0			30	75		95		110	ns
		4.5			8	15		19		22	
		6.0			7	13		16		19	
t_{PLH} t_{PHL}	Propagation Delay Time	2.0			24	75		95		110	ns
		4.5			8	15		19		22	
		6.0			7	13		16		19	

Figure 24: This table shows propagation delays for a same IC in figure 5

WAVEFORM : PROPAGATION DELAY TIMES (f=1MHz; 50% duty cycle)

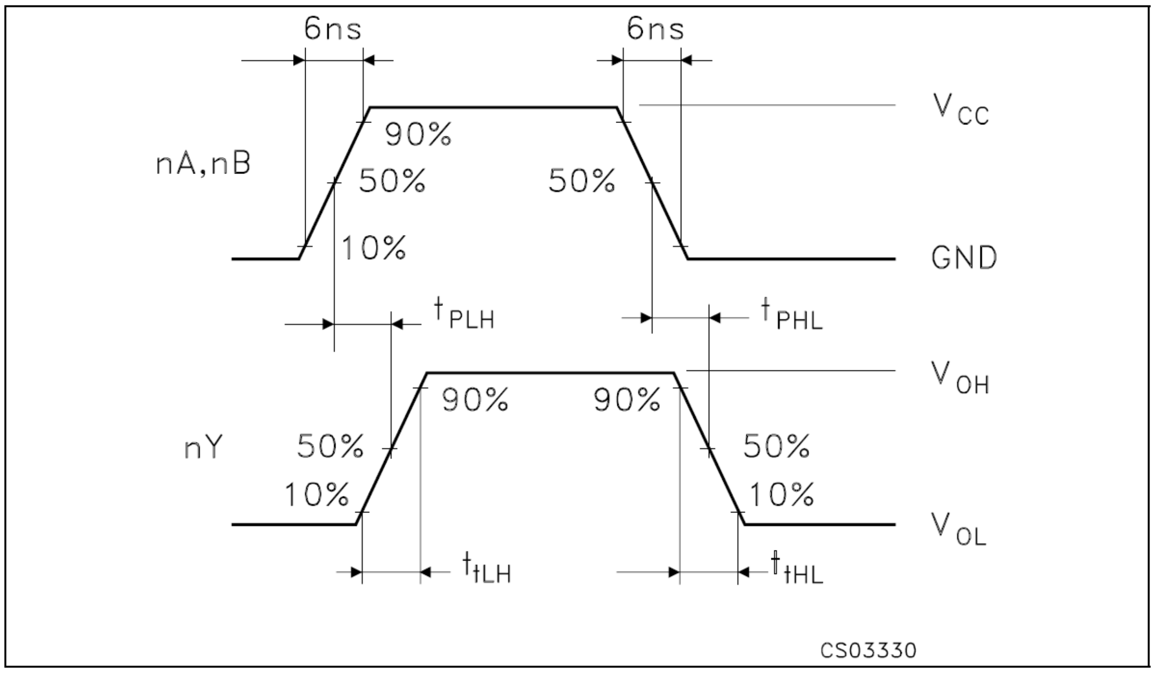


Figure 25: Chart depicting how the table in figure 24 is measured.

16 Logic Block Timing — 2.9 in Text

The *critical path* is the longest delay path through the logic block.
The *short path* is the shortest delay path through the logic block.
Use the contamination delays and the short path to find the contamination delay of the entire logic block.
Use the propagation delays and the critical path to find the propagation delay of the entire logic block.

Calculate the timing of these standard form circuits used during lecture 4.

Use logic minimization techniques to optimize the speed of both of these options.

17 Introduction — 3.1 in Text

Every circuit has so far been purely combinational in ECE 271. At any given the outputs are only determined by the current inputs. This chapter explores the design and construction of state machines. Counters, stoplights, and stepper motor drivers are all examples of these state machines.

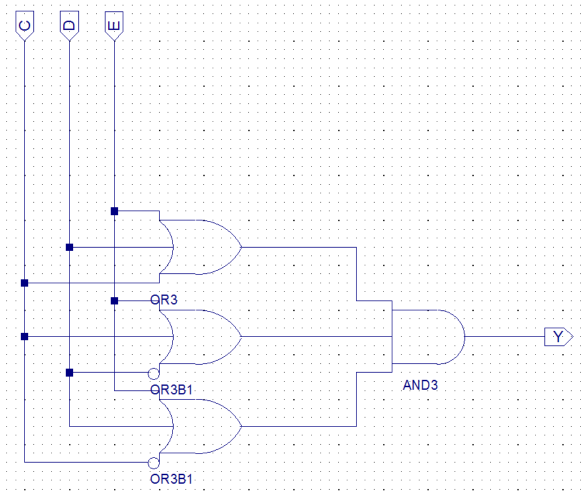


Figure 26: Product of Sums standard form for Y.

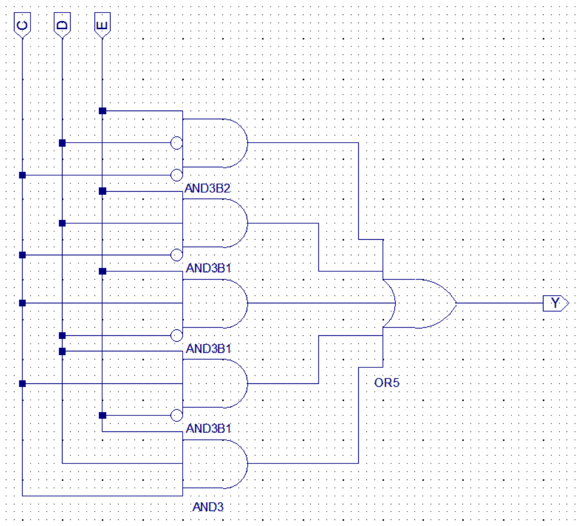


Figure 27: Sum of Products standard form for Y.

18 Sequential Circuits — 3.1 in Text

Sequential circuits depend on both the current inputs and the current state of the circuit. This means if there are 2 inputs and 3 state bits that the combinational logic has 5 inputs, a 5 input karnaugh map would be needed to minimize this logic. The outputs could turn on LEDs or control motors. There would also be 3 outputs to the memory to control what the next state would be.

This whole process is pictured in figure 28.

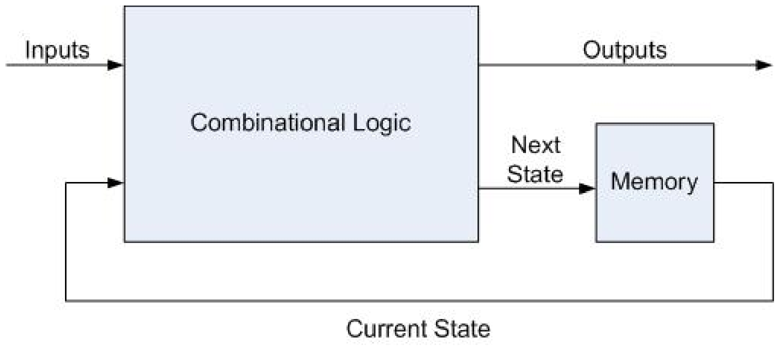


Figure 28: General structure for a sequential circuit

19 Latches — 3.2 in Text

Latches are the fundamental storage unit of memory. There are 3 basic latches below in figure 29, 30, and 31.

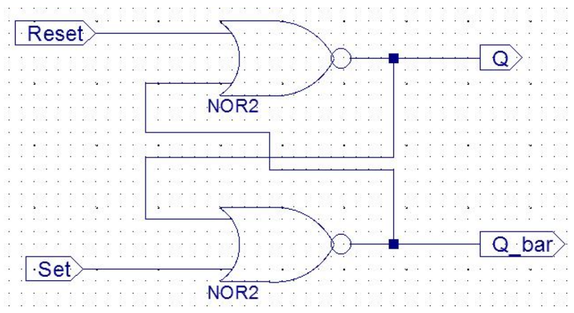


Figure 29: Active high SR latch

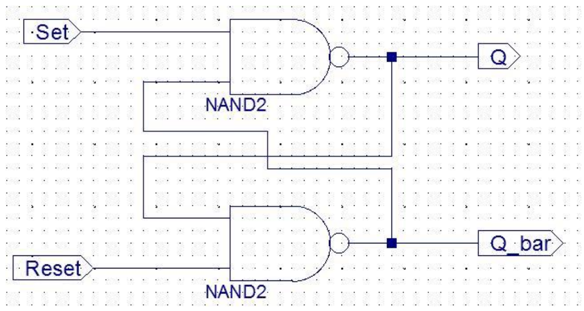


Figure 30: Active low SR latch

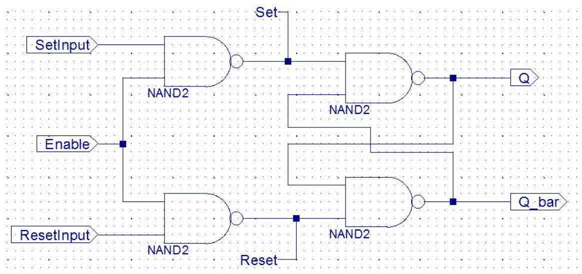


Figure 31: Active high SR latch with an enable input

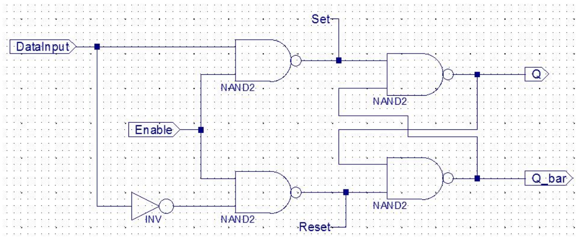
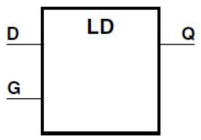


Figure 32: When Enable is high the DataInput is copied to Q in a D Latch



LD is a transparent data latch. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied.

Figure 33: Xilinx Information Sheet on the D Latch

20 Flip-Flops — 3.2 in Text

Flip-flops are the most commonly used storage unit in digital logic. They are more usable than latches, because they only change at a transition of the clock, either rising or falling edge. There are three different types of flip flops shown in figures 34, 36, 35.

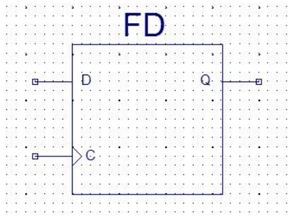


Figure 34: On the rising edge, D is copied to Q in a D Flip Flop.

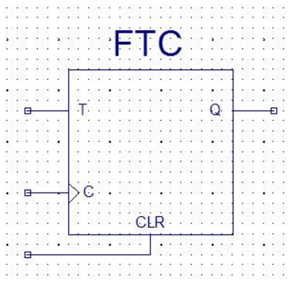


Figure 35: A T Flip Flop toggles the output whenever T is high.

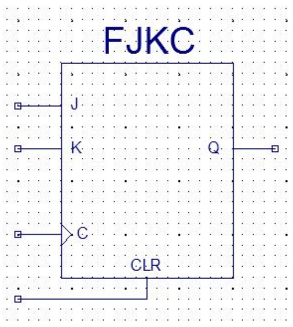


Figure 36: A JK Flip Flop can set, reset, toggle, or not toggle the outputs, based off of the JK inputs.