Physics 310
Lecture 8b – Digital Circuits

Mon. 3/15	**Ch's 11 & 12:** Digital Circuits	
Wed. 3/17	more of the same	
Thurs. 3/18	**Quiz** Ch's 11 & 12; **Lab 8:** Digital Circuits	**HW 8:** Ch11 Pr.2,8,9*; Ch12 Pr 1,5
Fri. 3/19	More of the same	**Lab 8 Notebook**

Project Proposals Due at the end of next Monday

Three levels of description:

Basic what it achieves (doorbell)

General Logic flow

Schematic

<u>Handouts</u>:

- Assignment #8, Lab #8
- TTL vs. CMOS levels
- Boolean algebra theorems
- Count-to-16 circuit
- Count-by-3 circuit design

Study List for Quiz #8:

1. Logic gates: AND, NAND, OR, NOR, XOR, XNOR, and NOT (inverter).
2. Know how to use the truth tables for logic gates.
3. Data and JK Flip-Flops

Logic Gates

- Symbols
- Truth Tables
- TTL & CMOS

Boolean Algebra

- can make anything with NAND or NOR gates!
- De Morgan's theorem

Numbering systems & codes – decimal, binary, BCD

(Open –collector & three-state logic – NO TIME!)

Flip-Flops

- RS
- Clocked RS – also the different types of clocking
- Data – count-by-two ciruit
- JK – count-to-16 circuit

BCD-to-decimal Decoder & Seven-segment Display

Count-by-3 circuit design (the decade counter is similar!)

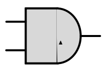
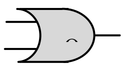
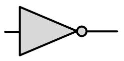
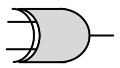
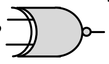
Synchronous vs. ripple counters

Physics 310
Lecture 8b – Digital Circuits

11-4 Boolean Algebra

As already mentioned, you already know all about addition, multiplication, and even integration and differentiation. However, you may not be as familiar with discrete, Boolean algebra. If that's the case, then a) you may have a limited sense of the use of logic gates and b) you may not appreciate how networks of simple gates can perform complex reasoning. So, here's a crash course in Boolean Algebra.

Common short hand for Boolean Logic operations

True = 1		
False = 0		
AND = *	$A*B$ reads "A AND B"	
OR = +	$A+B$ reads "A OR B"	
NOT = $\overline{}$	$\overline{A} = Out$ reads "NOT A"	
XOR = $\oplus$	A $\oplus$ B reads "A XOR B"	
X-NOR = $\odot$	A $\odot$ B reads "A X-NOR B"	

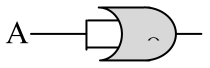
While the uses of * and + in Boolean statements bears some resemblance to their use in continuous algebra, you'd quickly get into trouble if you read them as "multiplication" and "addition." One thing that they *do* have in common with their regular-algebra counter parts is their commutative, distributive, and associative behavior. That is

A*B = B*A	A+B = B+A	commutative
A*(B+C)=A*B+A*C		distributive
A*(B*C) = (A*B)*C	A+(B+C)= (A+B)+C	associative

Boolean Theorems. Indeed, table 11.7, which shows some Boolean Theorems that would make absolutely no sense if we were talking addition and multiplication. The use of these theorems is that they allow you to find alternative ways of performing the same logical operation. Often, you want to perform a given operation in the *simplest* way possible since the logic is bound to be more transparent and you're apt to use fewer chips. Then again, sometimes you'd rather use a few more gates if it means you use less *variety* of types since logic gates often come in packs of two or four (one chip with two or four individual gates in the same package).

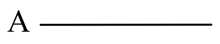
Some of the Boolean Theorems are so self-evident as to be laughable if you just recast them in English. Others are not at all obvious but can be proven by either employing a number of the more obvious ones or comparing logic tables for the two different operations.

1. A+A = A.
    a. "Do you (have an apple OR have an apple)?" You might as well just ask "Do you have an apple?" once.



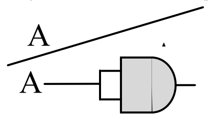
A	A	OR out put
1	1	1
0	0	0

Identical



Physics 310
Lecture 8b – Digital Circuits

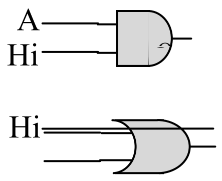
2. A*A=A.
    a. Similarly, "Do you (have an apple AND have an apple)?"; you might as well just ask "Do you have an apple?"



A	A	AND out put
1	1	1
0	0	0

Identical

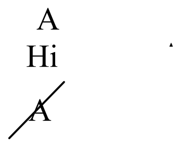
3. A+1 = 1
    a. "Are you here OR do you have an apple?"; the unilateral 'are you here' option renders moot apple-ownership condition.



A	Hi	OR out put
1	1	1
0	1	1

Identical

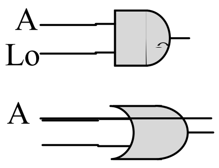
4. A*1=A
    a. "Are you here AND do you have an apple?" = "Do you have an apple?"



A	Hi	AND out put
1	1	1
0	1	0

Identical

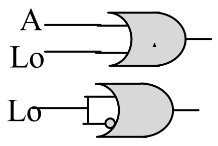
5. A+0=A
    a. "Are you absent OR do you have an apple?" = "Do you have an apple?"



A	Lo	OR out put
1	0	1
0	0	0

Identical

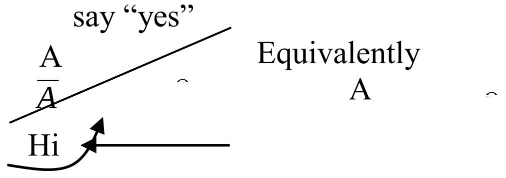
6. A*0=0
    a. "Are you absent AND do you have an apple?" = "Are you absent?"



A	Lo	AND out put
1	0	0
0	0	0

Identical

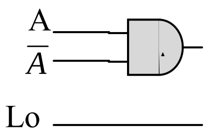
7. A+$\overline{A}$=1
    a. "Do you have an apple OR not have an apple?" there's no way someone won't say "yes"

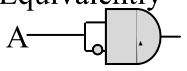


A	$\overline{A}$	OR out put
1	0	1
0	1	1

Physics 310
Lecture 8b – Digital Circuits

8. $A*\overline{A}=0$
   a. "Do you have an apple AND not have an apple?" No one could answer "yes" to that.

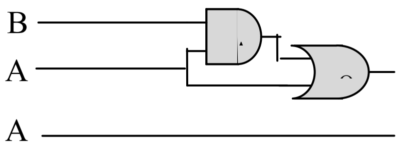


Equivalently


A	$\overline{A}$	AND out put
1	0	0
0	1	0

9. $\overline{\overline{A}}=A$
   a. "Do you not *not* have an apple?" You mean 'Do you *have* an apple.'

10. B*A+A=A
    a. "Do you (have a banana AND an apple) OR have an apple?" In other words, 'Do you have an apple?'

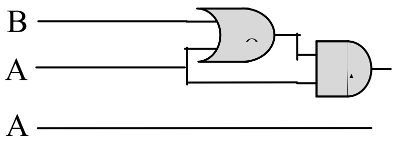


A	B	B*A	B*A + A
1	0	0	1
0	1	0	0
1	1	1	1
0	0	0	0

$$\text{A} \xleftrightarrow{\text{Identical}} \text{B*A + A}$$

11. A*(A+B) = A
    a. "Do you (have a banana or an apple) AND have an apple?" In other words, 'Do you have an apple?'

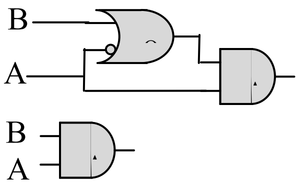
    b. Proof: A*(A+B) = A*A + A*B = A + A*B = A (last step by Th. 10)



A	B	B+A	A*(B+A)
1	0	1	1
0	1	1	0
1	1	1	1
0	0	0	0

$$\text{A} \xleftrightarrow{\text{Identical}} \text{A*(B+A)}$$

12. $(B+\overline{A})*A = B*A + \overline{A}*A = B*A + 0 = B*A$ (by appealing to simpler theorems)
    a. "Do you (have a banana OR Not have an apple) AND have an apple?" = "Do you have an apple AND a banana?"

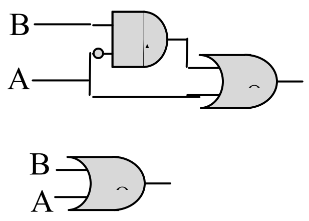


A	B	$\overline{A}$	$(B+\overline{A})$	$(B+\overline{A})*A$	B*A
1	0	0	0	0	0
0	1	1	1	0	0
1	1	0	1	1	1
0	0	1	1	0	0

$$(B+\overline{A})*A \xleftrightarrow{\text{Identical}} B*A$$

Physics 310
Lecture 8b – Digital Circuits

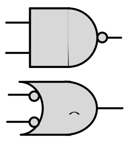
13. $A+\overline{A}*B = A + B$
   a. "Do you have an apple OR (Not have an apple AND have a banana)?" that is "Do you have an apple OR a banana?"



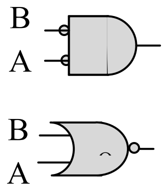
A	B	$\overline{A}$	$\overline{A}*B$	$A+\overline{A}*B$	$A+B$
1	0	0	0	1	1
0	0	1	0	0	0
1	1	0	0	1	1
0	1	1	1	1	1

Identical

The book proves Theorem 16 in this way: $\overline{A*B} = \overline{A}+\overline{B}$
   a. "Do you NOT have both an apple AND a Banana?" = "Do you NOT have an apple OR NOT have a banana?"



17: $\overline{A+B} = \overline{A}*\overline{B}$
   a. "Do you NOT have either an Apple OR a Banana?" = "Do you NOT have an apple AND NOT have a Banana?"

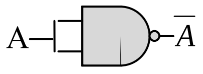


A	B	$\overline{A}$	$\overline{B}$	A+B	$\overline{A+B}$	$\overline{A}*\overline{B}$
1	0					
0	0					
1	1					
0	1					

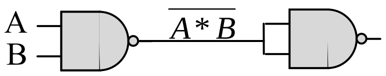
Boolean Theorems -> Circuit Equivalents
These last two theorems, are *particularly* useful for circuit design because they help us to use NAND gates to perform NOT, AND, OR, NOR and even X-OR operations. That may not seem particularly useful but gates often come packaged, four of the same together in one chip. So when it comes to actually building a circuit, it's often handy to just use another couple terminals on a chip you already need rather than making room for yet another chip on your board.

Figures 11.19 & 20 illustrate these designs

NOT

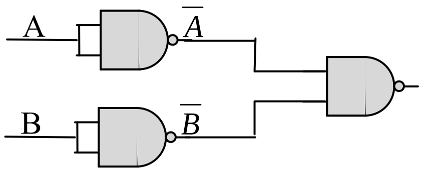


AND



$$\overline{\overline{A*B}} = A*B$$

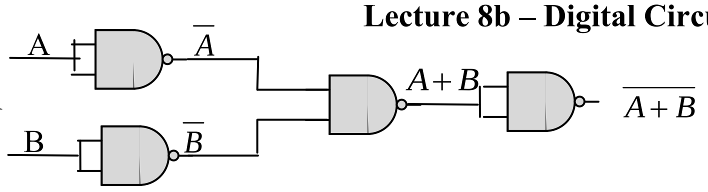
OR



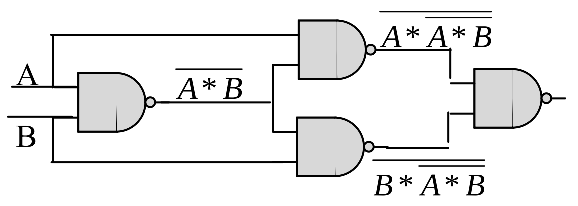
$$= \overline{\overline{A}*\overline{B}} = \overline{\overline{A}}+\overline{\overline{B}} = A+B$$

Physics 310
Lecture 8b – Digital Circuits

NOR

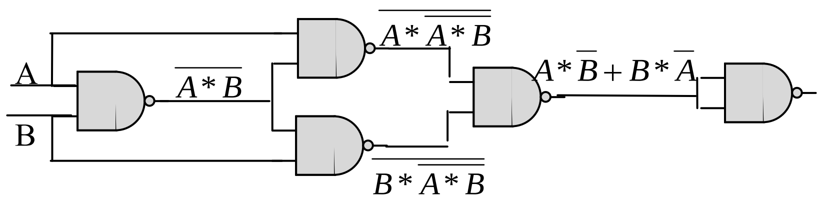


X-OR



$$\overline{\overline{B*\overline{A*B}}*\overline{A*\overline{A*B}}}=\left(\overline{\overline{B*\overline{A*B}}}\right)+\left(\overline{\overline{A*\overline{A*B}}}\right)$$
$$=(B*\overline{A*B})+(A*\overline{A*B})=(A+B)*(\overline{A*B})$$
$$=(A+B)*(\overline{A}+\overline{B})=0+A*\overline{B}+B*\overline{A}+0$$

X-NOR since it’s the inverse of X-OR (ask them to perform the logic of the NOT to get the final expression)


A
B
$\overline{A*B}$
$\overline{A*\overline{A*B}}$
$\overline{B*\overline{A*B}}$
$A*\overline{B}+B*\overline{A}$

$$\overline{A*\overline{B}+B*\overline{A}}=(\overline{A*\overline{B}})*(\overline{B*\overline{A}})$$
$$(\overline{A}+\overline{\overline{B}})*(\overline{B}+\overline{\overline{A}})=(\overline{A}+B)*(\overline{B}+A)$$
$$\overline{A}*\overline{B}+A*B+0+0$$

11-5 Numbering Systems

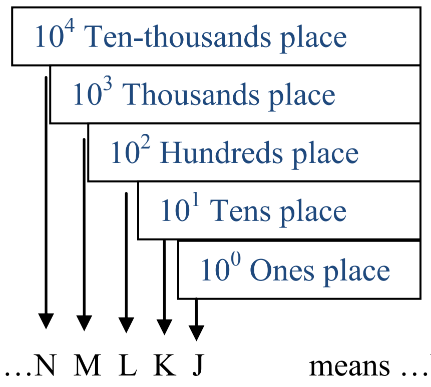
It’s quite obvious that our old continuous-math circuits could represent and manipulate a continuous range of numeric values: the signal can be a voltage of 10.01V or 10.0099V or… That’s particularly important for smooth operations like integration and differentiation. There are downsides though; for example, adders and integrators can’t distinguish between signal and noise – they add / integrate *whatever* their input is. Along those lines, if you want to represent number to seven digits, you need to have a signal-to-noise ratio of 10^7!

Now, it’s quite obvious that our new discrete-math circuits can represent and manipulate 0’s and 1’s, but what may not be immediately obvious is that, with 1’s and 0’s they can represent any discrete number to however many digits you’re willing to handle; the catch is that rather than being able to encode the value in one continuously varying signal, you encode it in N discretely varying signals. We just have to represent that value in Binary.

Decimal a.k.a. Base 10

First let’s quickly look at how we represent numbers in Decimal, then we’ll see that it’s not so very different in Binary. The two freedoms we have for representing different numbers are *symbols* and *placement* of those symbols. First we step through our basis of ten available symbols, 0-9, and then, if that’s not enough, we add a ‘place’ and do it again.

So, when we write 12,425 we've encoded the numeric value in the specific placement of specific symbols

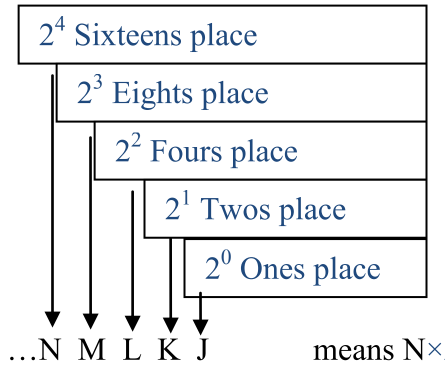


means $\ldots N\times10^4 + M\times10^3 + L\times10^2 + K\times10^1 + J\times10^0$

where N,M,L,K,J… can be any numeral from 0 to 9

Binary a.k.a. Base 2

Again, we've got the freedoms of symbols and placement with which to encode a number. Now, in Binary we only have a basis of two symbols, 0 and 1. So rather than having to move up to the next place after every 10th symbol, we have to do it after every 2nd symbol.



means $N\times2^4 + M\times2^3 + L\times2^2 + K\times2^1 + J\times2^0$

Where N, M, L, K, J… can be 0 or 1.

So, for example 10011 = $1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 16 + 0 + 0 + 2 + 1 = 19$

Vocabulary

Bit = one binary digit (which means one electronic 'line' carrying one signal)

Byte = eight binary digits (which means eight electronic 'lines', each carrying one signal)

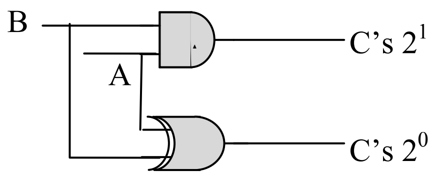
With eight places, largest value is 11111111 =
$1\times2^7+1\times2^6+1\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0 = 128+64+32+16+8+4+2+1=255$

and of course the smallest is 00000000

From 0 up to 255, there are $256=2^8$ discrete possible values. You may have encountered this language "8 bit", "16 bit", "32bit" and "64bit" when people talk about computers. For example, you can set your monitor to 16bit or 32bit color: 2^{16}=65,536 shades or 2^{32}=4,294,967,296 shades. 64bit logic means that a value can be represented across 64 discrete lines.

Physics 310
Lecture 8b – Digital Circuits

We'll get more experience with this later, but just to relate this 'binary' stuff back to our logic gates, here's a really simple operation: adding two one-bit numbers and encoding that as a two-bit number:



Say A = 0 and B = 0,then
C 2^1 2^0
0 0 = zero
Say A = 1 and B = 0, or the other way around, then
C 2^1 2^0
0 1 = one
Say A = 1 and B= 1
C 2^1 2^0
1 0 = two

11-6 Codes
The book notes that there are a variety of ways to encode a number when you only have freedoms of *place* and *two symbols.* For us, it suffices to appreciate this fact, and move on.

Chapter 12: digital Circuitry

Introduction. In chapter 11 we met the basic components but didn't talk much about actually achieving anything with them. Chapter 12 is more about applications. Chapter 13, which we'll briefly touch on in a few weeks, is about the most famous application of digital circuitry – computers. For now, we'll focus on counters, registers, and control circuits.

Physics 310
Lecture 8b – Digital Circuits

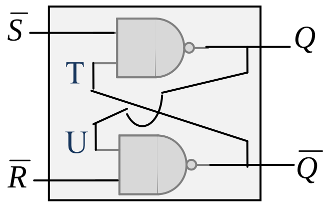
12-2 Flip-Flops

You first met a Flip-Flop inside the 555 timer. There are actually a few different styles of Flip-Flops that we'll explore. We'll start with the simple RS flip-flop and then see how it can be built upon and modified to give greater control.

RS (traditionally for Reset and Set, the "NOT" bars will make sense when we see the next model)

Let's use what we know about NAND gates to reason out how this thing behaves. The logic of the individual NANDs is

1	$\bar{S}$	T=$\bar{Q}$	Q
a	0	0	1
b	1	0	1
c	0	1	1
d	1	1	0



2	$\bar{R}$	U=Q	$\bar{Q}$
a	0	0	1
b	1	0	1
c	0	1	1
d	1	1	0

Now we try to merge these two logic tables, cutting out the middle-men of U and T

From the first table's rows a and c we see that Q is 1 as long as $\bar{S}$ is 0 (regardless of $\bar{Q}$)

From the second table's rows a and c we see that $\bar{Q}$ is 1 as long as $\bar{R}$ is 0 (regardless of Q)

Now, to fill in some more of the blanks, from the first table's row d, if $\bar{S}$ = 1 and T=$\bar{Q}$ = 1, then Q= 0

From the second table's row d, if $\bar{R}$ =1 and Q = 1, then $\bar{Q}$ =0.

3	$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$
a	0	0	1	1
b	1	0	0	1
c	0	1	1	0
d	1	1	hold	hold

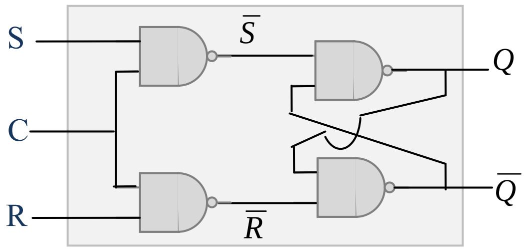
Now for the tricky row. We know that $\bar{S}$ = 1 and $\bar{R}$ =1, but we don't really have a foothold on $\bar{Q}$ and Q. If we assume $\bar{Q}$ = 1, then Q=0 according to table 1's row d; which is consistent with table 2's row b. But before we get wed to the idea, if we assume that $\bar{Q}$ = 0, then Q = 1 according to table 1's row b, which is consistent with table 2's row d. Which is it? The proper conclusion is that, if you start out in table 3's state b, and then flip $\bar{R}$'s value, you *hold* the output values of state b, but if you start out in table 3's state c, and then flip $\bar{S}$'s value, you *hold* the output values of state c.

The book notes that the top row can be unstable in some SR flip-flop designs, though that isn't the case in this one.

Physics 310
Lecture 8b – Digital Circuits

Clocked RS

What if the input is intended to change from $\overline{S}$ =1, $\overline{R}$ =0 to $\overline{S}$ = 0, $\overline{R}$ =1, so going from row b to row c, but say the $\overline{R}$ value switches a tad faster than the S value does; oops, you get a moment of row a! How can this be avoided? If the RS flip flop is "clocked." A third input is used to tell the flip flop when it's time to look at its inputs, the rest of the time it simply *holds* its outputs. This is kind of like the additional control on tri-state gates, except rather than telling the gate when it's time to *set the output*, the clock tells the flip-flop when it's time to *see the input*.



Now, when **C =Hi**, the top NAND gate gives $\overline{1*S} = \overline{S}$ and the bottom gate gives $\overline{1*R} = \overline{R}$, thus the naming we already used. From there, we've already figured out the device's truth table.

C=Hi	S	R	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
a	1	1	0	0	1	1
b	0	1	1	0	0	1
c	1	0	0	1	1	0
d	0	0	1	1	hold	hold
C=Lo	x	x	1	1	hold	hold

Now, when **C = Lo**, the top NAND gate gives $\overline{0*S} = \overline{0} = 1$and the bottom one similarly gives $\overline{0*R} = \overline{0} = 1$regardless of R or S's values. With the $\overline{S}$ and $\overline{R}$ both = 1, the **outputs just hold.**

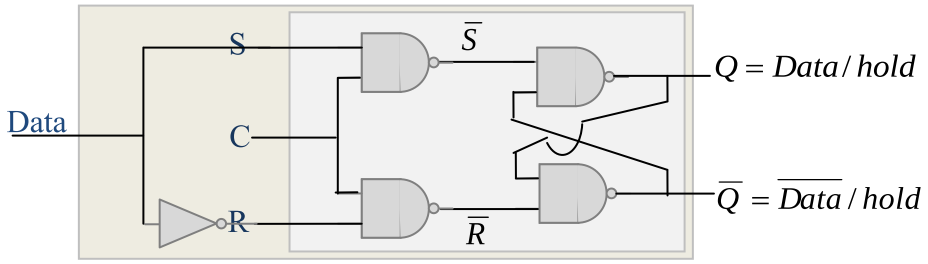
So, the outputs are responsive to the S and R values *only if C=Hi*. Otherwise, they just hold their values. Aside from avoiding unintentionally stumbling into a state like row a, this also allows one physical pair of A, B input lines to feed a lot of flip-flops, each one looks on the lines for *it's* appropriate input signals when its clock tells it to.

There are two valuable variations on the Clocked RS Flip-Flop. One is essentially two in series but out of synch so that information moves through the circuit in a slow, orderly fashion. The other essentially has only one 'input' signal.

Physics 310
Lecture 8b – Digital Circuits

Data Flip-Flop

This is a clocked RS flip-flop that's fed one signal, 'Data.' It goes right onto the S line and its inverse goes onto the R line.



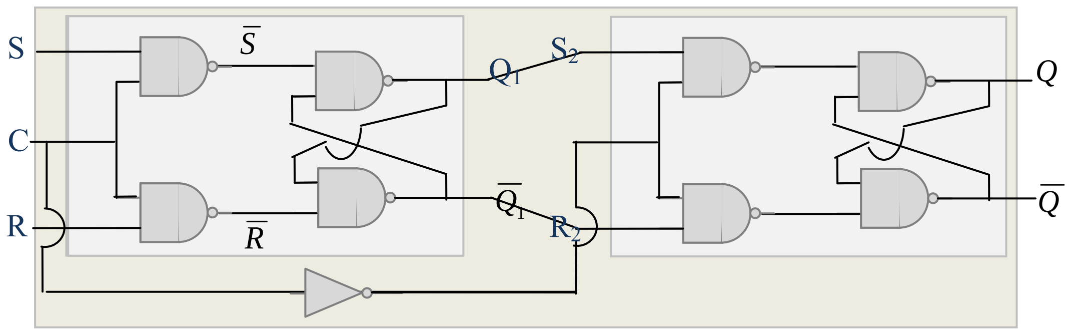
Looking at the data table, since we're guaranteed that R and S will be opposite, rows a and d just aren't options.

C=Hi	S=Data	R=$\overline{\text{Data}}$	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
a	1	1	0	0	1	1
b	0	1	1	0	0	1
c	1	0	0	1	1	0
d	0	0	1	1	hold	hold
C=Lo	x	x	1	1	hold	hold

Thus Q just passes the input signal / Data value and $\overline{Q}$ passes the opposite. Then why bother using the flip-flop at all? Because it passes these values *only* when the clock is Hi. When the clock is Lo, it holds its value regardless of what happens to the input signal. Why would you want to do that? One word: "memory." These gates 'remember' what the data line said when their clocks were high. They will hold that memory until they're told to over-write it by the clock going high again. That alone probably suggests a use for such gates. There are a couple of other nice uses For one thing, right when that input line transitions between high and low, if there's any noise on that signal, it could flutter back and forth across the threshold; meanwhile, this divice's output won't be effected by that fluttering if the clock is Lo during the moment of transition. Alternatively, you can have lots of these gates fed by one single "data" wire, then each gate can 'look' at the wire only when it's its turn / its clock goes high.

Master-Slave Flip Flop

This is essentially two clocked RS flip-flops in series, but the second runs off $\overline{C}$, is it *holds* while its predecessor updates, then it *updates* while its predecessor holds. All together, the device samples the input when C is Hi ('tick') and updates the output when C is Lo ('tock'). This slows down the transmission of information, which can be invaluable if, say, the output of the flip-flop is used in a feedback to influence its own next input.



	S	R	**C=Hi**		**C=Lo**	
			$Q_1=S_2$	$\overline{Q}_1=R_2$	$Q_1=S_2$	$\overline{Q}_1=R_2$
a	1	1	1	1	Hold	Hold
b	0	1	0	1	Hold	Hold
c	1	0	1	0	Hold	Hold
d	0	0	Hold	Hold	Hold	Hold

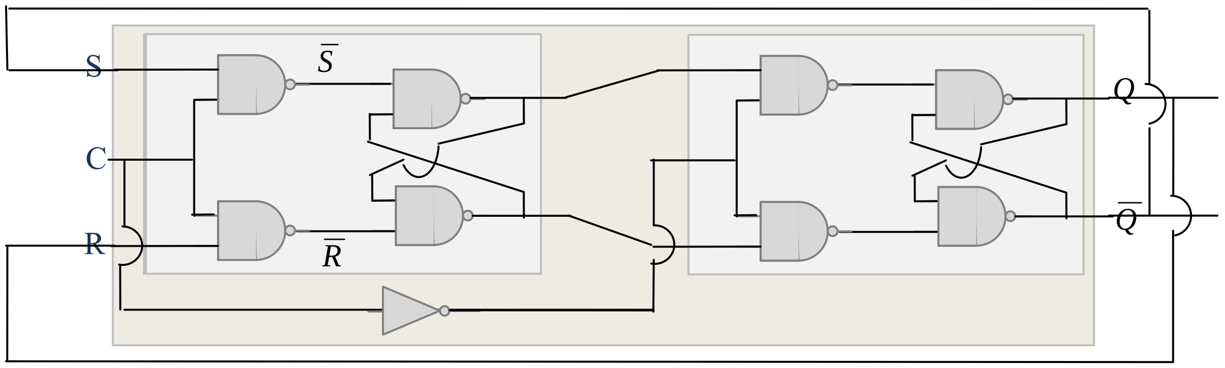
	S_2	R_2	**C=Hi**		**C=Lo**	
	Q_1	$\overline{Q}_1$	$Q_1=S_2$	$\overline{Q}_1=R_2$	Q	$\overline{Q}$
a	1	1	Hold	Hold	1	1
b	0	1	Hold	Hold	0	1
c	1	0	Hold	Hold	1	0
d	0	0	Hold	Hold	Hold	Hold

So, if the input changes on a 'tick' (clock going Hi), that change doesn't trickle through to the output until the 'tock' (clock going Lo). One way of looking at the master-slave flip-flop's operation is this: the output just *after* the clock signal goes low depends on the input just *before* it went low.

Physics 310
Lecture 8b – Digital Circuits

Toggle Flip-flop: Divide-by-two

So far, we've met a few different variations on the basic flip-flop. Now let's consider a very simple circuit using one. Mater-Slave with feedback of $\overline{Q} \to S$ and $Q \to R$. Then the clock is the only free parameter.

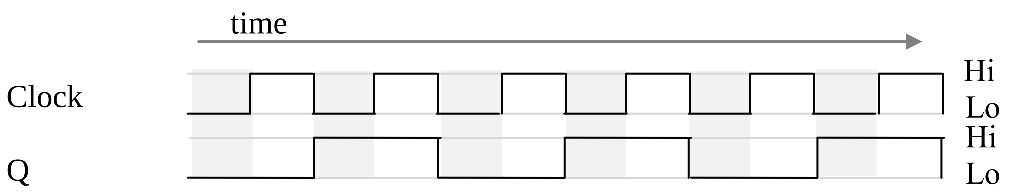


What the heck would this do? Plausibly, it's going to cycle its output, so just to jump into that cycle and see where it leads:

S/R		Q_1=S_2/$\overline{Q}_1$ =R_2		Q/$\overline{Q}$
Clock Lo=0 (tock)				
1		0	→	0
0		1	→	1
Clock goes Hi = 1 (tick)				
1	→	1		0
0	→	0		1
Clock goes Lo = 0 (tock)				
0		1	→	1
1		0	→	0
Clock goes Hi = 1 (tick)				
0	→	0		1
1	→	1		0

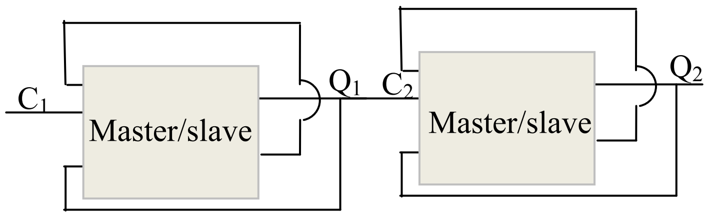
Notice that Q changes values *half* as frequently as the clock does (ditto for any of the other signals.)

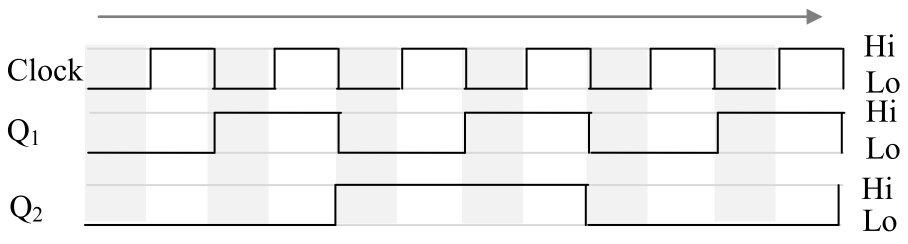
A convenient way of visualizing this is



Physics 310
Lecture 8b – Digital Circuits

Imagine what you'd get if you used this circuit's output as the clock for a similar circuit. Notice that the output signal changes each time the clock transitions from Hi to Lo, in keeping with the read that the output *after* the clock goes low depends upon the input *before* it went low.

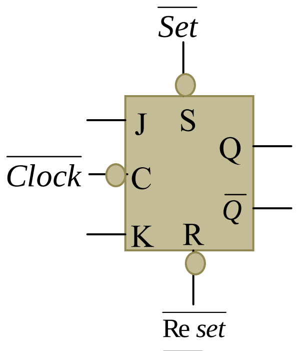




Q_2 changes states a *quarter* as often as the clock does.

JK Flip-Flop

The next version of flip-flop has the desirable 'output after down tick depends on input before' feature. It also has two *more* controls.



The book gives a schematic of its innards, but it's probably not worth our time to reason through it, just close the hood and accept that these are the rules of its operation:

$\overline{Q}$ = opposite of Q always (so we won't bother listing it separately)

Q = 1 if $\overline{Set} = 0$, regardless of J or K (bar indicates active when low)

Q = 0 if $\overline{\text{Re}\,set} = 0$, regardless of J or K (bar indicates active when low)

Otherwise, if $\overline{Set} = \overline{\text{Re}\,set} = 1$, then the output *after* the clock transitions from 1 to 0 depends on the inputs *before* the transition as

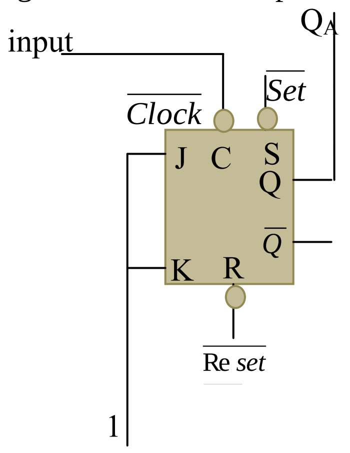
C = Hi before downstroke t_n		After downstroke t_{n+1}
J	**K**	**Q**
0	0	Q_n (What it already was)
1	0	1
0	1	0
1	1	$\overline{Q}_n$ (opposite of what it was)

The bar over "clock" indicates that input influences output *when clock goes low.*

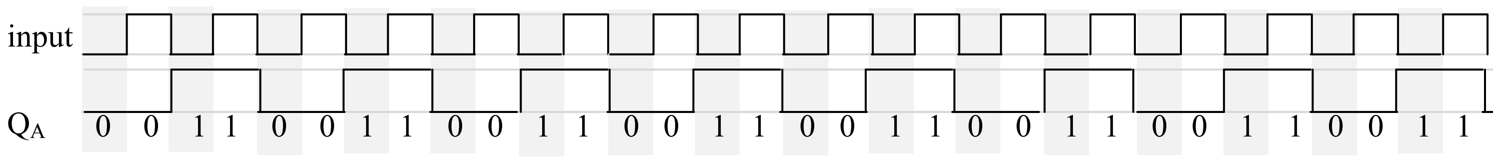
Physics 310
Lecture 8b – Digital Circuits

12-4 Count to 16 binary counter

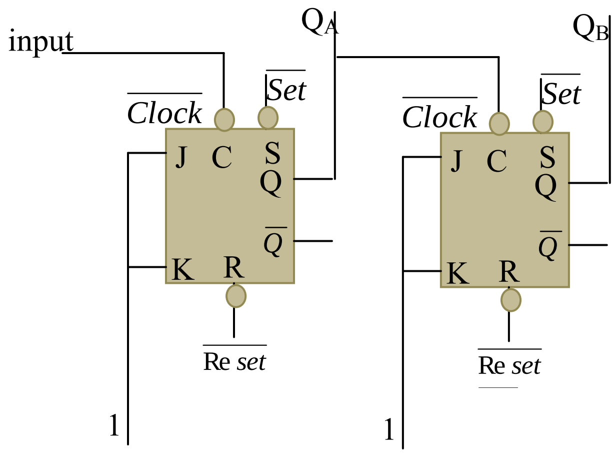
We've already seen how to make a 'divide by 2' and a 'divide by 4' circuit with master-slave flip-flops. Something similar can be achieved with these JK flip flops. For the sake of not twisting wires, the clock input is displayed at the top.

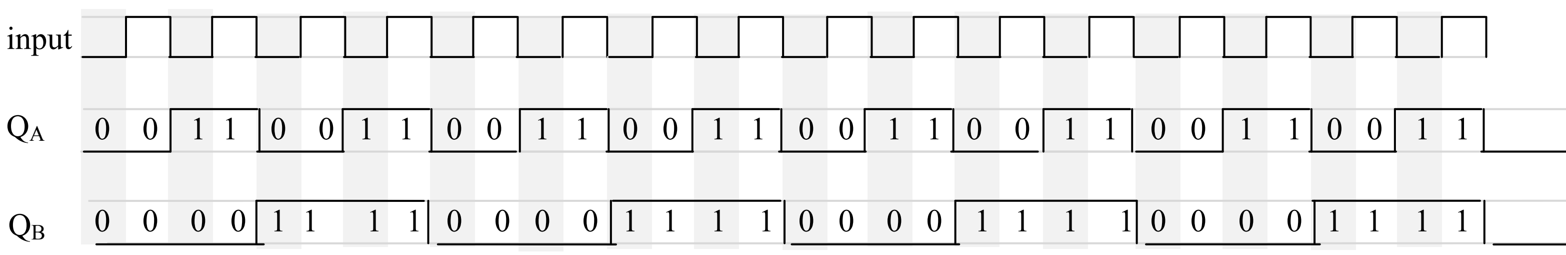


Looking at the truth table, with both J and K held high, the output, Q, is doomed to flip states every time the input transitions from Hi to L. Just to start somewhere, say that the output is initially low, then if the input regularly ticks and tocks, the output looks like



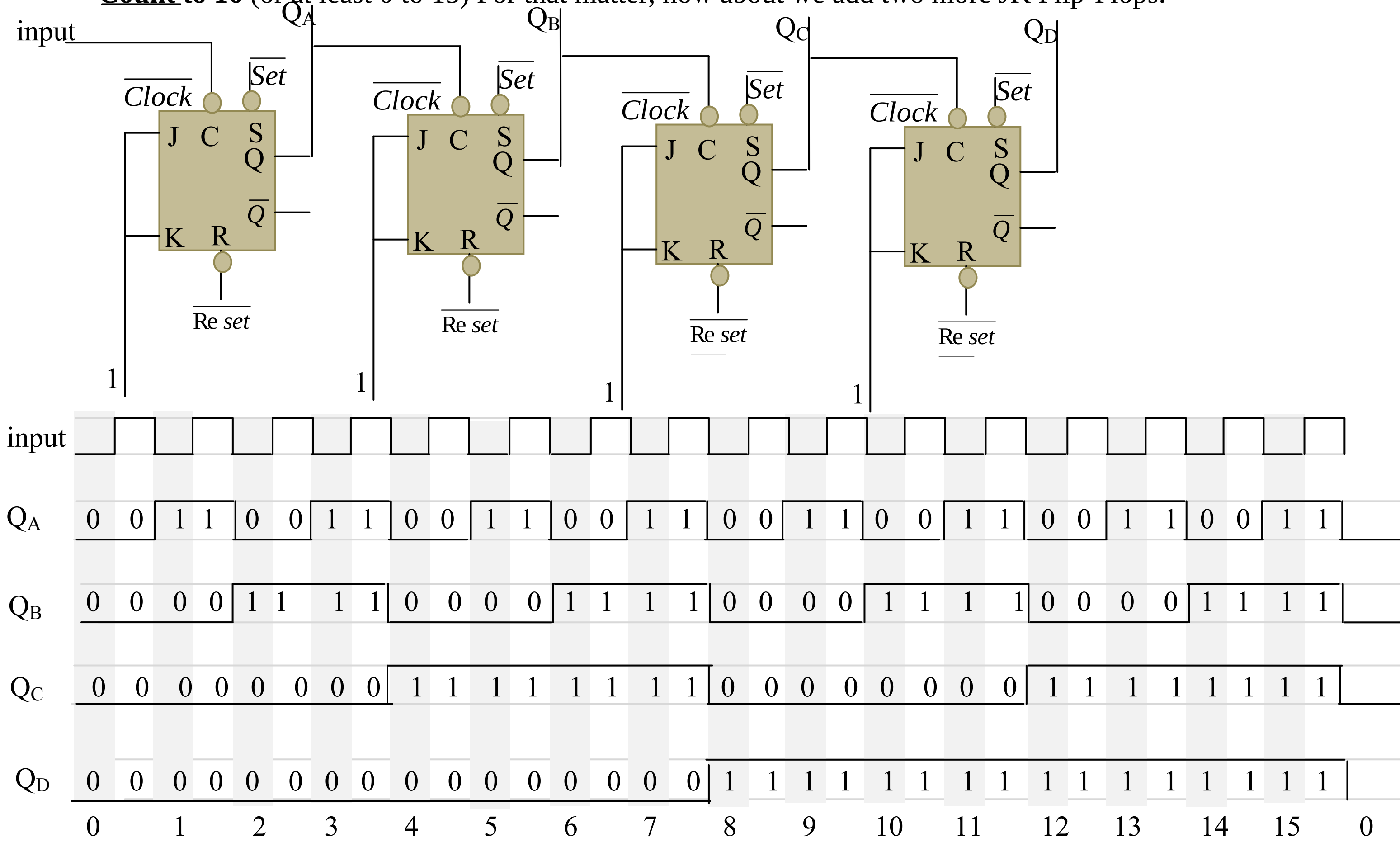
Now let's add a second JK Flip-Flop that's 'clocked' by the first one's input.





Physics 310
Lecture 8b – Digital Circuits

<u>Count</u> to 16 (or at least 0 to 15) For that matter, how about we add two more JK Flip-Flops.



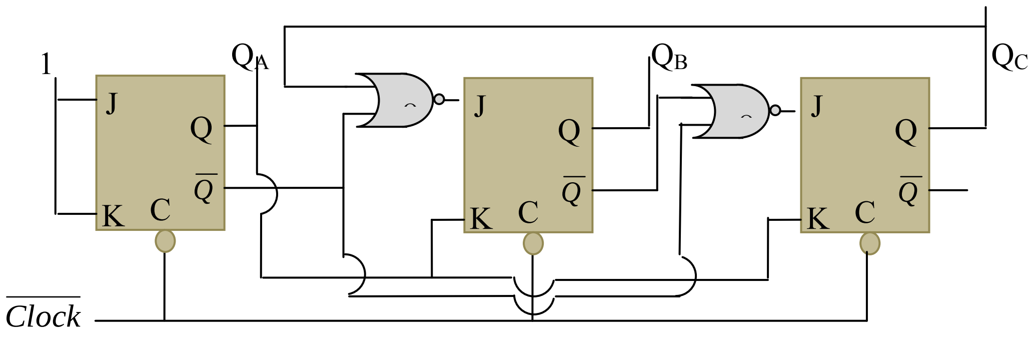
This was an example of an "asynchronous" counter, which means that the chips do *not* share a common 'clock' signal. This particular case, each chip's "clock" input is the previous chip's output, is known as a "ripple counter."

Physics 310
Lecture 8b – Digital Circuits

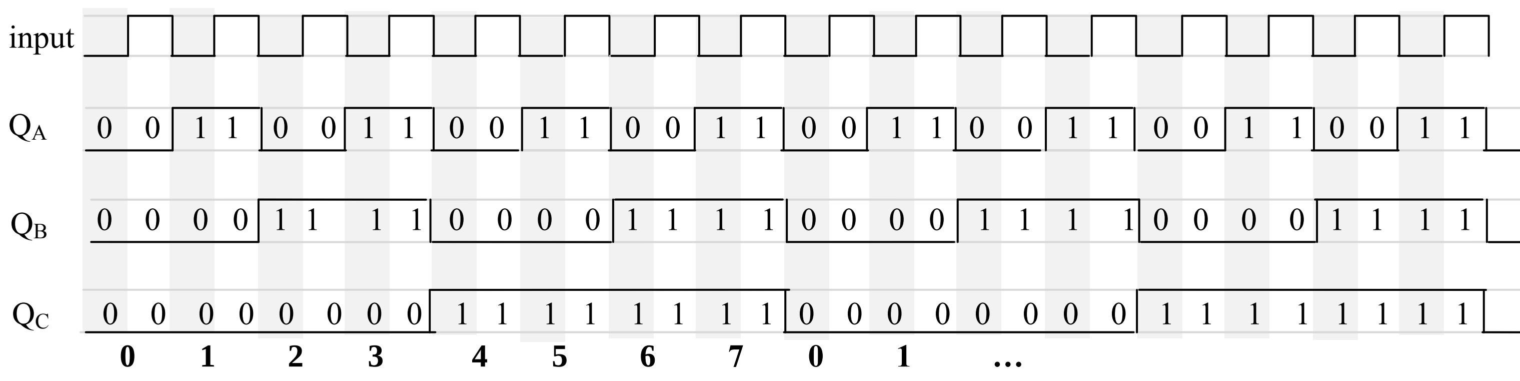
Alternatively, rather than holding the J & K inputs high for all chips and just manipulating the ‘clock’ signals, we could give all the chips the same clock signal and manipulate their J & K inputs in order to make them ‘count.’ This, with all chips running off the same clock, would be *synchronous*. Here’s an example of a synchronous shift / ring counter. (Warning: C, R, and S get shoved all around in these diagrams in order to make the external wiring look the simplest, don’t take the geometry too literally.)

Recall the logic table:

C = Hi	t_n	t_{n+1}	
J	**K**	$\mathbf{Q_{n+1}}$	$\overline{Q}_{n+1}$
0	0	Q_n	$\overline{Q}_n$
1	0	1	0
0	1	0	1
1	1	$\overline{Q}_n$	Q_n



Also recall that a NOR gate’s output is Hi *only* if both inputs are Lo. The outputs cycle as



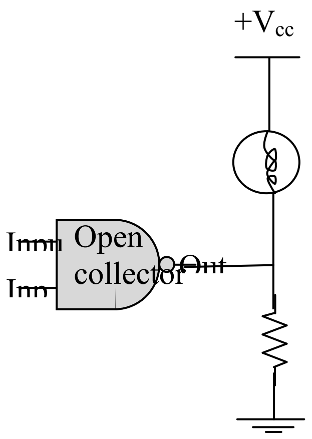
The text, on pages 276 through 277, gives a variety of different counters – they count up, they count down, they count through 8 states, they count through 16 states,…

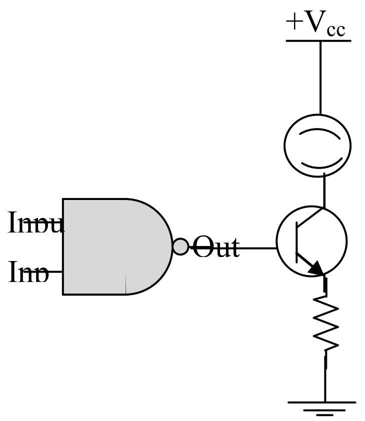
While one can construct a counter from flip-flops and gates, there are also single-chip counters such as the 74192 which you can set to count just to 1 or up to 9. Running two of *these* in series allows you to count to 99 – one of them handles the 1’s place while the other handles the 10’s place and gets updated each time the first one cycles through all its digits.

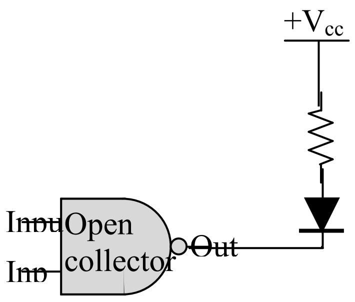
12-3 Digital Read-Outs

At this point, before getting more into actual circuits that use these chips, the book takes a little digression about ways of *displaying* the logic values. There are three basic types of lights: incadenscent (which takes a lot of current), florescent (which takes a large voltage) and Light Emitting Diode, LED, which takes neither a large current nor a large voltage. In lab last week you already employed these to visualize when the 555’s output was high or low. For reference, the book shows sample wirings using the three different types of lights.

Physics 310
Lecture 8b – Digital Circuits

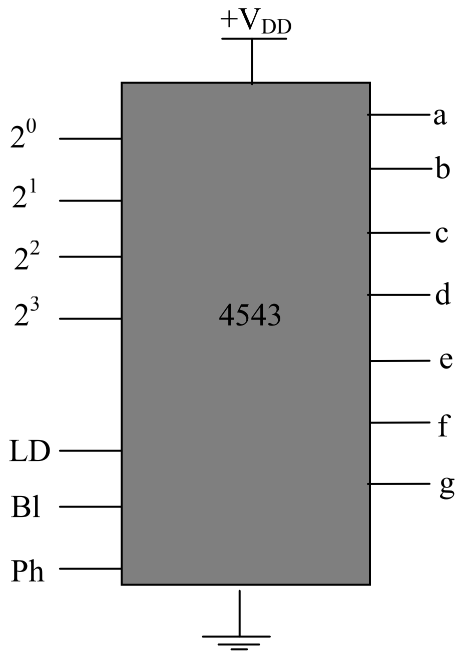


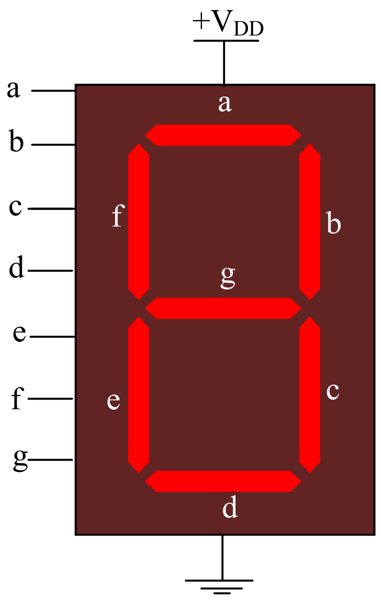




LED's come in a small range of colors: red, green, yellow, blue. Orange can be achieved with a diode that's built for red when biased one way and yellow another, then an A/C signal would give the appearance of orange.

A third common type of display is LCD. It doesn't actually generate light, but reflectivity can be varied by getting liquid crystals to line up or not. A *very* common display that uses LCD's is the Seven-Segment Display as is featured in those *nifty* digital watches.



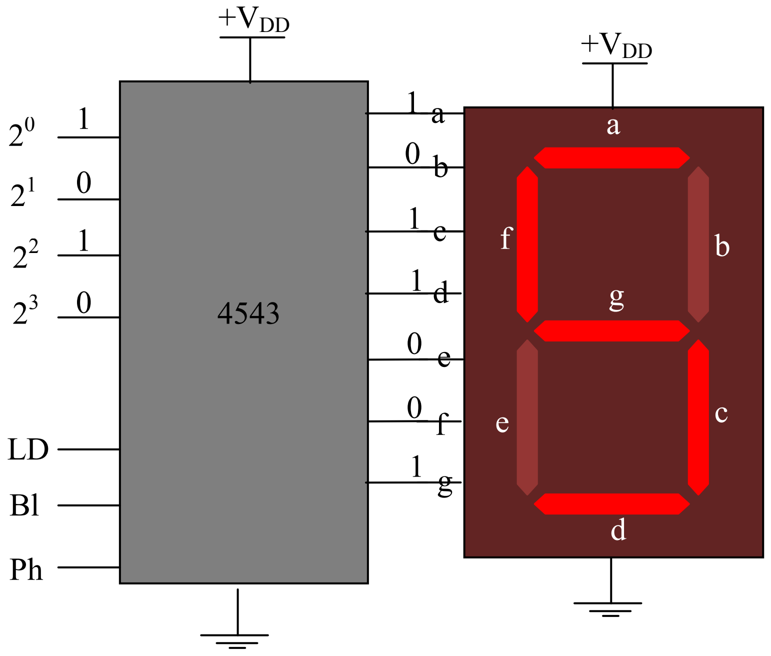


When the different input lines go high, the different segments get activated. One of these is often accompanied by a BCD-to-seven-segment to take something in binary and light up the appropriate decimal digit. As the labels suggest, if a binary number is represented on the top left lines, then the appropriate combination of segments get lit up. In the spirit of 'waste not, want not', this can easily make 16 different shapes, and so represent 0 – 15, but the shapes used for 10 and above are not ones we typically use.

LD = Lo: Locks the Display regardless of how the output may subsequently change
Bl = Hi: Blanks the display

Ph = Hi: flips the “Phase” of the output logic (what would be Hi is Lo and vice versa).

Here’s an example using these two chips together:



Binary		Decimal
1010	=	5