Physics 124: Lecture 5

Binary, Hexadecimal, and Logic

adapted from T. Murphy's lectures

Binary, Hexadecimal Numbers

- Computers store information in binary
  - 1 or 0, corresponding to V_{CC} and 0 volts, typically
  - the CC subscript originates from “collector” of transistor
- Become familiar with binary counting sequence

binary	decimal	hexadecimal
0000 0000	0	0x00
0000 0001	1	0x01
0000 0010	2	0x02
0000 0011	2+1 = 3	0x03
0000 0100	4	0x04
0000 0101	4+1 = 5	0x05
etc.		
1111 1100	128+64+32+16+8+4 = 252	0xfc
1111 1101	128+64+32+16+8+4+1 = 253	0xfd
1111 1110	128+64+32+16+8+4+2 = 254	0xfe
1111 1111	128+64+32+16+8+4+2+1 = 255	0xff

Binary to Hex: easy!

- Note separation of previous 8-bit (one-byte) numbers into two 4-bit pieces (nibbles)
  - makes expression in hex (base-16; 4-bits) natural

binary	hexadecimal	decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A (lower case fine)	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Hexadecimal, continued

- Once it is easy for you to recognize four bits at a time, 8 bits is trivial:
  - 01100001 is 0x61
  - 10011111 is 0x9f
- Can be handy because the ASCII code is built around hex:
  - ‘A’ is 0x41, ‘B’ is 0x42, …, ‘Z’ is 0x5a
  - ‘a’ is 0x61, ‘b’ is 0x62, …, ‘z’ is 0x7a
  - ‘^A’ (control-A) is 0x01, ‘^B’ is 0x02, ‘^Z’ is 0x1a
  - ‘0’ is 0x30, ‘9’ is 0x39

ASCII Table in Hex

first hex digit

second hex digit

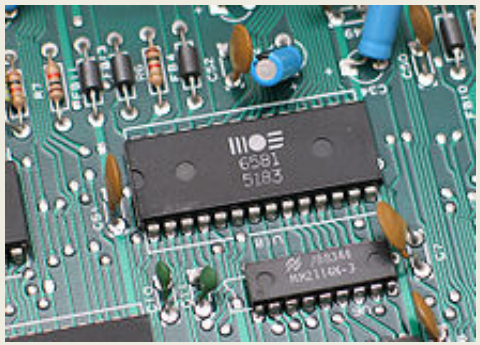
	0	1	2	3	4	5	6	7
0	NUL ^@ null (\0)	DLE ^P	SP space	0	@	P	`	p
1	SOH ^A start of hdr	DC1 ^Q	!	1	A	Q	a	q
2	STX ^B start text	DC2 ^R	"	2	B	R	b	r
3	ETX ^C end text	DC3 ^S	#	3	C	S	c	s
4	EOT ^D end trans	DC4 ^T	$	4	D	T	d	t
5	ENQ ^E	NAK ^U	%	5	E	U	e	u
6	ACK ^F acknowledge	SYN ^V	&	6	F	V	f	v
7	BEL ^G bell	ETB ^W	'	7	G	W	g	w
8	BS ^H backspace	CAN ^X	(	8	H	X	h	x
9	HT ^I horiz. tab (\t)	EM ^Y	)	9	I	Y	i	y
A	LF ^J linefeed (\r)	SUB ^Z	*	:	J	Z	j	z
B	VT ^K vertical tab	ESC escape	+	;	K	[	k	{
C	FF ^L form feed	FS	,	<	L	\	l	|
D	CR ^M carriage ret (\n)	GS	-	=	M	]	m	}
E	SO ^N	RS	.	>	N	^	n	~
F	SI ^O	US	/	?	O	_	o	DEL

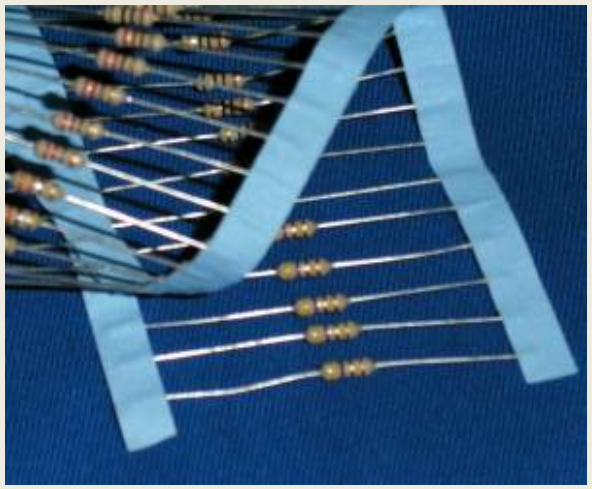
ASCII in Hex

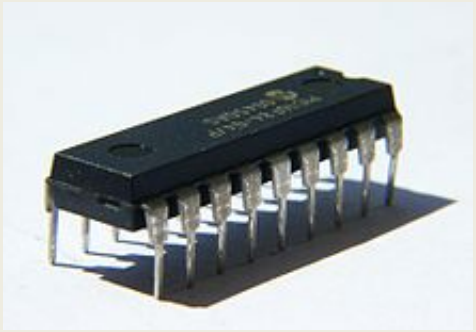
- Note the patterns and conveniences in the ASCII table
  - 0 thru 9 is hex 0x30 to 0x39 (just add 0x30)
  - A-Z parallels a-z; just add 0x20
    - starts at 0x41 and 0x61, so H is 8th letter, is 0x48, etc.
  - the first 32 characters are control characters, often represented as Ctrl-C, denoted ^C, for instance
    - associated control characters mirror 0x40 to 0x5F
    - put common control characters in red; useful to know in some primitive environments

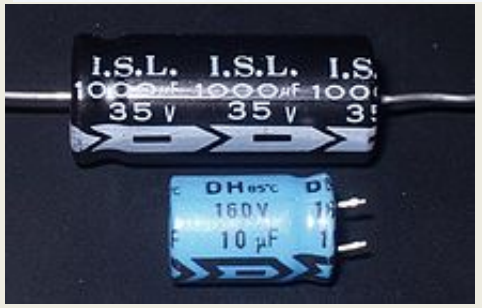
Parenthesis: Thru-hole vs SMT (Surface Mounted Technology)

Thru-hole

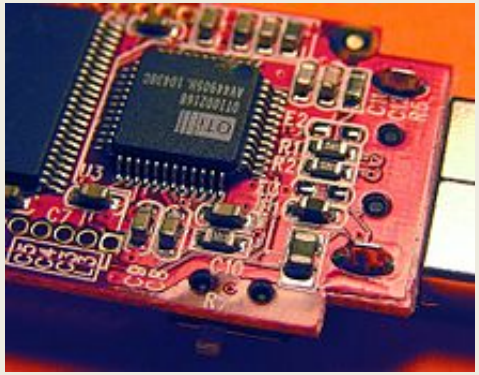


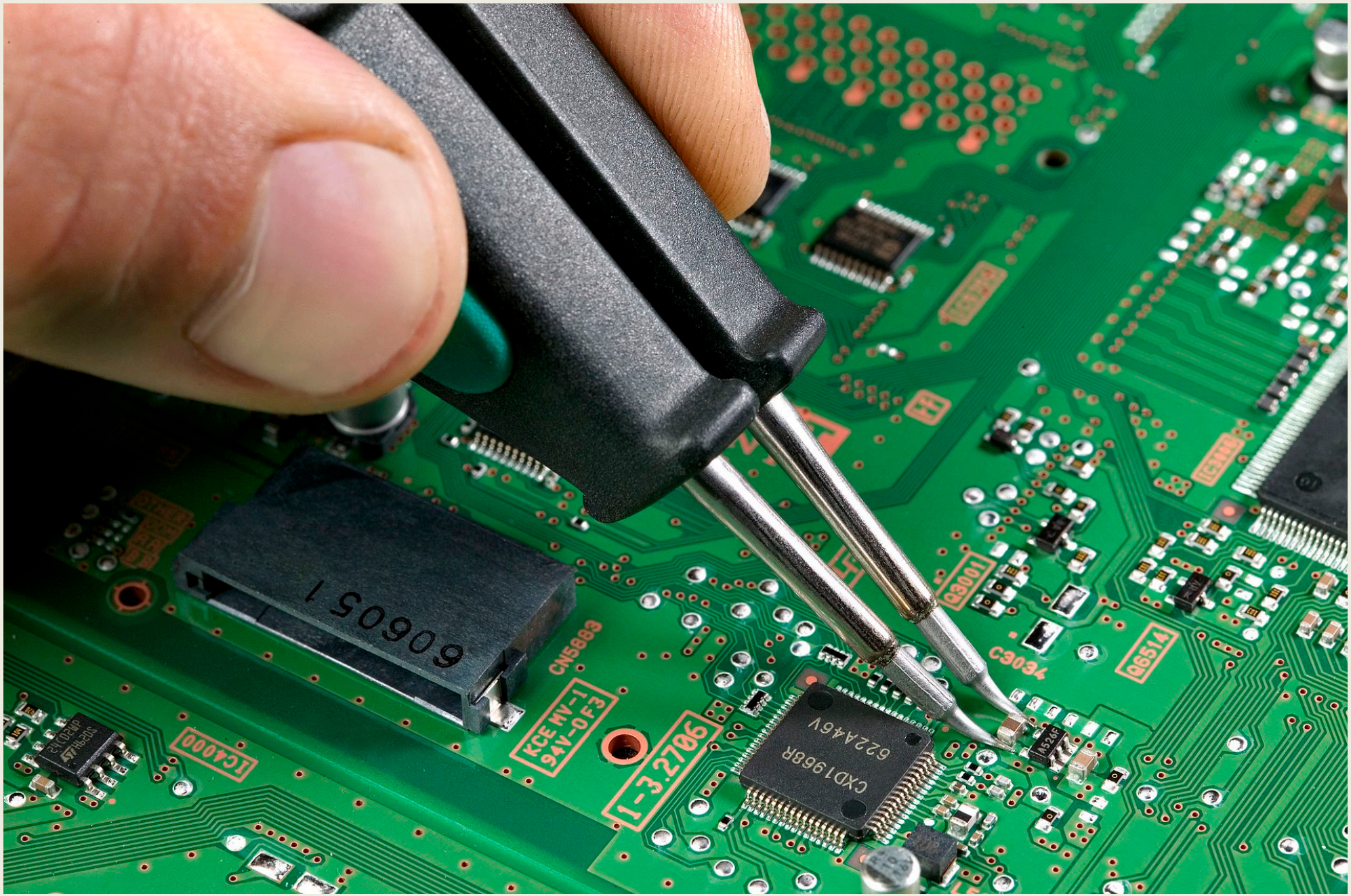






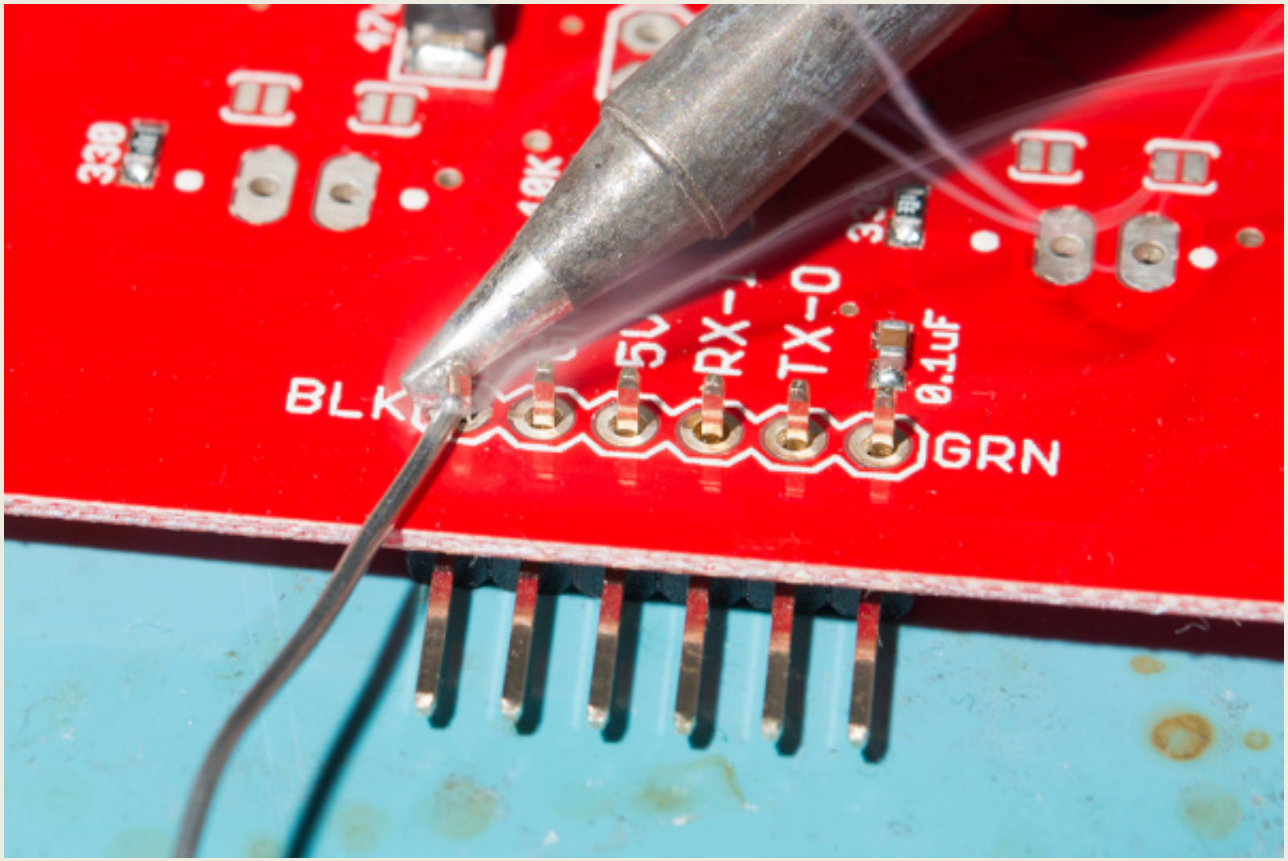
SMT




606051
CN5883
KCE MV-1
94V-0F3
1-3.2706
IC4000
Q3001
C3034
Q6514
CXD1968R
6224A6V

Intro to soldering

- https://learn.sparkfun.com/tutorials/how-to-solder-through-hole-soldering



Soldering equipment and supplies

solder

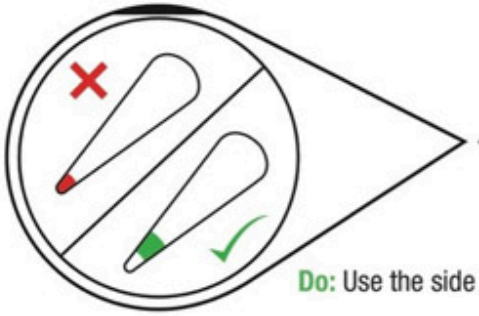
soldering irons

Soldering tips

soldering wand

brass sponge

“Dos and Don’ts”



Don’t: Use the very tip of the iron.

Do: Use the side of the tip of the iron, “The Sweet Spot.”

Do: Touch the iron to the component leg and metal ring at the same time.

Do: While continuing to hold the iron in contact with the leg and metal ring, feed solder into the joint.

Don’t: Glob the solder straight onto the iron and try to apply the solder with the iron.

Do: Use a sponge to clean your iron whenever black oxidization builds up on the tip.

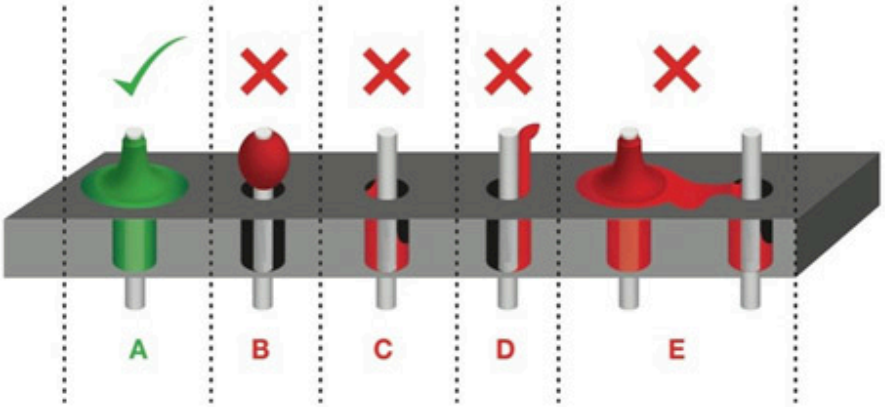
A Solder flows around the leg and fills the hole - forming a volcano-shaped mound of solder.

B Error: Solder balls up on the leg, not connecting the leg to the metal ring.
Solution: Add flux, then touch up with iron.

C Error: Bad Connection (i.e. it doesn’t look like a volcano)
Solution: Flux then add solder.

D Error: Bad Connection...and ugly...oh so ugly.
Solution: Flux then add solder.

E Error: Too much solder connecting adjacent legs (aka a solder jumper).
Solution: Wick off excess solder.



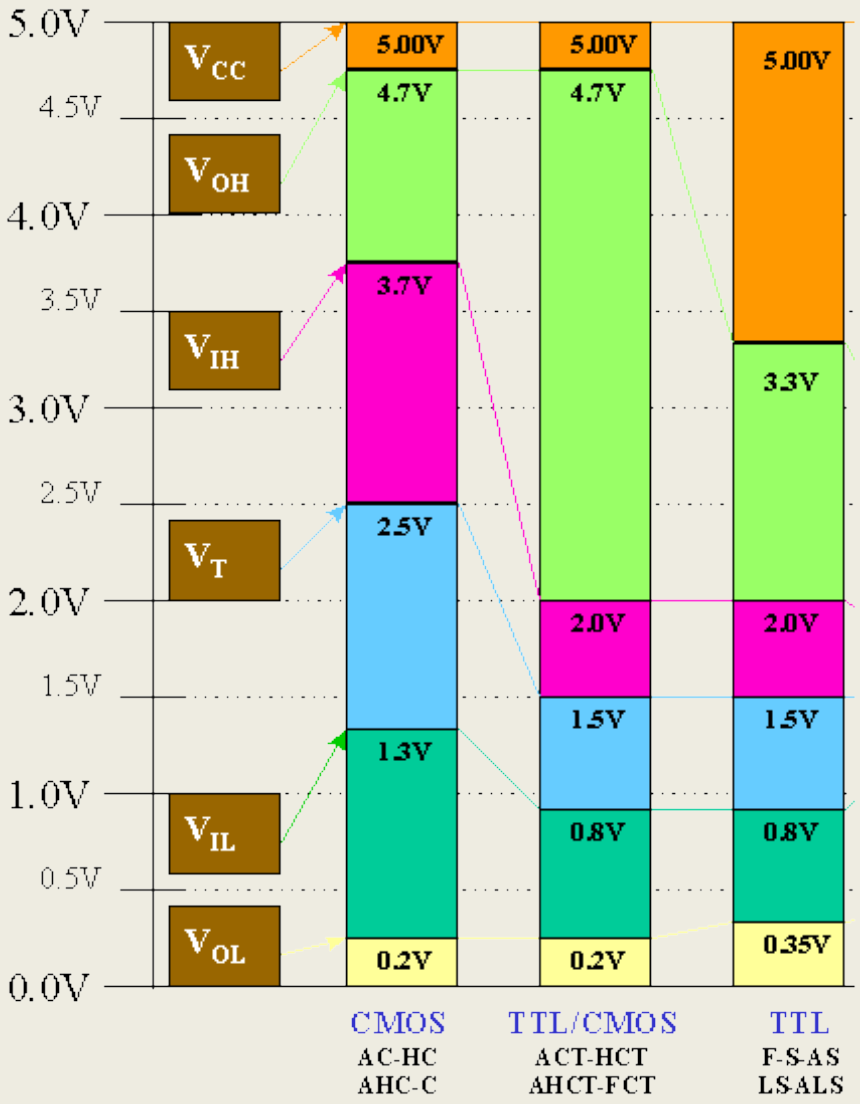
Logic Families

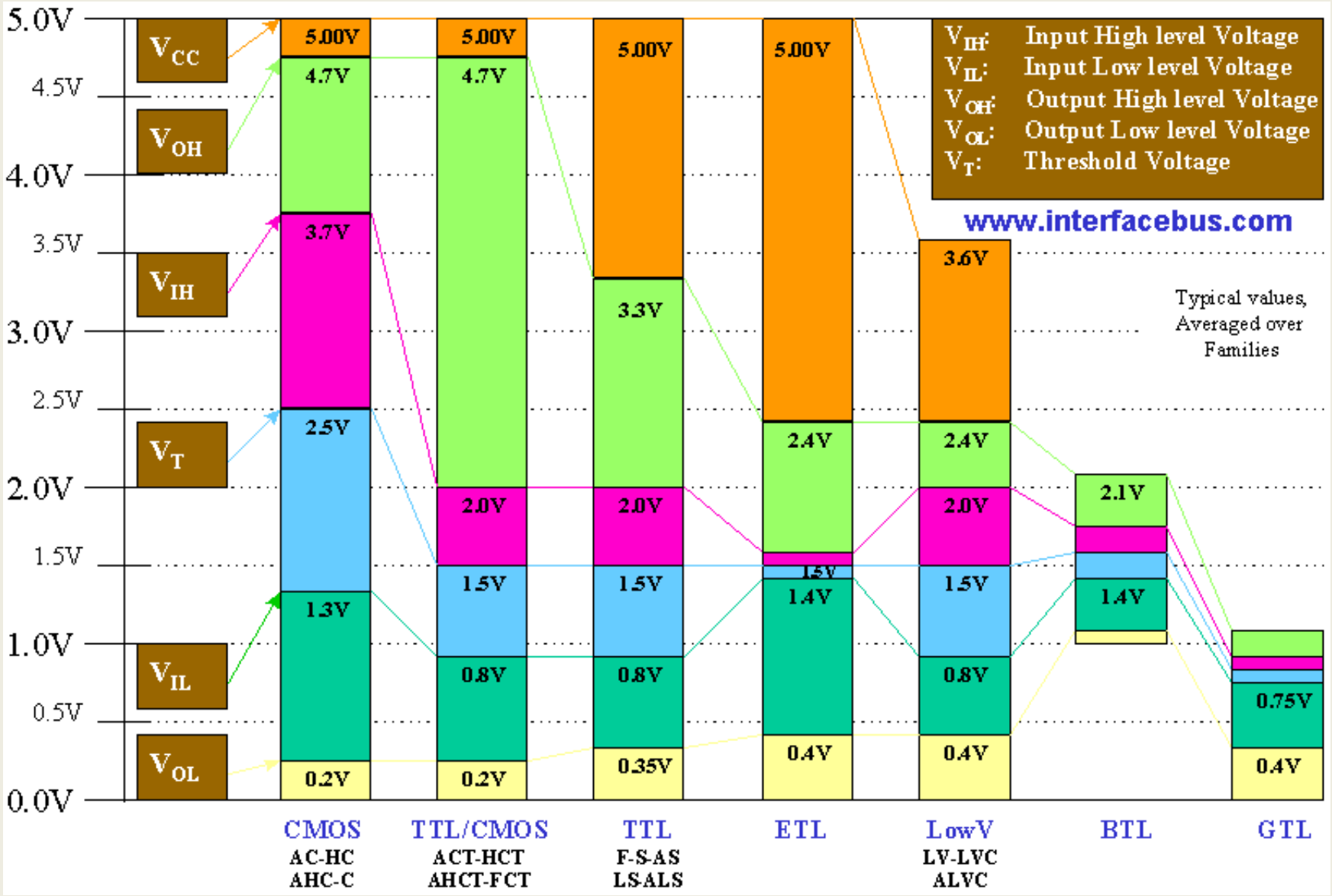
- TTL: transistor-transistor logic: BJT based
  - subtypes: L(ow power), LS(chotky), F(ast), A(dvanced)S, ALS, or H(igh-speed), LVTTL
  - output: logic high has $V_{OH} > 3.3$ V; logic low has $V_{OL} < 0.35$ V
  - input: logic high has $V_{IH} > 2.0$ V; logic low has $V_{IL} < 0.8$ V
  - dead zone between 0.8V and 2.0 V
    - nominal threshold: $V_T = 1.5$ V
- CMOS: complimentary MOSFET (Metal-Oxide-Semicond FET)
  - chips have HC or AHC designation (Advanced High-speed CMOS)
  - output: logic high has $V_{OH} > 4.7$ V; logic low has $V_{OL} < 0.2$ V
  - input: logic high has $V_{IH} > 3.7$ V; logic low has $V_{IL} < 1.3$ V
  - dead zone between 1.3V and 3.7 V
    - nominal threshold: $V_T = 2.5$ V
  - chips w/HCT designation are CMOS with TTL-compatible thresholds

Logic Family Levels

V_{IH}: Input High level Voltage
V_{IL}: Input Low level Voltage
V_{OH}: Output High level Voltage
V_{OL}: Output Low level Voltage
V_T: Threshold Voltage

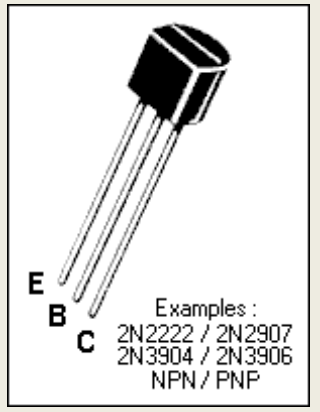
- CMOS is closer to the "ideal" that logic low is zero volts and logic high is 5 volts
  - and has a bigger dead zone
- Example: A TTL device must:
  - interpret any input below 0.8 V as logic low
  - interpret any input above 2.0 V as logic high
  - put out at least 3.3 V for logic high
  - put out less than 0.35 V for logic low
- The differing input/output thresholds lead to noise immunity

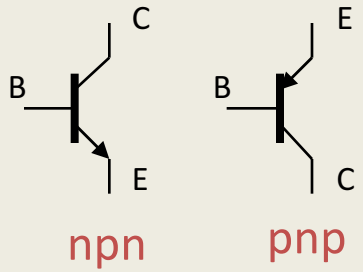



5.0V
4.5V
4.0V
3.5V
3.0V
2.5V
2.0V
1.5V
1.0V
0.5V
0.0V
V_{CC}
V_{OH}
V_{IH}
V_T
V_{IL}
V_{OL}
V_{IH}: Input High level Voltage
V_{IL}: Input Low level Voltage
V_{OH}: Output High level Voltage
V_{OL}: Output Low level Voltage
V_T: Threshold Voltage
www.interfacebus.com
Typical values, Averaged over Families
5.00V
4.7V
3.7V
2.5V
1.3V
0.2V
5.00V
4.7V
2.0V
1.5V
0.8V
0.2V
5.00V
3.3V
2.0V
1.5V
0.8V
0.35V
5.00V
2.4V
2.0V
1.5V
1.4V
0.4V
3.6V
2.4V
2.0V
1.5V
0.8V
0.4V
2.1V
1.4V
0.75V
0.4V
CMOS
AC-HC
AHC-C
TTL/CMOS
ACT-HCT
AHCT-FCT
TTL
F-S-AS
LS-ALS
ETL
LowV
LV-LVC
ALVC
BTL
GTL

Transistors

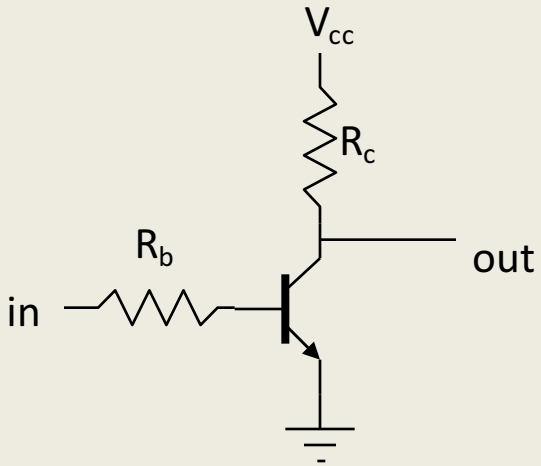
- Transistors are versatile, highly non-linear devices
- Two frequent modes of operation:
  - amplifiers/buffers
  - switches
- Two main flavors:
  - npn (more common) or pnp, describing doping structure
- Also many varieties:
  - bipolar junction transistors (BJTs) such as npn, pnp
  - field effect transistors (FETs): n-channel and p-channel
  - metal-oxide-semiconductor FETs (MOSFETs)
- We'll just hit the essentials of the BJT here
  - MOSFET later in lecture

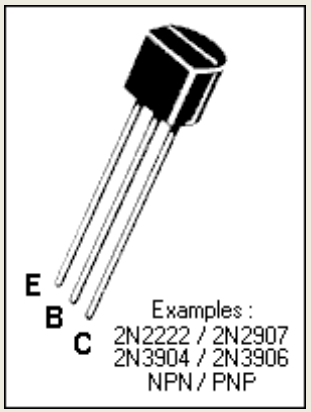


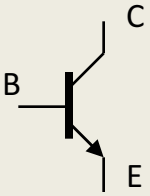


BJT Amplifier Mode

- Central idea is that when in the right regime, the BJT collector-emitter current is proportional to the base current:
  - namely, $I_{ce} = \beta I_b$, where the DC current gain β (or h_{fe}) is ~100
  - In this regime, the base-emitter voltage is ~0.6 V
  - below, $I_b = (V_{in} - 0.6)/R_b$; $I_{ce} = \beta I_b = \beta(V_{in} - 0.6)/R_b$
  - so that $V_{out} = V_{cc} - I_{ce}R_c = V_{cc} - \beta(V_{in} - 0.6)(R_c/R_b)$
  - ignoring DC biases, wiggles on V_{in} become $\beta\,(R_c/R_b)$ bigger (and inverted): thus amplified

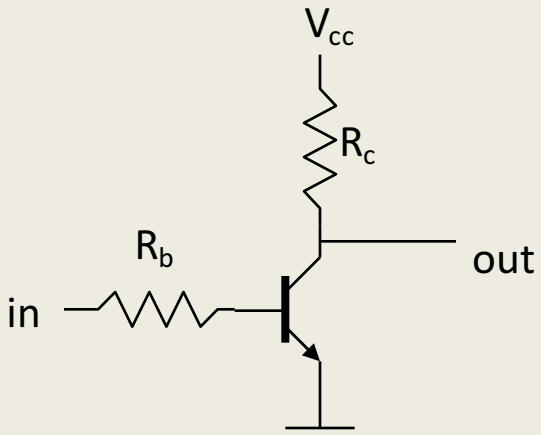






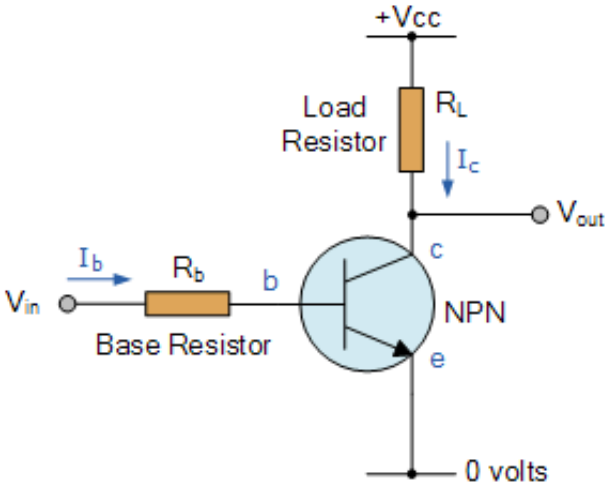
Switching: Driving to Saturation

- What would happen if the base current is so big that the collector current got so big that the voltage drop across R_c wants to exceed V_{cc}?
  - we call this saturated: $V_{CE} = V_c - V_e$ cannot dip below ~0.2 V
  - even if I_b is increased, I_c won't budge any more
- The example below is a good logic inverter
  - if V_{cc} = 5 V; R_c = 1 kΩ; I_c(sat) ≈ 5 mA; need I_b > 0.05 mA
  - so R_b < 20 kΩ would put us safely into saturation if V_{in} = 5V
  - now 5 V in → ~0.2 V out; < 0.6 V in → 5 V out

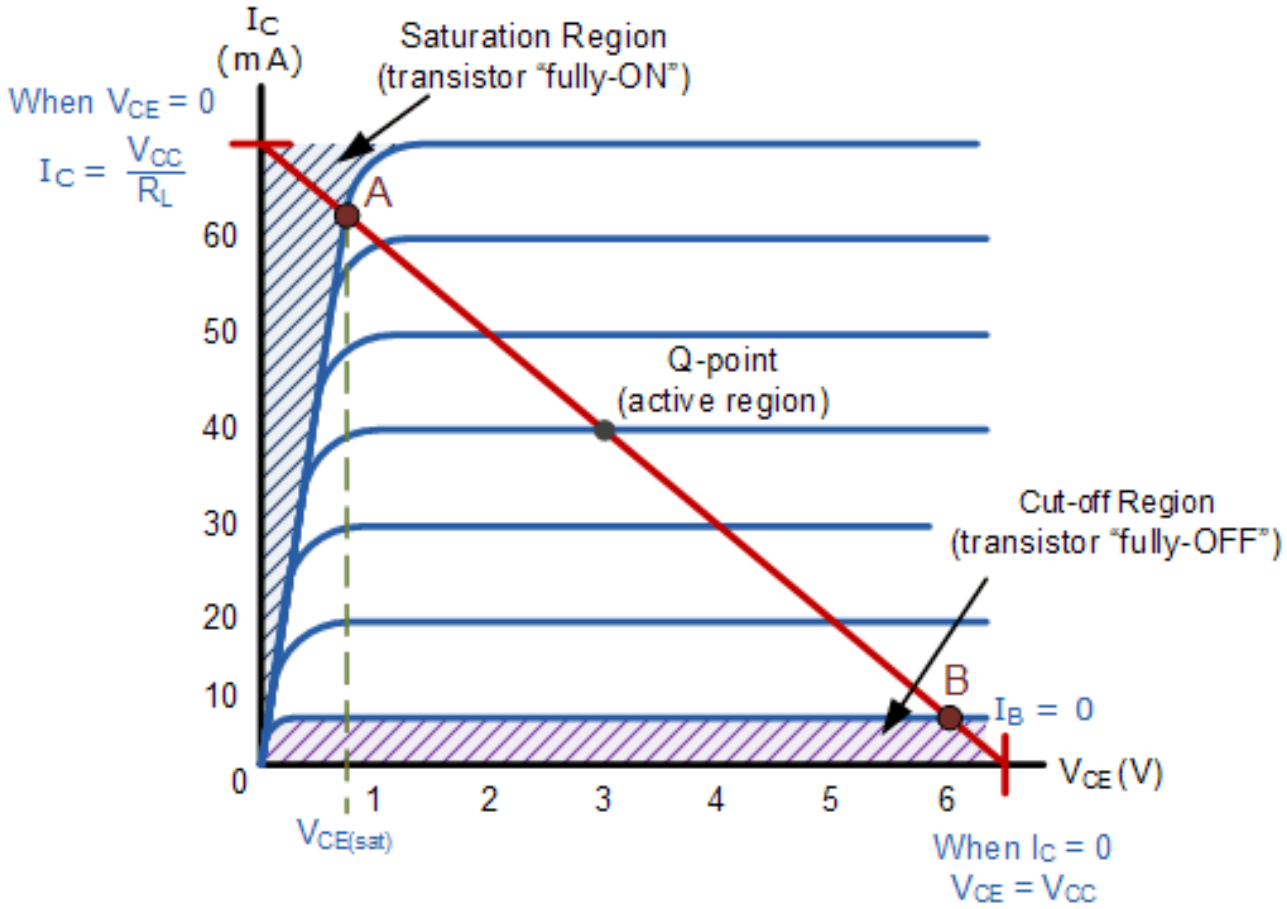


from Lecture 3: Transistor as a switch

Operate in either cut-off (OFF) or saturation (ON) regions



Example:
Bipolar junction transistor (BJT)

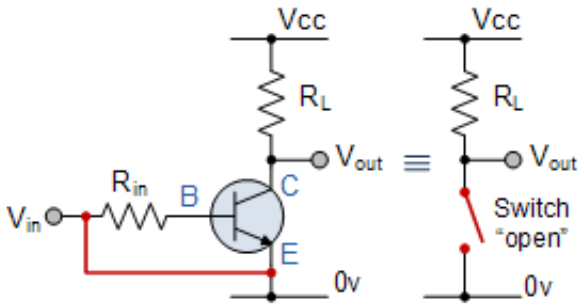


http://www.electronics-tutorials.ws/transistor/tran_4.html

from Lecture 3: Transistor as a switch

OFF

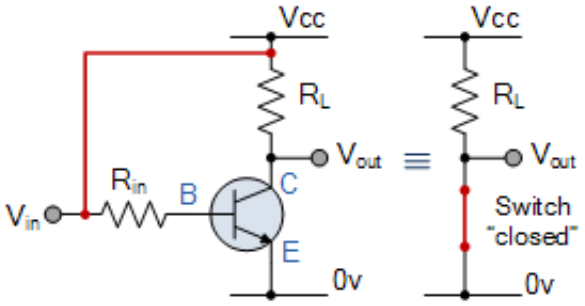
Cut-off Characteristics



- The input and Base are grounded (0v)
- Base-Emitter voltage $V_{BE} < 0.7v$
- Base-Emitter junction is reverse biased
- Base-Collector junction is reverse biased
- Transistor is "fully-OFF" (Cut-off region)
- No Collector current flows ($I_C = 0$)
- $V_{OUT} = V_{CE} = V_{CC} =$ "1"
- Transistor operates as an "open switch"

ON

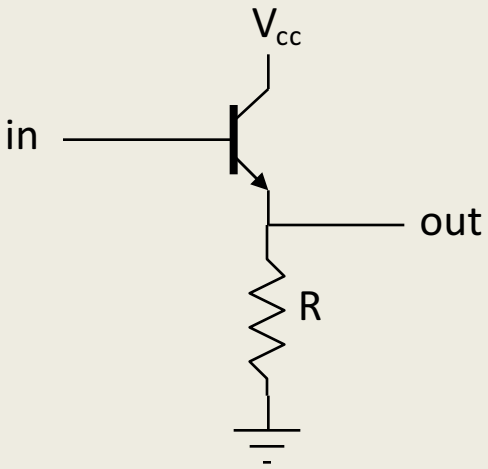
Saturation Characteristics



- The input and Base are connected to V_{CC}
- Base-Emitter voltage $V_{BE} > 0.7v$
- Base-Emitter junction is forward biased
- Base-Collector junction is forward biased
- Transistor is "fully-ON" (saturation region)
- Max Collector current flows ($I_C = Vcc/R_L$)
- $V_{CE} = 0$ (ideal saturation)
- $V_{OUT} = V_{CE} =$ "0"
- Transistor operates as a "closed switch"

http://www.electronics-tutorials.ws/transistor/tran_4.html

Transistor Buffer



- In the hookup above (emitter follower, or common collector), $V_{out} = V_{in} - 0.6$
  - sounds useless, right?
  - there is no voltage "gain," but there *is* current gain
  - Imagine we wiggle V_{in} by ΔV: V_{out} wiggles by the same ΔV
  - so the transistor current changes by $\Delta I_e = \Delta V/R$
  - but the base current changes $1/\beta$ times this (much less)
  - so the "wiggler" *thinks* the load is $\Delta V/\Delta I_b = \beta \cdot \Delta V/\Delta I_e = \beta R$
  - the load therefore is less formidable
- The "buffer" is a way to drive a load without the driver feeling the pain (as much): it's impedance isolation

Field-Effect Transistors

- The "standard" npn and pnp transistors use base-current to control the transistor current
- FETs use a field (voltage) to control current
- Result is no current flows into the control "gate"
- FETs are used almost exclusively as switches
  - pop a few volts on the control gate, and the effective resistance is nearly zero

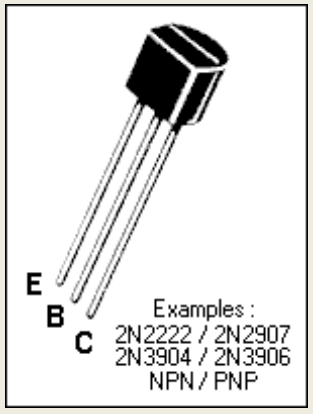
2N7000 FET

ON CHARACTERISTICS*						
$V_{GS(th)}$	Gate Threshold Voltage	$V_{DS} = V_{GS}$, $I_D = 1$ mA	0.8	2.1	3	V
$r_{DS(ON)}$	Static Drain-Source On-Resistance	$V_{GS} = 10V$, $I_D = 0.5A$		1.2	5	Ω
		$T_C = 125°C$		1.9	9	Ω
$V_{DS(ON)}$	Drain-Source On-Voltage	$V_{GS} = 10V$, $I_D = 0.5A$		0.6	2.5	V
		$V_{GS} = 4.5V$, $I_D = 75$ mA		0.14	0.4	V
$I_{D(ON)}$	On-State Drain Current	$V_{GS} = 4.5V$, $V_{DS} = 10V$	75	600		mA
g_{FS}	Forward Transconductance	$V_{DS} = 10V$, $I_D = 200$ mA	100	320		ms

FET Generalities

FET

BJT

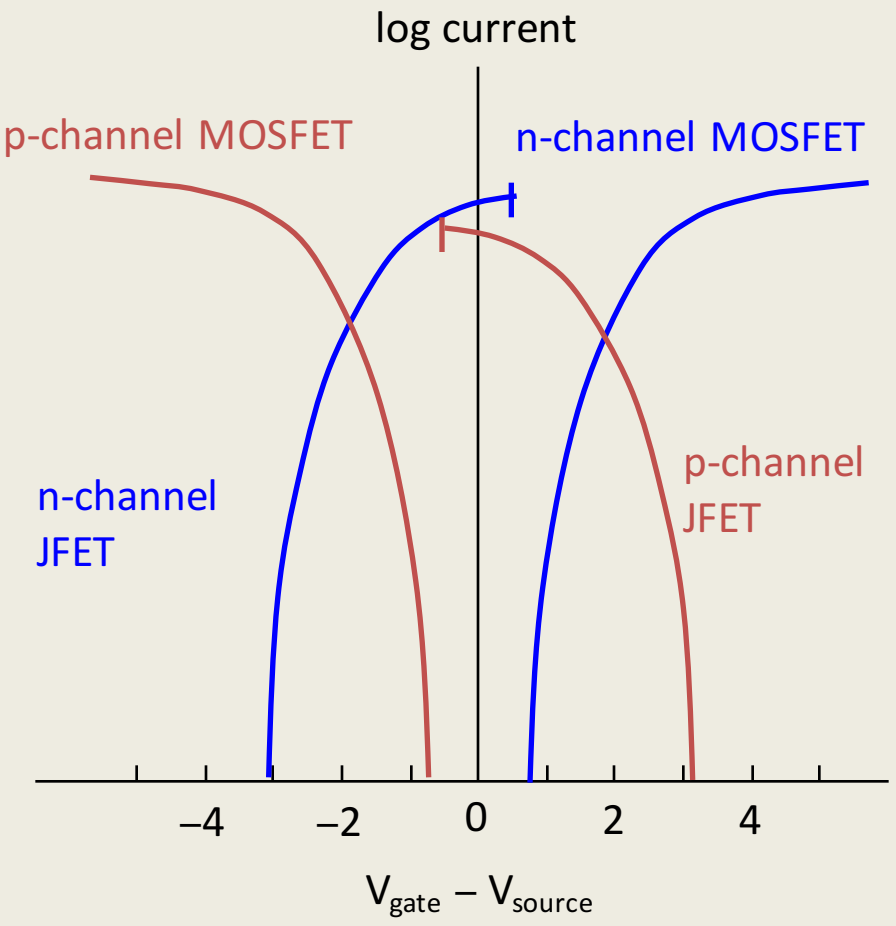


note pinout correspondence

- Every FET has at least three connections:
  - source (S)
    - akin to emitter (E) on BJT
  - drain (D)
    - akin to collector (C) on BJT
  - gate (G)
    - akin to base (B) on BJT
- Some have a body connection too
  - though often tied to source

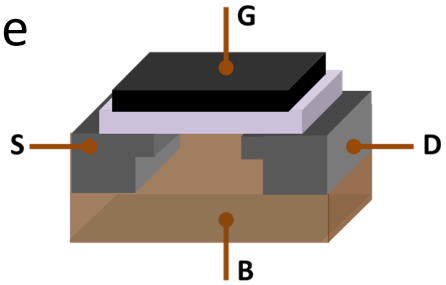
FET Types

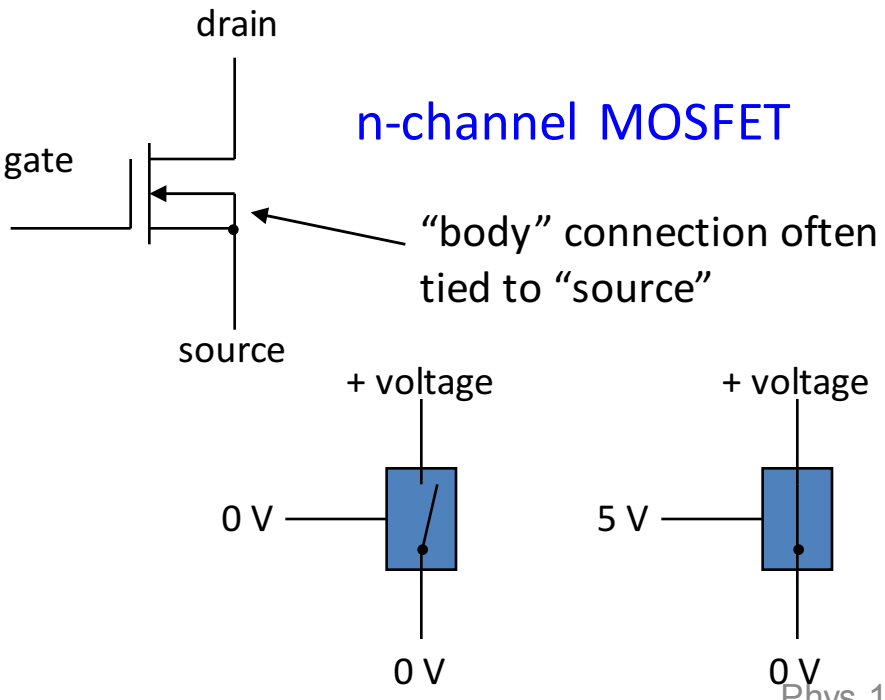
- Two flavors: n and p
- Two types: JFET, MOSFET
- MOSFETs more common
- JFETs conduct "by default"
  - when $V_{gate} = V_{source}$
- MOSFETs are "open" by default
  - must turn on deliberately
- JFETs have a p-n junction at the gate, so must not forward bias more than 0.6 V
- MOSFETs have total isolation: do what you want

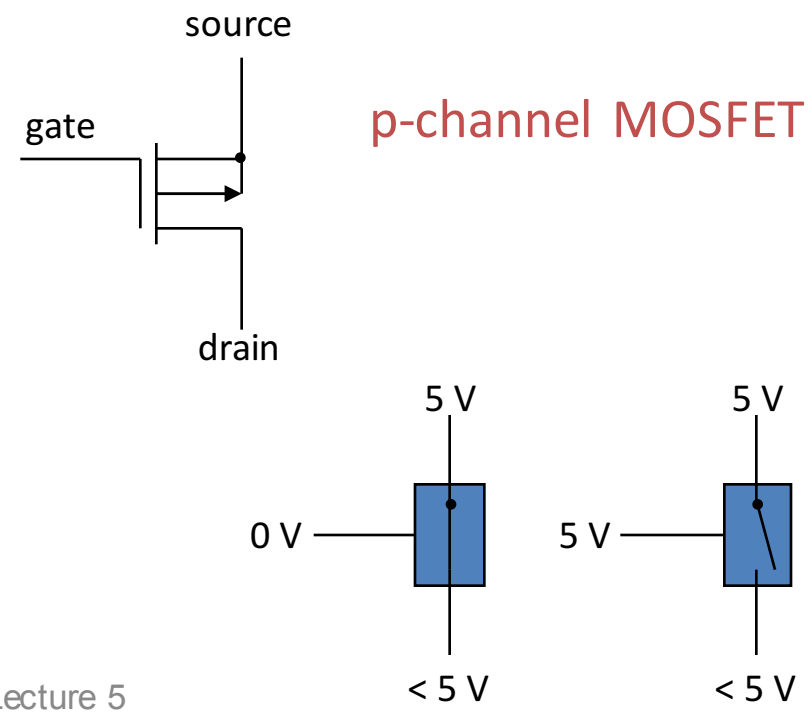


MOSFET Switches

- MOSFETs, as applied to logic designs, act as voltage-controlled switches
  - n-channel MOSFET is closed (conducts) when positive voltage (+5 V) is applied, open when zero voltage
  - p-channel MOSFET is open when positive voltage (+5 V) is applied, closed (conducts) when zero voltage
    - (MOSFET means metal-oxide semiconductor field effect transistor)

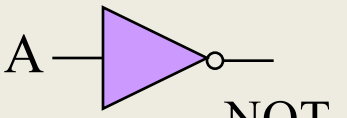






Data manipulation

- All data manipulation is based on *logic*
- Logic follows well defined rules, producing predictable digital output from certain input
- Examples:



NOT

A	C
0	1
1	0

AND

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

XOR

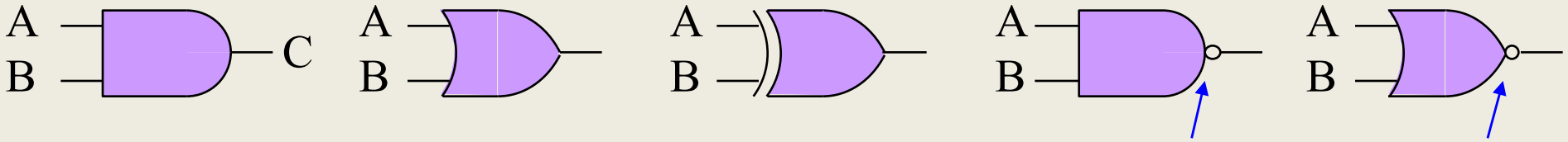
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

NAND

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

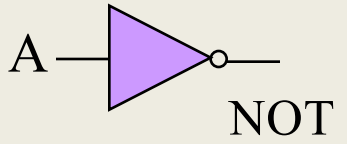
NOR

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

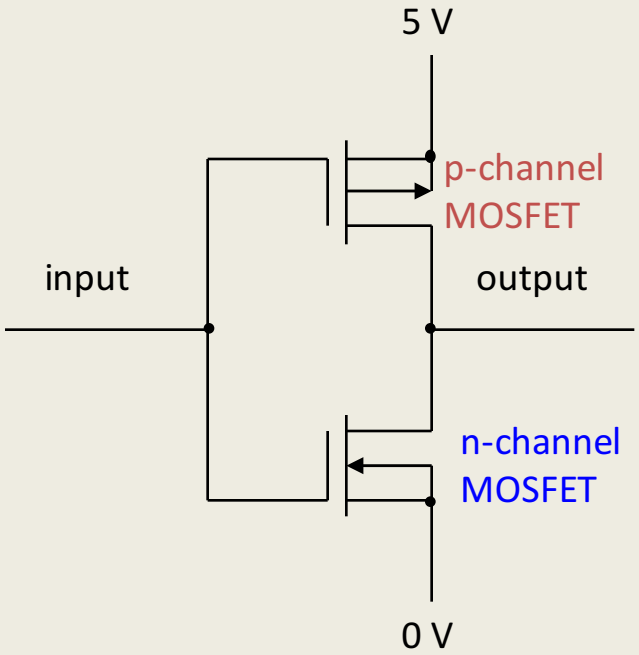


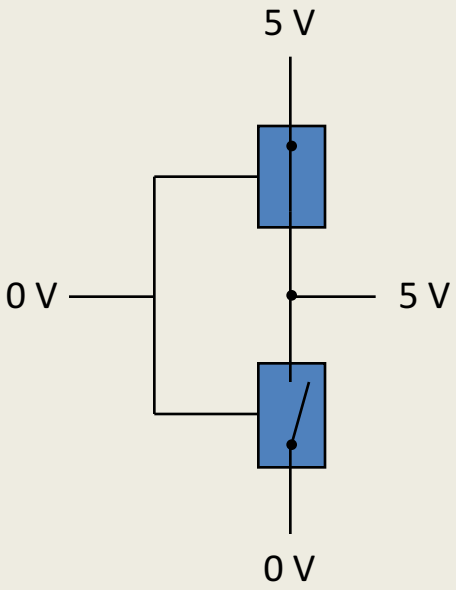
bubbles mean inverted (e.g., NOT AND → NAND)

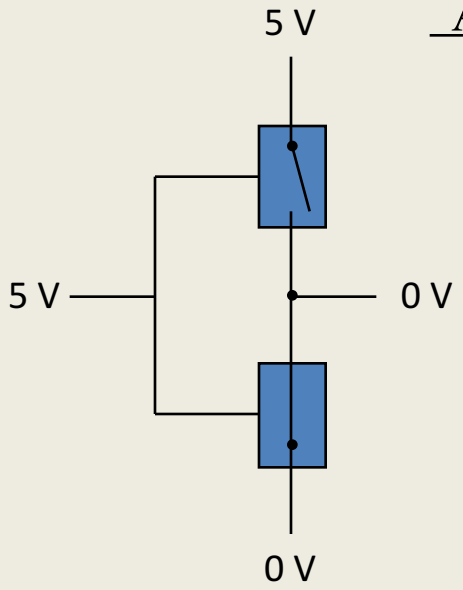
An inverter (NOT) from MOSFETS:



A	C
0	1
1	0

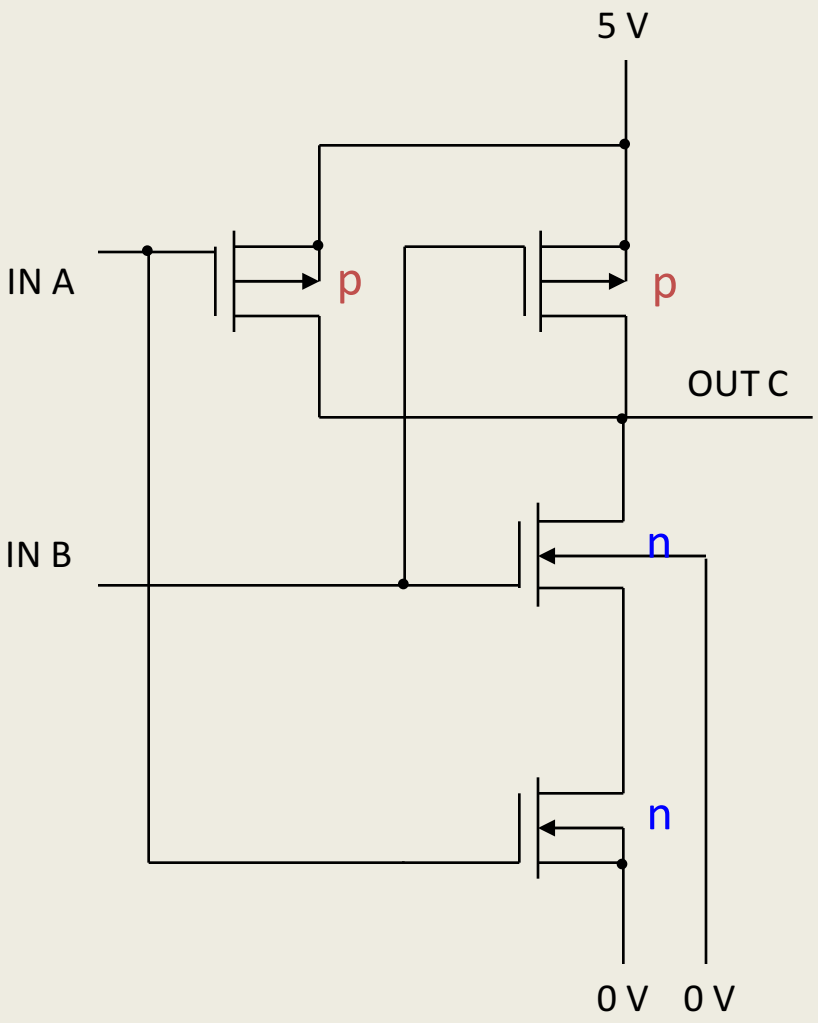






- 0 V input turns OFF lower (n-channel) FET, turns ON upper (p-channel), so output is connected to +5 V
- 5 V input turns ON lower (n-channel) FET, turns OFF upper (p-channel), so output is connected to 0 V
  - Net effect is logic inversion: 0 → 5; 5 → 0
- Complementary MOSFET pairs → CMOS

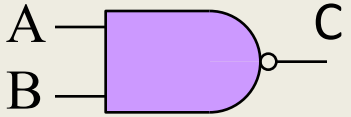
A NAND gate from scratch:



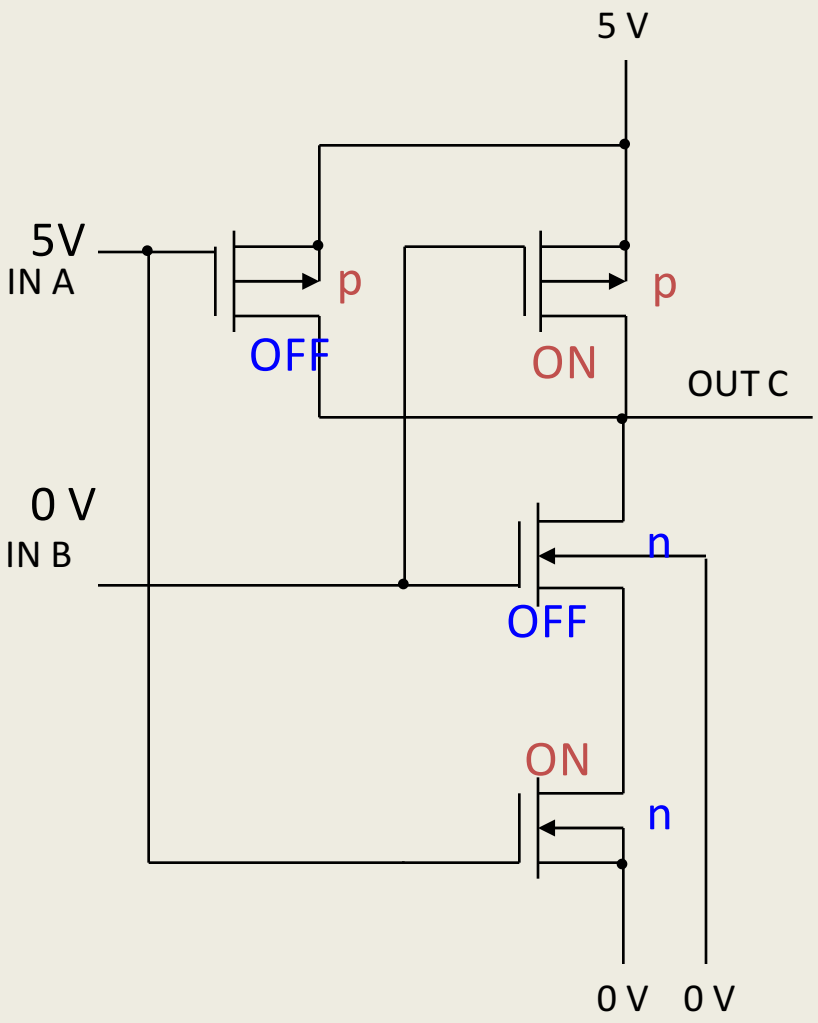
- Both inputs at zero:
  - lower two FETs OFF, upper two ON
  - result is output HI
- Both inputs at 5 V:
  - lower two FETs ON, upper two OFF
  - result is output LOW
- IN A at 5V, IN B at 0 V:
  - upper left OFF, lowest ON
  - upper right ON, middle OFF
  - result is output HI
- IN A at 0 V, IN B at 5 V:
  - opposite of previous entry
  - result is output HI

NAND

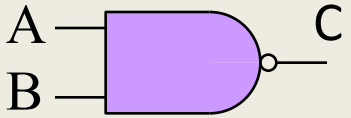
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0



A NAND gate from scratch:



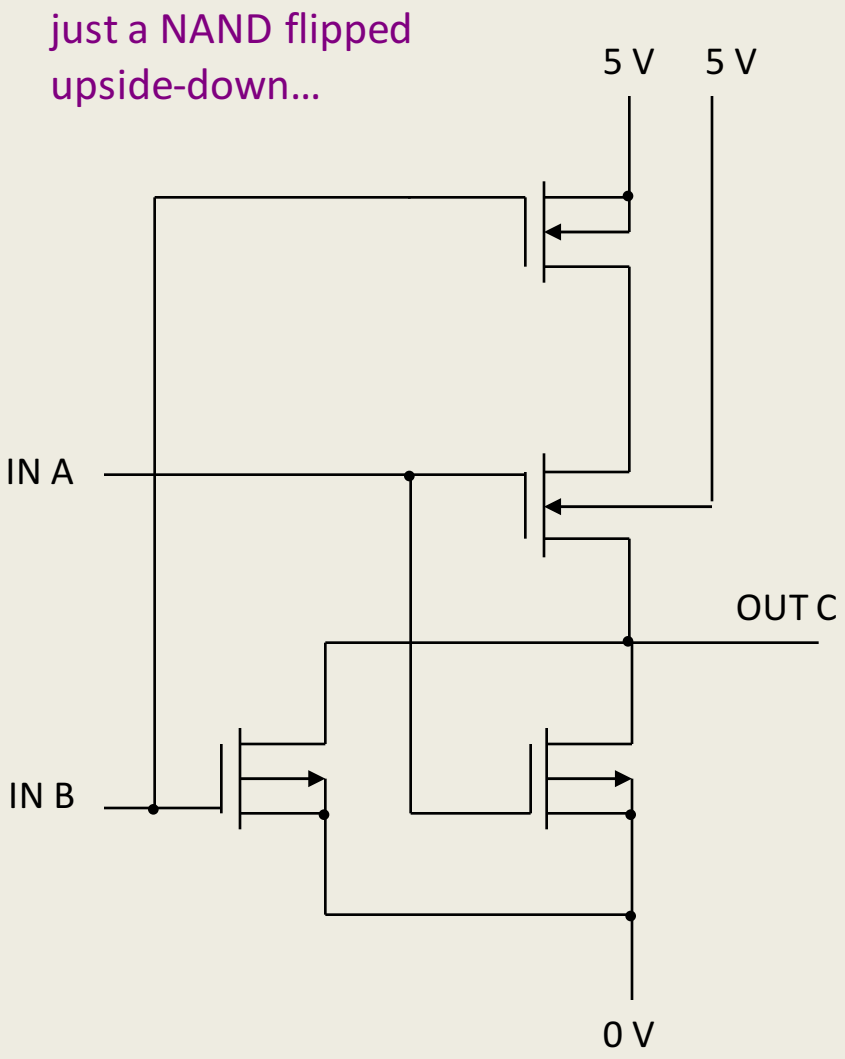
- Both inputs at zero:
  - lower two FETs OFF, upper two ON
  - result is output HI
- Both inputs at 5 V:
  - lower two FETs ON, upper two OFF
  - result is output LOW
- IN A at 5V, IN B at 0 V:
  - upper left OFF, lowest ON
  - upper right ON, middle OFF
  - result is output HI
- IN A at 0 V, IN B at 5 V:
  - opposite of previous entry
  - result is output HI



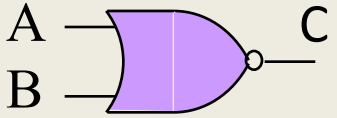
NAND

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

A NOR gate from scratch:



- Both inputs at zero:
  - lower two FETs OFF, upper two ON
  - result is output HI
- Both inputs at 5 V:
  - lower two FETs ON, upper two OFF
  - result is output LOW
- IN A at 5V, IN B at 0 V:
  - lower left OFF, lower right ON
  - upper ON, middle OFF
  - result is output LOW
- IN A at 0 V, IN B at 5 V:
  - opposite of previous entry
  - result is output LOW



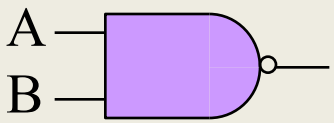
NOR

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

All Logic from NANDs Alone

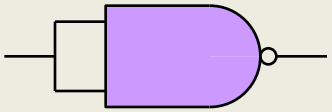
NAND

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0



NOT

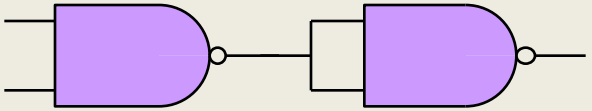
A	C
0	1
1	0

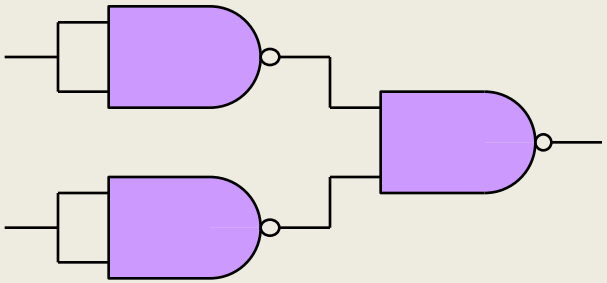


AND

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

invert output (invert NAND)





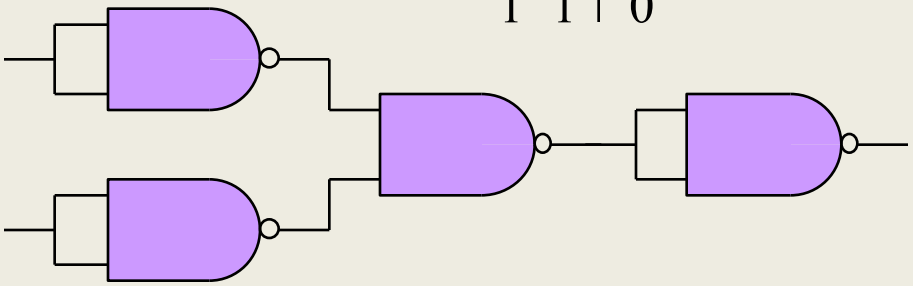
invert both inputs

OR

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

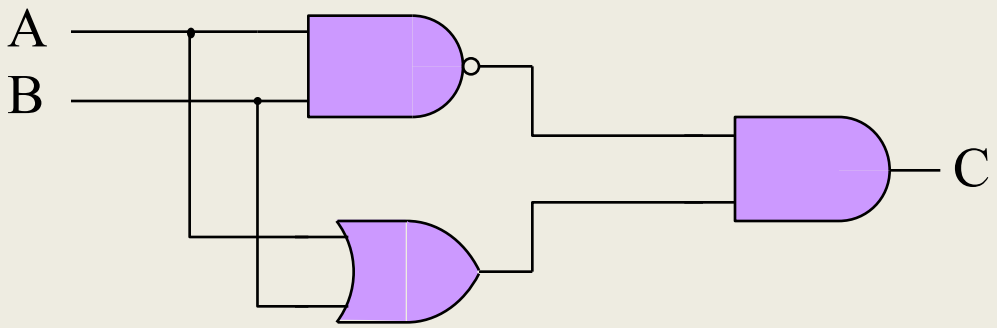
NOR

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0



invert inputs *and* output (invert OR)

One last type: XOR



- XOR = (A NAND B) AND (A OR B)
- And this you already know you can make from composite NAND gates (though requiring 6 total)
- Then, obviously, XNOR is the inverse of XOR
  - so just stick an inverter on the output of XOR

Rule the World

- Now you know how to build ALL logic gates out of n-channel and p-channel MOSFETs
  - because you can build a NAND from 4 MOSFETs
  - and all gates from NANDs
- That means you can build computers

- So now you can rule the world!

Arithmetic Example

- Let's add two binary numbers:

  00101110 = 0x2e = 46

  \+ 01001101 = 0x4d = 77

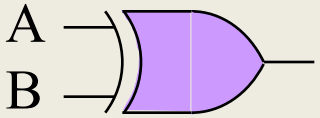
  01111011 = 0x7b = 123

- How did we do this? We have rules:

  0 + 0 = 0; 0 + 1 = 1 + 0 = 1; 1 + 1 = 10 (2): (0, carry 1);

  1 + 1 + (carried 1) = 11 (3): (1, carry 1)

- Rules can be represented by gates
  - If two input digits are A & B, output digit looks like XOR operation (but need to account for carry operation)

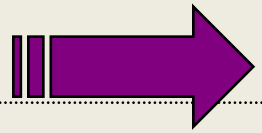


XOR

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

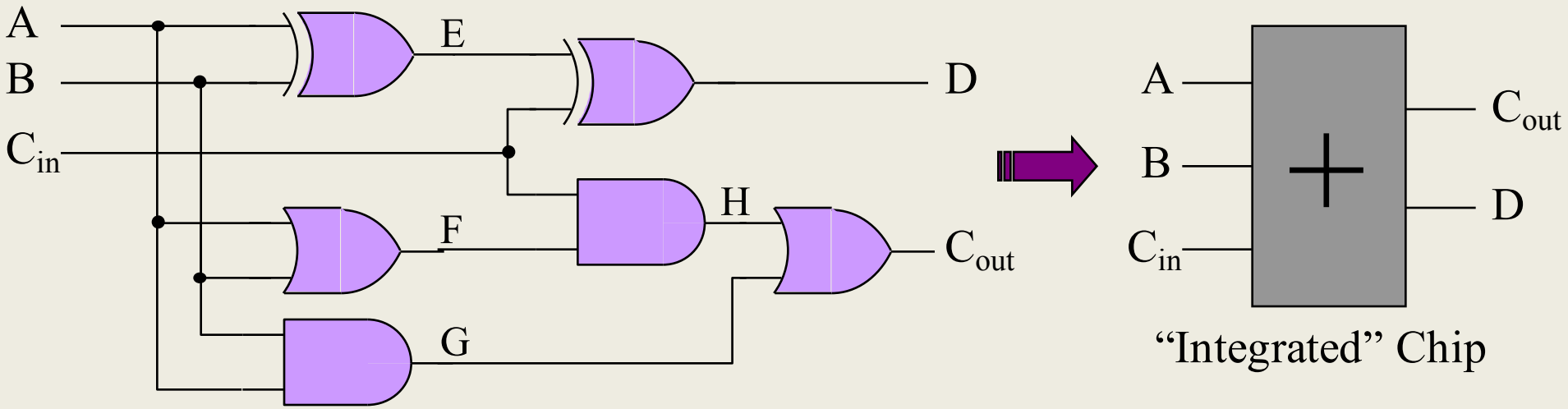
Can make rule table:

C_{in}	A	B	D	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



- Digits A & B are added, possibly accompanied by carry instruction from previous stage
- Output is new digit, D, along with carry value
  - D looks like XOR of A & B when C_{in} is 0
  - D looks like XNOR of A & B when C_{in} is 1
  - C_{out} is 1 if two or more of A, B, C_{in} are 1

Binary Arithmetic in Gates



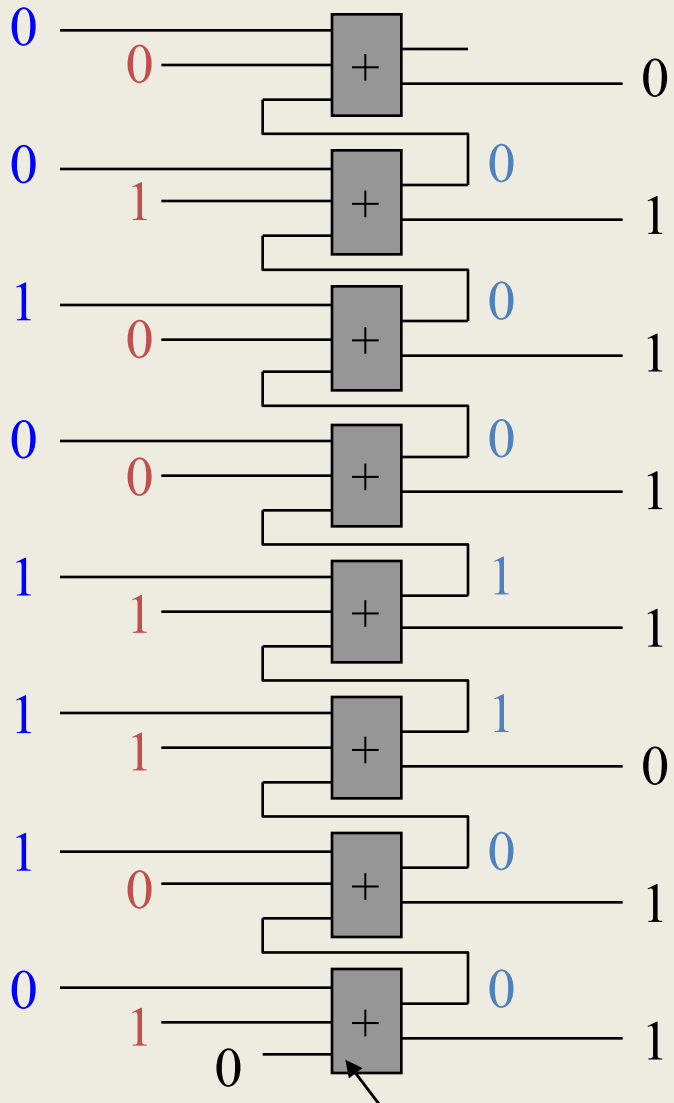
Input			Intermediate				Output	
A	B	C_{in}	E	F	H	G	D	C_{out}
0	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0
1	0	0	1	1	0	0	1	0
1	1	0	0	1	0	1	0	1
0	0	1	0	0	0	0	1	0
0	1	1	1	1	1	0	0	1
1	0	1	1	1	1	0	0	1
1	1	1	0	1	1	1	1	1

Each digit requires 6 gates

Each gate has ~6 transistors

~36 transistors per digit

8-bit binary arithmetic (cascaded)



MSB

LSB = Least Significant Bit

```
   1 1
  00101110 =  46
+ 01001101 =  77
  01111011 = 123
```

Carry-out tied to carry-in of next digit.

"Magically" adds two binary numbers

Up to ~300 transistors for this basic function. Also need –, ×, /, & lots more.

Integrated one-digit binary arithmetic unit (prev. slide)

Computer technology built up from pieces

- The foregoing example illustrates the way in which computer technology is built
  - start with little pieces (transistors acting as switches)
  - *combine* pieces into functional blocks (gates)
  - *combine* these blocks into higher-level function (e.g., addition)
  - *combine* these new blocks into cascade (e.g., 8-bit addition)
  - blocks get increasingly complex, more capable
- Nobody on earth understands every nit of modern CPU
  - Grab previously developed blocks and run
  - Let a computer design the gate arrangements (eyes closed!)

Bitwise logic operators in C

- Logical operators applied to integers or characters get applied bit-wise
  - operators include & (and), | (or), ^ (xor), ~ (not)
  - don't confuse with conditional && (AND), || (OR), etc. (doubled-up)
- Examples:
  - `21 & 7` → 5: 00010101 & 00000111 → 00000101
  - `21 & 0xff` → 21: 00010101 & 11111111 → 00010101
  - `21 & 0` → 0: 00010101 & 00000000 → 00000000
  - `21 | 7` → 23: 00010101 | 00000111 → 00010111
  - `21 ^ 7` → 18: 00010101 ^ 00000111 → 00010010
  - `~21` → 234: ~00010101 → 11101010
- Masking
  - `234 &= 0x1f` → 11101010 & 00011111 → 00001010 = 0x0a
- Bit shifting with >> or << operators
  - `01101011 >> 2` → 00011010 (effectively divide by 4)
  - `01101011 << 1` → 11010110 (effectively multiply by 2)