Prerequisites Students should feel comfortable using computers. A rudimentary knowledge of programming language concepts and electrical fundamentals (8.02) is assumed. Each student must have an Athena account to access the software used to complete the lab assignments.

Lectures *TR 1 - 2* in 32-123

Recitation Sections

#	Time	Room	Instructor
1	WF 10	26-322	Brad
2	WF 11	26-322	Brad
3	WF 11	34-303	Silvina
4	WF 12	34-303	Silvina
5	WF 12	34-304	Li-Shiuan
6	WF 1	34-304	Li-Shiuan
7	WF 1	34-303	Chris
8	WF 2	34-303	Chris
9	WF 2	36-155	David
10	WF 3	36-155	David

(Final assignments will be made after the first lecture)

Section assignments are made by the registrar; please plan to attend your assigned section.

If extraordinary circumstances make it impossible to attend the section you have been assigned, you may request a section reassignment by emailing 6004-headta@csail.mit.edu a request to be reassigned to a new recitation time.

Staff

Duties	Name	Email (@mit.edu)	Office	Phone
Lectures	Steve Ward	ward	32-G786	x3-6036
Head TA	Eben Kunz	ekunz@mit.edu	--	--
Admin Asst	Cree Bruins	cbruins@csail.mit.edu	32-G784A	x3-2629
Recitations	Chris Terman	cjt	32-G790	--x3-6038
	Li-Shiuan Peh	peh	--	--
	Silvina Hanono Wachman	silvina	--	--
	David Crowell	dcrowell	--	--
	Brad Gaffney	bgaffney	--	--

LAs	Becky Bianco	renminbi	--	--
	Daniel Gray	haiiro	--	--

The Head TA is in charge of most 6.004 operational issues, including section assignments. You can email the 6.004 Head TA at 6004-headta@csail.mit.edu.

Text There is no required text for the course this semester. Readings for some of the course material will be available on-line.

Handouts On-line versions of the handouts (in PDF format) can be found at this website.

Problem sets There are no weekly graded problem sets. Instead there are on-line tutorial problems with answers you can use to test your understanding of the material. The WF recitations give you a chance to work on these problems with the help of the course staff and to ask any questions that you may have.

Collaboration The assignments are intended to help you understand the material and should be done individually. You are welcome to get help from others but **the work you hand in must be your own**. Copying another person's work or allowing your work to be copied by others is a serious academic offense and will be treated as such. We do spot-check submissions to the on-line checkoff system for infractions of the collaboration policy. So please don't tempt fate by submitting someone else's work as your own; it will save us all a lot of grief.

Labs There are eight lab assignments due at various times during the term and an optional design project at the end of the term. Completing each part of a lab earns points that count toward your final grade. Points are determined during a short interview about each with a member of the course staff. Note that you can submit your work for a lab more than once, for example, as you complete each part. After completing the work on some of the labs, you'll be presented with some on-line lab questions to answer (these are different than the tutorial questions mentioned above). And you'll need to schedule a short lab checkoff meeting for each lab with a member of the course staff. This meeting can happen after the lab's due date but to receive full credit it must be completed within one week of the due date.

You must have a non-zero score for each required lab and all on-line lab questions must be checked-off as a prerequisite for passing the course. A missing required lab (i.e., a lab with a score of 0) will result in a failing grade; incompletes will *not* be given for unfinished laboratory work.

The lab gets crowded just before an assignment is due so plan accordingly. The lab will be staffed by the course staff during the late afternoon and evening M through R, and during the afternoon on F.

The 6.004 lab is located in 32-083 and is open 24 hours a day, 7 days a week. An access code is required for entry; it will be given out during lecture. The lab offers Linux-Athena workstations that can be used to complete the homework assignments. It is also possible to complete the assignments using

your own computer: the lab software is written in Java and runs in Sun's Java 2 Standard Edition (J2SE) environment (see Courseware for details). The lab serves as a meeting room for 6.004 students and staff; staff members keep their office hours in the lab, whose schedule is posted elsewhere on the 6.004 web site.

Late policy for labs: The on-line system will give you 50% of any points earned for submissions after the due date. So if your first submittal is late, you get 50% of the points. But if you submitted on-time for 15 points, and then late for 25 points, you'll get 20 points total for the lab. Note that points reported by JSim/BSim at check-in are for on-time submittals; you can check your on-line status page to see how many points count toward your total. This will be reported as "0" until you complete your checkoff meeting.

Late policy for checkoffs: The on-line system will allow only 50% of any points earned by your lab (including any late penalties you incurred) if you don't complete your checkoff before the checkoff deadline. So if you miss both the lab deadline and the checkoff deadline, you'll only get 25% of the total points.

Quizzes

There are 4 fifty-minute, closed-book quizzes. The questions will be similar (perhaps identical!) to the tutorial problems and will ask you to provide short, written answers and/or explanations. The quizzes are scheduled roughly every three weeks during recitation:

Quiz	Date given	Deadline for grade corrections
Quiz 1	Fri, 9/23	Wed, 10/5
Quiz 2	Fri, 10/14	Wed, 10/26
Quiz 3	Fri, 11/4	Wed, 11/16
Quiz 4	Fri, 12/2	Wed, 12/14

To ensure everyone has a seat, please attend your assigned section on quiz days. If exceptional circumstances make it impossible to take a quiz at your assigned time, please contact your recitation instructor *before* the quiz to see if other arrangements can be made. Requests for make-ups after the quiz has been given are unlikely to be successful.

There is no final exam.

Grading

The final grade is determined by performance on the quizzes (30 points/quiz, 120 points total), the labs and the design project (90 points total). In addition, **you must have a non-zero score for each of the required labs and all the on-line lab questions must be checked off as a prerequisite for passing the course**. A missing required lab will result in a failing grade; incompletes will

not be given for unfinished laboratory work.

Once your combined score has been computed as explained above, here's how grades will be assigned:

Grade	Requirement
A	175 ≤ total points
B	155 ≤ total points < 175
C	135 ≤ total points < 155
D	115 ≤ total points < 135
F	total points < 115, missing required lab

6.004 at a glance: Fall 2011

Tue	Wed	Thu	Fri
Registration Day	*NO CLASS*	**L1** Sep 08 Course overiew & mechanics. Basics of information.	**R1** Sep 09
L2 Sep 13 Digital abstraction, combinational logic, voltage-based encoding.	**R2** Sep 14	**L3** Sep 15 CMOS technology; gate design; timing	**R3** Sep 16
L4 Sep 20 Canonical forms; synthesis, simplification	*Holiday*	**L5** Sep 22 Sequential logic. **Lab 1 (CMOS) due**	**R4** Sep 23 **QUIZ 1**
L6 Sep 27 Storage elements, finite state machines.	**R5** Sep 28	**L7** Sep 29 Synchronization, metastability. **Lab 2 (Adder) due**	**R6** Sep 30
L8 Oct 04 Pipelining; throughput and latency.	**R7** Oct 05	**L9** Oct 06 Case study: multipliers. **Lab 3 (ALU) due**	**R8** Oct 07
Columbus Day	**R9** Oct 12	**L10** Oct 13 Models of computation, programmable architectures.	**R10** Oct 14 **QUIZ 2**
L11 Oct 18 Beta instruction set architecture, compilation.	**R11** Oct 19	**L12** Oct 20 Machine language programming issues. Lab 4 (TM) due	**R12** Oct 21
L13 Oct 25 Stacks and procedures.	**R13** Oct 26	**L14** Oct 27 Beta implementation. **Lab 5 (Assy Lang) due**	**R14** Oct 28
L15 Nov 01 Multilevel memories; locality, performance, caches	**R15** Nov 02	**L16** Nov 03 Cache design issues	**R16** Nov 04 **QUIZ 3**
L17 Nov 08 Virtual memory: mapping, protection, contexts	**R17** Nov 09	**L18** Nov 10 Virtual machines: timesharing, OS kernels, supervisor calls **Lab 6 (Beta) due**	*Veteran's Day*
L19 Nov 15 Devices and interrupt handlers, preemptive interrupts, real-time issues	**R18** Nov 16	**L20** Nov 17 Communication issues: busses, networks, protocols **Lab 7 (Trap Handler) due**	**R19** Nov 18
L21 Nov 22 Communicating processes: semaphores, synchronization, atomicity, deadlock	**R20** Nov 23	*Thanksgiving*	
L22 Nov 29 Pipelined Beta implementation, bypassing	**R21** Nov 30	**L23** Dec 01 Pipeline issues: delay slots, annulment, exceptions **Lab 8 (Tiny OS) due**	**R22** Dec 02
L24 Dec 06 Parallel processing, shared memory, cache coherence, consistency criteria	**R23** Dec 07	**L25** Dec 08 Wrapup Lecture!	**R24** Dec 09 **QUIZ 4**
No lecture	*No recitation* **PROJECT due**	*Finals - 6.004 is over!*	

generated by glancegen.py 8/30/2011 SAW

9/8

Welcome to 6.004!

I thought this course was called "Computation Structures"

Handouts: Lecture Slides, Calendar

Course Mechanics

Unlike other big courses, you'll have

NO evening quizzes

NO final exam

NO weekly graded problem sets

Instead, you'll face

Repository of tutorial problems (with answers)

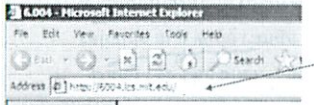
4 ~~FIVE~~ quizzes, based on these problems (in Friday sections)

EIGHT labs + on-line lab questions + Design Contest (extra credit)

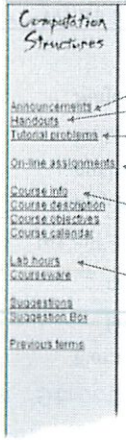
(all labs and olqs must be completed to pass!)

ALGORITHMIC assignment of your grade!

6.004: Course Clickables



http://6004.csail.mit.edu/



Announcements, corrections, etc.

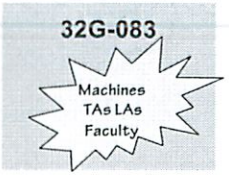
On-line copies of all handouts

Tutorial problems for each lecture

Current status, access to assignments

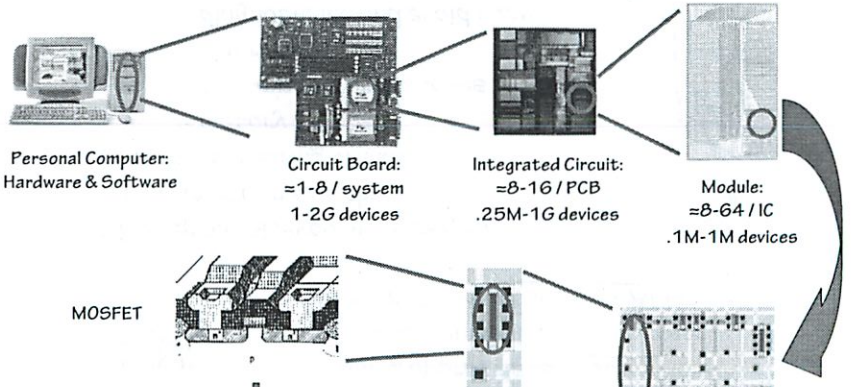
Course information, staff, policies, etc.

Where to find machines and help:



Combination Lock: 7755331

How do you build systems with >1G components?

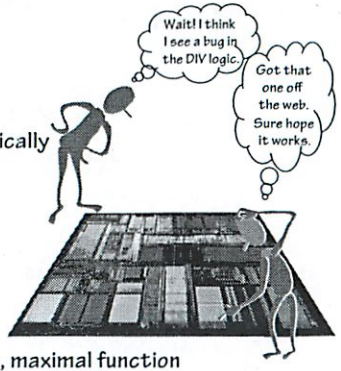


analog tech

9/8

What do we see?

- Structure
  - hierarchical design:
    - limited complexity at each level
    - reusable building blocks
- Interfaces
  - Key elements of system engineering; typically outlive the technologies they interface
  - Isolate technologies, allow evolution
  - Major abstraction mechanism
- What makes a good system design?
  - "Bang for the buck": minimal mechanism, maximal function
  - reliable in a wide range of environments
  - accommodates future technical improvements



robustness in real world

Our plan of attack...

- Understand how things work, bottom-up
- Encapsulate our understanding using appropriate abstractions
- Study organizational principles: abstractions, interfaces, APIs.

- Roll up our sleeves and design at each level of hierarchy
- Learn engineering tricks
  - history
  - systematic approaches
  - algorithms
  - diagnose, fix, and avoid bugs

First up: INFORMATION

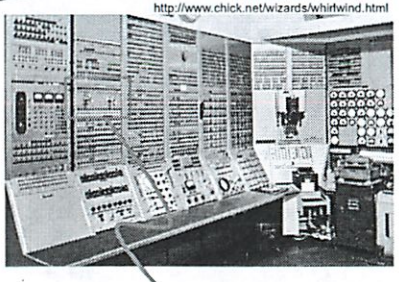
If we want to design devices to manipulate, communicate and store information then we need to quantify information so we can get a handle on the engineering issues. Goal:

good implementations

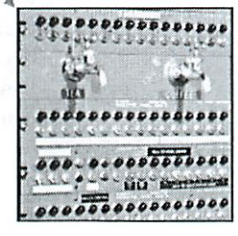
- Easy-to-use
- Efficient
- Reliable
- Secure
- ...

- Low-level physical representations
- High-level symbols and sequences of symbols

Whirlwind, MIT Lincoln Labs
http://www.chick.net/wizards/whirlwind.html



magnetic core memory



had tea+ coffee

What is "Information"?

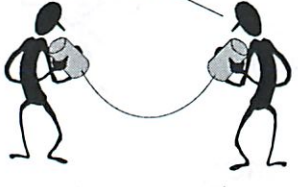
Claude Shannon formalized defn

information, *n.* Knowledge communicated or received *concerning a particular fact or circumstance.*

Pats WON their 18th game

Tell me something new...

... then lost the 19th.



Information resolves uncertainty.

- Information is simply that which cannot be predicted.

The less predictable a message is, the more information it conveys!

depends on how surprising info is to ya

Quantifying Information

(Claude Shannon, 1948)

Suppose you're faced with N equally probable choices, and I give you a fact that narrows it down to M choices. Then I've given you

Information is measured in bits (binary digits) = number of 0/1's required to encode choice(s)

$\log_2(N/M)$ *bits* of information

Examples:

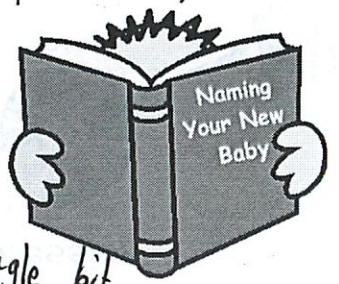
- information in one coin flip: $\log_2(2/1) = 1$ bit
- roll of 2 dice: $\log_2(36/1) = 5.2$ bits
- outcome of a Red Sox game: 1 bit

(well, actually, are both outcomes equally probable?)

(Why - still need to know who actually won)

Encoding

- Encoding describes the process of *assigning representations to information*
- Choosing an appropriate and efficient encoding is a real engineering challenge
- Impacts design at many levels
  - Mechanism (devices, # of components used)
  - Efficiency (bits used)
  - Reliability (noise)
  - Security (encryption)



Next lecture: encoding a bit.

What about longer messages?

need multiple games too

Fixed-length encodings

If all choices are equally likely (or we have no reason to expect otherwise), then a fixed-length code is often used. Such a code will use at least enough bits to represent the information content.

ex. Decimal digits 10 = {0,1,2,3,4,5,6,7,8,9}

4-bit BCD (binary coded decimal)

$\log_2(10) = 3.322 < 4$ bits

ex. ~86 English characters =

{A-Z (26), a-z (26), 0-9 (10), punctuation (11), math (9), financial (4)}

7-bit ASCII (American Standard Code for Information Interchange)

$\log_2(86) = 6.426 < 7$ bits

Encoding numbers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an n-bit number encoded in this fashion is given by the following formula:

$$v = \sum_{i=0}^{n-1} 2^i b_i$$

Octal shorter

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	1	1	1	1	0	1	0	0	0	0	$= 2000_{10}$

7 prefix 0

shorter still

0x7d0 prefix

Oftentimes we will find it convenient to cluster groups of bits together for a more compact notation. Two popular groupings are clusters of 3 bits and 4 bits.

03720

Octal - base 8
000 - 0
001 - 1
010 - 2
011 - 3
100 - 4
101 - 5
110 - 6
111 - 7

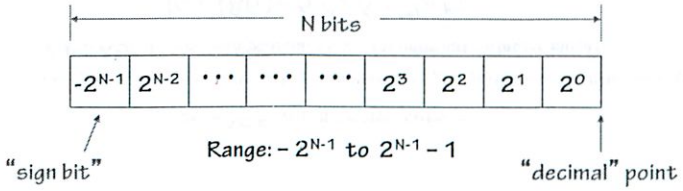
Hexadecimal - base 16	
0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - a
0011 - 3	1011 - b
0100 - 4	1100 - c
0101 - 5	1101 - d
0110 - 6	1110 - e
0111 - 7	1111 - f

3 groups of symbols

4 symbols

How to represent ⊖ #

Signed integers: 2's complement



8-bit 2's complement example:

$11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$

If we use a two's complement representation for signed integers, the same binary addition mod 2^n procedure will work for adding positive and negative numbers (don't need separate subtraction rules). The same procedure will also handle unsigned numbers!

By moving the implicit location of "decimal" point, we can represent fractions too:

$1101.0110 = -2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} = -8 + 4 + 1 + 0.25 + 0.125 = -2.625$

When choices aren't equally probable

When the choices have different probabilities (p_i), you get more information when learning of a unlikely choice than when learning of a likely choice

Information from $choice_i = \log_2(1/p_i)$ bits

Average information from a choice = $\sum p_i \log_2(1/p_i)$ ← best case

Example

$choice_i$	p_i	$\log_2(1/p_i)$
"A"	1/3	1.58 bits
"B"	1/2	1 bit
"C"	1/12	3.58 bits
"D"	1/12	3.58 bits

Average information
= (.333)(1.58) + (.5)(1)
+ (2)(.083)(3.58)
= 1.626 bits

Can we find an encoding where transmitting 1000 choices is close to 1626 bits on the average? Using two bits for each choice = 2000 bits

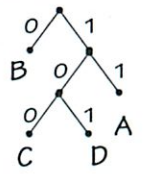
Variable-length encodings

(David Huffman, MIT 1951)

Use shorter bit sequences for high probability choices, longer sequences for less probable choices

$choice_i$	p_i	encoding
"A"	1/3	11
"B"	1/2	0
"C"	1/12	100
"D"	1/12	101

B C A B A D
010011011101



Huffman Decoding Tree

Average information
= (.333)(2)+(.5)(1)+(2)(.083)(3)
= 1.666 bits

Transmitting 1000 choices takes an average of 1666 bits... better but not optimal

To get a more efficient encoding (closer to information content) we need to encode sequences of choices, not just each choice individually. This is the approach taken by most file compression algorithms...

over the block - like zip

Data Compression

Key: *re-encoding to remove redundant information: match data rate to actual information content.*

40 MB

(mp3, divx, zip, ...)

4 MB

"Outside of a dog, a book is man's best friend. Inside of a dog, its too dark to read..."

-Groucho Marx

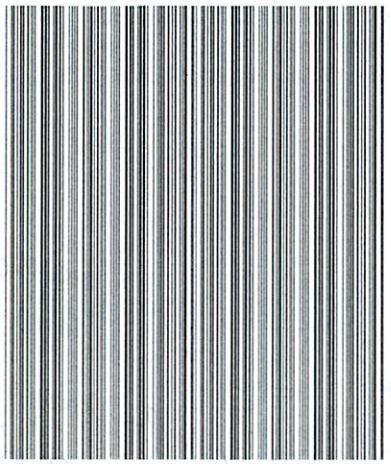
A84b!*m9@+M(p

Ideal: No redundant info – Only unpredictable bits transmitted. Result appears *random!*

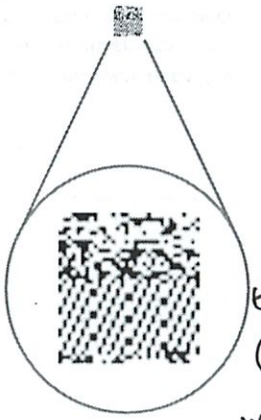
LOSSLESS: can 'uncompress', get back original.

"Able was I ere I saw Elba."*1024

Uncompressed: 27648 bytes



Compressed: 138 bytes



← I can compress more - still structure

Does *recompression* work?

If ZIP compression of a 40MB Bible yields a 4MB ZIP file, what happens if we compress *that*?

- Do we get a 400 KB file?

ZIP

- Can we compress *that*, to get a 40K file??

ZIP

- Can we compress the whole Bible down to a single bit???
- Is it a 0 or a 1????

ZIP

"?"

HINT: if the initial compression works perfectly, the result has *no* redundancy!

No!

Calmagoice Complexity
Size of smallest program to generate tree

Is redundancy *always* bad?

Simplifies tech

Encoding schemes that attempt to match the information content of a data stream are minimizing redundancy. They are *data compression* techniques.

"ground on which the human hand of man has never before set foot"

However, sometimes the goal of encoding information is to *increase redundancy*, rather than remove it. Why?

- Make the information easy to manipulate (fixed-sized encodings)
- Make the data stream resilient to noise (error detecting and correcting codes)

guard against noise corruption

Error detection and correction

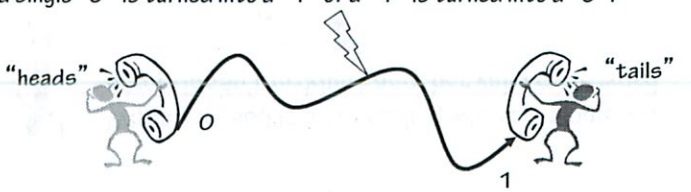
Suppose we wanted to reliably transmit the result of a single coin flip:

Heads: "0"

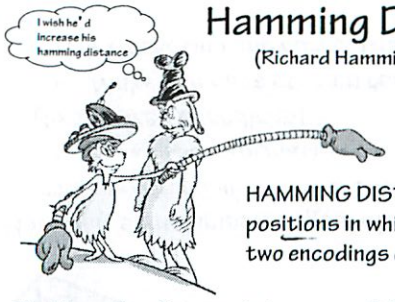
Tails: "1"

This is a prototype of the "bit" coin for the new information economy. Value = 12.5¢

Further suppose that during transmission a single-bit error occurs, i.e., a single "0" is turned into a "1" or a "1" is turned into a "0".



This is all identical to 6.02



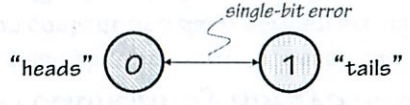
Hamming Distance

(Richard Hamming, 1950)

HAMMING DISTANCE: The number of digit positions in which the corresponding digits of two encodings of the same length are different

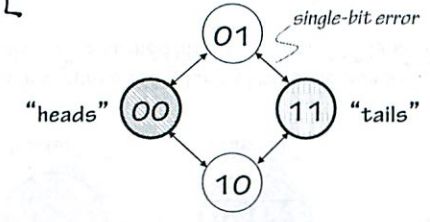
The Hamming distance between a valid binary code word and the same code word with single-bit error is 1.

The problem with our simple encoding is that the two valid code words ("0" and "1") also have a Hamming distance of 1. So a single-bit error changes a valid code word into another valid code word...



Error Detection

What we need is an encoding where a single-bit error doesn't produce another valid code word.



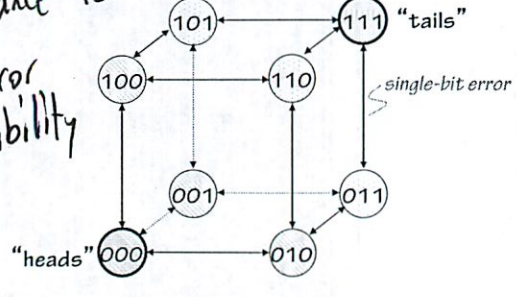
If D is the minimum Hamming distance between code words, we can detect up to (D-1)-bit errors

We can add single-bit error detection to any length code word by adding a *parity bit* chosen to guarantee the Hamming distance between any two valid code words is at least 2. In the diagram above, we're using "even parity" where the added bit is chosen to make the total number of 1's in the code word even.

Can we *correct* detected errors? Not yet...

increase Hamming distance to ↓ error possibility

Error Correction



If D is the minimum Hamming distance between code words, we can correct up to $\left\lfloor \frac{D-1}{2} \right\rfloor$ - bit errors

By increasing the Hamming distance between valid code words to 3, we guarantee that the sets of words produced by single-bit errors don't overlap. So if we detect an error, we can perform *error correction* since we can tell what the valid code was before the error happened.

- Can we safely detect double-bit errors while correcting 1-bit errors?
- Do we always need to triple the number of bits?

The right choice of codes can solve hard problems

Reed-Solomon (1960)

First construct a polynomial from the data symbols to be transmitted and then send an over-sampled plot of the polynomial instead of the original symbols themselves – spread the information out so it can be recovered from a subset of the transmitted symbols.

Particularly good at correcting bursts of erasures (symbols known to be incorrect)

Used by CD, DVD, DAT, satellite broadcasts, etc.

Viterbi (1967)

A dynamic programming algorithm for finding the most likely sequence of hidden states that result in a sequence of observed events, especially in the context of hidden Markov models.

Good choice when soft-decision information is available from the demodulator.

Used by QAM modulation schemes (eg, CDMA, GSM, cable modems), disk drive electronics (PRML)

Summary

- Information resolves uncertainty
- Choices equally probable:
  - N choices down to M⇒ $\log_2(N/M)$ bits of information
  - use fixed-length encodings
  - encoding numbers: 2's complement signed integers
- Choices not equally probable:
  - choice_i with probability p_i ⇒ $\log_2(1/p_i)$ bits of information
  - average number of bits = $\sum p_i \log_2(1/p_i)$
  - use variable-length encodings
- To detect D-bit errors: Hamming distance > D
- To correct D-bit errors: Hamming distance > 2D

Next time:

- encoding information electrically
- the digital abstraction
- combinational devices

Same as 6.02!

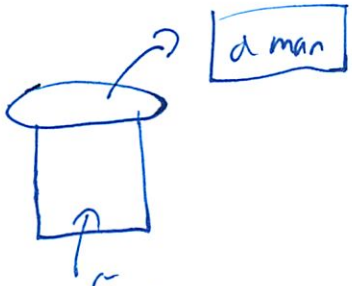
6.004 Recitation

9/8

Chris Terman

lots of options

4 fridays - quizzes on friday



How much info?

$\log_2\left(\frac{8}{5}\right)$ ← measure info/entropy in bits

8 ↓N

↑ can narrow to 5 M

$\log\left(\frac{1}{\text{Prob}}\right)$

Prob ↑ 5/8

k bits → 2^k different choices

4.1 bits → 5 bits if one

Can if have multiple, can use variable-length encoding

X = 8 bits

2^8 choices for value

= 256

(2)

Y | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Hamming distance = 1

So now narrow to 8 choices

The info is ~~base~~ $\log_2\left(\frac{1}{\frac{8}{256}}\right)$

$= \log_2(2^5)$

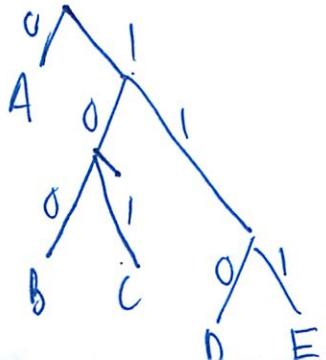
$= 5$ bits

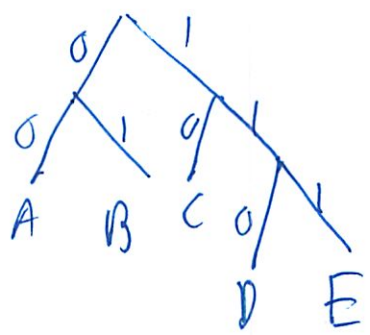
↑ abstract measure

Sometimes hard to rationalize

6.004 is his fav class

Huffman Code





(3)

Want to encode Bad Dad

Using 1st tree

100	0	110	110	0	110
B	A	D	D	A	D

Now decoding

0	100	0	111	etc
A	B	A	E	

Short sequence = high probability

Like $P(A) = .5$

$P(B) = P(C) = P(D) = P(E) = .125$

	Prob
A	.15
F	.4
I	.15
O	.15
U	.15

Build Tree

(4)

Build from bottom up

Then remake table

A I .3

E .4

~~O U .2~~

O .15

U .15

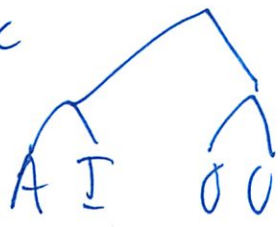
Etc

AI .3

E .4

OU .3

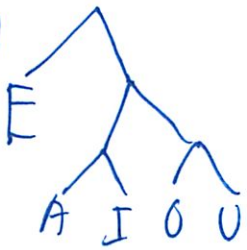
Etc



AIOU .6

E .6

Finish



Tree 1

(5)

Optimal $\log_2\left(\frac{1}{.15}\right) = 2.74$

So our code for 1 letter is worse than avg

Can code words to get better

What is avg lenght of 100 char message

$.4 \cdot 1 + .6 \cdot 3 =$

$\frac{40 + 180}{100} = 2.2$ bits/char

↑ Oh no / 100 since lenght of whole message

220 bits

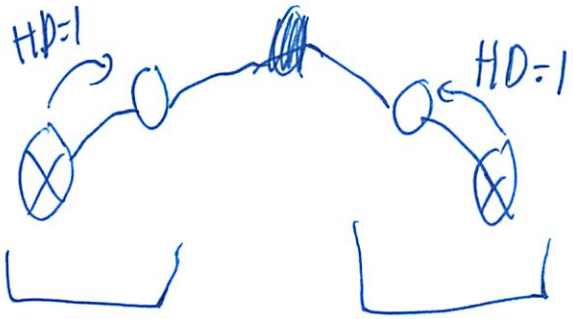
Fixed Length Code

	3 bit
A	000
E	010
I	011
O	100
U	101

6

Want to do single bit error correction?

- hamming distance must be 3



must be disjoint

Transmit everything twice?

- no still hamming distance = 2 in some cases

~~000~~

Could triple

Could use some codes

Hamming Correction Code

Did not review two's Complement arithic

9/9
9/10

Preface

6.004 Computational Structures

This text approaches the field of digital systems architecture with a particular bias, one kindled during years of observation of M.I.T. students and speculation about the factors that distinguish outstanding engineers from the merely very competent majority. We bring to the project a conviction that the distinction reflects attitudes and perspectives that are at most subtle by-products of the usual technical education: subliminal lessons occasionally administered to students by virtue of accidental discovery rather than deliberate pedagogy. The organization and approach of *Computation Structures* is geared to the explicit cultivation of these characteristics in every student and reader.

Scope and Mission

As computer systems become increasingly complex, their study inevitably relies upon more specific areas of specialization. We compartmentalize specific technologies—logic elements, processors, compilers, operating systems—as subdisciplines to be treated by separate courses, texts, and intellectual tools. Interfaces between the subdisciplines are powerful engineering abstractions that allow, for example, the designer of a processor to deal entirely in the domain of digital logic, naively exploiting the electronic circuitry in the abstraction below and providing interpretive services to software in the abstraction above.

The abstractions work sufficiently well that our curricula (and, for that matter, our job descriptions) accommodate specialists whose understanding of computer systems is cripplingly incomplete. We confer computer science degrees on theorists who may be helpless when programming is required, on hardware designers to whom the high-level structure of the system to which they contribute is mystery, and on programmers who lack a clear understanding of how or why their programs make real things happen. Often what passes for technical breadth takes the form of multiple specialties: Students exhibit pockets of local facility, unable to connect them into an effective understanding of the system as a whole. Such people can and do perform useful functions, but their potential for creativity is limited by their acceptance of the boundaries of their specialties. They shy from the challenge to venture beyond familiar approaches, to reshape problems, to develop new interfaces and revise existing ones; they accept the mysterious rather than unveiling it, oversensitized to their own limitations. They are, in effect, imprisoned by the abstractions they have been taught.

Outstanding students, in contrast, somehow develop a perspective that illuminates the interfaces between technologies, rather than the technologies themselves, as the

most important structural elements of their discipline. They view the interfaces with respect but not reverence; they examine both sides of an abstraction, treating it as an object of study rather than a boundary between the known and the intractably mystical. They have discovered the power of their own universality; typically they have built complete systems, often with meager resources, and made them work. They have the sophistication, and most importantly the self-assurance, to explore new abstractions and to appreciate the role and costs of existing ones.

Computation Structures is intended to produce renaissance engineers of this latter category. To this end, it deliberately telescopes much of computer science into a bottom-up progression that builds simple electronics into representative computer systems. The student is catapulted through one layer after another of implementation technology, an experience that accentuates technological interfaces and the structure induced by them. The breadth of coverage of the book reflects our compulsion to demystify its subject matter. We are unwilling to ask students to accept a technology on faith, or to postpone its penetration to some more advanced course. The spectrum of implementation technology is laid before them as a challenge to sample, to uncover, to discover their capacity to master its entirety.

The ultimate mission of *Computation Structures* is not so much to convey facts as to catalyze personal discoveries by each student: the ability to master a wide range of technical details, the self-assurance to dive through a technological boundary, the power to grab a wire-wrap gun or CAD tool or body of code and create new structures. The considerable volume of technical detail is present largely to serve this goal; it is not the subject of the book, but rather a tool by which we convey its more important lesson.

The Text

Computation Structures does not presume to replace the many excellent texts that focus on established subtopics such as logic design or operating systems; nor is it intended for an elite audience capable of digesting four semesters of course work in a single term. It covers topics that every engineer—certainly every computer scientist—should be aware of, in sufficient detail to establish a concrete connection between each topic and practical reality. In many cases that connection is provided by the MAYBE computer, a simple microarchitecture that we use as a running example throughout the text. The MAYBE is presented in sufficient detail to allow its actual construction and operation, an activity required of M.I.T. students; even in the absence of such experience, it provides a subject for close scrutiny and a context for concrete discussions and questions.

The text is introductory, presuming no formal background in digital systems; however, it generally assumes a level of technical sophistication consistent with that of an undergraduate engineering student. The text makes repeated connections to related technical areas. These should enrich the presentation for students with appropriate backgrounds, but are inessential to the sequel; for example, early chapters include a few circuit diagrams whose appreciation requires an Ohm's-law level of circuit sophistication. Programming ideas and constructs are introduced with a lack of fanfare that presumes some previous exposure to computers and programming on the part of the student. A subset of the C language, used for sample programs, is

presented in a terse appendix; we have found that students easily attain a level of C fluency that allows them to read and understand the examples.

The text does not make extensive use of real-world case studies, relying instead on the more coherent framework designed into our real but parochial example machines. Connections to the technical literature and to industrial practice are identified primarily in sections entitled Context; these provide pointers for deeper investigation of technical issues, links to adjacent fields, and occasional historical perspective.

Role at M.I.T.

Computation Structures is used at M.I.T. as the text for 6.004, a one-term, 15-hour-per-week sophomore "core" course required of all electrical engineering and computer science undergraduates. Three of the fifteen student hours are allocated to laboratory activities; the remainder are spent in classes and homework. Typical students have previously taken 6.001, a LISP-based programming course that uses Abelson and Sussman [1985] as a text, and 6.002, an introductory circuits course. The role of 6.002 is primarily to prepare the student for the 6.004 laboratory component, in which (for example) familiarity with an oscilloscope is assumed. 6.001 provides a first exposure to many ideas that recur in 6.004—programs, stacks, memory locations—albeit in a quite different context and often following a different idiom.

6.004 is a fast-moving course, stretching to accommodate the subject in its tightly packed single-term syllabus. Major topics are dealt with in about 25 lectures, which generally follow the order of the text and average one or two lectures per chapter. Substantial pruning is required to provide coherent if somewhat superficial coverage in lectures: While the major topics of the text are each dealt with in some depth, many of their ramifications and issues raised in the text are not addressed explicitly. Many of the optional (starred) sections are omitted, although the mix varies somewhat from one term to the next.

The big picture emerging from the lectures is embellished by smaller section meetings held twice weekly, by homework, and by laboratory assignments. These components provide a vehicle for sampling underlying technical detail, stimulating each student to relate the lecture topic with at least one example of practical reality. The cultivation of this connection is a key element of the course and this text. Rather than presenting a stack frame in abstract terms, for example, we encourage each student to come to grips with the entire set of nested implementation technologies that make it real—machine language, microcode, circuit diagrams, logic gates. Frequent probes to concrete reality reinforce the two major lessons of the course: first, the value of abstractions as a tool for structuring complex systems; and second, the capacity of the individual student to master *any* aspect of a system he or she cares to focus on.

Laboratory Work

These key lessons of *Computation Structures* are reinforced by the laboratory component of the course. Each student is required to construct, debug, and program a working MAYBE computer from simple components (at the level of ROMs and registers). Several machine architectures—stack and general-register machines—are implemented and programmed using a common microarchitecture.

Students' computers are constructed from reusable take-home kits. The kits have a selection of logic components as well as integral prototyping board, power supply, and very primitive input/output provisions. Workstations (with a variety of support programs) and oscilloscopes are available in the laboratory. Although the wiring and much of the debugging can be performed by students at home, final checkout and subsequent program development require the use of facilities in the lab.

The laboratory is structured as about eight assignments, each directed at a fairly tightly constrained set of goals. The directedness of the assignments differentiates the activity from more advanced "project" laboratories, which emphasize initiative and creativity; however, each assignment contains some nugget of creative challenge, such as the design of a machine instruction. A major creative outlet takes the form of an optional design contest, held at the end of each term, in which students are given a relatively free hand to modify their machines to improve performance on a benchmark program that is not revealed until the day of the contest. Each entrant is constrained to use only parts from the lab kit, although we allow n-student teams to combine n lab kits, normalizing the performance of such entries by a factor of n. (The best performance by this metric has come from two-kit entries.)

In order to make the student computer construction tractable and reasonably fail-safe, we have developed a set of printed-circuit modules (compatible with protoboard construction techniques) and a staged approach to construction whose early phases involve the use of a kernel subset of the machine to debug its incremental improvements. This methodology, involving the use of bootstrapping and scaffolding to develop complex systems, is among the major lessons of the course; it is one we have found no way to teach without hands-on experience.

The integration of a "complete" laboratory with a broad-spectrum architecture course makes an ambitious package, but one that produces students whose technical maturity and self-assurance is based on that ultimate educator: They have each built a computer and made it work. We enthusiastically recommend the approach.

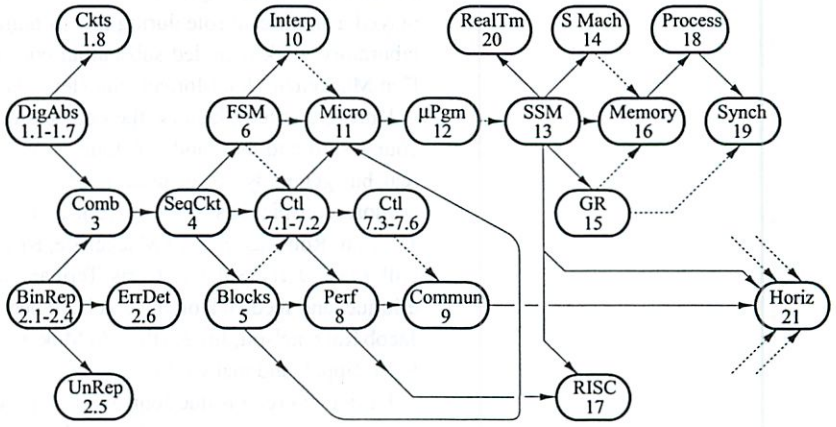
While we value the *omniware* breadth of the laboratory, much of its pedagogy can be captured by somewhat less extreme programs. The majority of the laboratory activity involves coding at some level, and can be carried out effectively using simulation software rather than hardware implementations. Several simulators and related system software (available both for UNIX workstations and for PCs) provide a convenient development test bed for microcode and software, even where the alternative of a hardware MAYBE machine exists. Since the architectural personality of our machine is largely dictated by the contents of a control ROM, the range of architectural options that can be explored using reprogramming and simulation techniques is relatively unconstrained. An effective laboratory can thus be fashioned using only commonly available computing resources.

In the absence of such resources, the complete implementations presented here can still be explored in a kind of ersatz laboratory whose medium is paper rather than machinery, that is, in assignments that probe, modify, and analyze the given structures. This approach was used at M.I.T. in years prior to the 6.004 laboratory; while the latter has clear advantages, a paper laboratory devoted to all levels of a single coherent example accomplishes many of the demystification and perspective-

building goals of our current course.

Alternative Paths through the Text

The subject of the text lends itself well to subsetting by suppression of detail, and we expect individual courses and readers to deemphasize or omit selected subtopics to suit their needs and backgrounds. The (widely varying) dependencies among topics are mapped crudely by the following graph:



Relatively inessential dependencies are shown as dotted edges. The level of detail shown in the diagram ignores certain subtleties, often bundling essential and optional issues together as single nodes. Some additional guidance to topic selection is provided by observing sections marked with an asterisk, indicating digressions that can be skipped without compromising subsequent (nonasterisked) coverage.

Topic selection and path through the text may be adjusted to accommodate a number of different course needs. A two-term undergraduate sequence based on the text would allow careful and relaxed treatment of its entire coverage and, given sufficient student time, is an attractive alternative to the single-term firehose-style course offered at M.I.T. An introductory graduate course might augment the text by selected readings from the literature; candidates appear in the Context sections and in the bibliography. Certain of the exercises appearing at the end of each chapter are marked with one or two asterisks to indicate that they may be challenging for an introductory undergraduate course. A single asterisk suggests that the marked problem requires more than routine application of the principles of the chapter; double asterisks identify problems that should challenge graduate students.

Acknowledgments

Computation Structures reflects the ideas, influence, and efforts of literally dozens of people who have contributed to 6.004 and its ancestors over the years. This lineage begins with the seminal course 6.032, which was developed in the late 1960s by Jack Dennis and subsequently evolved under the auspices of Jon Allen and Bob Gallager. The book owes its name and much of its sense of direction to these early roots; it

conveys, to a considerable extent, lessons that have been taught its authors by their predecessors.

The current course began to take shape during the spring of 1980, when the authors collaborated with Dave Reed in an effort to modernize 6.032 and its course notes. Subsequent years saw a succession of significant revisions to the course, each accompanied by a rewrite of the notes, exploring a variety of pedagogic formulae. The birth of the MAYBE in 1984 marks the point at which the book's current content and focus began to jell; the efforts of Dan Nussbaum and Chris Terman played a significant role during this formative period. Subsequent maturation of the laboratory work included substantial contributions to the MAYBE by Ziggy Blair, Ken Mackenzie, Joe Morgan, and Henry Houh.

During this long history, the course and its notes were scrutinized by dozens of course staff and thousands of students, whose contributions to the surviving text are real but generally so diffused as to be unidentifiable. Exceptions, whose impact is more conspicuous, include Tom Anderson, Andy Ayers, Dave Goddeau, Mark Johnson, Rob Kassel, Ken Mackenzie, Rishiyur Nikhil, Dan Nussbaum, Jon Powell, Gill Pratt, Jerry Saltzer, Chris Terman, and John Wolfe. Valuable discussions, critique, and feedback on the text have been provided by Anant Agarwal, Bill Dally, Jacob Katzenelson, Jim Kirtley, Al Mok, Bruce Musicus, Steve Seda, Gerry Sussman, Rich Zippel, and many others.

Particular credit is due John Wolfe for the massive effort he devoted to conversion of the source manuscript to TeX, to Eva Tervo and Sharon Thomas for the help they lent and the aggravation they sustained in preparing the manuscript, to Larry Cohen for his skilled and tireless editing, and to the Harris Corporation, whose HCX-9 was used to format the book with blazing speed. And to Don Knuth, that premier computer scientist whose contributions have been varied, prolific, and influential, we are indebted for TeX, which deserves simultaneous recognition as the text formatter of choice and the most idiosyncratic programming language known to us. It is impossible to account for the prenatal debt incurred by the book to the authors' wives, Debbie and Louise, whose loving encouragement nursed it through its long gestation.

The authors are grateful to M.I.T. for providing an environment that led to *Computation Structures*, to the Department of Electrical Engineering and Computer Science for support of the course development surrounding the text, and to Hewlett-Packard and Tektronix for their donations to the 6.004 laboratory. Finally, a decade of 6.004 and 6.032 students deserve sympathy and thanks for their role as subjects of a long succession of educational experiments as well as credit for the result. They, after all, are more than an influence; they are the reason.

1 The Digital Abstraction

The key to the orderly design of complex systems, digital or otherwise, lies in the decomposition of their function into modules whose behavior can be independently and concisely specified. To encourage this functional modularity and simplify the specifications for each module, we adopt a variety of engineering disciplines and agree to abide by the constraints they impose; in programming, for example, conventions for the representation of data and for passing parameters are typical self-imposed constraints. Each such discipline involves a set of primitive elements, each performing some prescribed basic function, together with rules for the construction of new elements by the composition of existing ones. The power of an engineering discipline derives from its simplification of the functional specifications of each module by the abstraction of essential function from nonessential detail; the description of a square-root procedure, for example, must confront the arithmetic relationship between its input and output but need not deal with the patterns of bits used to represent them.

We can observe two broad classes of activities in the structuring of complex systems. The first and more common of these involves working within a single discipline, enriching its repertoire by combining existing functions to define new ones. The appropriate software analogy is the construction, within the framework of a particular programming language, of a library of procedures to perform various useful functions. The second and more radical structuring activity involves the use of the modules of one discipline to define and support a new, higher-level abstraction with its own primitive elements and composition rules. In software design, such radical structuring typically involves the use of a lower-level language (together with its library of support procedures) to implement a higher-level language whose primitives and composition rules are in some (perhaps application-dependent) sense more powerful. The major characteristic of the new level is that it isolates its user from the underlying implementation technology, as opposed to simply adding new options to it.

Typically the structure of complex systems shows an alternation between these two activities. Functional extensions are made to some discipline until its complexity becomes unwieldy, at which point it is used to support a new abstraction with a manageable number of higher-level primitives and composition rules.

Such abstractions and the disciplines that support them are the central theme of computer science. Every significant digital system involves many levels of discipline, each implementing a more powerful set of primitive elements and composition rules

than the ones that underlie it. This chapter presents the fundamental abstraction that bridges the gap between the domain of the circuit designer (whose primitive elements are electronic components) and that of the computer architect.

1.1 Information and the Digital Abstraction

The primary use of electronic devices is the processing of information. Major appliances such as televisions and computers serve to translate information from one format to another—from an encoded radio frequency input to an output picture, or from a string of input symbols to a corresponding string of output symbols. Each such complex information processor is synthesized from components that transform, combine, store, and manipulate representations of information. Thus we can dissect a television and identify the information-processing functions of its major subsystems, specifying the way in which the input and output information of each is encoded into electronic signals. We might further scrutinize each subsystem, documenting the function performed by each component in information-handling terms, down to the level of fundamental circuit elements.

A capacitor, for example, is an energy storage device; in typical applications, its stored energy may be viewed as the electronic representation of stored information (such as elapsed time, in an RC timer circuit).[1] While the electronic representations of information in a television typically involve continuous variables (such as voltages or frequencies), the symbolic information processed by a computer consists of discrete units such as binary digits.

Digital systems, in general, are based on technology for the electronic representation of discrete information units; they offer the important advantage that the information content of each electronic signal is easy to quantify. They illustrate a powerful and important engineering principle: the pursuit of simplicity through constraint. Digital engineering involves a self-imposed design discipline that allows systems to be analyzed in the simple, abstract domain of logic rather than the vastly more complicated domain of underlying electronic principles. The advantages and mechanism of this abstraction are explored briefly in the following paragraphs.

The amount of information carried in a discrete-valued signal s may be defined as $\log_2 N_v$, where N_v is the number of distinct values of s that can be reliably set and measured. It is conventional to take the logarithm in base 2 and express the result in *bits*, or binary digits, of information. We might propose, for example, to communicate each decimal digit d by means of a d-volt signal on a particular wire, so that N_v is 10. If our measurement technology is accurate enough to distinguish reliably between the ten values, the information conveyed by each signal is $\log_2 10 = 3.322$ bits. If our measurement tells us only whether or not the signal is below 5 V, however, N_v is 2 and the amount of information conveyed is $\log_2 2$ or 1 bit.

A single bit is the minimum amount of information necessary to distinguish reliably between two values; it is the smallest convenient quantum for discrete information

[1] There are situations in which this view is inappropriate, such as applications in which a capacitor is used as a storage medium for energy to power other devices.

representations and serves as the basic unit of information storage and communication in most digital systems. It is noteworthy that 1 bit is exactly the information content of a single digit in a binary number system, which makes binary (rather than, say, decimal) an attractive and popular choice for the representation of numbers in digital systems.

Electronic parameters such as voltages are, to a good first approximation, continuous variables; we quantify them by real numbers rather than by integers. Each such parameter can in theory assume infinitely many distinct values, even over a bounded range; a voltage between 0 and 1 V may be 0.23 V or 0.6713 V or 0.6713456278 V. If we were able to set and measure such a voltage exactly, it would carry an infinite amount of information (since N_v would be infinite). Of course, noise and other physical realities limit our ability to constrain and measure physical parameters. A measurement of 0.23 V indicates that 0.23 V is a more likely value than, say, 0.3 V, which in turn is more likely than 0.4 V. The actual amount of information conveyed by our measurement is awkward to quantify exactly; it depends on the accuracy and reliability characteristics of the measuring device as well as electrical noise and other detriments to the validity of the signal itself. A typical signal level might carry 10 or 12 bits of useful information, on the average, with moderate reliability. Such representations are ideally suited to applications (such as television) involving large volumes of information flow and in which occasional errors are not catastrophic. The synthesis and analysis techniques used in these situations typically assume that each signal represents the corresponding real number to an acceptable accuracy, thus avoiding the need for precise quantification of its information content.

1.2 Representation of Discrete Variables

In many applications, it is convenient to use components whose behavior can be simply and precisely characterized in informational terms. A printer device connected to a computer can take as input the representation of a character to be printed, from an alphabet of, say, 128 possible characters. We would like the communication from the computer to the printer to be highly reliable: The printer must be able to determine precisely the character to be printed from the signals it receives. The amount of information conveyed to the printer is thus $\log_2 128 = 7$ bits for each character printed. The communication technique used will of course involve wires carrying continuously variable signals (such as voltages), but to reach our reliability goals we may be willing to sacrifice much of the information-carrying potential of each signal. A typical *parallel* printer interface, for example, involves seven independent wires, each carrying 1 bit of information encoded as a voltage. Each wire selects one of two alternative values (say, a 1 or 0 as the value of a corresponding binary digit), and the seven-wire aggregate thus identifies one of $2^7 = 128$ possible characters.

The reliable translation between a discrete variable such as a binary digit and its representation as the approximate value of a continuous variable such as a voltage provides the key to what we call the *digital abstraction*. It allows us to use conventional circuit elements to build a family of *digital devices* whose information-processing characteristics can be specified as a simple logical function involving discrete inputs

and outputs.

1.3 Combinational Devices

The simplest and most fundamental abstraction in the repertoire of the digital engineer is the *combinational device*, which we formalize as follows:

A combinational device is a circuit element having the following properties:

- one or more discrete-valued *input* terminals;
- one or more discrete-valued *output* terminals;
- a *functional specification*, detailing the value of each output for each of the possible combinations of input values; and
- a *timing specification*, consisting (at minimum) of an upper bound t_{pd} on the time required for the device to compute the specified output values from an arbitrary set of input values.

The usual interpretation of the *propagation delay* t_{pd} is that whenever a particular combination of input values is applied to the device and maintained for at least t_{pd} seconds, a corresponding set of output values will appear. Moreover, these output values will remain at the output terminals (at least) until the inputs change. Thus a set of input values applied for t seconds results in corresponding output values for a period of at least $t - t_{\mathrm{pd}}$ seconds. Note that t_{pd} is a *maximum* time required for new input values to be reflected at the output terminals; in general, the *minimum* such time is assumed to be zero.[2] One result of the latter assumption is that output values are contaminated immediately by any change at the inputs, regardless of t_{pd}.

An important feature of combinational devices is the simplicity with which they may be interconnected to synthesize new combinational functions. In particular, new combinational devices may be created by combining combinational elements in acyclic circuits, so long as care is taken not to connect outputs together.

More precisely, we can construct a combinational device by exploiting the basic rule of composition:

Combinational composition A circuit is a combinational device if it consists of interconnected circuit elements such that

- each circuit element is itself combinational,

---

[2] A refinement of this assumption, in which a nonzero *contamination delay* becomes part of the device specification, is occasionally used in the design of performance-critical circuitry.

- every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element, and
- the circuit contains no directed cycles (that is, every path through the circuit that traverses elements only in the input-to-output direction visits each circuit node at most once).

Any of the circuit nodes may be designated as outputs of the new device.

It is easy to verify that this construction yields devices that conform to our combinational criteria. The acyclic constraint allows us to proceed systematically from the circuit's input terminals through successive circuit elements, assigning to each node both a functional specification (as a function of the input values) and a time bound. The functional specification of each noninput node is derived from the specification of the element driving that node together with the functional specification assigned to that element's inputs; similarly, the propagation delay associated with such a node is computed by adding the element's t_{pd} to the maximum of the delays associated with its input nodes. In general, t_{pd} for the constructed device becomes the *maximum* cumulative propagation delay encountered on a path from any input to any output.

1.4 The Static Discipline: Logic Levels

The fact that we can synthesize arbitrarily complicated combinational devices using acyclic circuits of combinational elements gives us a bootstrapping technique for extending the usefulness of a primitive initial set of such devices; this is vaguely analogous to the induction step of an inductive proof. A remaining task is the development of a basis—an initial set of combinational elements from which we can synthesize more ambitious ones. Clearly we must begin with at least one combinational device that is defined in terms of technologies other than combinational devices. It is in the design of these basic elements that fundamental decisions regarding the representation of the discrete values of our abstraction are confronted.

Suppose, for example, that we are to represent a binary (or *logical*) variable d using a voltage v that can be varied between 0 and 5 V. We might choose to represent $d = 0$ by a voltage of 0 and $d = 1$ by 5 V. The efficacy of this representation then depends on our ability to set and measure v with sufficient reliability that the representation of $d = 0$ is never mistaken for the representation of $d = 1$ and vice versa, a reasonably easy demand to meet. We can afford to be fairly sloppy about setting and measuring v and still deduce the correct value of d. If we guarantee that v will be set to a value below 1 V when $d = 0$ and above 4 V when $d = 1$, and if we ensure that our measurement of v will be accurate to within 0.5 V, we shall always measure a voltage greater than 3.5 V when $d = 1$ and below 1.5 V when $d = 0$. We might distinguish between the two cases by comparing the measured voltage with a 2.5-V threshold voltage V_t and assuming a logical 1 as the value of d if the measured value is above the threshold and a logical 0 if it is below.

Figure 1.1 shows the mapping of the continuous variable v onto the discrete (binary digit) d. If the measured voltage is near or at the 2.5-V threshold, we will be unable

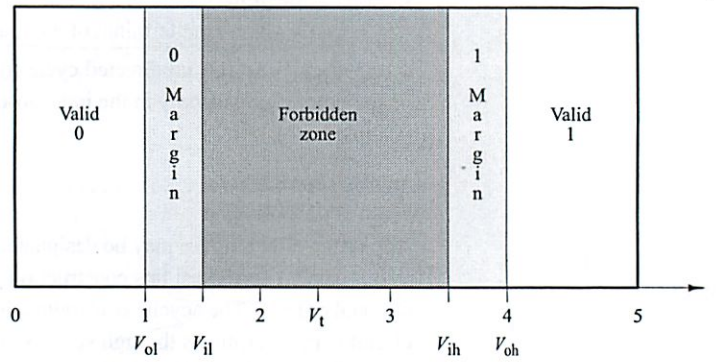


Figure 1.1 Mapping of logic levels to voltage.

to distinguish reliably between the two values of d. We avoid this embarrassing circumstance by outlawing it; we designate the region between 1.5 V and 3.5 V as a *forbidden zone* that does not correspond to the valid representation of a binary variable. Since we can reliably discriminate between voltages below 1.5 V and those above 3.5 V, adhering to the discipline that v must be set above 4 V or below 1 V ensures that forbidden values are avoided. In fact, we have allowed an extra half-volt margin for error on each side of the forbidden zone; these *noise margins* allow our scheme to function in the presence of a certain amount of noise or other unanticipated error.

In general, the reliable representation of discrete variables using continuously variable physical quantities requires such ranges of excluded values between valid representations. This gives rise to the possibility of *invalid signals* that correspond to none of the logical values; a wire carrying V_t represents neither a logical 1 nor a logical 0 by the convention outlined above. Such invalid signals are foreign to the digital abstraction and are unaccounted for in the analysis and synthesis methodologies commonly used by the digital engineer.

In order to protect digital designers from unanticipated invalid signals in their circuits, logic elements are commonly designed to conform to a *static discipline*:

how to ensure?
how to test?

A static discipline is a guarantee that, given logically valid inputs, every circuit element will (after an appropriate propagation delay) deliver logically valid outputs.

The interpretation of "logically valid" is relative to a particular representation convention, which in turn varies with the implementation technology. Typically, however, a single representation convention applies throughout an entire digital system. A static discipline offers a measure of assurance that invalid signals will not arise spontaneously in a well-designed combinational circuit: If the inputs to each device are

well behaved, each device's outputs will also be well behaved after some bounded delay.

Noise margins amount to imposing a more stringent validity requirement on output signals from a device than apply to its inputs, ensuring that marginally valid input signals are restored to unquestionable signals at the output. The threshold V_{il} represents the highest input voltage that must be interpreted as a logical 0; it is higher than V_{ol}, the highest output voltage that can be asserted to indicate a 0. This ensures that every valid output 0 will be interpreted as a 0 at connected inputs, even after some degradation. A similar relationship holds between V_{ih} (the lowest input representing a valid 1) and V_{oh} (the lowest output asserted to indicate a 1).

1.5 Rate of Information Flow

The representation scheme we have developed provides for the *static* transmission of a fixed amount of information—a single bit—as an electric signal over a wire. In general, we expect this value to change from time to time; if it *never* changes, as would occur if the value communicated is a constant, the need for the wire should be questioned! Thus, while at any instant a signal might convey only a single bit of information, it typically carries a *sequence* of bits over a period of time. If we make provisions for changing a signal to represent a new bit of information every second on the second, that signal may be viewed as carrying a *flow* of information at the rate of 1 bit/s; it could be used to transmit a 100-bit binary number in 100 s.

Any scheme we choose for communicating discrete variables places practical limits on the frequency with which we can change their values and thereby constrains the rate of information flow. Rates of information flow in digital systems are often specified in *baud*, or value transitions per second, a parameter that is constrained by the communication *bandwidth* of the underlying (analog) communication medium. A fundamental theorem by Nyquist places an upper bound on a communication channel's useful baud rate of twice its bandwidth, even in the absence of noise. In the case of binary representations, where each value is a binary digit, 1 baud is equivalent to 1 bit/s. Thus the theoretical maximum rate of information flow over a 3000-Hz voice-grade line carrying binary values as two different voltage levels is 6000 bits/s. This limit can be exceeded, in theory, by the use of more than two discrete values at a baud rate of less than 3000 changes per second; using four voltages, for example, raises the theoretical limit of our voice-grade line to 12,000 bits/s.

Of course, the effective rate of information flow can always be increased by using multiple signals; a 100-bit number can be transmitted in 1 s *serially* using a single 100-baud binary signal or *in parallel* by means of a hundred 1-baud binary signals, each encoding a single digit. Such choices are faced frequently by designers of digital systems, and a judgment typically depends strongly on underlying technological issues. For example, digital communication between geographically distant subsystems, such as terminals of an airline's flight reservation system, might rely on leased telephone lines whose bandwidth characteristics limit transmitted signals to audio frequencies. Reliable digital communication over such a line is limited to a few thousand baud, but increasing the communication rate by using multiple lines

is expensive. Since communication cost is likely to be a dominant factor in the design of such systems, it is worthwhile devoting considerable effort to minimizing the required communication rates.

Even in localized systems, communication costs are an important design consideration. As logic elements become cheaper and perform more complex functions, the cost of running wires (or other media) to interconnect them becomes an increasingly important element of system costs. Although nearby modules of a system can communicate at rates measured in hundreds of megabaud by using many parallel wires, such high bandwidth is expensive in device terminals and interconnections and is not to be squandered. Even within a single integrated-circuit chip, where the economies are based largely on space (chip area), interconnection costs often dominate: More of the chip area is devoted to lines transmitting signals from one logic element to another than to the logic elements themselves.

1.6 Transitions and Validity

In a dynamic system, where logical variables change in value from time to time, we cannot in practice avoid brief excursions through the forbidden zone. Because of stray capacitance and other such physical constraints, the voltage representing a logical variable v cannot be changed instantaneously from one valid logic representation to the other, although the transition may be quite fast—a few nanoseconds is typical.

We therefore enforce on ourselves an additional discipline: We avoid asking whether v represents a logical 1 or 0 at about the time when it may be making a transition between values. This in turn requires that the beholders of v—those devices to which v is an input—each have available information about when v might change. Most commonly this requirement is met by constructing *synchronous systems*, in which logic values are constrained to make transitions only at prescribed times (for example, every microsecond on the microsecond) keyed to a globally available clock signal. For truly asynchronous systems, it is impossible to guarantee that no logic level will be sampled at an inauspicious time (for example, during a transition); however, such events can be made quite improbable in practice. This general topic is visited again in section .

1.7 Logic Families

Some of the parameters in figure 1.1 seem arbitrary. The association of low voltage with logical 0 and high voltage with logical 1 is a convention termed *positive logic*; the dual convention, *negative logic*, represents logical 0 by the higher of the two voltages. This choice may be made arbitrarily to suit one's taste; a digital system can be analyzed in terms of positive logic, negative logic, or a mixture of the two.

Other parameters, such as the choice of logically valid voltage ranges and threshold voltages, typically reflect a combination of design goals and characteristics of the implementation technology. Several distinct "families" of digital devices have been developed over the years, each with its own set of characteristics and parameters for mapping continuous electronic variables (usually voltage) onto logic (nearly always

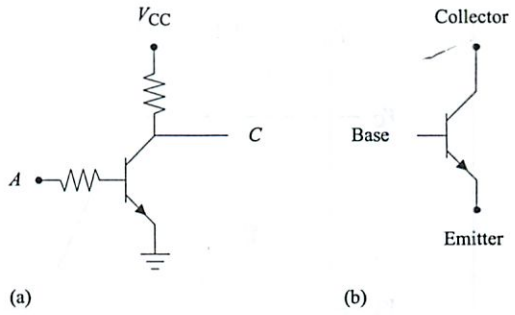


Figure 1.2 Resistor-transistor logic: (a) RTL inverter; (b) NPN transistor.

binary) values. Each such family includes a set of rules for the valid representation of logical values; each device in the family has the property that, so long as its inputs obey the rules, its outputs will also. They provide in each case a set of Tinkertoy-like modules that can be simply plugged together to build arbitrarily complex logical systems, with relative disregard for the underlying electronics. The resulting simplicity is an enormous advantage to the designer, who can deal with the concise logical characterization of devices whose specification in electrical terms would be nearly intractable.

1.8 Digital-Circuit Implementation

Figure 1.2(a) depicts a simple logic device from an early implementation technology called *resistor-transistor logic*, or RTL. The device is an *inverter*; its logical output at C is 1 if its input (at A) is 0, and 0 if its input is 1. The transistor in the circuit of figure 1.2(b) may be viewed roughly as a switch controlled by its base (input) current, which in turn depends on the voltage at A. If the voltage at input A is high (representing, using positive logic, a logical 1 input), then the switch is closed (the collector and emitter are short-circuited) and the output C is effectively connected to ground (yielding a logical 0 output). If the input voltage is close to 0 (logical 0 input), the switch is open, leaving C connected to the supply voltage V_{CC} through a small resistor; this results in a logical 1 output.

1.8.1 Logic Levels and Noise

A slightly more sophisticated model of the transistor yields the voltage-transfer characteristics shown in figure 1.3 for our simple inverter.

Over a portion of its input-voltage range, the device behaves like an amplifier with a negative gain (proportional to the slope of the center segment of the plot). This is termed the *active* or *linear* region of operation. When the input voltage is sufficiently low, the transistor is said to be at *cutoff* and effectively presents an open circuit between its emitter and collector. When the input voltage is sufficiently high, the

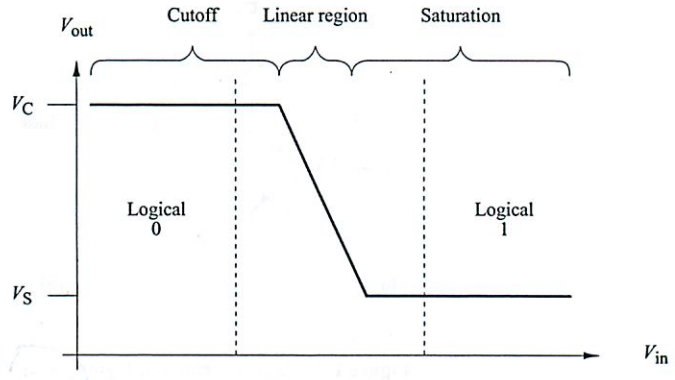


Figure 1.3 Voltage-transfer characteristics of an RTL inverter.

transistor becomes *saturated* and effectively short-circuits its emitter to its collector. Details of the sizes of each region of operation (cutoff, active, and saturation) and the exact placement of the "knees" of the transfer characteristic depend on parameters of the transistor (such as its gain) as well as on values of the resistors in the device. The ranges of voltages specified for valid logic levels of 0 and 1, in turn, reflect these details and hence constrain their choice.

It is important, for example, that the design parameters be such that a valid logic level at the input places the transistor either in the cutoff or saturation region of operation; this avoidance of the active region guarantees reliable performance in the presence of noise. In its active region, the circuit behaves like an amplifier with high gain; if, for example, a valid logical 1 at the input to the device biases the transistor into its linear region, it will amplify small perturbations of the input. Thus a 10-mV noise spike at the input might (assuming a gain of 10) become an 0.1-V noise spike at the output, as sketched in figure 1.4.

The output signal presumably connects to the input of other devices, where its noise component may be amplified further, until the resulting voltages become invalid representations of logic values.

Thus we constrain input voltages representing valid logic values to fall well within the cutoff and saturation regions, where the output voltage will be relatively insensitive to small perturbations of the input. In general, we leave a gap between each range of valid logic levels and the active region of the transistor in order to provide noise margins.

In the RTL logic family, the standard supply voltage V_{CC} is 3.6 V. Valid logical 0 must be below 1 V, and logical 1 must be above 2.5 V; the resulting noise margins are each about 0.5 V.

1.8.2 Fanout Restrictions

It is necessary to define the valid logic levels so as to include the device's output values when it has logically valid inputs. Having insisted that valid inputs will

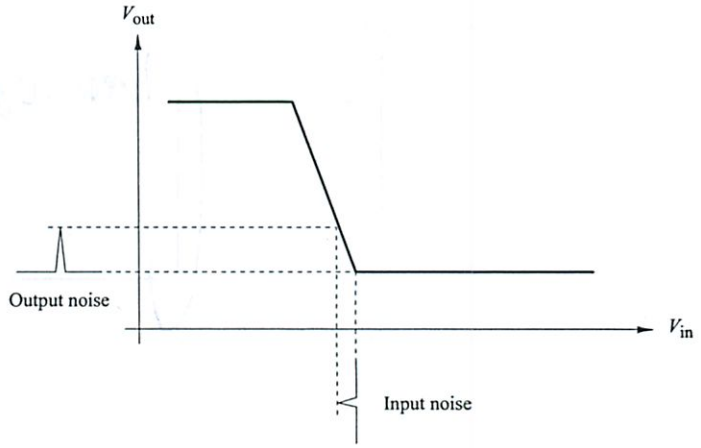


Figure 1.4 Amplification of a noise spike.

result in either saturated or cutoff transistor behavior, we find that this reduces to a requirement that the cutoff output voltage V_C and the saturation output voltage V_S be included as valid logical 1 and 0 levels, respectively.

Unfortunately, the characteristic plotted in figure 1.3 oversimplifies the behavior of our device in several ways. One of these is the assumption, made for the purposes of the preceding discussion, that the load on the output of the device imposed by external devices connected to terminal C is negligible. In fact, the voltage drops across the collector resistor and the transistor itself are functions of the output current; the effect of a heavy load on the output of the inverter is, roughly, to move the cutoff voltage V_C and the saturation voltage V_S toward each other and hence toward the forbidden zone between valid logic representations. As a result of this effect, each family of logic implementations imposes a restriction on the number of device inputs to which each device output can be connected; this is called the *fanout* limitation. In the case of RTL, device fanout is only about 5. This is primarily due to the voltage drop across the collector resistor while the transistor is at cutoff (in the logical 1 state), which lowers the output voltage as the fanout (and hence the output current) is increased.

1.8.3 Nonlinearities and Gain

We have seen that the continuous range of voltages between minimum (zero or ground, in our example) and maximum (the supply voltage V_{CC}) is divided into several regions: The valid logic levels are at the extremes, a forbidden zone occupies the center portion, and the remaining two gaps constitute the noise margins. It is generally desirable to center the forbidden zone between the valid logic levels, yielding comparable noise margins for logical 0 and 1 values; it is the *minimum* of the noise margins that dictates the maximum tolerable electrical noise level for a logic family.

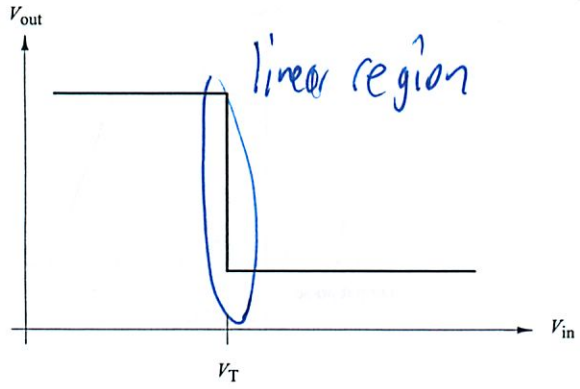


Figure 1.5 Ultra-high-gain switching device.

The reader may have noticed that the active region of the transistor, which constitutes its useful range in linear applications (such as in a high-fidelity amplifier) is a positive annoyance in digital applications. The fact that it must be avoided in the assignment of valid logic levels motivates us to minimize its size, thereby allowing more freedom to assign logic levels and noise margins. From this standpoint, an ideal transistor for digital purposes has an infinitesimal linear region, as depicted in figure 1.5.

In this hypothetical device, the linear range is reduced to a vertical line, corresponding to a (negative) infinite gain at a single threshold voltage V_T.

The gain of active elements such as transistors plays a crucial role in digital logic, in that it allows a device with marginally valid input levels to produce output levels that are well within the valid ranges. It is possible to build logic devices using only passive devices such as diodes or even resistors, but the outputs of such devices will in general be closer to the forbidden zone than the inputs. Hence a logic signal passing through several such devices will gradually deteriorate until it is no longer valid. A passive device fails to guarantee that every valid input produces a valid output; one can always find a marginally valid input value for which the output value is forbidden. Such a device fails to meet the static discipline unless the validity standards applied to its output signals are less rigorous than those applied to its inputs.

In order to apply a set of validity constraints such as those depicted in figure 1.1 uniformly to both inputs and outputs of a combinational device, we need an active device exhibiting both gain and nonlinearity.

Consider figure 1.6, which shows the static voltage-transfer curve of an inverter. The shaded regions of the diagram represent valid inputs that generate invalid outputs and are excluded by the static discipline. Consequently, the transfer curve of the inverter must enter the center rectangle at the top and leave it at the bottom, in order to constrain invalid outputs to occur only on invalid inputs. Since the width $V_{ih} - V_{il}$ of this rectangle is less than its height $V_{oh} - V_{ol}$ by the sum of the noise margins,

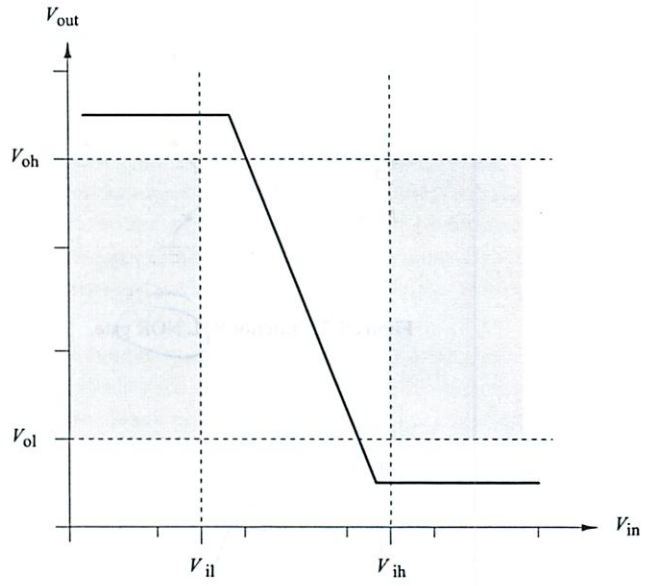


Figure 1.6 Forbidden V_{in}-V_{out} regions.

the slope of the transfer characteristic—hence the gain—must be greater than 1 (in magnitude) over this center section of the curve. Moreover, since the difference between the maximum and minimum input values is at least as great as the difference between the maximum and minimum output values, the *average* gain over the entire transfer curve is at most 1; this implies that the gain outside of the central forbidden region must be less than 1. Thus differing slopes are required along various regions of the transfer characteristic, dictating a nonlinear input/output relationship.

These considerations lead us to the inescapable conclusion that only *nonlinear* devices are suitable for the implementation of logic families. Linear devices, whose DC gain is constant over their entire operating range, will result in logic elements that fail to meet the static discipline for certain values.

Passive and linear devices are occasionally used to perform logic functions, although this practice requires that active ("restoring") logic elements be interspersed to restore marginal signals to valid logic values. This technique amounts to a change of logic-level representations between the inputs to a device and its outputs, choosing in the latter case an ad hoc representation that accommodates the signal deterioration imposed by the device (such as lower signal levels due to low gain).

1.8.4 Input Gating

A logic family must provide devices that effect logical functions of several input variables. Devices that combine logical values in elementary ways are often called *gates*, a name that stems from the early days of switching theory. As the name

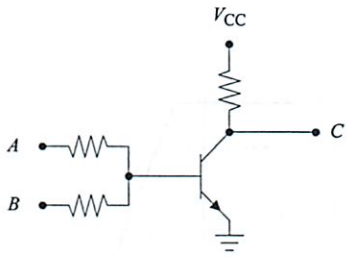


Figure 1.7 Inferior RTL NOR gate.

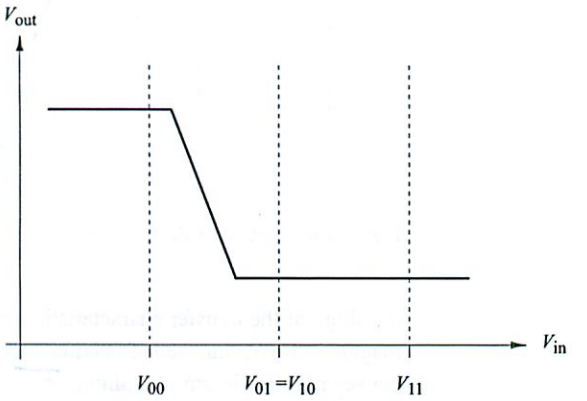


Figure 1.8 NOR gate voltage-transfer characteristic.

suggests, gating can be viewed as a mechanism by which one logical signal can be blocked or passed, depending on the value of another signal.

In the RTL family, the simplest such device to implement is the two-input *NOR gate*. This device has the property that its output is a logical 0 (using positive logic) if either or both of its inputs are logical 1s; its output is 1 if and only if both inputs are 0s. An early implementation strategy for such devices used resistors to take a linear combination of the two input voltages and directed the result to a transistor with a strategically placed active region. Figure 1.7 shows such a circuit, and figure 1.8 shows a plausible voltage-transfer characteristic.

Note that the input circuitry (resistors) produces a weighted sum of the input voltages; the weights are made equal because of the symmetry of the operands to the logical NOR function. The effect is to select the transistor's input from one of three equally spaced values, corresponding to input combinations containing zero, one, or two logical 1s, respectively.

This implementation strategy is an example of *threshold logic*, in which a logic

function is computed by comparison of a weighted sum of binary inputs against a fixed threshold value. The inherent weakness of this approach stems from the way that *fanin*, or number of inputs, compromises its noise-immunity characteristics. Referring to figure 1.8, we note that the transistor now has three valid input ranges rather than two; there is consequently less room between any two of them to squeeze both the forbidden zone (including the active transistor region) and noise margins. As the number of inputs is increased, the problem clearly gets worse, until the fundamental limit is reached in which pairs of distinct linear combinations of valid inputs are separated by less than the active range of the transistor. Primarily as a result of these considerations, the NOR gate implementation shown in figure 1.7 was abandoned early in the history of RTL.

The problem with this circuit stems from its use of linear circuit elements—namely resistors—to combine logic values. Although it was resolved in the case of RTL by moving from *input gating* (in which transistor inputs are combined) to the *output gating* strategy described in the next section, modern logic families use input gating extensively. The difference is that they use nonlinear circuit elements to combine the inputs. Diode-transistor logic (DTL) uses input diodes to effect logic functions, and the popular transistor-transistor logic (TTL) families use multiemitter transistors to perform input gating. It should be noted that input gating typically involves passive devices such as diodes or resistors and hence must be followed by an active "restoring" device to ensure unambiguous output values despite possibly marginal inputs.

1.8.5 Output Gating: Wired-OR

An alternative implementation of multi-input RTL gates is shown in figure 1.9. This circuit has noise characteristics similar to those of the inverter of figure 1.2, since the inputs are combined by the nonlinear output transistors rather than by the linear input resistors. A reasonable understanding of its operation may be obtained by viewing the transistors as switches connected in parallel, so that either or both switches being closed effectively connects the output C to ground. While there are second-order effects that limit potential fanin with this approach, they are much less serious than those inherent in threshold logic techniques.

The NOR circuit of figure 1.9 differs only slightly from the result of connecting the outputs of two RTL inverters together; the difference is simply a reduction of the collector resistance in the latter case. In fact, the interconnection of outputs is common practice when using RTL and certain other amenable logic families. The technique has come to be known as *wired-OR*, although in the present case (RTL positive logic) *wired-AND* would be a more appropriate term. Thus two RTL logic levels A and B, each the output of some RTL device, can be simply wired together to yield their logical conjunction A AND B (or AB, using the usual Boolean algebraic notation).

There are, of course, limits to the number of outputs that can be wired together. If n RTL 1 levels are connected to a single RTL 0 level, for example, then one saturated transistor must carry the sum of the currents flowing through the $n + 1$ collector resistors. This increased current not only degrades the output voltage (moves it

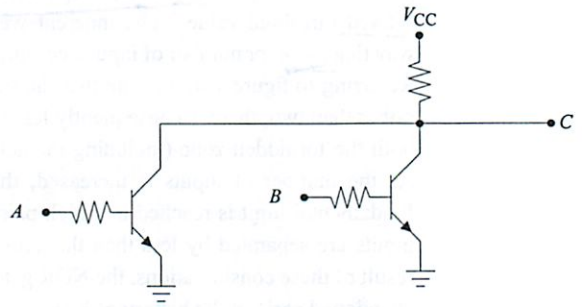


Figure 1.9 Improved RTL NOR gate.

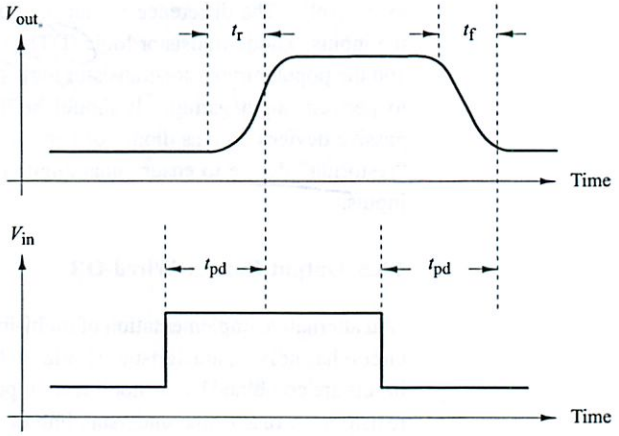


Figure 1.10 Signal timing.

closer to the forbidden zone) but may, for sufficiently large n, approach the current limits of the output transistor.

1.8.6 Timing Considerations

If we use a rectangular voltage waveform such as the V_{in} in figure 1.10 as input to an inverter, we are likely to observe an output waveform similar to that plotted as V_{out}.

We notice first that, in contrast to the "ideal" rectangular shape of the V_{in} curve, the corners of the output waveform have become rounded and the rising and falling edges are not vertical. These effects reflect fundamental physical properties, such as capacitive effects, of the device. In order to make a transition between the valid logic levels, electrical charge must be moved—for example, to and from transistor

junctions. This motion of charge constitutes work (in the sense of physics), and since we are constrained to apply finite power to the device, the transitions require finite amounts of time. While very fast logic families such as emitter-coupled logic (ECL) attempt to minimize the amounts of stored charge to be moved and maximize the amount of power used to move it, in practice transitions between valid logic levels are not instantaneous. The time t_r taken for a transition from low to high logic level is called *rise time*, and the time t_f for the reverse transition is called *fall time*. Typical rise and fall times are measured in very small numbers of nanoseconds. They are dependent not only on the technology of particular logic families, but on the load placed on particular signals; higher load tends to add capacitance and hence prolong transitions. A pleasant characteristic of rise and fall times is their resistance to accumulation as a signal passes through a number of cascaded logic devices. Thus the waveform at the end of a long string of series-connected inverters will have rise and fall times similar to that of a single inverter. This is attributable to the high gains of the transistors, each of which tends to increase the slope of its input transitions.

The output waveform is delayed from the input by an approximately constant interval, corresponding to the propagation delay t_{pd} of the device. This delay also reflects the time necessary for the motion of charge that accompanies transitions and is thus closely related to the rise and fall times. However, propagation delay *does* accumulate through cascaded devices. Thus, if the propagation delay of an RTL inverter is 12 ns, the output of n series-connected inverters will be delayed (with respect to its input) by $12n$ ns. Of course, the output will be inverted if n is odd and will be a delayed replica of the input if n is even. Propagation delay is an important characteristic of digital devices, leading ultimately to performance limitations (in the sense of the number of computations that can be executed per unit of time) in complex digital systems.

1.8.7 Propagation and Contamination Delays

The delay involved in a complex digital computation is typically determined by the *longest* time necessary for signals to propagate from the inputs of a system to its outputs. As a consequence, digital designers often compute the total propagation delay along each signal path (adding the delays of each device it includes) to determine the limiting *critical path* of information flow. The usefulness of t_{pd} specifications for component devices, then, stems from their validity as an *upper bound* on the delay of data through each device: System designers need a guarantee that the time taken by signals to propagate along each path is no longer than the value they compute from device specifications. An overly conservative t_{pd} specification may cause a system to be designed for suboptimal performance, but it will operate as its designer predicts; an optimistic t_{pd} specification is likely to cause faulty operation. To optimize performance, it is valuable to have the tightest possible bounds on propagation delays in order to know precisely when signals are valid; however, we must insist that these bounds be conservative.

Very aggressive designs sometimes require us to pin down signal validity periods even more precisely. Glancing back at figure 1.10, we note that the output voltage remains in a logically valid range for a brief period after the input changes; this reflects

our acceptance of a *range* of valid voltages as well as the physical propagation time of the leading edge of the input signal. This delay represents an additional period of output signal validity that can be exploited in carefully engineered designs: A designer might assume that the output of a gate remains valid for a nanosecond or two *after* its input changes, in contrast to the more conservative assumption that output validity is immediately corrupted by an input change. To facilitate such highly engineered designs, the specification of a combinational device might include, in addition to the usual propagation delay t_{pd}, a *contamination delay* t_{cd}.[3] The contamination delay of a combinational device is the minimum interval following an input change during which the validity of previous outputs is guaranteed to remain intact.

While t_{pd} is an *upper* bound on a period of output *invalidity*, t_{cd} is a *lower* bound on a period of output *validity*. Like propagation delay, the contamination delay specifications must be conservative; however, a conservative t_{cd} is *smaller* than necessary, whereas a conservative t_{pd} is *larger* than necessary. Like propagation delays, contamination delays can be specified independently for each input-to-output path of a device. The contamination delay between an input-output pair reflects the *fastest* path between that input and output; the propagation delay reflects the *slowest* path. The contamination delay is always less than the corresponding propagation delay; typically, it is much less.

Designs that substantively exploit nonzero contamination delays are rare, and for good reason. Squeezing the last nanosecond out of a path requires painstaking timing analysis and effectively returns the engineer to the analog domain of continuous variables and physical effects. It also tends to compromise system manufacturability by depending on subtle characteristics that are quite variable from device to device. Unless otherwise noted, we shall assume zero contamination delays throughout the remainder of this text.

1.8.8 Speed and Power

A major weakness of RTL stems from its use of a collector resistor (called a *pullup resistor*) to establish its high-voltage output level. First, this resistor limits collector current and hence prolongs the time necessary to charge stray capacitances associated with a low-to-high transition, yielding poor rise times and consequently high propagation delays. Second, it provides a substantially poorer path for current flow to V_{CC} than a saturated transistor provides to ground; hence it introduces an asymmetry between the rise and fall times of the device.

Adjustments in the value of the pullup resistor affect both the switching speed and the power dissipated by the gate. For a resistance R and total stray capacitance C, the switching times are on the order of RC seconds; hence halving the value of the pullup resistor speeds up the device. However, the power dissipated by the resistor

[3] Contamination delays are occasionally referred to as *minimum* propagation delays, which is misleading. Propagation and contamination generally happen at different times, contamination being earlier.

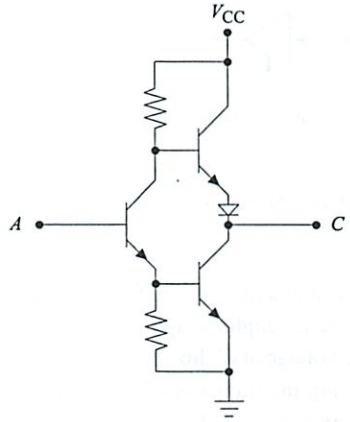


Figure 1.11 Simplified TTL inverter.

on a low-voltage output is roughly

$$P = \frac{(V_{CC})^2}{R}.$$

Thus the product of the switching time and the power dissipation, often cited as the *speed-power product*, is relatively independent of the value of R and is approximately constant for a particular logic family or implementation strategy. Speed and power dissipation are important parameters in digital-system design decisions, and the speed-power product is sometimes used as a measure of efficiency by which alternative families are compared.

The asymmetry between the resistive pullup and saturated transistor pulldown tends to make opposite-going transitions propagate at differing rates through RTL devices, yielding in effect separate propagation delays for rising and falling edges. Aside from its other disadvantages, this disparity complicates enormously the conceptual model of the devices necessary to use them effectively; in particular, it casts suspicious light on our lumping of propagation delay into a simple single parameter. Figure 1.11 illustrates the more symmetrical output circuitry used in TTL logic.

The two output transistors are stacked in a *totem-pole output* configuration, arranged so that valid output levels are asserted by one transistor being saturated while the other is cut off. When the A input is low, the input transistor is cut off. As a result, the base of the upper (pullup) output transistor is pulled high and that transistor saturates, while the base of the lower (pulldown) output transistor is pulled low and that transistor is cut off. Consequently, the voltage at the output terminal C rises close to the collector supply voltage V_{CC}. Conversely, when the A input is high, the input transistor saturates. The bases of the two output transistors are effectively wired together and pulled to a low voltage by current going into the base of the pulldown transistor, which saturates. The diode in series with the pullup transistor's emitter causes the voltage at that emitter to be higher than the voltage at C by the voltage drop

review 8.02 material

- Capictance
- capacitor
- RLL Circuit

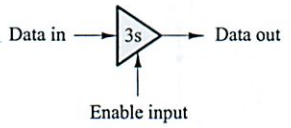


Figure 1.12 Three-state driver.

across a conducting silicon diode (typically 0.7 V). As a result, the pullup transistor's emitter is at a high enough potential, relative to its base, that the transistor cuts off, and the voltage at C drops to nearly zero.

Viewing the transistors as switches, we find that the circuit can provide a valid high output voltage if it connects C to V_{CC} about as definitively as it can short C to ground to force it low. The passive resistive pullup of RTL logic has effectively been replaced by an active switch (transistor), resulting in a faster device with more similar rise and fall times.

A disadvantage of this scheme is that it rules out wired-OR interconnection of outputs, since the interconnection of a high to a low output would result in a low impedance path (through two saturated transistors) between V_{CC} and ground. This constraint is partially alleviated by the availability of *three-state drivers* (often called *tristate drivers*), which feature a logic-level *enable* input. When *enable* carries a logical 1, the driver behaves as a normal TTL driver; when *enable* carries a logical 0, both of the totem-pole output transistors are cut off, leaving the output terminal floating. The symbol for a three-state driver is shown in figure 1.12. Several three-state outputs may be connected, so long as the logic that drives their enable inputs is designed to guarantee that at most one connected driver is enabled at any time. Of course, a line connected only to disabled three-state drivers is left floating and should generally be considered to carry an invalid logic level.

The goal of fast transitions dictates that the pair of totem-pole output transistors change between cutoff and saturation as nearly simultaneously as possible; thus there tends to be a brief period during each transition when both transistors are conducting. This in turn presents (momentarily) a nearly short-circuit path between V_{CC} and ground, tending to generate annoying supply voltage spikes that can interfere with other logic devices sharing the same power supply. For this reason, it is common practice to proliferate small filtering capacitors in TTL designs to smooth out supply-voltage variations.

1.8.9 Saturation Charge

It has already been noted that electric charge stored in semiconductor junctions and stray capacitances requires both time and power to move, and hence limits the performance of digital circuits. This effect is due in large part to charge (and hence delay) associated with switching a transistor in and out of saturation. Several techniques have been developed to avoid delays stemming from *saturation charge*; in each case, output transistors remain in their active region rather than becoming

saturated. This leads to faster switching times, generally at the cost of increased power consumption.

Schottky TTL is similar in design to (and compatible with) standard TTL, except for the incorporation of a very fast diode (a *Schottky diode*) between the base and collector of each transistor. This diode prevents the transistor from saturating and effectively provides nonlinearity at the low-voltage end of its voltage-transfer characteristic, but with greatly reduced charge storage.

Emitter-coupled logic (ECL) avoids nonlinearities at the saturation end of transistor voltage-transfer curves altogether. Instead, it combines input transistors with output transistors in such a way that either input or output transistors are cut off, thus using the relatively fast cutoff nonlinearities to stabilize both logical output values. This results in very fast switching times (on the order of a nanosecond) and ensures that each gate will provide some current path to ground at all times, thus increasing power dissipation.

1.8.10 Power and Density

The density at which logic gates can be packed onto a silicon chip is an important factor in both cost and performance. Integrated circuits that can be reliably manufactured involve a marginal production cost that is relatively independent of the complexity of the circuit they contain. Thus, to a first approximation, a small-scale integrated circuit containing a few dozen gates might have a price similar to that of a complete computer chip in high-volume production. Moreover, for a variety of reasons, electric signals take longer to propagate between chips than to propagate between gates on the same chip. Thus a complete single-chip computer is likely to outperform one that distributes the same technology among several chips, because of the increased interchip communication time. These and other factors have stimulated a strong push to increase the densities of integrated circuits; these have improved from the level of a few gates per chip in the mid-1960s to numbers in the hundreds of thousands by the late 1980s.

Density, however, is limited by several factors. Among these is the optical technology used to inscribe circuits on silicon, which limits (currently to a micron or so) the width of the on-chip lines that make up the circuit. Clearly the complexity of the basic gate design further limits the number of gates on a chip. Finally, the power consumed by each gate, coupled with the physical constraints on the amount of heat an integrated-circuit package can dissipate, is a fundamental limitation and provides a principal motivation for our concern with power dissipation as a parameter of gate technology.

Currently the highest densities are obtainable using *metal-oxide semiconductor* (MOS) technologies. The basic switching element in an MOS gate is a *field-effect transistor*, or FET, whose principal characteristic is its very high input impedance: An input to an FET draws virtually no current and hence consumes virtually no power. The combination of very low power consumption and a very simple gate design allows high densities and hence very complex functions on a single integrated circuit. Nearly all of the single-chip computers, for example, use MOS technologies.

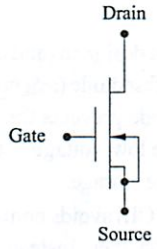


Figure 1.13 n-channel FET.

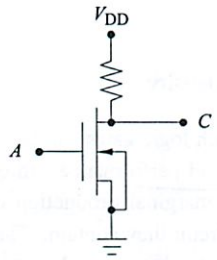


Figure 1.14 NMOS inverter.

MOS transistors, like bipolar ones, come in two polarities, called *n-channel* and *p-channel*, respectively. Figure 1.13 shows an n-channel FET. Note that it is a three-terminal device, which for our purposes can be viewed as a voltage-controlled switch. So long as the *drain* has a positive voltage with respect to the *source*, current flows from the drain to the source when the *gate* voltage is high (approaching the drain voltage) but not when the gate voltage is low (near the source voltage).

A common digital MOS technology—NMOS—uses n-channel FETs as the basic switching element. The basic NMOS gate (figure 1.14) is similar to the RTL gate, using a resistive pullup[4] and an active pulldown; the big difference between NMOS and RTL gates is the much greater currents (and lower resistances) of the latter.

An MOS technology using active pullups, called *CMOS* (for complementary MOS), combines n- and p-channel FETs to form a gate with very low power dissipation. A CMOS gate may be viewed as two series-connected switches (similar to the totem-pole outputs of TTL), arranged so that one switch or the other is always open.

Thus a quiescent CMOS gate provides no path for current to flow between V_{DD}

[4] In fact, since resistors are very awkward to implement using NMOS technology, a fixed-bias *depletion-mode FET* is used as the pullup. To a good first approximation, the depletion-mode pullup may be viewed as a resistor.

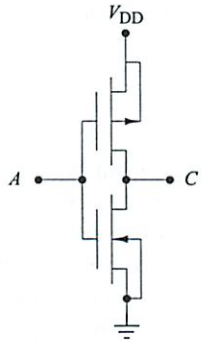


Figure 1.15 CMOS inverter.

and ground; furthermore, because of the high impedances of FET inputs, negligible amounts of current are drawn by other CMOS devices connected to its output. The result is that CMOS logic consumes very little power except during transitions, leading to a speed-power product several orders of magnitude better than those of other logic families. CMOS offers high noise immunity, allows a wide range of power-supply voltages, and is generally easy to design with; for these reasons, it has become the technology of choice for large-scale integrated circuits.

Each of the MOS technologies requires relatively complex buffers to interface the very-high-impedance MOS gates to external (off-chip) logic. As a result, they are generally used only for complex functions (such as calculator chips and microprocessors) where the number of external connections is small compared with the number of MOS gates involved.

1.9 Summary

Wait for class for end part

The technology discussed in this chapter provides the basis for the fundamental abstraction of digital systems. It allows systems to be constructed by designers who are largely ignorant of the underlying electronics, who deal with the primitive elements (such as gates) strictly in the digital domain, and who agree to abide by the simple rules (such as fanout and synchrony) upon which we insist.

It should be emphasized that our goal has not been to deal seriously with the topic of logic gate design. Although the examples given were based on electronic implementations, the reader should recognize that the fundamental issues are not electronic; they apply equally to the realization of digital values as mechanical tension, fluid pressures, or any other continuously variable physical parameter. In each case nonlinearities, gain, and the "forbidden zone" will play essential roles. Each will involve speed-power trade-offs, noise, and propagation delays. An appreciation of the electronic details of the preceding examples is not essential to the understanding or even the effective design of digital systems; indeed, the degree of naivete they allow their users is precisely their value as an abstraction. It *is* important, however, to

recognize the limits to the abstraction and the physical constraints from which they derive.

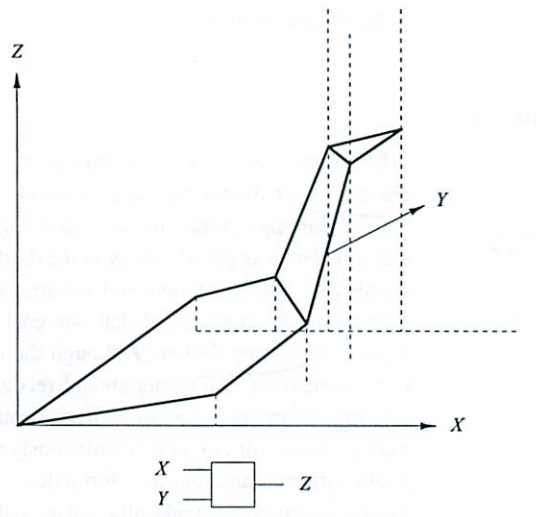
1.10 Context

The RTL and NMOS logic families were the progenitors of logic ICs (early 1960s) and VLSI (early 1970s), respectively, but each is now obsolete. Surviving families of off-the-shelf logic chips include ECL and the still ubiquitous TTL; the venerated *TTL Data Book* [TI 1984] will provide the uninitiated reader with a glimpse of the Tinkertoy-set world that has been available to the digital designer for a quarter century. Many contemporary textbooks on digital design, such as Hill and Peterson [1987], treat the subject of this and the following several chapters at a more detailed but still introductory level.

Logic implementation issues are treated in the context of NMOS technology in the classic Mead and Conway [1980], which precipitated something of a revolution by popularizing VLSI design beyond the engineering elite. Weste and Eshraghian [1985] perform a similar service for CMOS, the current vogue among custom chip design technologies.

1.11 Problems

Problem 1.1 Logic Functions:



A. Using suitable values for valid logic levels and noise margins, what logic function does the device with the above transfer characteristic compute? Assume the convention of positive logic.

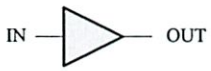
B. What function is computed by the above device if we assume negative logic?

Problem 1.2 Base 3 Numbering:

Ternary is a term referring to the number system in base 3. Consider a convention in which a ternary digit is represented as an electric voltage between 0 and 10 V. Let 0–1 V represent a valid "0" output, 4–6 V a valid "1" output, and 9–10 V a valid "2" output.

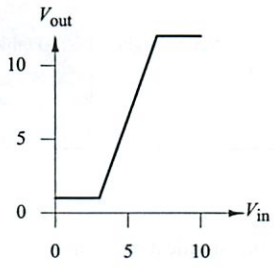
A. Assuming noise margins 1 V wide, show the mapping of logic levels to voltages for this ternary system. Include valid logic-level outputs, noise margins, and forbidden zones. Your result should resemble figure 1.1.

B. Graph the transfer characteristic for a device capable of acting as a ternary logic *buffer*, that is, a device that produces at its output the same logic level present at its input, as shown below:



IN	OUT
0	0
1	1
2	2

C. Can a device with the following transfer characteristic be used as a ternary logic buffer? Why or why not?



D. How many bits of information are carried in a ternary signal on a single wire?

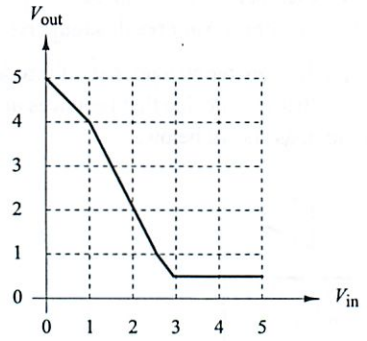
E. How many different combinations of valid logic levels can be encoded on three ternary wires? How many bits of information does this represent? How many wires would be needed to carry this same amount of information in binary?

F. What is the information flow in bits per second for three ternary wires if a new set of values is sent every 10 ms?

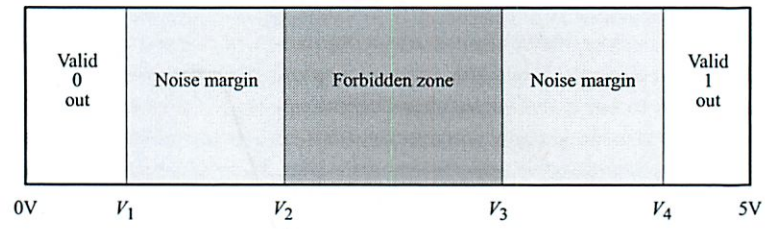
G. What is the information flow in bits per second for three *binary* wires if a new set of values is sent every 10 ms?

Problem 1.3 RTL Logic:

An RTL inverter circuit has the transfer characteristic diagrammed below:

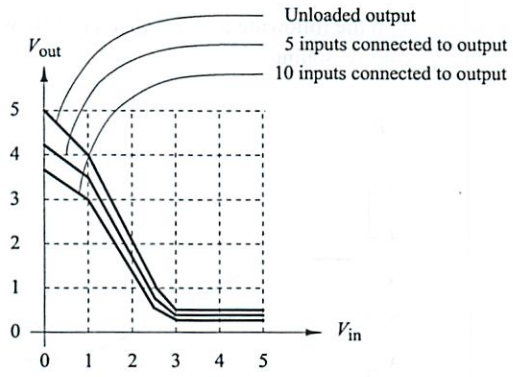


For suitable values of the threshold voltages V_1, V_2, V_3, and V_4 in the figure below, this RTL inverter obeys the static discipline.



A. What is the smallest possible width for the forbidden zone (that is, what is the smallest possible value of $V_3 - V_2$) if the noise margins are each at least 0.5 V wide? Give values of V_1, V_2, V_3, and V_4 corresponding to this minimum width. Why can't the forbidden zone be made any smaller than this?

The above transfer characteristic is for an *unloaded* RTL inverter, that is, an inverter with no other logic circuits connected to its output. If other logic circuits are connected to this output, they will draw current from it and affect the shape of the transfer characteristic. Some of these altered transfer characteristics are shown below.

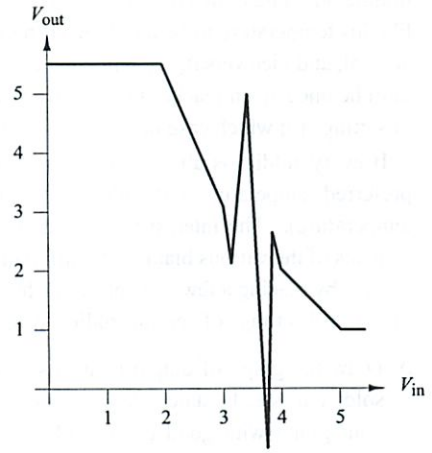


B. Pick threshold voltages V_1, V_2, V_3, and V_4, as before, so that the static discipline is obeyed where individual inverters have a fanout of no more than 5. In other words, the transfer curve of any individual inverter may be that of an unloaded inverter, or an inverter loaded with five logic inputs, or anywhere in between. Continue to allow noise margins of at least 0.5 V. Briefly explain the constraints on your selection (such as "V_1 cannot be greater than 2 V because...").

C. If the fanout of some inverters is increased to 10, is it still possible to pick V_1, V_2, V_3, and V_4 to allow noise margins of 0.5 V? Why or why not?

Problem 1.4 Transfer Characteristics:

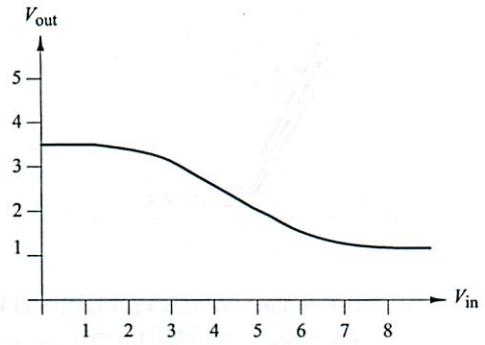
Is it possible to assign valid logic level and noise margin boundaries so that a device with the following transfer characteristic would serve as an inverter? Briefly explain your reasoning.



Problem 1.5 Transfer Characteristics II:

Is it possible to assign valid logic levels and noise margin boundaries so that a device

with the following transfer characteristic would serve as an inverter? Briefly explain your reasoning.



Problem 1.6 Barracks Logic:

Barracks logic is built out of sleeping soldiers covered by electric blankets. Each blanket has a control switch with discrete control settings ranging in 5-degree (Fahrenheit) intervals from 0 to 50 degrees. The temperature of a soldier covered by one or more electric blankets will be the sum of the ambient temperature in the barracks plus the setting on the controller for each blanket.

Each soldier has a preferred sleeping temperature, which varies from individual to individual but is always within the range of 60 to 80 degrees, inclusive. If a soldier's temperature departs from his preferred temperature, he will wake up once every minute and adjust the control by one 5-degree increment in the direction he would like his temperature to be modified (if he is cold, he will increase the setting on his control, and vice versa). He will continue these adjustments by 5-degree increments until he once again reaches his preferred temperature (and goes to sleep) or runs out of settings (in which case he grumbles angrily in bed).

If every soldier is allowed to control his own blanket, each will soon reach his preferred temperature and slide into nocturnal bliss (assuming a suitable ambient temperature). The interesting aspects of barracks logic result from switching the controls of the various blankets to different soldiers. Inputs to the system are accomplished by placing a few controls in the hands of outsiders, and outputs are read from the control settings of certain soldiers designated by the logic designer.

A. Draw the graph of output control setting vs. input control setting for a typical soldier in steady state. Assume an ambient temperature of 40 degrees. Mark your graph with good choices of the valid regions for the two logical values, the forbidden zone, and the noise margins. Let logical 0 be when a control is completely off and logical 1 be when the control is completely on (or at the highest setting).

B. List some sources of noise that justify the need for noise margins.

C. Even though it is the middle of February, a sudden warm spell raises the ambient temperature in our barracks logic system to 55 degrees. Sketch a new graph of output control setting vs. input control setting in the warmer barracks.

D. Over what range of ambient temperatures will barracks logic function reliably?

E. Does the following arrangement perform a useful function? What is it?

F. A model HOT-1 electric blanket control can power 1200 W of blankets. A model HOT-BED electric blanket requires 275 W. What is the fanout of a HOT-1? What might happen if this rating is exceeded? Is there more danger of exceeding the rating at an output value of logical 0 or logical 1?

G. To create a system with multiple inputs, we allow several blankets to be placed over a single soldier. What is the maximum fanin possible in barracks logic if 170 degrees is the highest temperature a soldier can tolerate without his characteristics being permanently altered?

H. Show how to build a NOR gate and an AND gate in barracks logic.

I. Explain what causes rise time, fall time, and propagation delay in barracks logic. Give worst-case numerical values for each of these parameters for the barracks logic inverter. (Let *rise time* be the time from when an output leaves the valid logical 0 range until it enters the valid logical 1 range, and vice versa for fall time. For propagation delay, use the time from when the input switches to a valid logic level to the time when the output switches to a valid logic level.)

Problem 1.7

An inverter has propagation delay t_d and rise time t_r. What is the propagation delay and rise time of three inverters connected in series?

Problem 1.8

Consider a family of logic in which

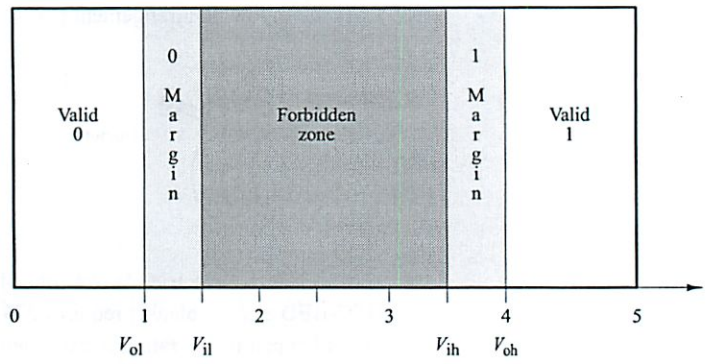
$$\begin{aligned} V_{ol} &= 0.6\ \text{V}, \\ V_{oh} &= 2.8\ \text{V}, \\ V_{il} &= 1.0\ \text{V}, \\ V_{ih} &= 2.2\ \text{V}. \end{aligned}$$

A. How wide are the 0 and the 1 noise margins (in volts)?

B. What is the smallest average voltage gain that a buffer in this family must exhibit over the range $V_{il} < V_{in} < V_{ih}$? (Recall that a buffer is a combinational device that asserts on its output the same logical value that it senses at its input.)

Problem 1.9

Consider a logic buffer (output logic value = input logic value) in a family of logic parameterized by the following mapping of voltages to logic levels:



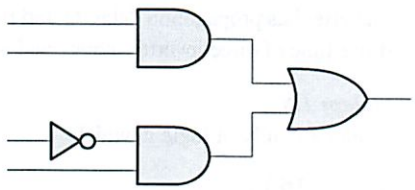
A. What is the minimum gain a *buffer* must exhibit over the forbidden zone in order to obey the static discipline with this convention?

B. Sketch the voltage-transfer characteristic of a buffer in this logic family.

C. Let V_f be a voltage that, when applied to the inverter's input terminal, yields V_f at the buffer's output. Give a range of possible values for V_f.

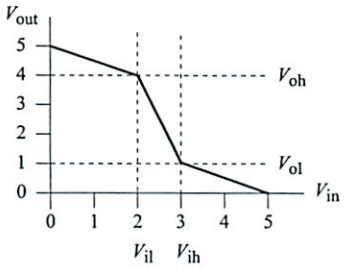
Problem 1.10 Propagation Delay and Rise Time:

Assume that all gates have a propagation delay t_{pd} and a rise time t_r. What is the maximum propagation delay and rise time of the following circuit?



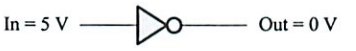
Problem 1.11

Suppose we have an inverter with the following transfer characteristic, where V_{oh}, V_{ih}, V_{ol}, and V_{il} are defined as in figure 1.1.

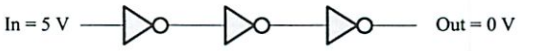


A. What are the sizes of the noise margins for this inverter?

B. Consider a single inverter having the above transfer characteristic, conventionally diagrammed as shown below. To what value could IN drop before OUT would cease to be a valid low output?



C. Now consider three cascaded inverters, each having the transfer characteristic shown above:



To what value could IN drop before OUT would cease to be a valid low output?

D. Suppose we put a box around these three cascaded inverters and called the box a NEW INVERTER. If $V_{oh} = 4$ V and $V_{ol} = 1$ V, what are the noise margins for this NEW INVERTER?

E. Give two reasons why this type of cascading is not a good way to increase noise margins.

MOSFET Readings

9/10

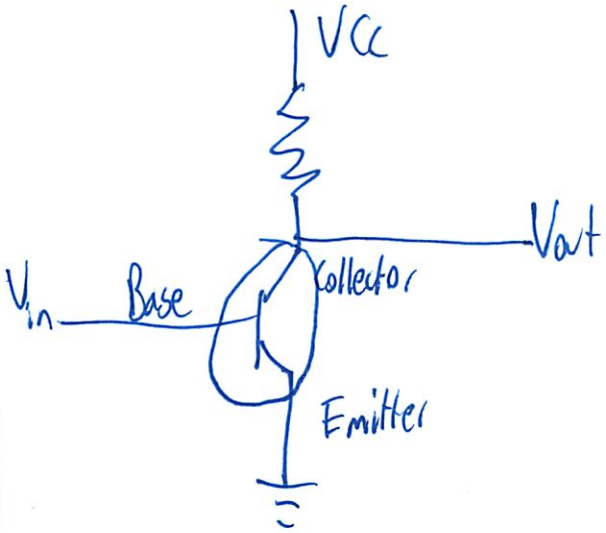
WP

transistor → that
[amplifying or switching electric signals

transistor

amplifies / switches electric signal

(did we do in 6.01?)



Small signal to control larger signal → gain

Output in proportion to input signal → amplifier

Diff types/channels

Lab started
Labs 1+2 up

next week - quiz 1

9/13

The Digital Abstraction

1. Making bits concrete
2. What makes a good bit
3. Getting bits under contract

representing individual bits

Handouts: Lecture Slides

Physical phenomena

Concrete encoding of information

To this point we've discussed encoding information using bits. But where do bits come from?

If we're going to design a machine that manipulates information, how should that information be physically encoded?

He said to his friend, "If the British march
By land or sea from the town to-night,
Hang a lantern aloft in the belfry arch
Of the North Church tower as a signal light,--
One if by land, and two if by sea;
And I on the opposite shore will be,
Ready to ride and spread the alarm
Through every Middlesex village and farm,
For the country folk to be up and to arm."

ways to represent bits

What makes a good bit?
- cheap (we want a lot of them)
- stable (reliable, repeatable)
- ease of manipulation
  (access, transform, combine, transmit, store)

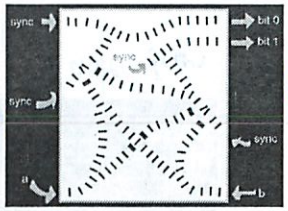
Substrates for computation

1 bit adder

We can build upon almost any physical phenomenon...

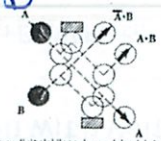
Wait!
Those last ones
might have potential...

~~lanterns~~
~~dominos~~ not practical
~~engraved stone tablets~~
~~Billiard balls~~
~~E. Coli~~
~~polarization of a photon~~



http://www.pinkandaint.com/oldhome/comp/dominoes/

http://www.americanscientist.org/template/AssetDetail/assetid/14340



http://www.digitalphilosophy.org/physicists_model.htm

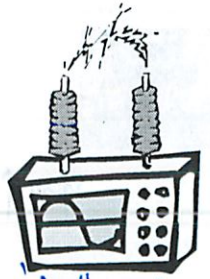
But, since we're EE's...

Stick with things we know about:

voltages	phase
currents	frequency

This semester we'll use voltages to encode information. But the best choice depends on the intended application...

Voltage pros:
- easy generation, detection
- lots of engineering knowledge
- potentially low power in steady state
  zero dissipate no energy unless something interesting happens



Voltage cons:
- easily affected by environment
- DC connectivity required?
- R & C effects slow things down

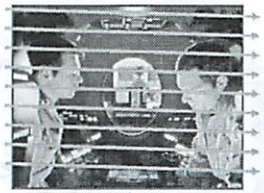
9/13

Representing information with voltage

Representation of each point (x, y) on a B&W Picture:

- 0 volts: BLACK
- 1 volt: WHITE
- 0.37 volts: 37% Gray
- etc.

Representation of a picture:
Scan points in some prescribed raster order... generate voltage waveform



How much information at each point?

Information Processing = Computation

First let's introduce some processing blocks:


v → Copy → v


v → INV → 1-v

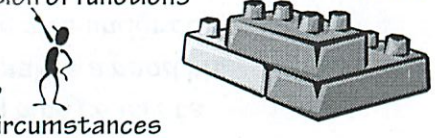
Why have processing blocks?

The goal of modular design:

Abstraction

What does that mean, anyway?

- Rules simple enough for a 6-3 to follow...
- Understanding BEHAVIOR
- without knowing IMPLEMENTATION
- Predictable composition of functions
- Tinker-toy assembly
- Guaranteed behavior,
- under REAL WORLD circumstances



Let's build a system!

input

Copy → INV

Copy → INV

Copy → INV

Copy → INV

(Reality)

output

Why did our system fail?

Why doesn't reality match theory?

1. COPY Operator doesn't work right
2. INVERSION Operator doesn't work right
3. Theory is imperfect
4. Reality is imperfect
5. Our system architecture stinks

unrealistic to do perfect

ANSWER: **all** of the above!

Noise and inaccuracy are inevitable; we can't reliably reproduce infinite information-- we must design our system to tolerate some amount of error if it is to process information reliably.

↑ the hard problem

The Key to System Design

A system is a structure that is guaranteed to exhibit a specified behavior, assuming all of its components obey their specified behaviors.

How is this achieved?

Contracts!

Every system component will have clear obligations and responsibilities. If these are maintained we have every right to expect the system to behave as planned. If contracts are violated all bets are off.

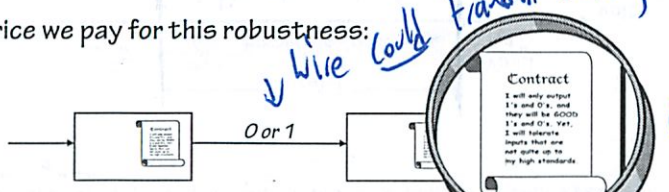
like to be able to try to build systems that work provably correct if components provably correct

The Digital Panacea ...

Why digital?

... because it keeps the contracts simple!

The price we pay for this robustness:



wire could transmit anything

but restrict to 1,0

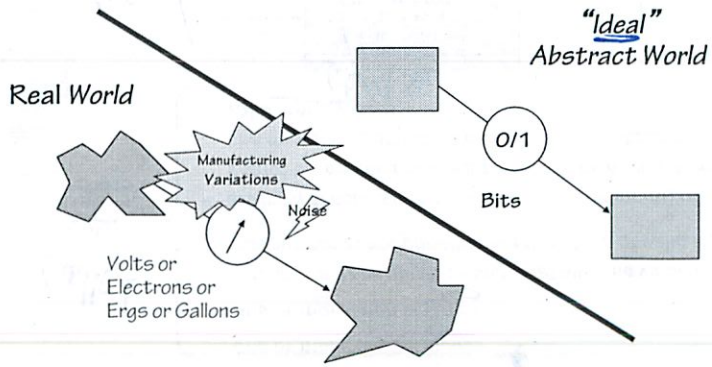
analog vs digital

All the information that we transfer between modules is only 1 crummy bit!

But, we get a guarantee of reliable processing.

can't get perfect op-amp - so handle it

The Digital Abstraction

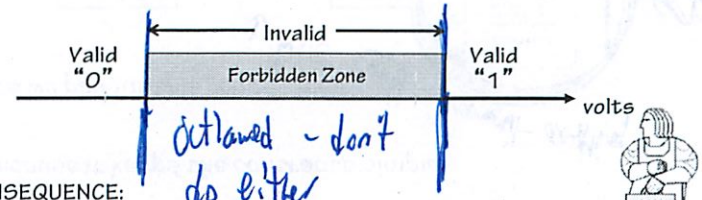


volts, noise etc have been abstracted away

Keep in mind that the world is not digital, we would simply like to engineer it to behave that way. Furthermore, we must use real physical phenomena to implement digital designs!

Using Voltages "Digitally"

- Key idea: don't allow "0" to be mistaken for a "1" or vice versa
- Use the same "uniform representation convention" for *every* component and wire in our digital system
- To implement devices with high reliability, we outlaw "close calls" via a representation convention which forbids a range of voltages between "0" and "1".



CONSEQUENCE:

Notion of "VALID" and "INVALID" logic levels

Fundamental abstraction

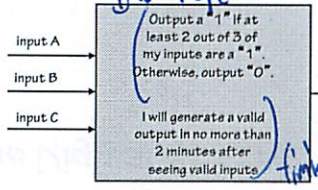
A Digital Processing Element

A combinational device is a circuit element that has

- one or more digital *inputs*
- one or more digital *outputs*
- a *functional specification* that details the value of each output for every possible combination of valid input values
- a *timing specification* consisting (at minimum) of an upper bound t_{pd} on the required time for the device to compute the specified output values from an arbitrary set of stable, valid input values

Called Static discipline

funct spec



some ins

some outs

timing spec

A Combinational Digital System

A set of interconnected elements is a combinational device if

- each circuit element is combinational
- every input is connected to exactly one output or to some vast supply of constant 0's and 1's
- the circuit contains no directed cycles

we can hook them up

if true for each part can think about entire system

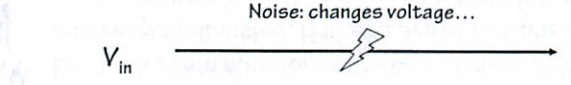
Why is this true?

Given an acyclic circuit meeting the above constraints, we can derive functional and timing specs for the input/output behavior from the specs of its components!

We'll see lots of examples soon. But first, we need to build some combinational devices to work with...

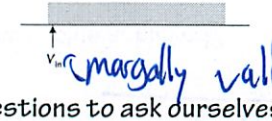
Wires: theory vs. practice

Does a wire obey the static discipline?



(voltage close to boundary with forbidden zone)

(voltage in forbidden zone: Oops, not a valid voltage!)



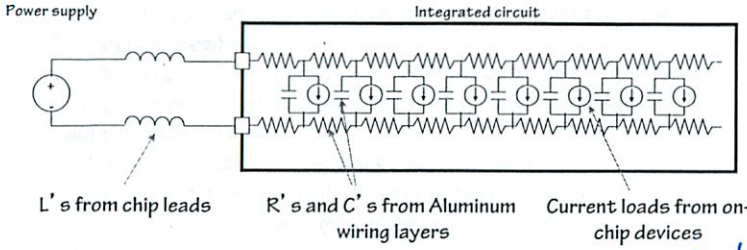
marginally valid

So not combinational device

Questions to ask ourselves:

In digital systems, where does noise come from?
How big an effect are we talking about?

Power Supply Noise



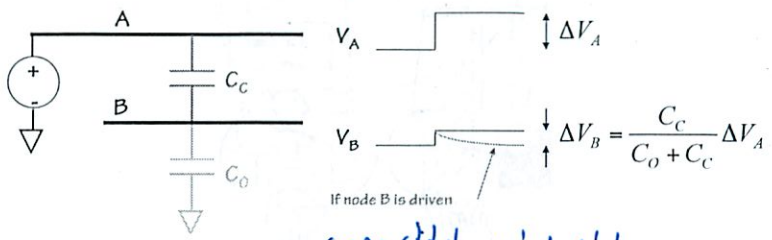
stray inductances & resistances

sensitive to current load

ΔV from:

- IR drop
  (between gates: 30mV, within module: 50mV, across chip: 350mV)
- L(dI/dt) drop
  (use extra pins and bypass caps to keep within 250mV)
- LC ringing triggered by current "steps"

Crosstalk



capacitively & inductively

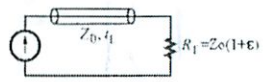
This situation frequently happens on integrated circuits where there are many overlapping wiring layers. In a modern integrated circuit ΔV_A might be 2.5V, C_0 = 20fF and C_C = 10fF → ΔV_B = 0.83V! Designers often try to avoid these really bad cases by careful routing of signals, but some crosstalk is unavoidable.

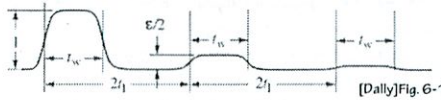
Sequential Interference

different value at different times

ΔV from energy storage left over from earlier signaling on the wire:

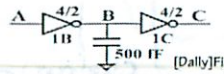
- transmission line discontinuities
  (reflections off of impedance mismatches and terminations)



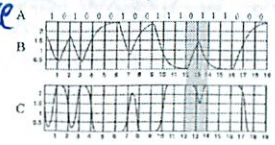


[Dally]Fig. 6-17

- charge storage in RC circuit
  (narrow pulses are lost due to incomplete transitions)

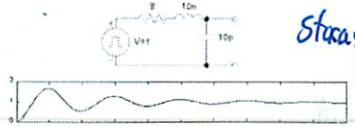


[Dally]Fig. 6-19



[Dally]Fig. 6-20

- RLC ringing (triggered by voltage "steps")



stray reactances

Fix: slower operation, limiting voltage swings and slew rates

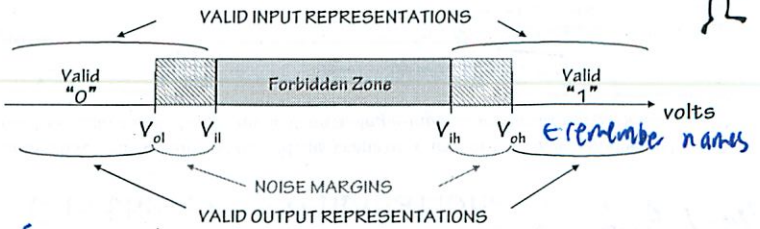
stray capacitance

performance cost

Needed: Noise Margins!

Does a wire obey the static discipline?

No! A combinational device must restore marginally valid signals. It must accept marginal inputs and provide unquestionable outputs (i.e., to leave room for noise).

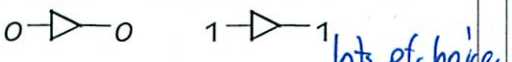


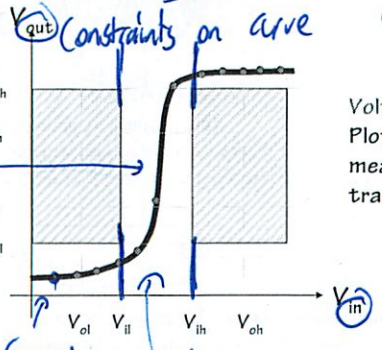
remember names

Can accept in margin but can't output in margin - must restore

A Buffer

A simple *combinational device*:





Voltage Transfer Characteristic (VTC):
Plot of V_{out} vs. V_{in} where each measurement is taken after any transients have died out.

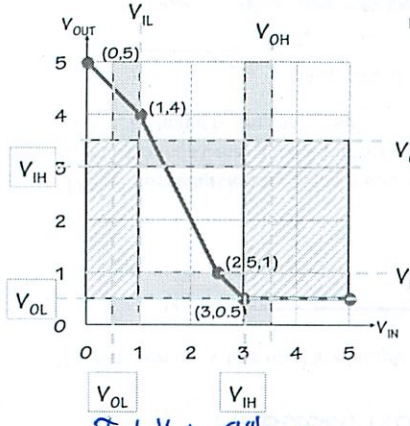
Note: VTC does not tell you anything about how fast a device is — it measures static behavior not dynamic behavior

Static Discipline requires that the VTC avoid the shaded regions (aka "*forbidden zones*"), which correspond to *valid* inputs but *invalid* outputs.

Net result: combinational devices must have GAIN > 1 and be NONLINEAR.

Can this be a combinational device?

Suppose that you measured the voltage transfer curve of the device shown below. Could we build a logic family using it as a single-input combinational device?



Hmmm, it had better be an *INVERTER*...

The device must be able to actually produce the desired output level. Thus, V_{OL} can be no lower than 0.5 V.

Try V_{OL} = 0.5 V

V_{IH} must be high enough to produce V_{OL}

Try V_{IH} = 3 V

Now, choose noise margins – find an N and set

$V_{OH} = V_{IH} + N$

$V_{IL} = V_{OL} + N$

Such that

V_{IH} IN generates V_{OL} or less out; AND

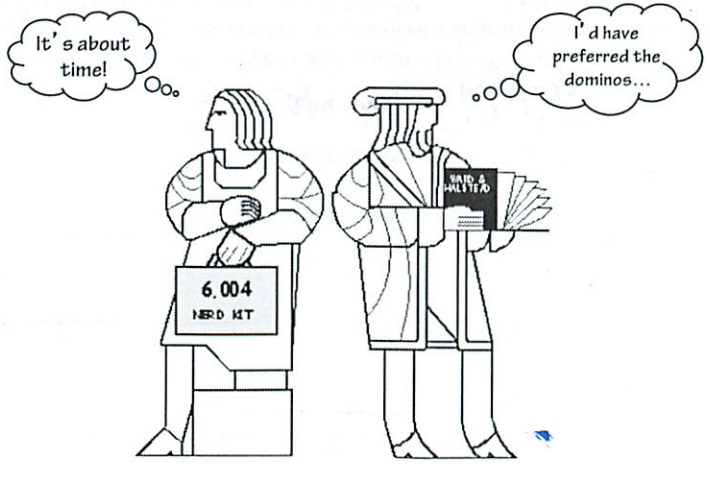
V_{IL} IN generates V_{OH} or more out.

Try N = 0.5 V

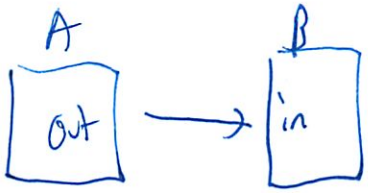
Summary

- Use voltages to encode information
- "Digital" encoding
  - valid voltage levels for representing "0" and "1"
  - forbidden zone avoids mistaking "0" for "1" and vice versa
  - gives rise to notion of signal VALIDITY.
- Noise
  - Want to tolerate real-world conditions: NOISE.
  - Key: tougher standards for output than for input
  - devices must have gain and have a non-linear VTC
- Combinational devices
  - Each logic family has Tinkertoy-set simplicity, modularity
  - predictable composition: "parts work → whole thing works"
  - static discipline
    - digital inputs, outputs; restore marginal input voltages
    - complete functional spec
    - valid inputs lead to valid outputs in bounded time

Next time:
Building Logic w/ Transistors



6.004 Recitation 2



binary signaling scheme 0 or 1

- voltage to encode base

0 : 0V

1 : 1V

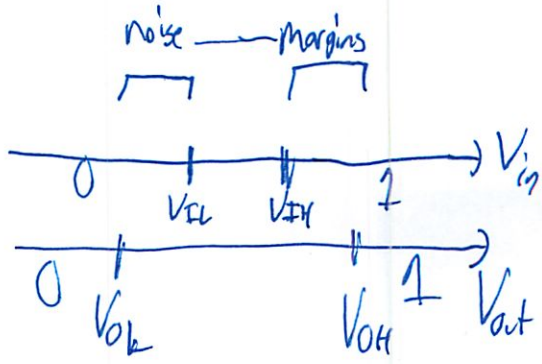
In real world, things not perfect

- noise on wire
- imperfect components
- environmental
  - temp

Have seperate specs for in + out

	in	out
0	$\leq V_{OL}$	$\leq V_{IL}$
1	$\geq V_{OH}$	V_{IH}

input "high"

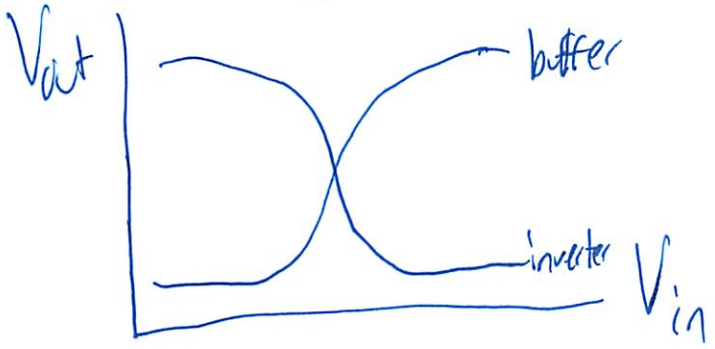


②

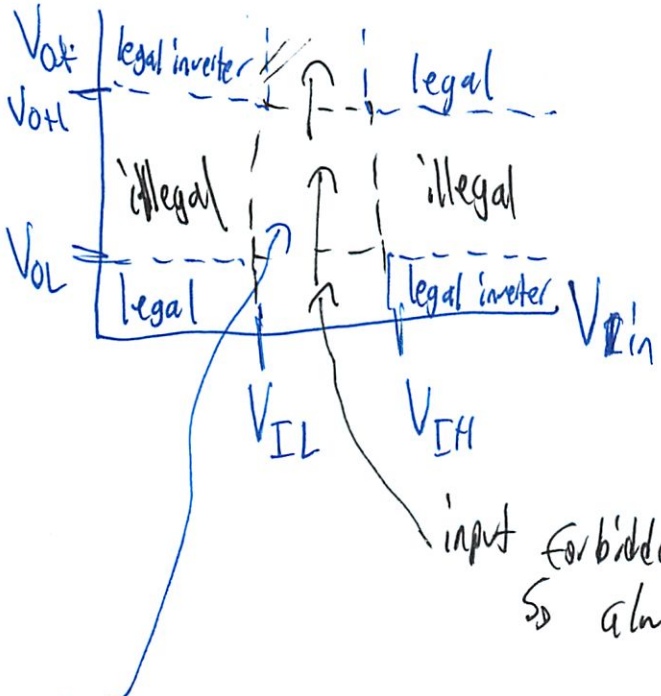
Noise margin for allowing noise on wire b/w components

Perfect

Build Voltage transfer curve



~~valid~~ invalid in > valid at



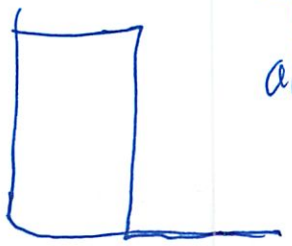
input forbidden so output unspecified
So always legal

Center box is taller than is wider since noise margin
- because of this gain must be > 1

(3)

- So the device must be powered
  - and non-linear
  - next lecture → transistors

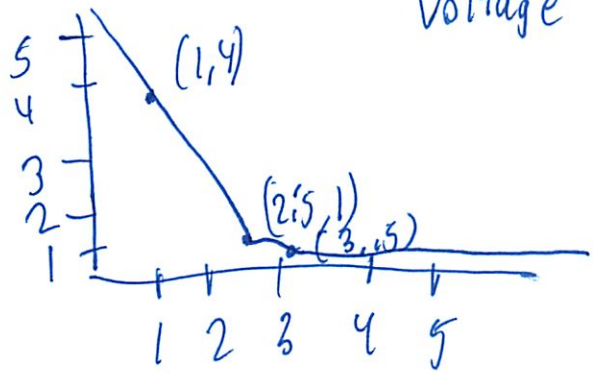
Perfect



- CMOS almost has this

Problems

Voltage transfer curve



What are we trying to do?

Find V_{IL}, etc so it works w/ graph

- can do fat + narrow
- or tall + skinny

*But H > W

$$V_{OL} < V_{IL}$$
$$V_{OH} > V_{IH}$$

Could we pick $V_{OL} = 0V$?

no does not work w/ device

(4)

must be
$V_{OL} \geq .5V$

↓ if $V_{OL} = .5V$

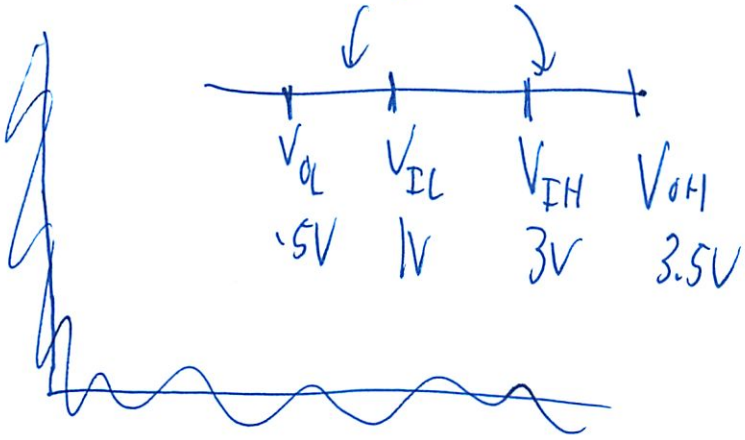
$V_{IH} \geq 3$

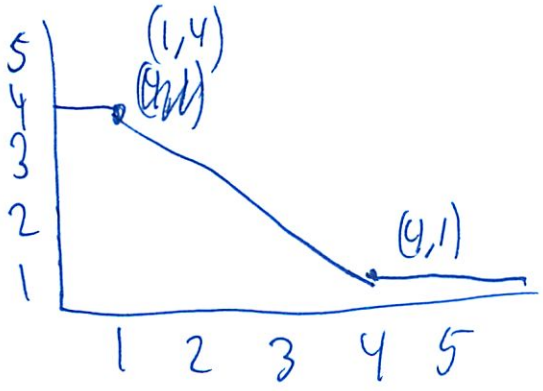
↓ then we can see $V_{OH} = 3.5$

$V_{IL} = 1V$

← So first we are gathering constraints

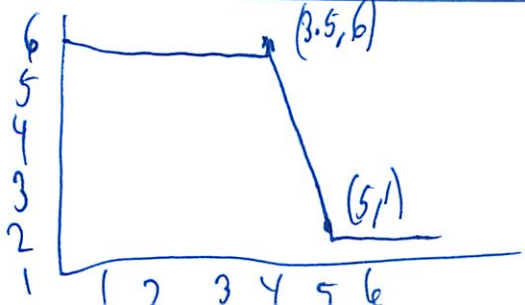
don't need to be symmetric
- can do some optimization





Not valid since max gain = 1
So 0 noise margin

Must be > 1



Could we draw noise margins?

(5)

Yes - and now what are they?

$V_a = 1$

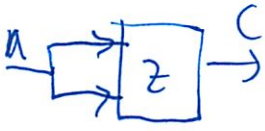
etc

What's the goal?

- depends on problem:
  - max noise margin
  - is it legal?
  - or just a legal noise margin

Weird device

$$V_C = \begin{cases} V_A \cdot V_B & \text{prod} \leq 2 \\ 2 & \text{prod} \geq 2 \end{cases}$$



Would these be legal choices?

V_{OL}	V_{IL}	V_{IH}	V_{OH}
1.5	.16	1.7	1.8

(6)

So is this device a legal buffer?

in	out
0	0
1	1

~~Yes~~

(X) No

V_{OL}	V_{IL}	V_{IH}	V_{OH}
.7	.9	1.1	1.3

Plug in a valid low voltage $0 \rightarrow .9$

- will I get something $< .7$?

So $.9 \cdot .9 = .81$

which is $> .7$ so (X) No

V_{OL}	V_{IL}	V_{IH}	V_{OH}
1.1	.3	1.7	1.9

$.3 \cdot .3 = .09$ which is $< .1$ ← small noise margin

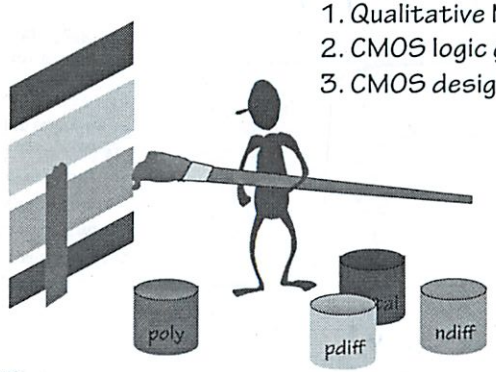
$1.7 \cdot 1.7 = 2$ > 1.9 .2 Volts

Works

9/15

CMOS Technology

1. Qualitative MOSFET model
2. CMOS logic gates
3. CMOS design issues

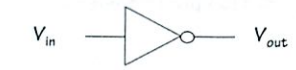


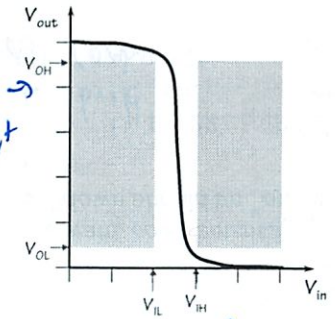
NEXT WEEK:
- THU: Lab 1 due!
- FRI: QUIZ 1!!!

Representing bit as voltage

Combinational Device Wish List

must gain > 1





more stringent →

- ✓ Design our system to tolerate some amount of error
  ⇒ Add positive noise margins
  ⇒ VTC: gain > 1 & nonlinearity — make up for horiz section
- ✓ Lots of gain ⇒ big noise margin
- ✓ Cheap, small
- ✓ Changing voltages will require us to dissipate power, but if no voltages are changing, we'd like zero power dissipation
- ✓ Want to build devices with useful functionality (what sort of operations do we want to perform?)

must be active

restore marginally valid bits

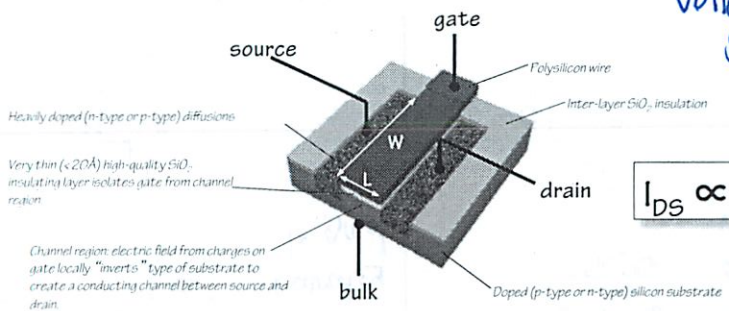
Shaded - has valid 'ins but not valid outs

everything on the wishlist

4 terminals

MOSFETS: Gain & non-linearity

voltage controlled gate



$I_{DS} \propto W/L$

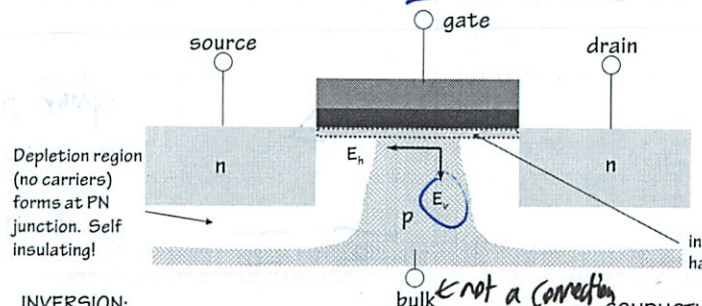
if V on gate > than source drain

MOSFETs (metal-oxide-semiconductor field-effect transistors) are four-terminal voltage-controlled switches. Current flows between the diffusion terminals if the voltage on the gate terminal is large enough to create a conducting "channel", otherwise the mosfet is off and the diffusion terminals are not connected.

physics beyond this class

FETs as switches

The four terminals of a Field Effect Transistor (gate, source, drain and bulk) connect to conductors that generate a complicated set of electric fields in the channel region which depend on the relative voltages of each terminal.



dielectric

not a connection

INVERSION:
A sufficiently strong vertical field will attract enough electrons to the surface to create a conducting n-type channel between the source and drain. The gate voltage when the channel first forms is called the *threshold voltage* -- the mosfet switch goes from "off" to "on".

CONDUCTION:
If a channel exists, a horizontal field will cause a drift current from the drain to the source.

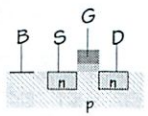
conducting channel

Voltage differential

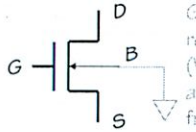
Same manuf process

FETs come in two flavors

NFET: n-type source/drain diffusions in a p-type substrate. Positive threshold voltage; inversion forms n-type channel

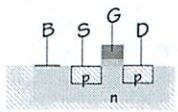


Very good Connecting to ground

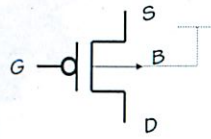


Connect B to GND to keep PN reverse-biased (Vp < Vn); keeps D and S insulated from B

PFET: p-type source/drain diffusions in a n-type substrate. Negative threshold voltage; inversion forms p-type channel.



Very good at selectively connecting wire to power supply



Connect B to VDD to keep PN reverse-biased

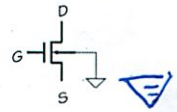
The use of both NFETs and PFETs – complimentary transistor types – is a key to CMOS (complementary MOS) logic families.

CMOS Recipe

If we follow two rules when constructing CMOS circuits then we can model the behavior of the mosfets as simple switches:

Rule #1: only use NFETs in pulldown circuits (paths from output node to GND)
Rule #2: only use PFETs in pullup circuits (paths from output node to V_{DD})

Only 3 terminals we care about



Power supply

NFET Operating regions:

"off": $V_G < V_{TH,NFET}$ S — D

Few tenths of volt

"on": $V_G > V_{TH,NFET}$ S — "resistor" — D

$\sim V_{DD}/5$

Small gate resistance

PFET Operating regions:

"off": $V_G > V_{DD} + V_{TH,PFET}$ S — D

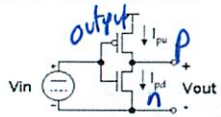
"on": $V_G < V_{DD} + V_{TH,PFET}$ S — "resistor" — D

$\sim -V_{TH,NFET}$

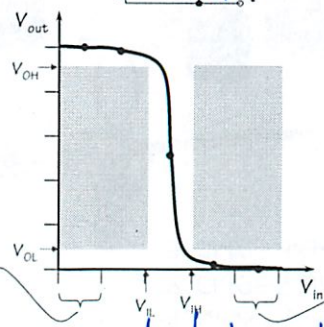
Can completely control voltage on wire b/w high + gnd

CMOS Inverter VTC

typical CMOS inverter



Steady state reached when V_{out} reaches value where $I_{pu} = I_{pd}$.



When V_{IN} is low, the nfet is off and the pfet is on, so current flows into the output node and V_{OUT} eventually reaches V_{DD} (= V_{OH}) at which point no more current will flow.

pfet "on" nfet "off"

When V_{IN} is high, the pfet is off and the nfet is on, so current flows out of the output node and V_{OUT} eventually reaches GND (= V_{OL}) at which point no more current will flow.

pfet "off" nfet "on"

When VIN is in the middle, both the pfet and nfet are "on" and the shape of the VTC depends on the details of the devices' characteristics. CMOS gates have very high gain in this region (small changes in V_{IN} produce large changes in V_{OUT}) and the VTC is almost a step function.

not specified - both transistors could be on so path power to ground

So want to avoid this region - transistors could melt!

Beyond Inverters: Complementary pullups and pulldowns

Now you know what the "C" in CMOS stands for!

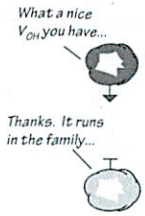
We want *complementary* pullup and pulldown logic, i.e., the pulldown should be "on" when the pullup is "off" and vice versa.

Can build all combos

pullup	pulldown	F(A1,...,An)
on	off	driven "1"
off	on	driven "0"
on	on	driven "X"
off	off	no connection

Since there's plenty of capacitance on the output node, when the output becomes disconnected it "remembers" its previous voltage -- at least for a while. The "memory" is the load capacitor's charge. Leakage currents will cause eventual decay of the charge (that's why DRAMs need to be refreshed!).

takes while for charge to leak



Can combine for more interesting gates

CMOS complements

N P

pull down

conducts when V_{GS} is high

conducts when V_{GS} is low

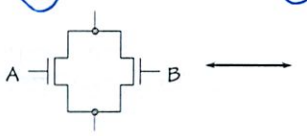
pull up

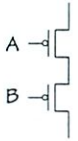
The dual

Complement

conducts when A is high and B is high: $A \cdot B$

conducts when A is low or B is low: $\overline{A}+\overline{B} = \overline{A \cdot B}$





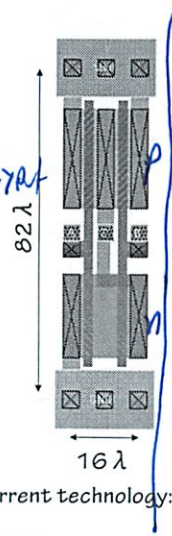
conducts when A is high or B is high: $A+B$

conducts when A is low and B is low: $\overline{A} \cdot \overline{B} = \overline{A+B}$

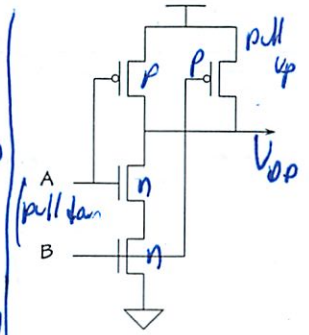
logical Os

actual CMOS gate

A pop quiz!



mask layer



What function does this gate compute?

truth table

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

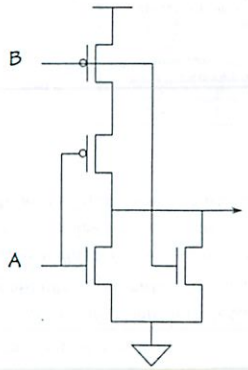
NAND

need to check pull up as well

COST:
- $3500 per 300mm wafer
- 300mm round wafer = $\pi(150e^{-3})^2 = .07m^2$
- NAND gate = $(82)(16)(45e^{-9})^2 = 2.66e^{-12}m^2$
- $2.6e^{10}$ NAND gates/wafer (= 100 billion FETS!)
- marginal cost of NAND gate: 132n$

Current technology: λ = 45nm

Here's another...



What function does this gate compute?

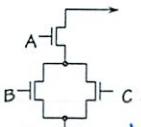
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

NOR

Give specific the same as Windows X

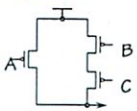
General CMOS gate recipe

Step 1. Figure out pulldown network that does what you want, *e.g.*, $\overline{F} = A \bullet (B+C)$ (What combination of inputs generates a low output)



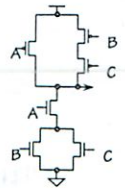
must be duals - look at other on

Step 2. Walk the hierarchy replacing nfets with pfets, series subnets with parallel subnets, and parallel subnets with series subnets



must be complementary - only 1 conducts

Step 3. Combine pfet pullup network from Step 2 with nfet pulldown network from Step 1 to form fully-complementary CMOS gate.



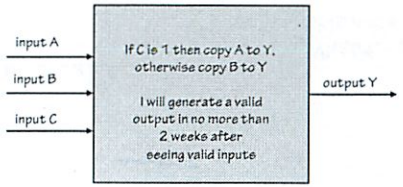
So, whats the big deal?

2 issues: timing, heat

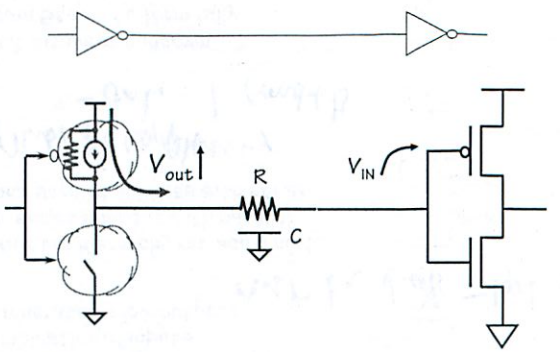
A Quick Review

- A *combinational device* is a circuit element that has
  - one or more digital *inputs*
  - one or more digital *outputs*
  - a *functional specification* that details the value of each output for every possible combination of valid input values
  - a *timing specification* consisting (at minimum) of an upper bound t_{PD} on the required time for the device to compute the specified output values from an arbitrary set of stable, valid input values

Static discipline



Big Issue 1: Time



Things don't happen instantly

Wire delays:

$\tau_{RC} \approx 50$ps/mm

Implies > 1 ns to traverse a 20mm x 20mm chip

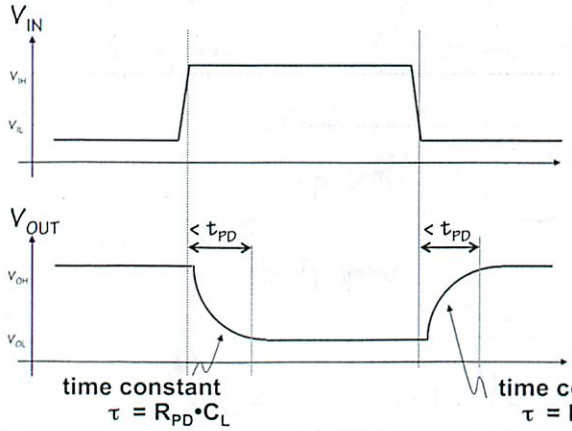
This is a long time in a 2GHz processor

Stray resistance, capacitance

Due to unavoidable delays…

Propagation delay (t_{PD}):

An UPPER BOUND on the delay from valid inputs to valid outputs.



GOAL:

minimize propagation delay!

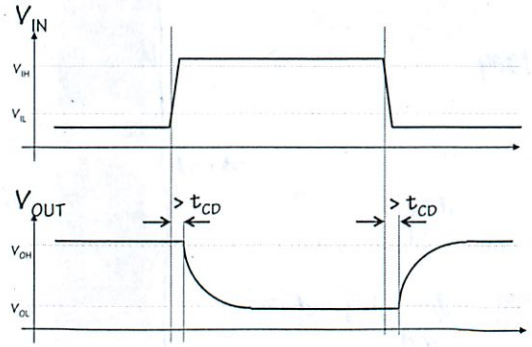
ISSUE:

keep Capacitances low and transistors fast

Contamination Delay

an optional, additional timing spec

INVALID inputs take time to propagate, too...



Do we really need t_{CD}?

Usually not… it'll be important when we design circuits with registers (coming soon!)

If t_{CD} is not specified, safe to assume it's 0.

CONTAMINATION DELAY, t_{CD}

A LOWER BOUND on the delay from any invalid input to an invalid output

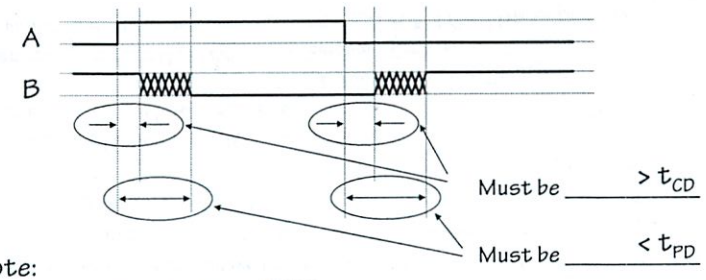
fine till output becomes invalid

The Combinational Contract

A ▷o B

A	B
0	1
1	0

t_{PD} propagation delay
t_{CD} contamination delay



Note:
1. *No Promises* during ≋
2. Default (conservative) spec: $t_{CD} = 0$

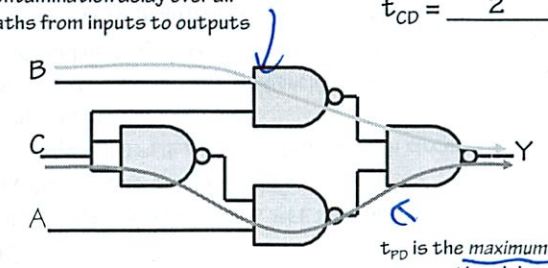
Acyclic Combinational Circuits

If NAND gates have a $t_{PD} = 4nS$ and $t_{CD} = 1nS$

t_{PD} = 12 nS

t_{CD} = 2 nS

t_{CD} is the *minimum* cumulative contamination delay over all paths from inputs to outputs

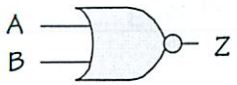


t_{PD} is the *maximum* cumulative propagation delay over all paths from inputs to outputs

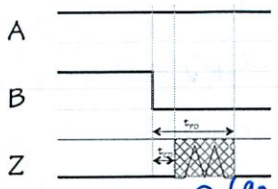
longest path

Oh yeah... one last timing issue

NOR:



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



↑ Can produce anything

1 Old info

2. Then prop delay

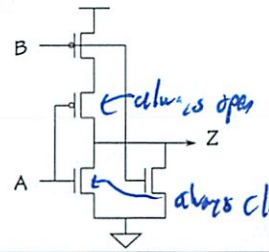
Recall the rules for *combinational devices*:

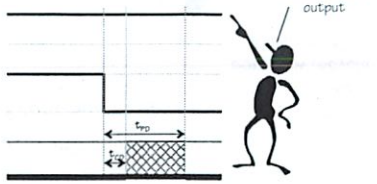
Output guaranteed to be valid when *all* inputs have been valid for at least t_{PD}, and, outputs may become invalid no earlier than t_{CD} after an input changes!

Many gate implementations--e.g., CMOS—adhere to even tighter restrictions.

What happens in this case?

CMOS NOR:





Input A alone is sufficient to determine the output

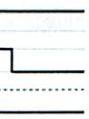
LENIENT Combinational Device:
Output guaranteed to be valid when any combination of inputs sufficient to determine output value has been valid for at least t_{PD}.
Tolerates transitions -- and invalid levels -- on irrelevant inputs!

NOR:

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

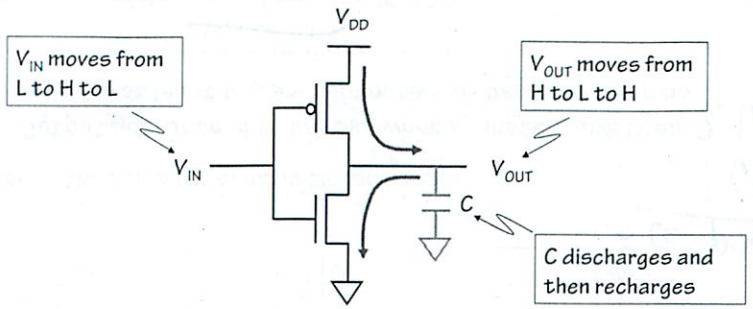
Lenient NOR:

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



Can be anything

Big Issue 2: Power



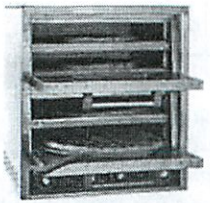
Energy dissipated = $C V_{DD}^2$ per cycle every time cycle

Power consumed = $f n C V_{DD}^2$ per chip value

where f = frequency of charge/discharge
n = number of gates /chip

on change

Unfortunately…



32 Amps (@220v)

Modern chips (UltraSparc III, Power4, Itanium 2) dissipate from 80W to 150W with a Vdd = 1.2V (Power supply current is = 100 Amps)

Cooling challenge is like making the filament of a 100W incandescent lamp cool to the touch!

MIT Computation Center and Pizzeria

Worse yet…
- Little room left to reduce Vdd
- nC and f continue to grow

MUST computation consume energy?

(a tiny digression…)

How energy-efficient can we make a gate? It seems that switching the input to a NAND gate will always dissipate some energy…

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

NAND GATE: 2 bits → 1 bit *(information Loss!)*

Landauer's Principle (1961): *discarding information* is what costs energy!

Bennett (1973): Use rev… bound to energy use!

Landauer: 1 bit = K·T·log(2)
Einstein: E = M·C²
⇒ 1.45 * 10^{37} bits/pound?

A	B	P	Q
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

FEYNMAN GATE: 2 bits → 2 bits *(information Preserving!)*

Bennett, Fredkin, Feynman, others: Computer systems constructed from info-preserving elements.

Theory: NO lower bound on energy use!

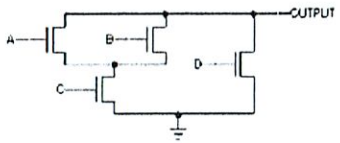
Practice: Research frontier (qubits, etc.)

Summary

- CMOS
  - Only use NFETs in pulldowns, PFETs in pullups → mosfets behave as voltage-controlled switches
  - Series/parallel Pullup and pulldown switch circuits are complementary
  - CMOS gates are naturally inverting (rising input transition can only cause falling output transition, and vice versa).
  - "Perfect" VTC (high gain, $V_{OH} = V_{DD}$, V_{OL} = GND) means large noise margins and no static power dissipation.
- Timing specs
  - t_{PD}: upper bound on time from valid inputs to valid outputs
  - t_{CD}: lower bound on time from invalid inputs to invalid outputs
  - If not specified, assume $t_{CD} = 0$
  - Lenient gates: output unaffected by some input transitions
- Next time: logic simplification, other canonical forms

CMOS technology

Problem 1. The following diagram shows a schematic for the pulldown circuitry for a particular CMOS gate:

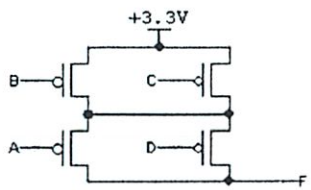


A. ★ What is the correct schematic for the pullup circuitry?

B. ★ Assuming the pullup circuitry is designed correctly, what is the logic function implemented this gate?

C. ★ Assuming the pullup circuitry is designed correctly, when the output of the CMOS gate above is a logic "0", in the steady state what would we expect the voltage of the output terminal to be? What would be the voltage if the output were a logic "1"?

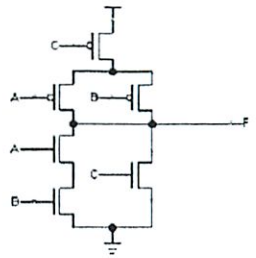
Problem 2. The following diagram shows a schematic for the pullup circuitry for a particular CMOS gate:



A. ★ Draw a schematic for the pulldown circuitry for this CMOS gate.

B. ★ Assuming the pulldown circuitry is designed correctly, give an expression for the logic function implemented by this gate.

Problem 3. Consider the following circuit built from nfets and pfets:



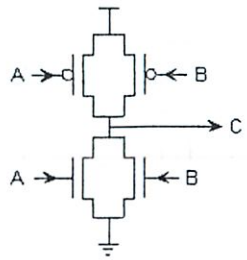
A. ★ Can this circuit be used as a CMOS gate? If not, explain why. If so, what function does it compute?

B. ★ If we wanted the output voltage to change more quickly when going from a logic "0" to a logic "1", what changes would we make to the fets?

Problem 4. Consider the 4-input Boolean function Y = (A*B) + (C*D) where "*" is AND and "+" is OR.

A. ★ Implement the function with a single 4-input CMOS gate and an inverter.

Problem 5. Anna Logue, a circuit designer who missed several early 6.004 lectures, is struggling to design her first CMOS logic gate. She has implemented the following circuit:



Anna has fabricated 100 test chips containing this circuit, and has a simple testing circuit which allows her to try out her proposed gate statically for various combinations of the A and B inputs. She has burned out 97 of her chips, and needs your help before destroying the remaining three. She is certain she is applying only valid input voltages, and expects to find a valid output at terminal C. Anna also keeps noticing a very faint smell of smoke.

A. What is burning out Anna's test chips? Give a specific scenario, including input

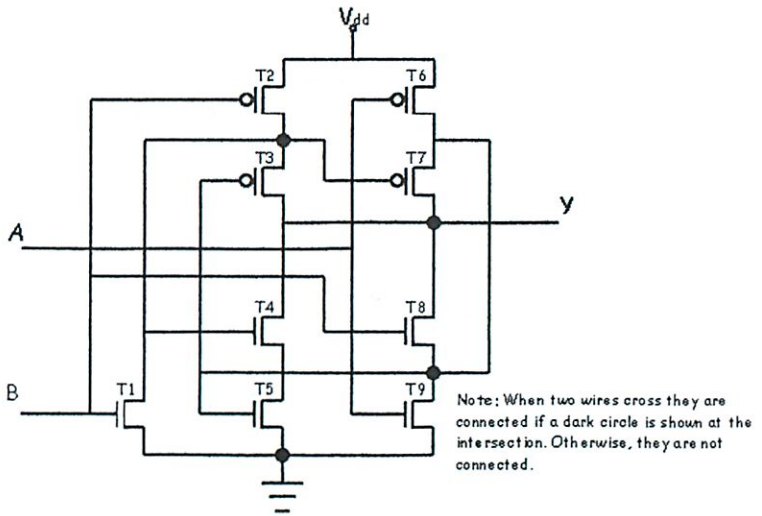
values together with a description of the failure scenario. For what input combinations will this failure occur?

B. Are there input combinations for which Anna can expect a valid output at C? Explain.

C. One of Anna's test chips has failed by burning out the pullup connected to A as well as the pulldown connected to B. Each of the burned out FETs appears as an open circuit, but the rest of the circuit remains functional. Can the resulting circuit be used as a combinational device whose two inputs are A and B? Explain its behavior for each combination of valid inputs.

D. In order to salvage her remaining three chips, Anna connects the A and B inputs of each and tries to use it as a single-input gate. Can the result be used as a single-input combinational device? Explain.

Problem 6. Occasionally you will come across a CMOS circuit where the complementary nature of the n-channel pull-downs and p-channel pull-ups are not obvious, as in the circuit shown below:



A. Construct a table that gives the on-off status of each transistor in the circuit above for all combinations of inputs A and B.

B. Compute the output, Y, for each input combination and describe the function of the above circuit.

Problem 7. In lecture there was a brief overview of the CMOS fabrication process.

A. What keeps the source/drain diffusions of a MOSFET from shorting out to the substrate or to each other?

B. Why does reducing the thickness of the thin oxide layer improve the performance of the mosfets?

C. Why is silicon dioxide (SiO_2) deposited right before a new wiring layer is added to the surface of the wafer?

D. How are connections between the wiring layers made?

E. If one wanted to *increase* I_{DS} for a NFET, how should it's dimensions be changed?

F. Suppose there are two mosfets of width W and length L connected in parallel, i.e., all their terminal connections are identical. Given that I_{DS} of a mosfet is proportional to W/L, what would be the appropriate dimensions for a *single* mosfet that would have the same I_{DS} as the pair connected in parallel?

6.004 Recitation

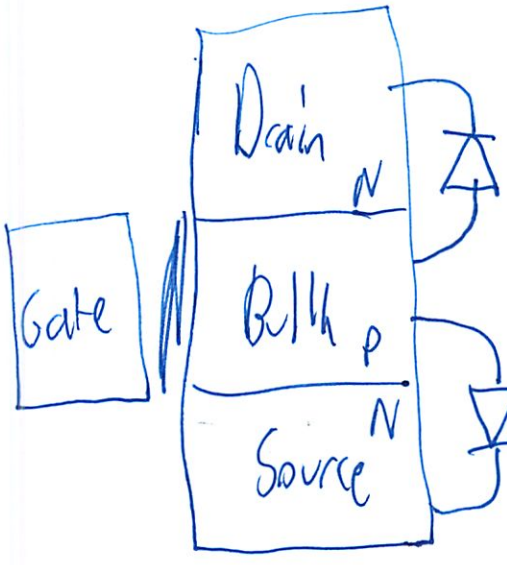
9/16

Eben not Chris today

Today: last material for Quiz 1
- last semester more material
- look at tutorial question

▽ = ⊥ = ground

n-type MOS



diode = conducts current 1 direction

if hook up wrong way get fire

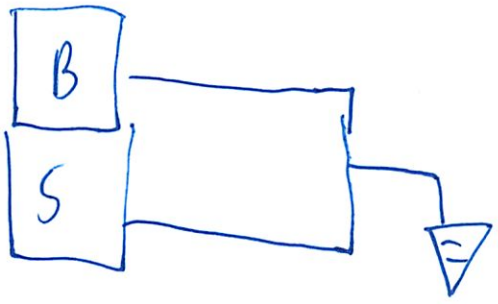
For now no difference b/w drain + source

If bulk Voltage > Source Voltage
- a lot of current flows
- transistor on fire 10^{15} amps

(2)

Control where bulk is connected

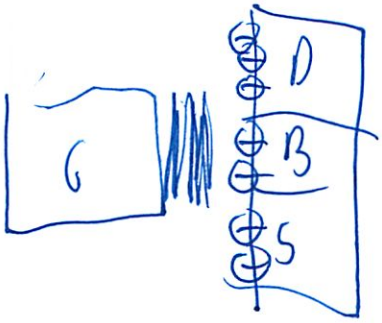
You usally connect bulk to source



Connect everything to ground so no current can flow

Bulk is the most of transistor physically

No current goes into gate - insulated
- but if add high voltage on gate
- draws negitive electron to it
- so conducts current from drain to source



Act like a capacitor
- is capacitance
- but don't need to worry about

(3)

(Makes a lot more sense now)

So has on + off condition

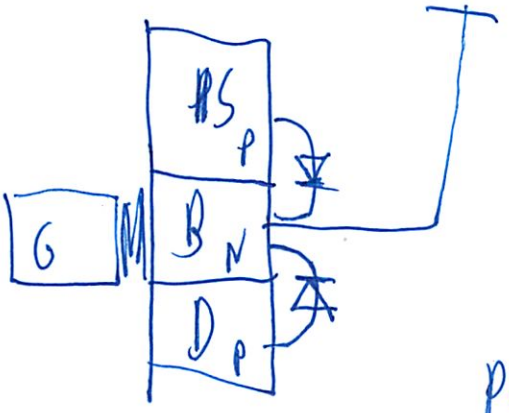
don't think of it as 1 or 0

Can swap Drain + Source

- it always goes from high to low potential
- Standard is high potential / drain at top

"Low potential" is Gate to source (which is ground for us)

P-Mos



tie bulk to high
So no conduction on dioeds
often bulk + source connected

people draw it upside down

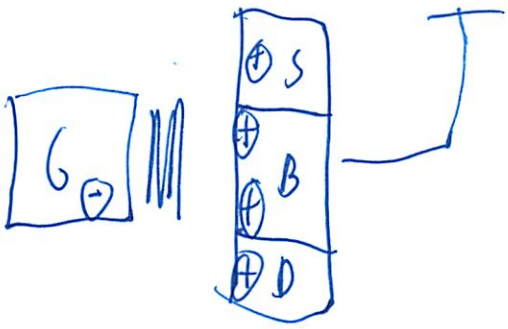
dioeds are inherent → side effect

(review p-n type silicon)

(4)

In this one want to attract (+) charge

So connect gate to low voltage

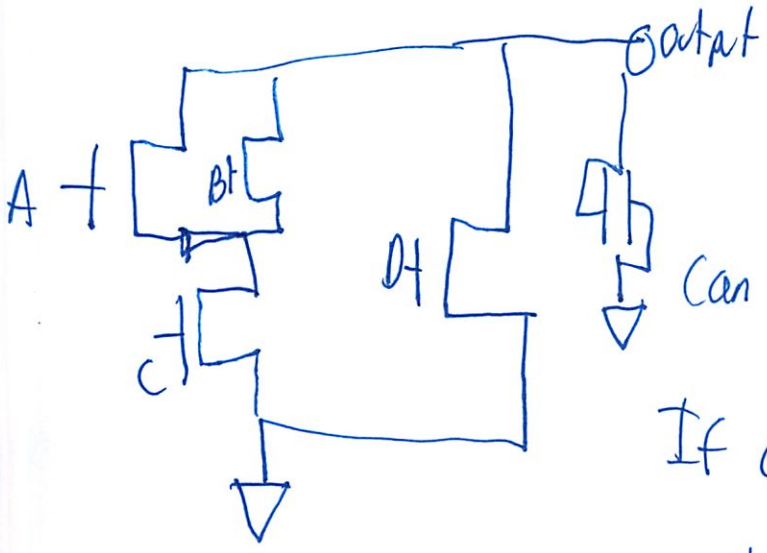


Current flows from high → low

source drain

De Morgan's Law

#1 Pulldown network



n-mos networks

Can knock output to ground

If connect to ground capacitor will discharge to 0

(5) (so confused!)

Also need to charge V_D

(I was thinking that
- must not be as confused
- need to remember 6.01)

When whole thing conducts - its "on" not 1 or 0

A	B	C	D	PD
0	0	0	0	Off ← no gates open
				Since no voltage differential over it
1	1	1	1	on

If wanted to do this w/ P-Mos
would need voltage below ground

~~D · C (A + B)~~

$D \cdot C (A + B)$

↑ the logic formula

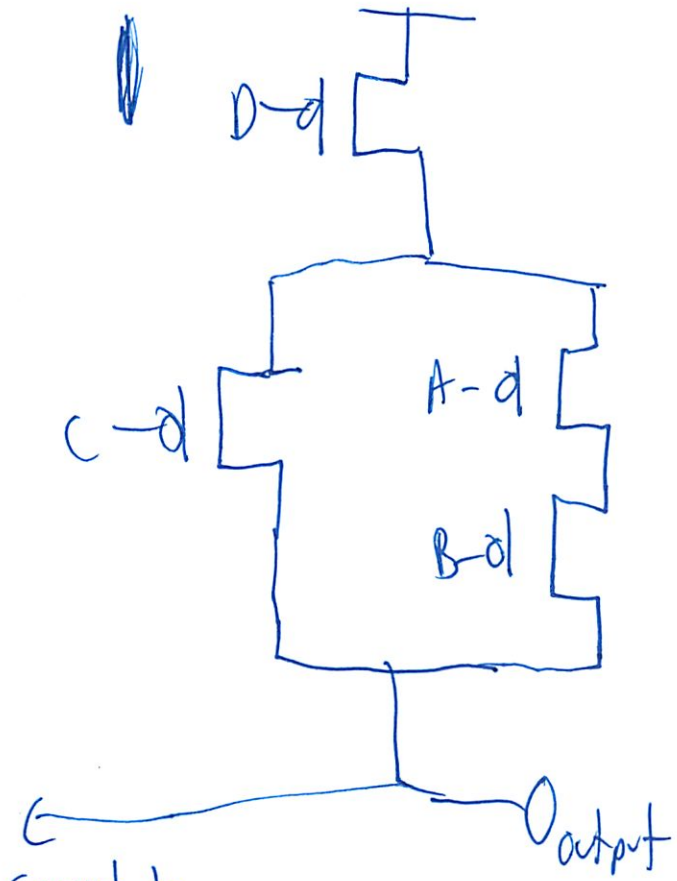
· and
+ or
− negation

can fill
at table

(6)

Pull up network would be opposite on/off

- anything serial, make parallel
- " parallel, " serial



Connect to PD output previous page

Want to describe network as $\bar{A}\ \bar{B}\ \bar{C}\ \bar{D}$

$\bar{D} \cdot (\bar{C} + (\bar{A} \cdot \bar{B})) =$ "on"

↑ d is low (not 0)

(so many diff ways to think about things!

(7)

Call output $\bar{F}$

$\bar{F}$ = "on. PD" $= D + C \cdot (A + B)$

F = "PU on" $= D \cdot (\bar{C} + (\bar{A} \cdot \bar{B}))$

Need to prove are the same via Mr. De Morgan

examples

$$\overline{G + H} = \bar{G} \cdot \bar{H}$$

$$\overline{G \cdot H} = \overline{\bar{G} + H}$$

So 1. invert op And → Or, Or → And
2. invert input
3. invert result

So $\bar{F} = D + C \cdot (A + B)$

$$F = \overline{D + C \cdot (A + B)}$$

$$F = \bar{D} \cdot \underbrace{\overline{C \cdot (A + B)}}_{\text{whole section of Or}}$$

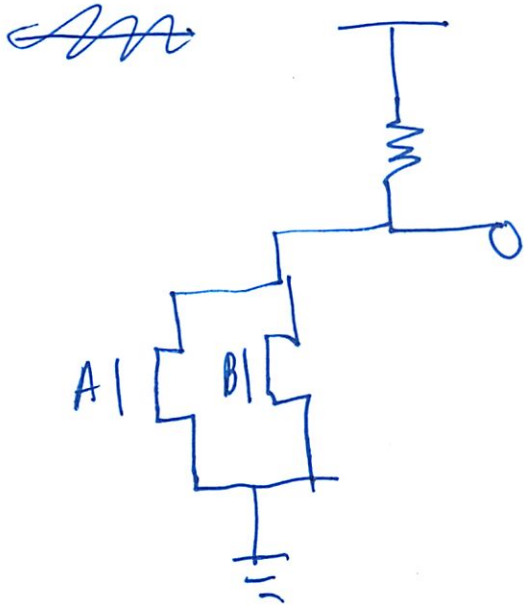
(8)

$F = \bar{D} \cdot (\bar{C} + \overline{A+B})$

$F = \bar{D} \cdot (\bar{C} + (\bar{A} \cdot \bar{B}))$ ← matches what we got w/ graphical cheater network

✓ done - confirmed

Need both to do full logic operation
- need to be complementary
  - reverse it visually
  - confirm w/ DeMorgans



$F = \overline{A+B}$

? if A or B is high it will 0

if output at static high is anything happening

So output high when not A and not B

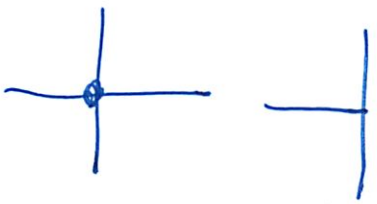
$F = \bar{A} \cdot \bar{B}$

A+B are "externally" controlled
- are wires running to it
- controlled by other parts of circuit

(9)

Note

not connected

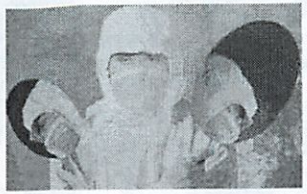


connected

Quiz is ~~[illegible]~~ next Friday

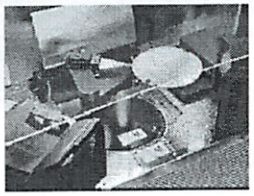
Optional

Let's build a MOSFET



Start with a 500μ slice of a silicon ingot that has been doped with an acceptor (typically boron) to increase the concentration of holes to $10^{14}/cm^3$ - $10^{18}/cm^3$. At room temperature, all the dopants in this *p-type* material are ionized, turning the silicon into a *semiconductor*.

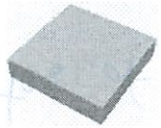
We'll build many copies of the same circuit onto a single wafer. Only a certain percentage of the chips will work; those that work will run at different speeds. The yield decreases as the size of the chips increases and the feature size decreases.



Wafers are processed by automated fabrication lines. To minimize the chance of contaminants ruining a process step, great care is taken to maintain a meticulously clean environment. So put on your bunny suits and let's begin…

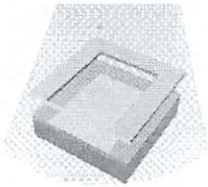
Creating patterns on the wafer

Images from: http://www.intel.com/education/chips/index.htm



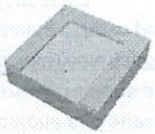
A "thick" (0.4u) layer of SiO_2 is formed by oxidizing the surface of the wafer with wet oxygen (we rust it!). The SiO_2 will serve as insulation between the conductive substrate and subsequent conductive layers we'll build on top of the oxide.

Now we'll form a pattern in the SiO_2 using a mask & etch process. First the wafer is coated with a layer of photoresist. Photoresist becomes soluble when exposed to ultraviolet light… like making t-shirts

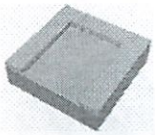


Using a mask to protect parts of the wafer, we'll expose those portions of the wafer where we want to remove the photoresist. We'll use different masks when creating each of the different structures on the wafer.

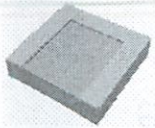
The etching process



The exposed photoresist is removed with a solvent. The unexposed photoresist remains, masking portions of the underlying SiO_2 layer.

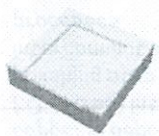


A chemical etch is then used to remove the revealed silicon dioxide.

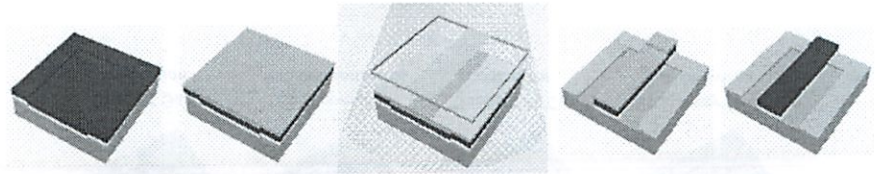


Finally, the remaining photoresist is removed with a different solvent and we're left with pattern of insulating SiO_2 on top of exposed p-type substrate.

Gate oxide & polysilicon



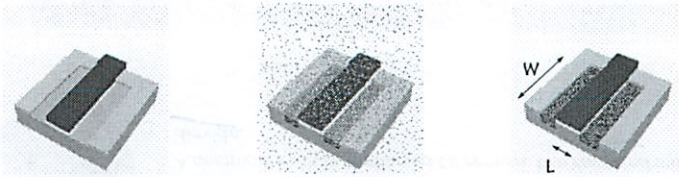
Now a "thin" (20 Å) layer of SiO_2, called *gate oxide*, is grown on the surface. The gate oxide needs to be of high quality: uniform thickness, no defects! The thinner the oxide, the more oomph the FET will have (we'll see why soon) but the harder it is to make it defect-free. Coming soon to a fab near you: 12 Å gate oxide…



On top of the thin oxide a 0.7u thick layer of polycrystalline silicon, called *polysilicon* or *poly* for short, is deposited by CVD. The poly layer is patterned and plasma etched (thin ox not covered by poly is etched away too!) exposing the surface where the source and drain junctions will be formed. Poly has a high sheet resistance of 20 Ω/sq which can be reduced by adding a layer of a silicided refractory metal such titanium ($TiSi_2$), tantalum ($TaSi_2$) or molybdenum ($MoSi_2$) => 1, 3 or 5 Ω/sq.

CVD?

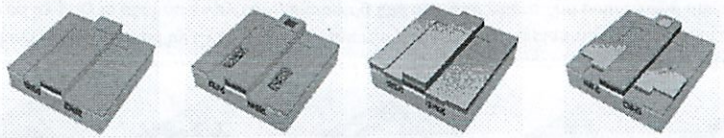
Source/drain diffusions



Donor implants are used to create self-aligned MOSFET source/drain diffusions and substrate contacts. Usually As is preferred to obtain shallow N-type diffusions and minimal lateral diffusion. High doses are needed to make low resistance (25 Ω/sq) diffusion wires. Afterwards a short thermal annealing step is performed to repair surface damage caused by the implantation.

This completes the construction of the MOSFET itself. Now we'll add the metal wiring layers…

Wires: metal interconnect

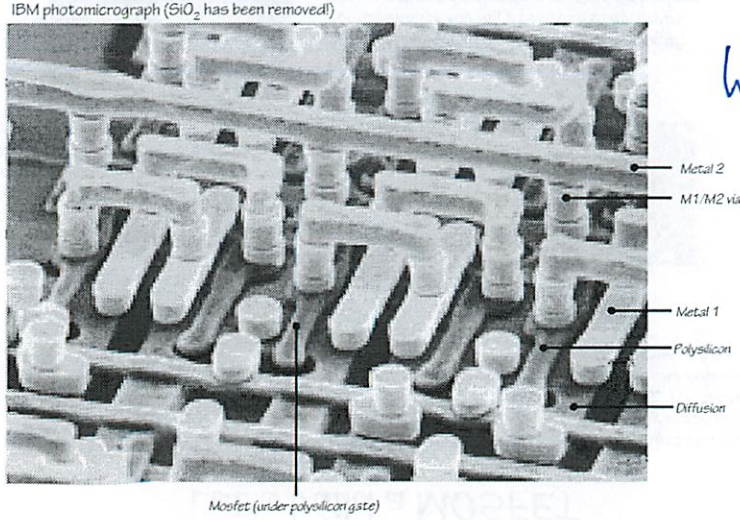


Deposit SiO_2 insulation | Etch openings for vias/contacts | Deposit Al/Cu conductor | Etch away Al/Cu leaving wires

After a layer of SiO_2 insulation has been deposited, aluminum or copper is deposited, patterned, then etched to form low-resistance (.07 Ω/sq) interconnect. With planarization of the SiO_2 (a mechanical polishing step that creates a flat surface), multiple levels of metal interconnect are possible -- 6 to 8 layers are common in today's processes.

Multiple interconnect layers

IBM photomicrograph (SiO_2 has been removed!)



WOW

nice camera

tons of engineering

CMOS Review on own via Internet/Wikipedia 9/18

CMOS - the system

MOSFET - actual transistor

Looked it up and that is right

CMOS - has = # of p and n MOSFETS
- way of design circuits

FET - transistor

MOS - adds insulator so uses less current

field effect transistor - relies on electric field to control shape and thus conductivity of channel in a semiconductor material

Source - carriers enter channel
Drain - " leave
) V_{DS}

Gate - controls V_{DS}, current is I_G

(I need to relearn all my electricty terms)
(w/ the mess of wikipedia extra info)

② Remember Voltage = pressure
Current = flow

Voltage difference in electrical potential
Work to do against static electric field to move charge

Electric Charge (coulomb)
property of matter that causes matter to experience a force when near other electrically charged matter

Current - flow of electric charge thrag medium (ampere/amp)

~~Semiconductor~~

Transistor - semiconductor to amplify + swich electric signals
(i Have I ever ~~read~~ learned??)
At least 3 terminals
Voltage or current to one pair of terminals changes current flowing thrayh other terminals
If output > input can amplify
Thinking of why needed - not just normal circuit (minor)
But is it mostly good to control other parts so can make complicated (MOS - state systems

(Making connections - read 1st sentence closly)

(3)

So 1st transistor was FET

? so that w/ field is most basic/simplest way
not just most efficent

Needs semiconductors ~1945

What before that?

kicked off low cost mass computers

Replaced · electronechanical devices

General purpose Micro controller easier than building basic circic

[Oh so now I kinda understand hardware/why general purpose PC important (epiphany)

[This weekend I am learning a lot - but not getting P-Sets done)

Vaccume tubes Compuison

smaller, easier to build
lower voltage
No warm up
higher reliabiy
insetive for shaky

but no >1600 V
high power, high freq broadcasting worse
Vulnerable to nuclear explosions

(4)

Vacuum Tubes device that relies on flow of electrons current through a vacuum

- amplification or switching
- thermonic emission of electrons from a hot filament (cathod) to annode (plate)
- add. electrodes can amplify + swich
- before that mechical computers

Semiconductor - material w/ electic conductivity due to electron flow intermediate b/w that of a conductor and a insulator

- Current carried by flow of electrons in wires
- here flow of electrons or flow of (+) charged "holes"
- Silicon most common
- add dopants

Holes

- electron absence as charge carrier
- electrons need to fall into holes
- "the lack of an electron"

(5)

~~MOSFET Transistor~~ ~~Transistor~~

MOSFET

n-type extrinsic semiconductor

provide extra conduction electrons to host material

So excess of negitive (n-type) charge carriers

(Remember 3.091)

↑ But abstractions there were very different

P-type (P = positive)

increase # free charge carriers

takes away weakly bound other electrons

↳ leaves behind holes

↑ lots of them

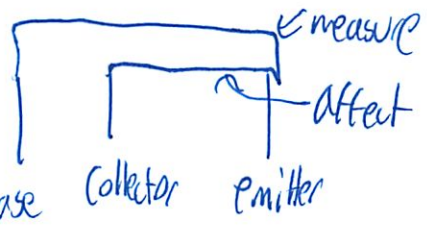
Charge Carriers - ä Free (mobile, unbound) particle carrying an electric charge

- electrons, ions

(6)

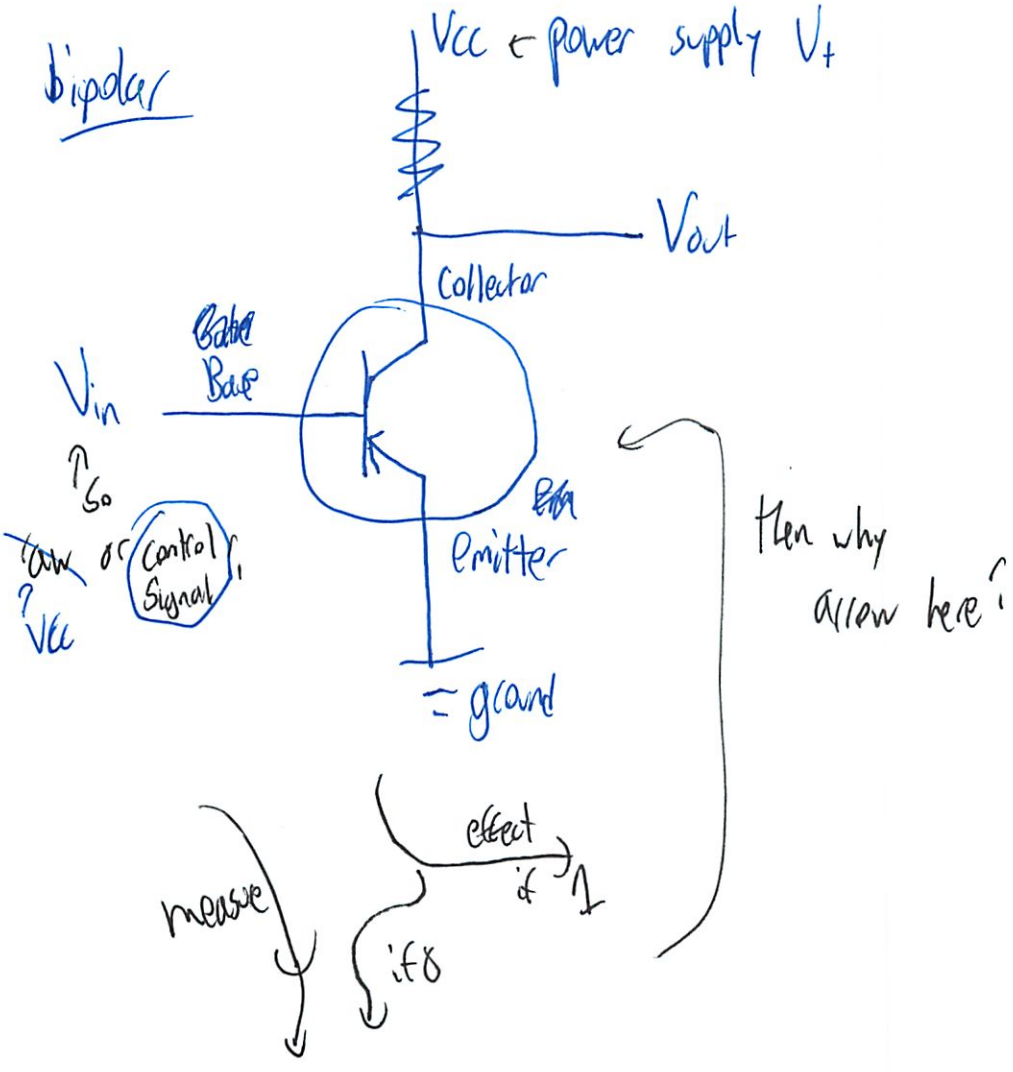
Transistor * ability to ~~sense~~ use small signal applied b/w one pair of terminals to control much larger signal at another pair of terminals * gain

- can be amplifier or switch
- bipolar



- fet gate source drain ← (; so just diff terms?)

bipolar



(1)

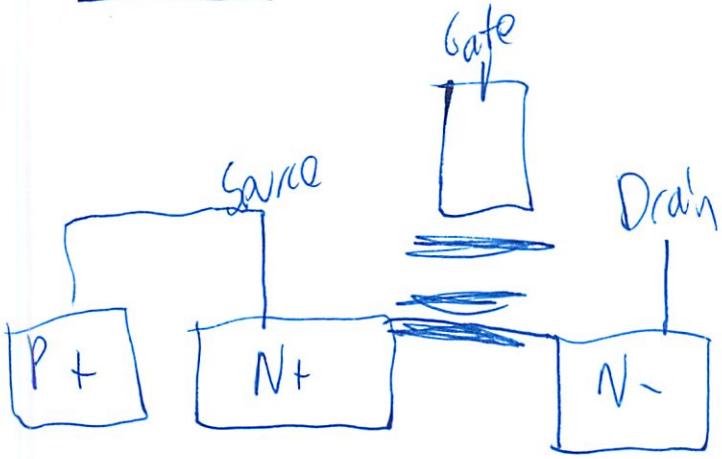
Current can be defined multiple ways

⊕ charge flow high → low potential ← usual

⊖ " " low → high "

↑ but usually electrons flow

MOSFET

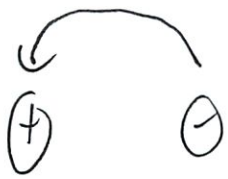


← N-type

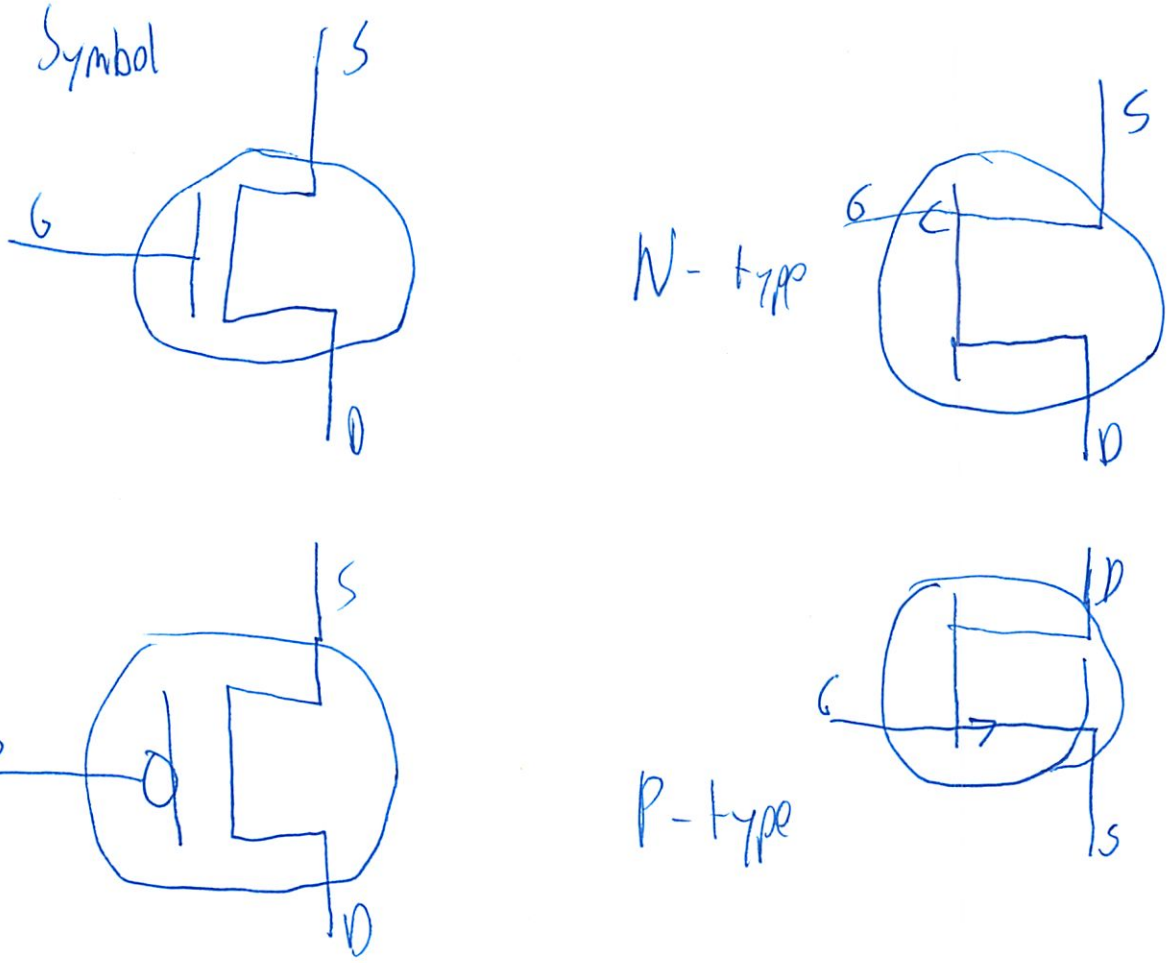
P substrate

When gate is ⊕ attracts electrons so electrons flow b/w n-doped terminals

∴ So from low to high potential



(8)



What about bulk?

- often not shown
- Connects to back
- w/ arrow ⊕ → ⊖
  P → N

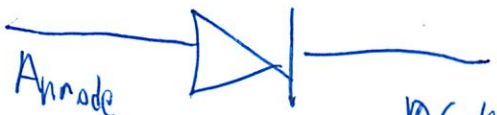
Mosfet Bulk (: what is 4th connection for it can do in 3?)

Oh I don't think its a gate
The base of it

(9)

CMOS Logic that uses p-, n- MOSFETs as building blocks
- no current flows (ideally) so no power consumed
- except when input switched
- connecting both n-, p- gates/drains together
- so one always connects, while the other does not
- otherwise fire as ~~the other one conducts~~ lots of current would flow

Logic Diode - two terminal electronic component w/ non linear current-voltage characteristic
- allows current to pass in 1 direction (forward)
- while blocking



Logic Gate - idealized or physical device implementing a Boolean function
- ideal: zero rise-time (time to go from low to high) 10% → 90%
unlimited fan out

(10)

fan out — the ability of a logic gate out to drive a # of logic gate inputs of same type = $\min\left(\left\lfloor \frac{I_{out\ high}}{I_{in\ high}} \right\rfloor, \left\lfloor \frac{I_{out\ low}}{I_{in\ low}} \right\rfloor\right)$

- perfect ∞ input impedance - "seen"

0 output impedance - exhibited

impedance - measure of opposition to AC

extends context of resistance to AC

in DC impedance = resistance

logic gate - implemented w/ diodes or transistors acting as electric switches

- can also do mechically

diode logic - simplest AND, OR only

- but no NOT / inverter

RTL - logic - resistor, transistor

TTL - transistor, transistor

Now MOSFETs

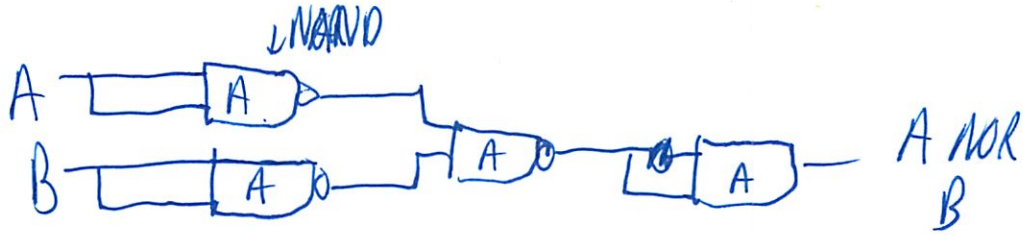
(11)

NAND gate not and

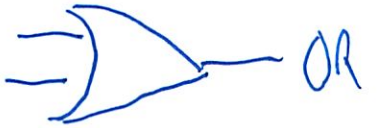
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

easy to construct w/ MOS

NOR



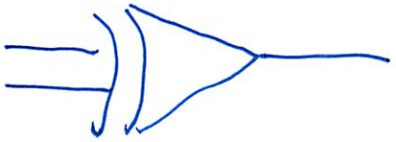
Logic gates All AND, OR, NOT, XOR, XNOR can be built from NAND



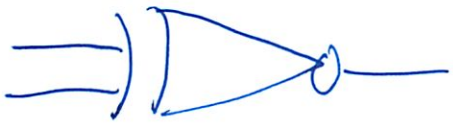
(12)

NAND $\overline{A \cdot B}$

NOR $\overline{A+B}$



XOR $A \oplus B$

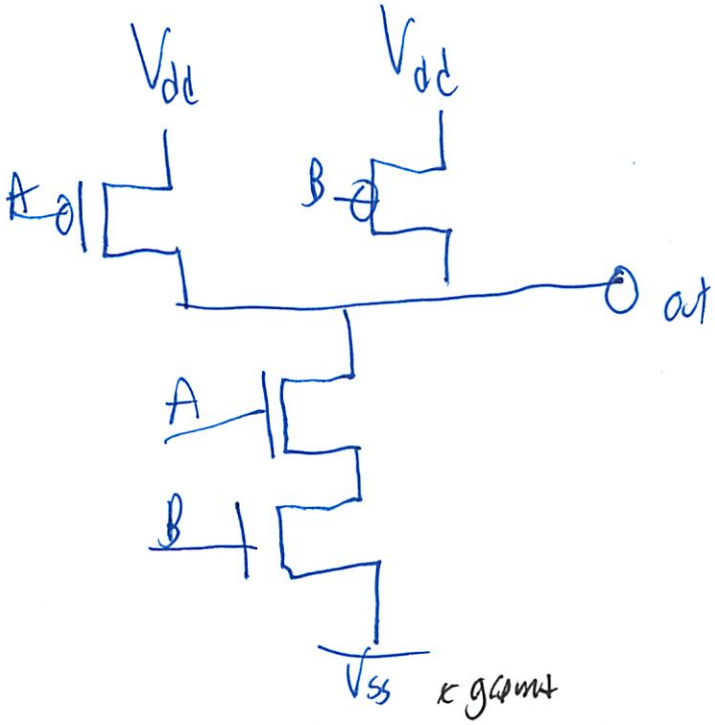


NXOR $\overline{A \oplus B}$ $A \cdot B$

CMOS

All P-MOS need in from P-Mos or V_{cc}
N-Mos V_{cc}

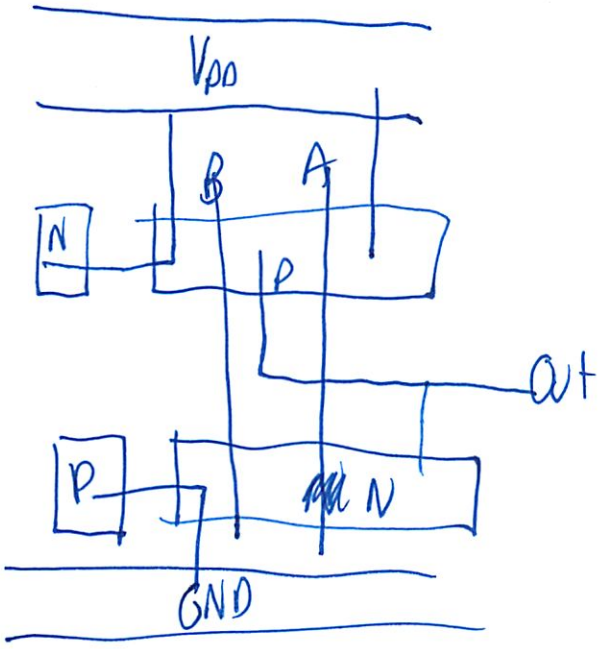
Always path from output ← source V_{cc}
ground



NAND ←

(13)

NAND physically



A, B = poly

Notes

Oh do connect base

N Connect B to Gnd to keep reversed biased $P_v < V_n$, keeps D + S isulated From B

P Connect B to VDP to keep PN reversided biased

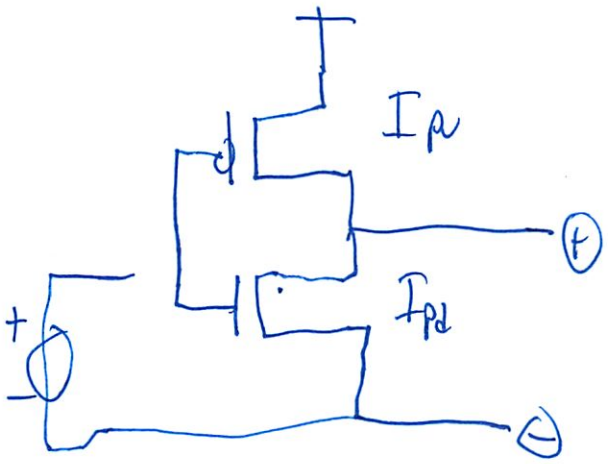
But is rarly drawn - assmed

(14)

N - pull down node to GND

P - Pull up node to V_{DD}

Inverter



steady state

$I_{pu} = I_{pd}$

V_{in} low - nfet off

p fet flows "on"

So Vat reaches V_{DD}

9/10

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Lab #1

Due 9/22

General Information

Lab assignments are due on Thursdays; check the on-line course calendar for the actual due date for each lab. Look at the on-line questions before you start to see what information you should collect while working on each lab. The on-line questions can be loaded from the "On-line assignments" page accessible from the navigation sidebar on the course website (http://6004.csail.mit.edu). If you have difficulties, questions or suggestions with the on-line system please send email to 6004-labs@csail.mit.edu.

You can visit each on-line question page as many times as necessary to complete the assignment – you do not have to answer all the questions in a single session. Click on the "Save" button at the bottom of the question page to save your answers. You can then come back to the page later on and pick up where you left off. You can also check your answers at any time by clicking on the "Check" button. When the system detects that all your answers are correct (either because of a "Check" or "Save"), it will give you credit for completing the assignment.

all must be complete

To receive credit for a lab, you'll need to have a short **lab checkoff meeting** with a member of the course staff and answer some questions about your work. Just come by the lab after you've completed your check-in and talk with one of the on-duty staff. The meeting can happen after the due date of the lab but to receive full credit you must **complete the meeting within one week of the lab due date**. To avoid long waits choose a time other than Wednesday or Thursday evenings.

The lab gets crowded just before an assignment is due and some of the problems are too long to be done the night before the due date, so plan accordingly. There will be course staff in the lab during the late afternoon and evening; check the course website for this semester's schedule.

7755331

The 6.004 lab is located in 32-083 and is open 24 hours-a-day, 7 days-a-week. An access code is required for entry; it was given out in the email that included your section assignment. The lab has Linux-Athena workstations that can be used to complete the homework assignments. The lab software is written in Java and should run on any Java Virtual Machine supporting JDK 1.3 or higher. The courseware can be run on any Linux-Athena workstation. You can also download the courseware for your Linux, Windows, or Macintosh computer – see the "Courseware" page on the 6.004 website.

1: Introduction to JSim

In this lab, we'll be using a *simulation program* (JSim) to make some measurements of an N-channel mosfet (or "nfet" for short). JSim uses mathematical models of circuit elements to make predictions of how a circuit will behave both statically (DC analysis) and dynamically (transient analysis). The model for each circuit element is parameterized, e.g., the mosfet model includes parameters for the length and width of the mosfet as well as many parameters characterizing the physical aspects of the manufacturing process. For the models we are using, the manufacturing parameters have been derived from measurements taken at the integrated circuit fabrication facility and so the resulting predictions are quite accurate.

The (increasingly) complete JSim documentation can be found at the course website. But we'll try to include pertinent JSim info in each lab writeup.

To run JSim, login to an Athena console. We recommend using the computers in the 6.004 lab (32-083) since JSim has been tested and is known to run with satisfactory performance in that environment. Another benefit of using the 6.004 lab is that there's plenty of help around, both from your fellow students and the course staff. After signing onto the Athena station, add the 6.004 locker to gain access to the design tools and model files (you only have to do this once each session):

```
athena% add 6.004
```

Start JSim running in a separate window by typing

```
athena% jsim &
```

It can take a few moments for the Java runtime system to start up, please be patient! JSim takes as input a *netlist* that describes the circuit to be simulated. The initial JSim window is a very simple editor that lets you enter and modify your netlist. You may find the editor unsatisfactory for large tasks—it's based on the JTextArea widget of the Java Swing toolkit that in some implementations has only rudimentary editing capabilities. If you use a separate editor to create your netlists, you can have JSim load your netlist files when it starts:

```
athena% jsim filename … filename &
```

Two limitations of the JSim editor are that **Ctrl-S does not save** and **there is no undo.**

Say what!

There are various handy buttons on the JSim toolbar:

Exit. Asks if you want to save any modified file buffers and then exits JSim.

New file. Create a new edit buffer called "untitled". Any attempts to save this buffer will prompt the user for a filename.

Open file. Prompts the user for a filename and then opens that file in its own edit buffer. If the file has already been read into a buffer, the buffer will be reloaded from the file (after asking permission if the buffer has been modified).

Close file. Closes the current edit buffer after asking permission if the buffer has been modified.

Reload file. Reload the current buffer from its source file after asking permission if the buffer has been modified. This button is useful if you are using an external editor to modify the netlist and simply want to reload a new version for simulation.

Save file. If any changes have been made, write the current buffer back to its source file (prompting for a file name if this is an untitled buffer created with the "new file" command). If the save was successful, the old version of the file is saved with a ".bak" extension.

Save file, specifying new file name. Like "Save file" but prompts for a new file name to use.

Save all files. Like "save file" but applied to all edit buffers.

Stop simulation. Clicking this control will stop a running simulation and display whatever waveform information is available.

Device-level simulation. Use a Spice-like circuit analysis algorithm to predict the behavior of the circuit described by the current netlist. After checking the netlist for errors, JSim will create a simulation network and then perform the requested analysis (i.e., the analysis you asked for with a ".dc" or ".tran" control statement). When the simulation is complete the waveform window is brought to the front so that the user can browse any results plotted by any ".plot" control statements.

Fast transient analysis. This simulation algorithm uses more approximate device models and solution techniques than the device-level simulator but should be much faster for large designs. For digital logic, the estimated logic delays are usually within 10% of the predictions of device-level simulation. This simulator only performs transient analysis.

Gate-level simulation. This simulation algorithm only knows about gates and logic values (instead of devices and voltages). We'll use this feature later in the term when trying to simulate designs that contain too many mosfets to be simulated at the device level.

Switch to waveform window. In the waveform window this button switches to the editor window. Of course, you can accomplish the same thing by clicking on the border of the window you want in front, but sometimes using this button is less work.

What is the difference b/w these?

Using information supplied in the checkoff file, check for specified node values at given times. If all the checks are successful, submit the circuit to the on-line assignment system.

The waveform window shows various waveforms in one or more "channels." Initially one channel is displayed for each ".plot" control statement in your netlist. If more than one waveform is assigned to a channel, the plots are overlaid on top of each using a different drawing color for each waveform. If you want to add a waveform to a channel simply type the appropriate signal name in the list appearing to the left of the waveform display (the name of each signal should be on a separate line). You can also add the name of the signal you would like displayed to the appropriate ".plot" statement in your netlist and rerun the simulation. If you simply name a node in your circuit, its voltage is plotted. You can also ask for the current through a voltage source by entering "I(Vid)".

The waveform window has several other buttons on its toolbar:

Select the number of displayed channels; choices range between 1 and 16.

Print. Prints the contents of the waveform window (in color if you have a color printer!). If you are using Athena, you have to print to a file and then send the file to the printer: select "file" in the print dialog, supply the name you'd like to use for the plot file, then click "print". You can send the file to one of the printers in the lab using "lpr", e.g., "lpr -Pcs foo.plot".

You can zoom and pan over the traces in the waveform window using the control found along the bottom edge of the waveform display:

zoom in. Increases the magnification of the waveform display. You can zoom in around a particular point in a waveform by placing the cursor at the point on the trace where you want to zoom in and typing upper-case "X".

zoom out. Decreases the magnification of the waveform display. You can zoom out around a particular point in a waveform by placing the cursor at the point on the trace where you want to zoom out and typing lower-case "x".

surround. Sets the magnification so that the entire waveform will be visible in the waveform window.

The scrollbar at the bottom of the waveform window can be used to scroll through the waveforms. The scrollbar will be disabled if the entire waveform is visible in the window. You can recenter the waveform display about a particular point by placing the cursor at the point which you want to be at the center of the display and typing "c".

The JSim netlist format is quite similar to that used by Spice, a well-known circuit simulator. Each line of the netlist is one of the following:

A *comment line*, indicated by an "*" (asterisk) as the first character. Comment lines (and also blank lines) are ignored when JSim processes your netlist. You can also add comments at the end of a line by preceding the comment with the characters "//" (C++- or Java-style comments). All characters starting with "//" to the end of the line are ignored. Any portion of a line or lines can be turned into a comment by enclosing the text in "/*" and "*/" (C-style comments).

A *continuation line,* indicated by a "+" (plus) as the first character. Continuation lines are treated as if they had been typed at the end of the previous line (without the "+" of course). There's no limitation on the length of an input line but sometimes it's easier to edit long lines if you use continuation lines. Note that "+" also continues "*" comment lines!

A *control statement*, indicated by a "." (period) as the first character. Control statements provide information about how the circuit is to be simulated. We'll describe the syntax of the different control statements as we use them below.

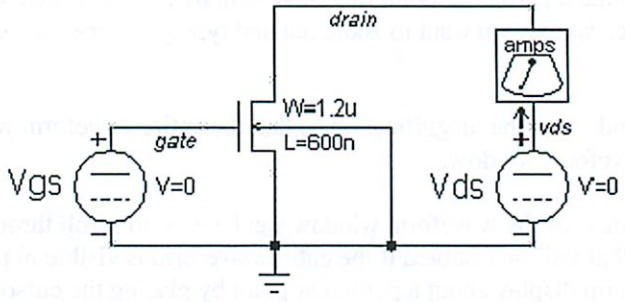
A *circuit element,* indicated by a letter as the first character. The first letter indicates the type of circuit element, e.g., "r" for resistor, "c" for capacitor, "m" for mosfet, "v" for voltage source. The remainder of the line specifies which circuit nodes connect to which device terminals and any parameters needed by that circuit element. For example the following line describes a 1000Ω resistor called "R1" that connects to nodes A and B.

```
R1 A B 1k
```

Note that numbers can be entered using engineering suffixes for readability. Common suffixes are "k"=1000, "u"=1E-6, "n"=1E-9 and "p"=1E-12.

2: Characterizing MOSFETs

Let's make some measurements of an nfet by hooking it up to a couple of voltage sources to generate different values for V_{GS} and V_{DS}:



We've included an ammeter (built from a 0v voltage source) so we can measure I_{DS}, the current flowing through the mosfet from its drain terminal to its source terminal. Here's the translation of the schematic into our netlist format:

```
* plot Ids vs. Vds for 5 different Vgs values
.include "/mit/6.004/jsim/nominal.jsim"
Vmeter vds drain 0v
Vds vds 0 0v
Vgs gate 0 0v
* N-channel mosfet used for our test
M1 drain gate 0 0 NENH W=1.2u L=600n
.dc Vds 0 5 .1 Vgs 0 5 1
.plot I(Vmeter)
```

The first line is a comment. The second line is a control statement that directs JSim to include a netlist file containing the mosfet model parameters for the manufacturing process we'll be targeting this semester. The pathname that's shown will work when running on Athena; if you're running at home you'll need to specify the directory where you downloaded the 6.004 tools. To make your life easier, you may want to copy nominal.jsim and other ".include" files into your Athena directory and use a relative pathname so that you don't have to modify your files when moving them back and forth.

The next three lines specify three voltage sources; each voltage source specifies the two terminal nodes and the voltage we want between them. Note that the reference node for the circuit (marked with a ground symbol in the schematic) is always called "0". The "v" following the voltage specification isn't a legal scale factor and will be ignored by JSim – it's included just remind ourselves that last number is the voltage of the voltage source. All three sources are initially set to 0 volts but the voltage for the Vds and Vgs sources will be changed later when JSim processes the ".dc" control statement.

We can ask JSim to plot the current through voltage sources which is how we'll see what I_{DS} is for different values of V_{GS} and V_{DS}. We could just ask for the current of the VDS voltage source, but the sign would be wrong since JSim uses the convention that positive current flows from the

positive to negative terminal of a voltage source. So we introduce a 0-volt source with its terminals oriented to produce the current sign we're looking for.

The sixth line is the mosfet itself, where we've specified (in order) the names of the drain, gate, source and substrate nodes of the mosfet. The next item names the set of model parameters JSim should use when simulating this device; specify "NENH" to create an nfet and "PENH" for a P-channel mosfet ("pfet"). The final two entries specify the width and length of the mosfet. Note that the dimensions are in microns (1E-6 meters) since we've specified the "u" scale factor as a suffix. **Don't forget the "u" or your mosfets will be meters long!** You can always use scientific notation (e.g., 1.2E-6) if suffixes are confusing.

The seventh line is a control statement requesting a DC analysis of the circuit made with different settings for the Vds and Vgs voltage sources: the voltage of Vds is swept from 0V to 5V in .1V steps, and the voltage of Vgs is swept from 0V to 5V in 1V steps. Altogether 51 * 6 separate measurements will be made.

The eighth and final line requests that JSim plot the current through the voltage source named "Vmeter". JSim knows how to plot the results from the dual voltage sweep requested on the previous line: it will plot I(Vmeter) vs. the voltage of source Vds for each value of voltage of the source Vgs—there will be 6 plots in all, each consisting of 51 connected data points.

After you enter the netlist above, you might want to save your efforts for later use by using the "save file" button. To run the simulation, click the "device-level simulation" button on the tool bar. After a pause, a waveform window will pop up where we can take some measurements. As you move the mouse over the waveform window, a moving cursor will be displayed on the first waveform above the mouse's position and a readout giving the cursor coordinates will appear in the upper left hand corner of the window. To measure the delta between two points, position the mouse so the cursor is on top of the first point. Now click left and drag the mouse (i.e., move the mouse while holding its left button down) to bring up a second cursor that you can then position over the second point. The readout in the upper left corner will show the coordinates for both cursors and the delta between the two coordinates. You can return to one cursor by releasing the left button.

We're now ready to make some measurements:

On-line question 1A:

To get a sense of how well the channel of a turned-on mosfet conducts, let's estimate the effective resistance of the channel while the mosfet is in the linear conduction region. We'll use the Vgs = 5V curve (the upper-most plot in the window). The actual effective resistance is given by $\partial V_{DS}/\partial I_{DS}$ and clearly depends on which V_{DS} we choose. Let's use V_{DS} = 1.2V. We could determine the resistance graphically from the slope of a line tangent to the I_{DS} curve at V_{DS} = 1.2V. But we can get a rough idea of the channel resistance by determining the slope of a line passing through the origin and the point we chose on the I_{DS} curve, i.e., 1.2V/ I_{DS}.

Of course, the channel resistance depends on the dimensions of the mosfet we used to make the measurement. For mosfets, their I_{DS} is proportional to W/L where W is the

width of the mosfet (1.2 microns in this example) and L is the length (0.6 microns in this example). When reporting the effective channel resistance, it's useful to report the *sheet resistance*, i.e., the resistance when W/L = 1. That way you can easily estimate the effective channel resistance for size device by scaling the sheet resistance appropriately. Since W/L = 2 for the device you measured, it conducted twice as much current and has half the channel resistance as a device with W/L = 1, so you need to double the channel resistance you computed above in order to estimate the effective channel sheet resistance.

So just standardize measurements Yes

Use the on-line questions page for this lab to report the value for I_{DS} that you measured and the effective channel sheet resistance you calculated from that measurement.

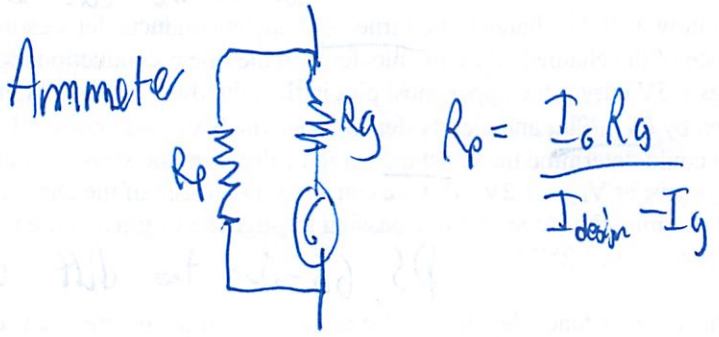
On-line question 1B:

Now let's see how well the mosfet turns "off." Take some measurements of I_{DS} at various points along the V_{GS}=0V curve (the bottom-most plot in the window). Notice that they aren't zero! Mosfets do conduct minute amounts of current even when officially "off", a phenomenon called "subthreshold conduction." While negligible for most purposes, this current is significant if we are trying to store charge on a capacitor for long periods of time (this is what DRAMs try to do). Make a measurement of I_{DS} when V_{GS}=0V and V_{DS}=2.5V. Based on this measurement report how long it would take for a .05pF capacitor to discharge from 5V to 2.5V, i.e., to change from a valid logic "1" to a voltage in the forbidden zone. Recall from 6.002 that Q = CV, so we can estimate the discharge time as $\Delta t = C(\Delta V / I_{OFF})$. So if our mosfet switch controls access to the storage capacitor, you can see we'll need to refresh the capacitor's charge at fairly frequent intervals.

So V_{GS} is power source

V_{DS} is where measure at??

Yeah for ammeter



$R_p = \frac{I_G R_g}{I_{desire} - I_g}$

3: Gate-level timing

The following JSim netlist shows how to define your own circuit elements using the ".subckt" statement:

```
* circuit for Lab#1, parts C thru F
.include "/mit/6.004/jsim/nominal.jsim"

* 2-input NAND: inputs are A and B, output is Z
.subckt nand2 a b z
MPD1 z a 1 0 NENH sw=8 sl=1
MPD2 1 b 0 0 NENH sw=8 sl=1
MPU1 z a vdd vdd PENH sw=8 sl=1
MPU2 z b vdd vdd PENH sw=8 sl=1
.ends

* INVERTER: input is A, output is Z
.subckt inv a z
MPD1 z a 0 0 NENH sw=16 sl=1
MPU1 z a vdd vdd PENH sw=16 sl=1
.ends
```

The ".subckt" statement introduces a new level of netlist. All lines following the ".subckt" up to the matching ".ends" statement will be treated as a self-contained subcircuit. This includes model definitions, nested subcircuit definitions, electrical nodes and circuit elements. The only parts of the subcircuit visible to the outside world are its terminal nodes which are listed following the name of the subcircuit in the ".subckt" statement:

```
.subckt name terminals…
* internal circuit elements are listed here
.ends
```

In the example netlist, two subcircuits are defined: "nand2" which has 3 terminals (named "a", "b" and "z" inside the nand2 subcircuit) and "inv" which has 2 terminals (named "a" and "z").

Once the definitions are complete, you can create an instance of a subcircuit using the "X" circuit element:

```
Xid nodes… name
```

where *name* is the name of the circuit definition to be used, *id* is a unique name for this instance of the subcircuit and *nodes*... are the names of electrical nodes that will be hooked up to the terminals of the subcircuit instance. There should be the same number of nodes listed in the "X" statement as there were terminals in the ".subckt" statement that defined *name*. For example, here's a short netlist that instantiates 3 NAND gates (called "g0", "g1" and "g2"):

```
Xg0 d0 ctl z0 nand2
Xg1 d1 ctl z1 nand2
Xg2 d2 ctl z2 nand2
```

The node "ctl" connects to all three gates; all the other terminals are connected to different nodes. Note that any nodes that are *private* to the subcircuit definition (i.e., nodes used in the

subcircuit that don't appear on the terminal list) will be unique for each instantiation of the subcircuit. For example, there is a private node named "1" used inside the nand2 definition. When JSim processes the three "X" statements above, it will make three independent nodes called "xg0.1", "xg1.1" and "xg2.1", one for each of the three instances of nand2. There is no sharing of internal elements or nodes between multiple instances of the same subcircuit.

It is sometimes convenient to define nodes that are shared by the entire circuit, including subcircuits; for example, power supply nodes. The ground node "0" is such a node; all references to "0" anywhere in the netlist refer to the same electrical node. The included netlist file nominal.jsim defines another shared node called "vdd" using the following statements:

```
.global vdd
VDD vdd 0 3.3v
```

The example netlist above uses "vdd" whenever a connection to the power supply is required.

The other new twist introduced in the example netlist is the use of symbolic dimensions for the mosfets ("SW=" and "SL=") instead of physical dimensions ("W=" and "L="). Symbolic dimensions specify multiples of a parameter called SCALE, which is also defined in nominal.jsim:

```
.option SCALE=0.6u
```

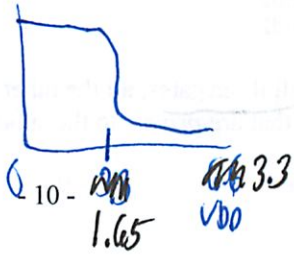
So with this scale factor, specifying "SW=8" is equivalent to specifying "W=4.8u." Using symbolic dimensions is encouraged since it makes it easier to determine the W/L ratio for a mosfet (the current through a mosfet is proportional to W/L) and it makes it easy to move the design to a new manufacturing process that uses different dimensions for its mosfets. Note that in almost all instances "SL=1" since increasing the channel length of a mosfet reduces its current carrying capacity, not something we're usually looking to do.

We'll need to keep the PN junctions in the source and drain diffusions reverse biased to ensure that the mosfets stay electrically isolated, so the substrate terminal of nfet (those specifying the "NENH" model) should always be hooked to ground (node "0"). Similarly the substrate terminal of pfet (those specifying the "PENH" model) should always be hooked to the power supply (node "vdd").

With the preliminaries out of the way, we can tackle some design issues:

On-line question 1C:

To maximize noise margins we want to have the transition in the voltage transfer characteristic (VTC) of the nand2 gate centered halfway between ground and the power supply voltage (3.3V). To determine the VTC for nand2, we'll perform a dc analysis to plot the gate's output voltage as a function of the input voltage using the following additional netlist statements:



```
* dc analysis to create VTC
Xtest vin vin vout nand2
Vin vin 0 0v

Vol vol 0 0.3v   // make measurements easier!
Voh voh 0 3v     // see part (D)

.dc Vin 0 3.3 .005
.plot vin vout voh vol
```

Combine this netlist fragment with the one given at the start of this section and run the simulation. To center the VTC transition, keep the size of the nfet in the nand2 definition as "SW=8 SL=1" and adjust the width of both pfets until the plots for vin and vout intersect at about 1.65 volts. Just try different integral widths (i.e, 9, 10, 11, …). Report the integral width that comes closest to having the curves intersect at 1.65V.

On-line question 1D:

The *noise immunity* of a gate is the smaller of the low noise margin ($V_{IL} - V_{OL}$) and the high noise margin ($V_{OH} - V_{IH}$). If we specify $V_{OL} = 0.3V$ and $V_{OH} = 3.0V$, what is the largest possible noise immunity we could specify and still have the "improved" NAND gate of part (C) be a legal member of the logic family?

Hint: to measure the low noise margin, use the VTC to determine what V_{IN} has to be in order for V_{OUT} to be 3V, and then subtract V_{OL} (0.3V) from that number. To measure the high noise margin, use the VTC to determine what V_{IN} has to be in order for V_{OUT} to be 0.3V, and then subtract that number from V_{OH} (3.0V). We've added some voltage sources corresponding to V_{OL} and V_{OH} to make it easier to make the measurements on the VTC plot.

NOTE: make these measurements using your "improved" nand2 gate that has the centered VTC, i.e., with the updated widths for the PFETS.

Now that we have the mosfets ratioed properly to maximize noise immunity, let's measure the contamination time (t_C) and propagation time (t_P) of the nand2 gate. The *contamination delay*, t_{CD}, for the nand2 gate will be a lower bound for all the t_C measurements we make. Similarly, the *propagation delay*, t_{PD}, for the nand2 gate will be an upper bound for all the t_P measurements.

Recall that the contamination time is the period of output validity after the inputs have become invalid. So for nand2:

$t_{C\text{-FALL}}$ = time elapsed from when input > V_{IL} to when output < V_{OH}
$t_{C\text{-RISE}}$ = time elapsed from when input < V_{IH} to when output > V_{OL}
$t_C = \min(t_{C\text{-RISE}}, t_{C\text{-FALL}})$

Similarly the propagation time is the period of output invalidity after the inputs have become valid. So for nand2:

$t_{P\text{-}RISE}$ = time elapsed from when input ≤ V_{IL} to when output ≥ V_{OH}
$t_{P\text{-}FALL}$ = time elapsed from when input ≥ V_{IH} to when output ≤ V_{OL}
$t_P = \max(t_{P\text{-}RISE}, t_{P\text{-}FALL})$

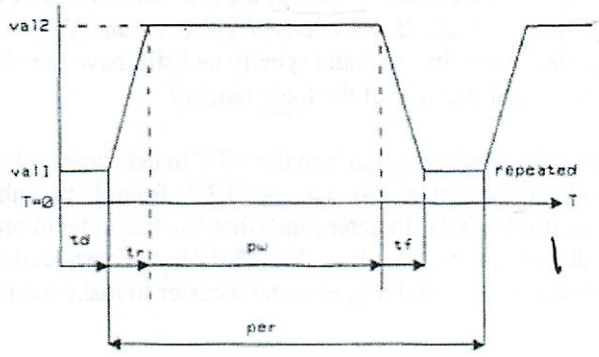
Following standard practice, we'll choose the logic thresholds as follows:

V_{OL} = 10% of power supply voltage = .3V
V_{IL} = 20% of power supply voltage = .6V
V_{IH} = 80% of power supply voltage = 2.6V
V_{OH} = 90% of power supply voltage = 3V

You can use a voltage source with either a pulse or piece-wise linear waveform to generate test signals for your circuit. Here's how to enter them in your netlist:

```
Vid output 0 pulse(val1 val2 td tr tf pw per)
```

This statement produces a periodic waveform with the following shape:



Don't forget to specify your times in nanoseconds (use an "n" suffix)! **Do not specify zero rise and fall times** since the simulation will probably fail to converge. To construct a piece-wise linear waveform, you need to supply a list of time,voltage pairs:

```
Vid output 0 pwl(t1 v1 t2 v2 … tn vn)
```

The voltage will be v1 for times before t1, and vn for times after tn.

On-line question 1E:

Replace the netlist fragment from (C) with the following test circuit that will let us measure various delays:

```
* test jig for measuring tcd and tpd
Xdriver vin nin inv
Xtest vdd nin nout nand2
Cload nout 0 .02pf
Vin vin 0 pulse(3.3,0,5ns,.1ns,.1ns,4.8ns)
```

```
Vol vol 0 0.3v   // make measurements easier!
Vil vil 0 0.6v
Vih vih 0 2.6v
Voh voh 0 3.0v

.tran 15ns
.plot vin
.plot nin nout vol vil vih voh
```

NOTE: make these measurements using your "improved" nand2 gate that has the centered VTC, i.e., with the updated widths for the pfets.

We use an inverter to drive the nand2 input since we would normally expect the test gate to be driven by the output of another gate (there are some subtle timing effects that we'll miss if we drive the input directly with a voltage source). Run the simulation and measure the contamination and propagation delays for both the rising and falling output transitions. (You'll need to zoom in on the transitions in order to make an accurate measurement.) Combine as described above to produce estimates for t_C and t_P.

On-line question 1F:

We mentioned several times in lecture the desire to have our circuits operate reliably over a wide range of environmental conditions. We can have JSim simulate our test circuit at different temperature by adding a ".temp" control statement to the netlist. Normally JSim simulates the circuit at room temperature (25°C), but we can simulate the circuit at, say, 100°C by adding the following to our netlist:

```
.temp 100
```

For many consumer products, designs are tested in the range of 0°C to 100°C. Repeat your measurements of part (E) at 100°C and report your findings. Recompute your estimates for t_C and t_P indicating which measurement(s) determined your final choice for the two delays.

Based on your experiment, if a Pentium 4 processor is rated to run correctly at 3Ghz at 100°C, how fast can you clock it and still have it run correctly at room temperature (assuming t_{PD} is the parameter that determines "correct" behavior)? This is why you can usually get away with overclocking your CPU—it's been rated for operation under much more severe environmental conditions than you're probably running it at!

4: CMOS logic-gate design

As the final part of this lab, your mission is to design and test a CMOS circuit that implements the function F(A,B,C) = C + A·B using nfets and pfets. The truth table for F is shown below:

A	B	C	F(A,B,C)
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

Your circuit must contain no more than 8 mosfets. Remember that **only nfets should be used in pulldown circuits** and **only pfets should be in pullup circuits**. Hint: using six mosfets, implement the complement of F as one large CMOS gate and then use the remaining two mosfets to invert the output of your large gate.

Enter the netlist for F as a subcircuit that can be tested by the test-jig built into lab1checkoff.jsim (a file that we supply and which can be found in the course locker). Note that the checkoff circuitry expects your F subcircuit to have exactly the terminals shown below – the inside circuitry is up to you, but the ".subckt F..." line in your netlist should match exactly the one shown below.

```
.include "/mit/6.004/jsim/nominal.jsim"
.include "/mit/6.004/jsim/lab1checkoff.jsim"

… you can define other subcircuits (eg, INV or NAND gates) here …

.subckt F A B C Z
… your circuit netlist here
.ends
```

lab1checkoff.jsim contains the necessary circuitry to generate the appropriate input waveforms to test your circuit. It includes a .tran statement to run the simulation for the appropriate length of time and a few .plot statements which will display the input and output waveforms for your circuit.

For faster simulation, use the  (fast transient analysis) button!

5: Checkoff

When you are satisfied your circuit works, you can start the checkoff process by making sure your top-level netlist is visible in the edit window and that you've complete a successful simulation run. Then click on the green checkmark in the toolbar. JSim proceeds with the following steps:

1. JSim verifies that a .checkoff statement was found when your netlist was read in.

2. JSim processes each of the .verify statements in turn by retrieving the results of the most recent simulation and comparing the computed node values against the supplied expected values. It will report any discrepancies it finds, listing the names of the nodes it was checking, the simulated time at which it was checking the value, the expected value and the actual value.

3. When the verification process is successful, Jsim asks for your 6.004 user name and password (the same ones you use to login to the on-line assignment system) so it can send the results to the on-line assignment server.

4. JSim sends your circuits to the on-line assignment server, which responds with a status message that will be displayed for you. If you've misentered your username or password you can simply click on the green checkmark to try again. Note that the server will check if your circuit has at most 8 mosfets – if it contains more, you'll see a message to that effect and your check-in will not complete.

If you have any difficulties with checkoff, talk to a TA, or send email to 6004–labs@csail.mit.edu. **Remember to schedule a lab checkoff meeting with a member of the course staff after you complete the on-checkoff and on-line questions. This meeting can happen after the due date of the lab but must be completed within one week of the lab's due date in order to receive full credit.**

6.004 On-line: Questions for Lab 1

When you're done remember to save your work by clicking on the 'Save' button at the bottom of the page. You can check if your answers are correct by clicking on the 'Check' button.

When entering numeric values in the answer fields, you can use integers (1000), floating-point numbers (1000.0), scientific notation (1e3), or JSim numeric scale factors (1K).

Problem 1.

A. Report the mosfet Ids you measured from the device curves for Vgs = 5V and Vds = 1.2V.

Ids (in amps):

Compute the effective channel sheet resistance using (Vds)/(Ids) as an estimate for the channel resistance of the test mosfet. Don't forget to correct for the W/L of the test device!

Sheet resistance (in ohms):

B. Report the mosfet Ids you measured from the device curves for Vgs = 0V and Vds = 2.5V.

Ids (in amps):

Calculate the time it would take to discharge a 0.05pF capacitor from 5V to 2.5V

Discharge time (in seconds):

C. Determine a scaled width (SW) for the two pullup mosfets so that the Vin and Vout curves for the NAND gate intersect at VDD/2 (1.65V).

Scaled width:

D. What is the largest noise immunity we could specify and still have the NAND gate qualify as a legal device? Please fill in your answer with a precision of .01 volts.

Hint: to measure the low noise margin, use the VTC to determine what Vin has to be in order for Vout to be 3V, then subtract Vol (0.3V) from that number. To measure the high noise margin, use the VTC to determine what Vin has to be in order for Vout to be 0.3V, then subtract that number from Voh (3.0V).

Maximum noise immunity (in volts):

E. Following standard practice, choose the logic thresholds as follows:

Vol = 10% of power supply voltage = 0.3V
Vil = 20% of power supply voltage = 0.6V
Vih = 80% of power supply voltage = 2.6V
Voh = 90% of power supply voltage = 3.0V

Measure the contamination and propagation times for your NAND gate using both the falling and rising output transitions.

t_C for falling output (in seconds):

t_P for falling output (in seconds):

t_C for rising output (in seconds):

t_P for rising output (in seconds):

Use these measurements to compute estimates for an upper bound for t_{CD} and a lower bound for t_{PD}.

upper bound for t_{CD} (in seconds):

lower bound for t_{PD} (in seconds):

F. Recompute your bounds for t_{CD} and t_{PD} this time including the measurements you made at 100 degrees C.

Recomputed upper bound for t_{CD} (in seconds):

Measurement used: --select answer--

Recomputed lower bound for t_{PD} (in seconds):

Measurement used: --select answer--

Check Save

source: on_line_questions.py, lab1questions.xdoc

6.004 Lab 1

9/18

On wiki w/ help
- not much

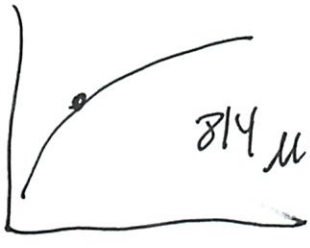
VP: Sheet Resistance - Resistance of films uniform in thickness

W/L=2 is twice current, half resistance

$$R_s = \frac{\rho}{t}\frac{L}{W} = R\frac{L}{W}$$

↑ scaling factor

1A) I_{ds}



814 μA = .000 814 ✓

Sheet resistance

$$\frac{V_{ds}}{I_{ds}}\frac{W}{L}$$

$$\frac{1.2}{.000814}\frac{1.2\mu}{.6\mu} = 2948\ \Omega$$ ✓

(2)

1b) $V_{GS} = 0\ V$

$V_{DS} = 2.5\ V$

$I = 5.155\ pA$ ✓ Can't enter in sci notation

How long for .05 pF cap to discharge 5V → 2.5V

$Q = CV \qquad \Delta t = C \frac{\Delta V}{I_{off}}$

$= .05p \frac{(5-2.5)}{5.155p}$

$= .024\ sec$ ✓

1D This is a VTC

Oh the #s are weird

~~to 2h~~

Vout: 1.5V ← at actually 3V

(3)

low noise margin $V_{IL} - V_{OL}$

high $V_{OH} - V_{IH}$

$V_{OH} = 3V$

$V_{OL} = .3V$

V_{IL} = we just found = ~~1.5V~~ 1.456

$V_{IH} = 1.872$

So low 1.456 $1.5 - .3 = 1.2$ ← smaller so 1.2V ✗ (since most restrictive)

high $3 - 1.872 = 1.128$ ← smaller now ✓

So minor graph reading error

1D val l = 0 or .3 ← since worst case

- oh they tell you no V_{IL} since in .6
- fd we want to have some time to see
  - guess 5 ns

And pulse lenght

(4)

Now plot?

how????

emailed in

This has me stumped

Skip for now

So pulse and pwd does not alternate anything...

E) Ok this was the previos qu? - There was no qu

Than what was that text before?

Where we used a pulse

And never spectied an output!!! for pulse

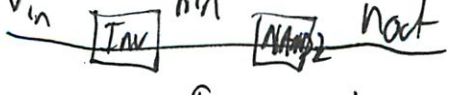
- know the syntax

So they did give specific directions

Now measure

What is diff n in and vin?

Ok nin is output of an inverter + input nand 2



↑ measure here - ~~output~~ want for nand 2

(5)

N_{in} starts changing $5.0392\,ns$

Vout starts " $5.0856\,ns$ ↘ $.0464\,n = t_c$

No - not what supposed to do

min { when input $> V_{IL}$ to when output $< V_{OH}$ fall
$< V_{IH}$ $> V_{OL}$ rise

↑ do both + take min

input $> V_{IL} = 5.0848$ output $< V_{OH} = 5.1122\,ns$ Fall

$= .0272$ ← smallest (X)

Input $< V_{IH} = 9.9464$ ouput $> V_{OL} = 9.9896$ rise

$= .0434$ (X) ← not symetrical

Think mixed rising falling

$.0272n$ = rising output (✓) ~~(which is falling input)~~

but this one ✓ for falling output!

(6)

falling output / rising input

5.0824 5.112

↘ .0296 ✓ measure error

So did not mix rising / falling — stumbled on answer

---

t_p rise time input $\leq I_{IL}$ output $\geq V_{OH}$

max t_p fall $\geq I_{IH}$ $\leq V_{OL}$

propagation time

rise

time input $\leq I_{IL}$ = 9.9832

output $\geq V_{OH}$ = 10.08 } .0968n ✓

fall

time in $\geq I_{IH}$ = 5.1256

out $\leq V_{OL}$ = 5.1808 } .0552n ✓

If

t_c fall input $\geq V_{IL}$ = 5.088

out $\leq V_{OH}$ = 5.12 } .0312 ε min ✓

rise in $\leq V_{IH}$ = ~~9.956~~ 9.956

out $\geq V_{OL}$ = 9.996 } .040

(7)

But correct measurement used = t_c Rise @ 25°

Why ???

Oh - I think qv was pick best t_{cd} for both temp ranges

.1072n - t_c rise 25°

and it accepted it as close enough

t_{pHL}

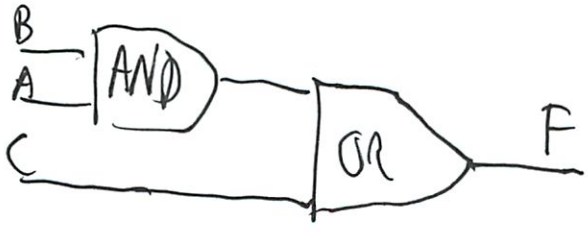
rise input $\leq V_{IL}$ 9.984
output $\geq V_{OH}$ 10.12) .136 n Emax of all ✓

fall input $\geq V_{IH}$ 5.136
out $\leq V_{OL}$ 5.208) .072 n

✓ complete; Online Qv

Part 4 (I should look at tutorial problems - MOSFET combo (CMOS) many)

So do



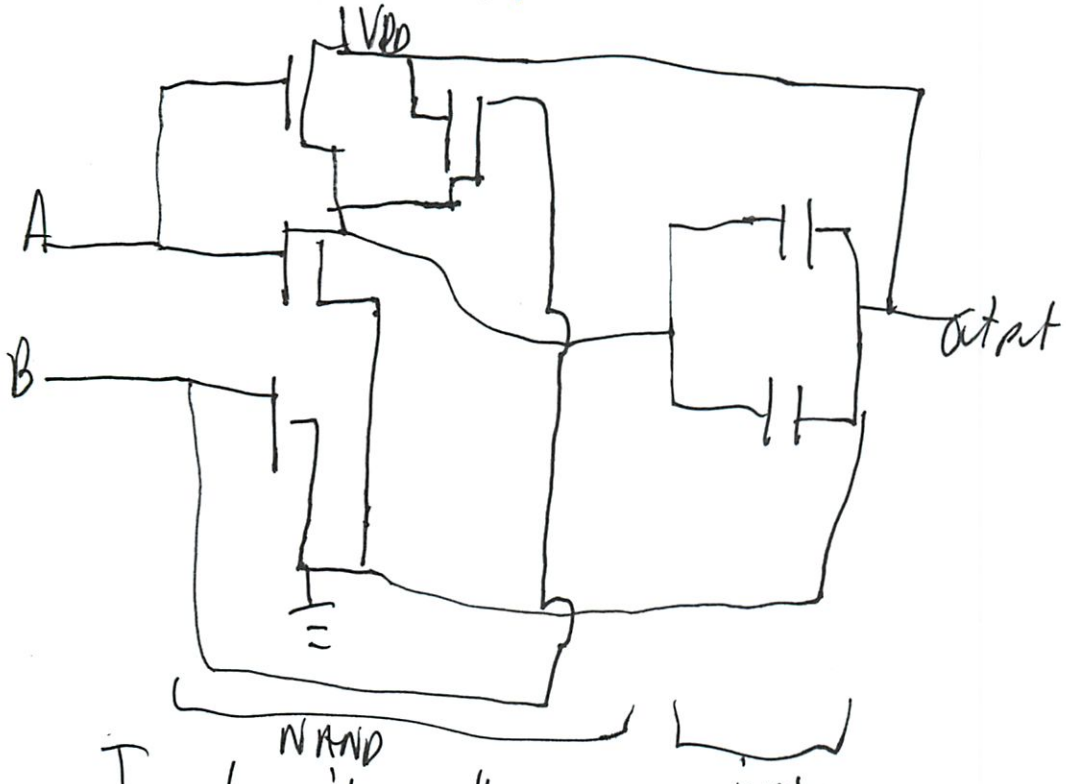
but how to do AND - can do everything w/ NANDs

(8)

Can do we need some DeMorgan's clevereas? how hard have they made it for us?

So you can either do everything w/ lots of NANDS or a few more specific

Webi And is NAND + inverter



I drew it really wrong

This has 6

- design from scratch
  - no never learned I don't think

(9)

NOR = 4 fets

OR = 6 fets (4 + 2 inv)

Try DeMorgan

$C + A \cdot B$

Oh this was kinda part of lab

$\tilde{F} = \overline{C \cdot (A+B)}$

↑ invert ↑ NAND ↑ OR

does class have combos:

└ notes has $F = A \cdot (B+C)$

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

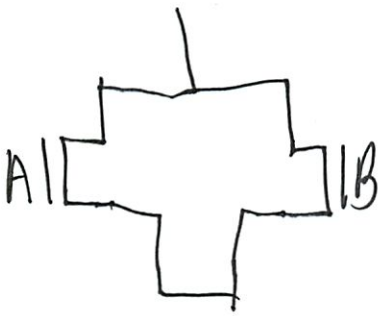
↑ do just like + invert

(6)

More basic AND / or parts. online
- I see half of it

So A · B

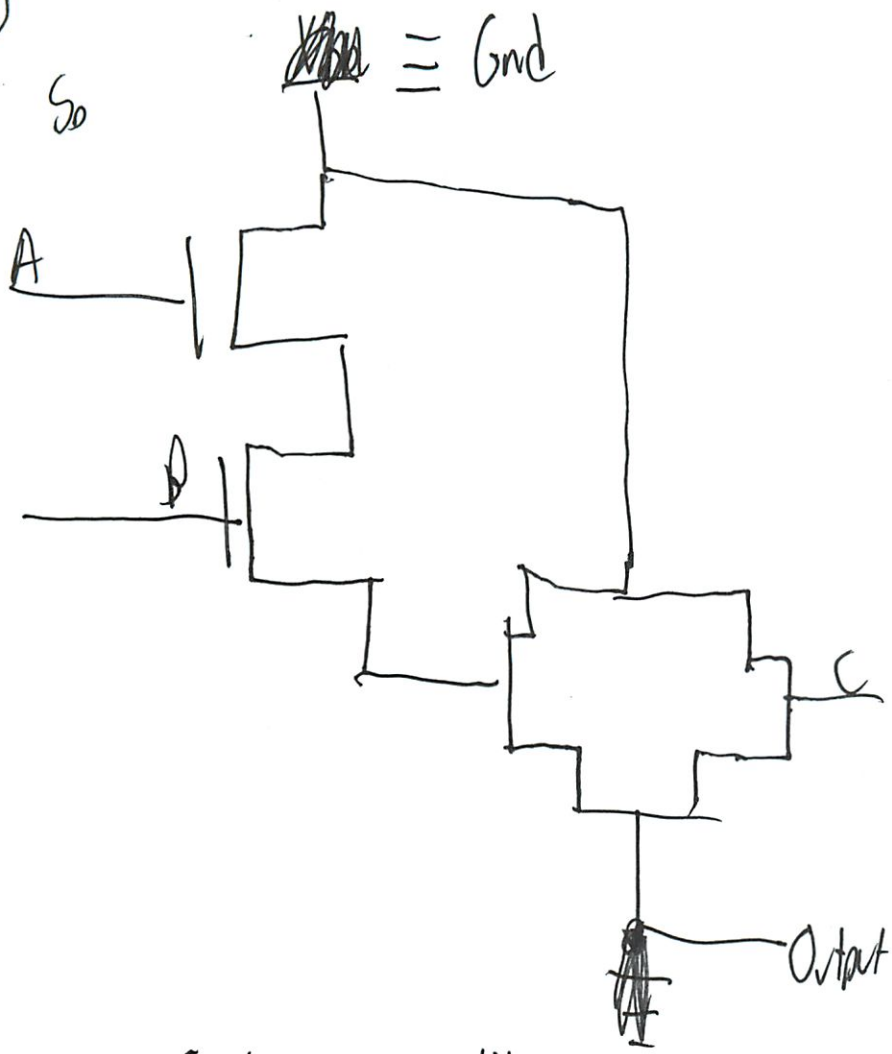
A + B



That is actually really obvious...
But is only 1 side

So

(16)



So that is pulll down gnd → output w/ n

Pull up P-type | V_{dd}

so here to the d Morgan

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

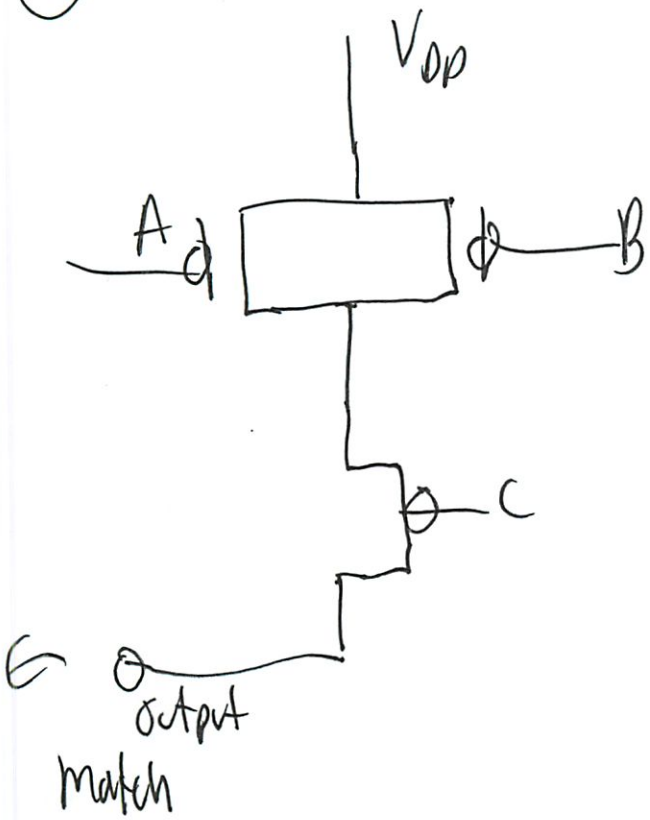
↑ So the not means P-fet;

So we want $F = \overline{F}$;

or $F = F$;

Output |

(12)



so this is 6e fets

Now need to convert to their format

MPD1 z a 1 0 NEWH gw=8 sl=1

name drain gate source bulk/substrate type N or P Size

(so just make up letters to connect)

(13)

G
D
S
N
on when V high
G
S
D
P
on when V low
B
A
H
G
1
0
C
0
Output F
A
V_{DD}
1
F
C
B

(14)

(X) Error: Can't find Z

What is that?

Output?

- but what is F then?

So renamed F → Z

Runs now

(X) Node verification error

Look closely at output

A	B	C	F	
0	0	0	0	✓
0	0	1	0	X
0	1	0	.5	X
0	1	1	0	X
1	0	0	.5	X
1	0	1	0	X
1	1	0	1	✓
1	1	1	0	X

Why .5

(15)

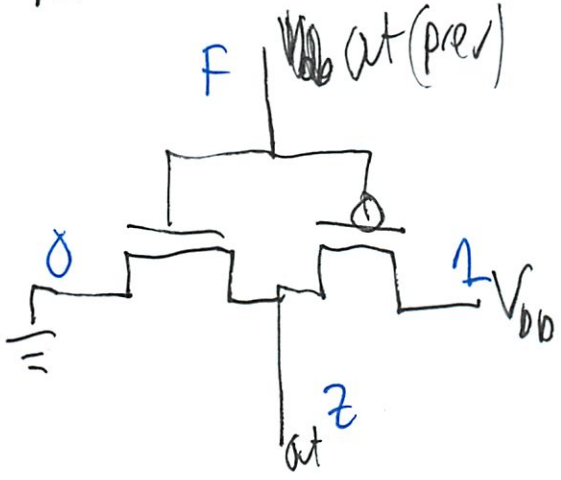
So what is wrong?
May need to ask TA

Well think about it - if 0 opens + floats w/ 0

Pull down - conducts w/ V_{gate} is high
but then pulls down

But from example did
- but example out is $\overline{F}$

So add an inverter
to out at:



but then I would have 9 Fets
But at least get working

N - on when V high
P - on when V low

(16)

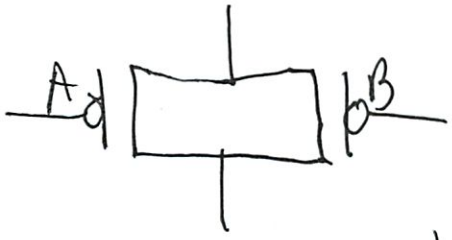
Now

A	B	C	Z
0	0	0	0
1	0	0	.5 then decline?
0	1	0	0
1	1	0	0 ⊗
0	0	1	lower 0?
1	0	1	1 ✓
0	1	1	1 ✓
1	1	1	higher 1 ✓

So redo aiming for complement of F

$\overline{F} = \overline{C} \cdot (\overline{A} + \overline{B})$ - before our new inverter

So $(\overline{A} + \overline{B})$



Wait $\overline{A \cdot \overline{B}} \neq \overline{A \cdot B}$

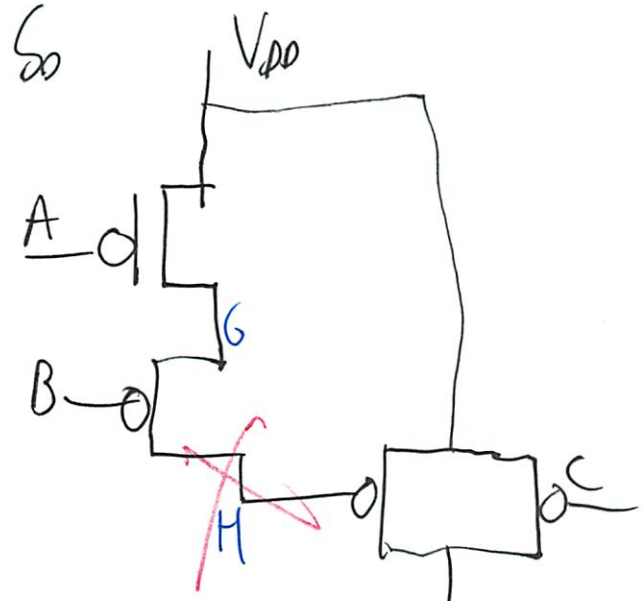
So actually

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

(17)

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

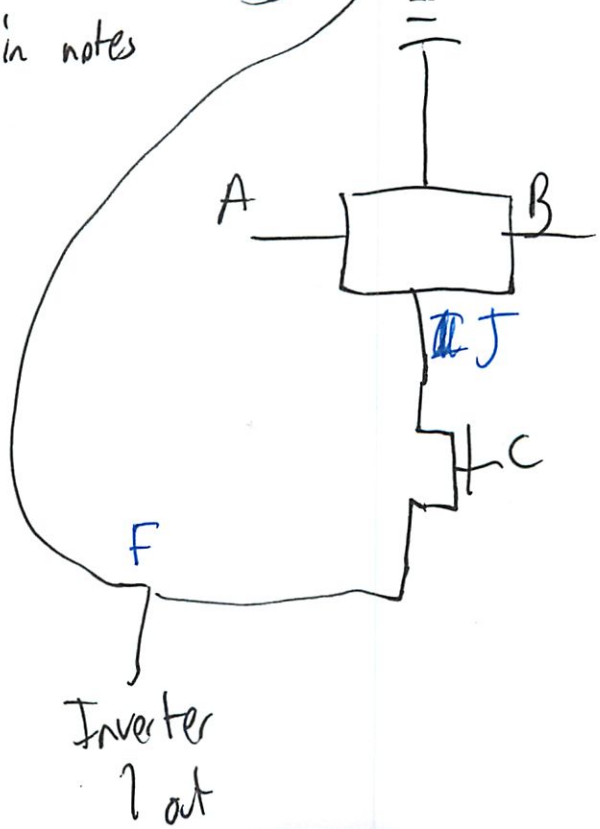
So



Swap $(A+B) \cdot C$

↑ what had originally – no

But is in notes



(18)

Try that

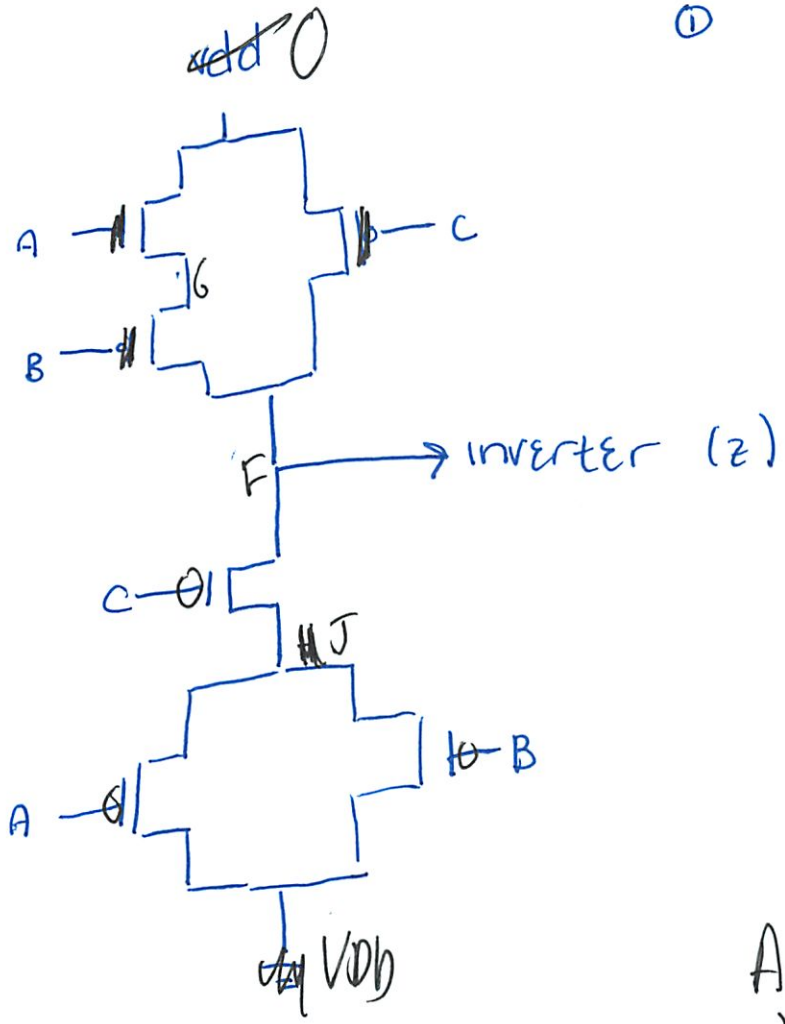
Thats the exact scene!

- ok go in to lab for help mon

TA Help

9/19

	A	B	C	Z
①	0	0	1	0
	1	1	0	
	1	0	0	



DGSB

Actually had this first but I had that weird H connection + no inverter

T and this uses 1 less

just connect together don't need extra

- where did I get that idea?

Xgate (inputs) (outputs) name

.subckt (name) (inputs) (outputs)

F vs Z qv

↑ name of ckt

↑ output

(2)

$F = C + (A + B)$

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

$F \neq \overline{F}$

Do have to invert

So pull down $= \overline{F}$
pull up $= F$ } inverter $\overline{\overline{F}} = F$ $\overline{F}$

That is not pure Demorgan
- but

(3)

Try

A	B	C	F	
0	0	0	0	✓
1	0	0	kinda 0	✓
0	1	0	0	✓
1	1	0	1	✓
0	0	1	0	✗
1	0	1	1	✓
0	1	1	1	✓
1	1	1	1	✓

∈ all that matters
's close enough

1 ∈ perhaps I read it wrong?

N Sorta

Drain is towards out

Source " 0

P

Drain towards out

Source " 1

That did not help/change anything

Changing to width 10 did not help either

So I should have used VDD not 1
since it was like 3.3V

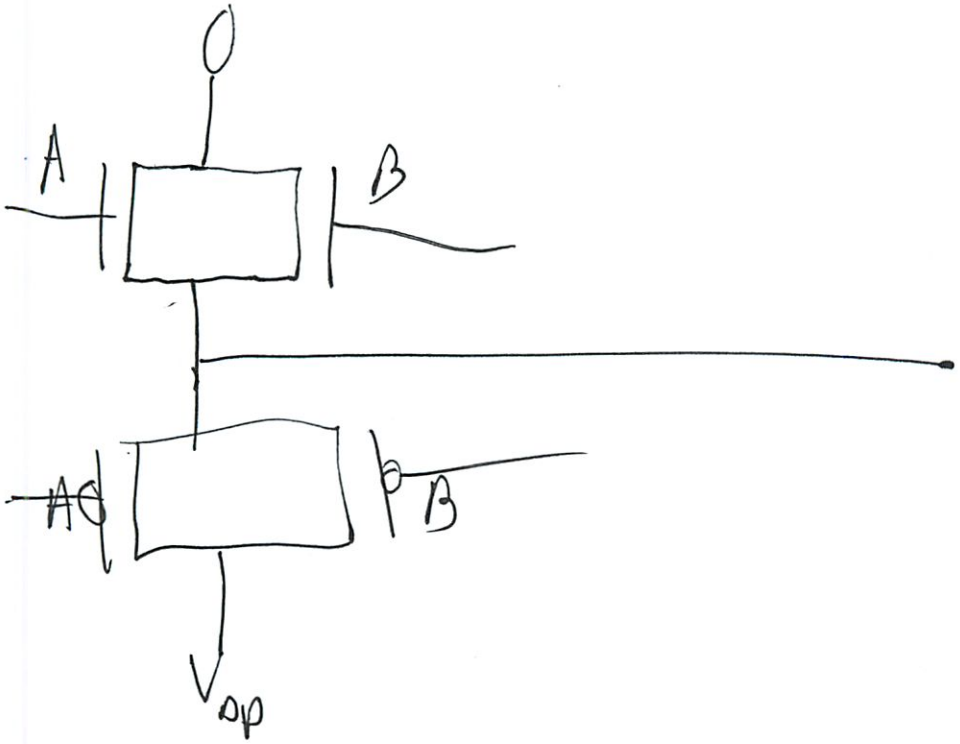
✓ checked

9/19

Check off

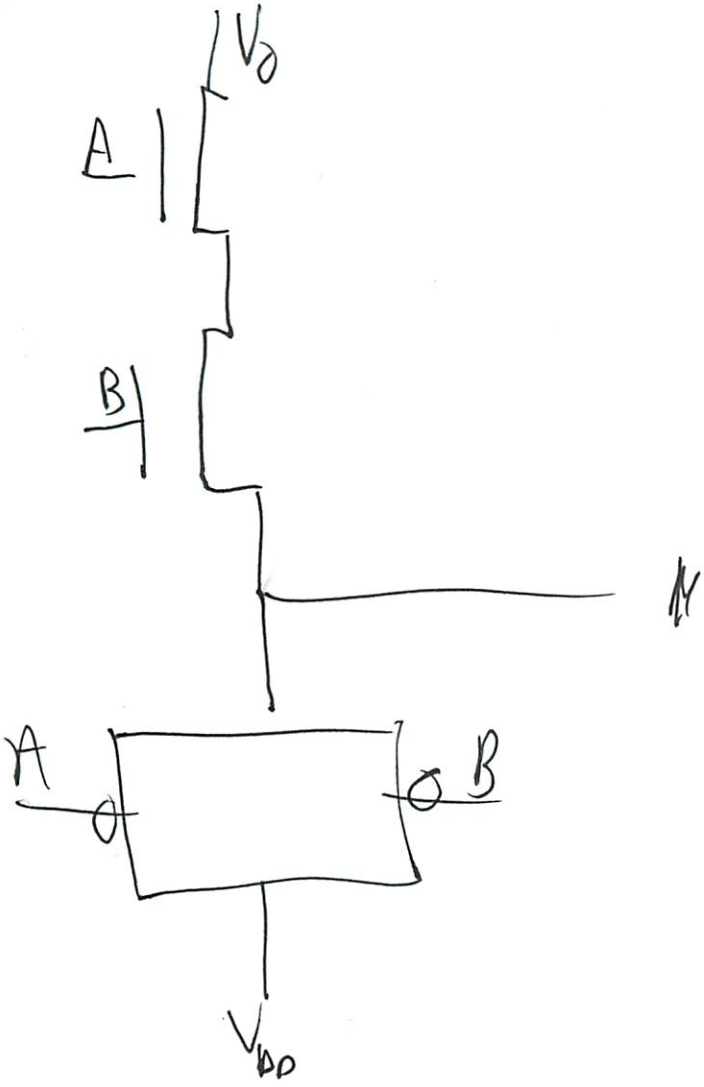
NAND or NOR - general purpose

AND



Can't do 'kna AND
in 1 stage
tell him why

(2)

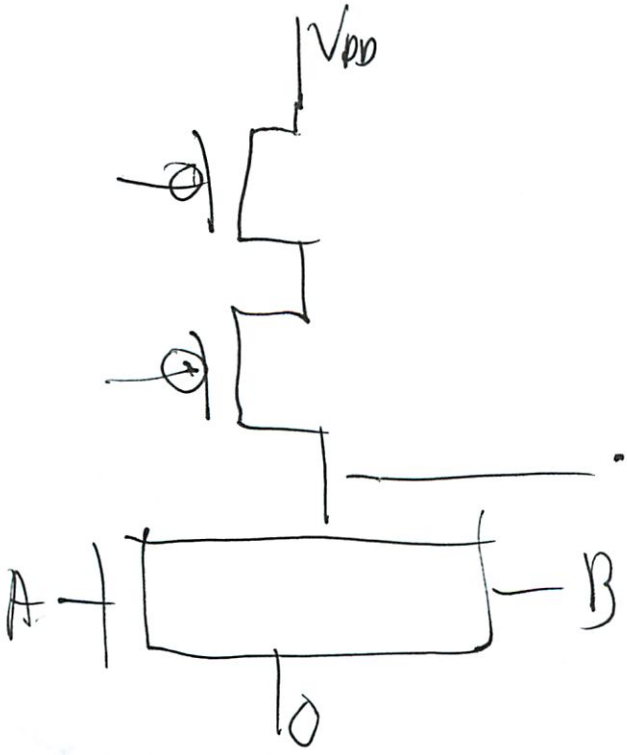


A=0 A=1 1 0
B=1 B=1 0 0
Out=1 Out=0 out=1
wrong but that is right

NAND

So can't do it
i how else would ya?

Even if - what if flip:



A=1 1 0 0
B=1 0 1 0
0 0 1

NOR

(3)

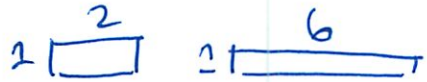
~~But why can do a more complex way?~~

So why can't ya not do it?
No possible combos

; Can't get power to flow that way
w/ a fire

Can only do 0 logic
b/c it always switches

2 1s → 0
2 0s → 1



↑ faster

↑ slower

↓ Prop delay ↑ width or ↓ length

Static power dissipations - very small

✓ cheloff

6.004 Quiz 1 Review

9/22

Lectures 1-3

1. Basics of digital info
   - intro
   - Information
   - encoding information
   - Shanon limit $\log_2\left(\frac{N}{M}\right)$ N choices, M = narrowed choices (= probable) bits
   - Fixed length
   - Octal 0,3720
   - Hex 0x7d0
   - Ints -2^{N-1} to $2^{N-1}-1$
   - When choices not = propable

     $\log_2\left(\frac{1}{p_i}\right)$ bits

     - avg info from choice $\sum p_i \log_2\left(\frac{1}{p_i}\right)$
   - Compression
   - Huffman code

(2)

Need to add redundancy back in

Hamming distance

detect D-bit errors HD > D

correct D-bit if HD > 2D

Lecture 2 Digital Abstraction

encode data/state somehow

using voltage

∞ # of points in analog

analog degrades

built on abstractions + contracts

Combonational device

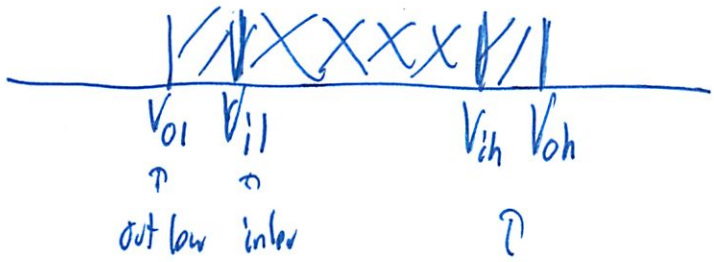
- 1 or more inputs
- 1 or more outputs
- functional spec
- timing spec
  - upper bound t_{pd} = time to get at

(3)

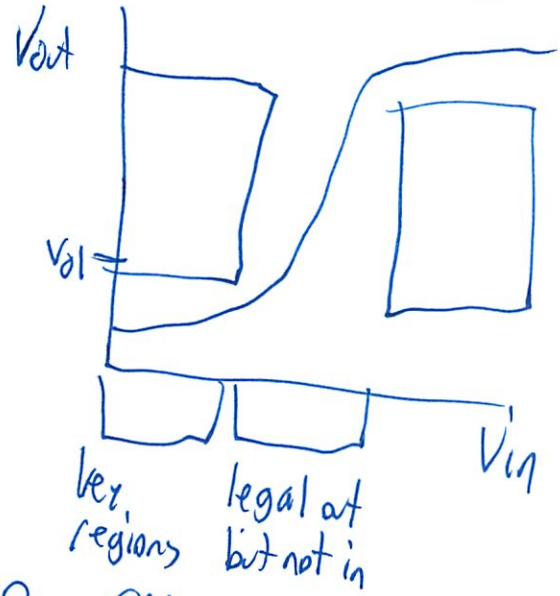
Lots of problem w/ voltage

- Cross talk, Power Supply noise, sequential interfearnce

Noise margins - can input, but not input

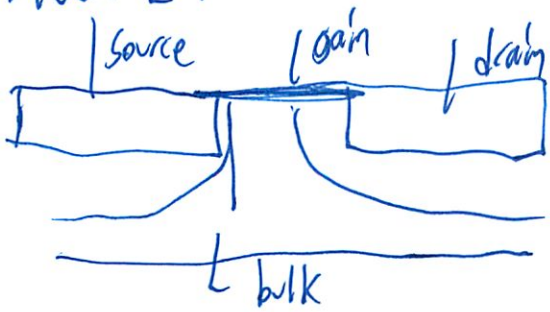


So that makes our VTC



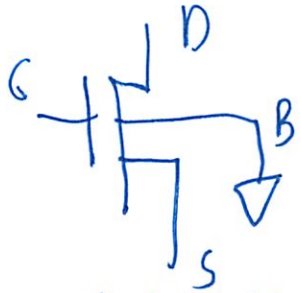
Lecture 3 CMOS

- need gain >1 so non linearity for noise margin
- MOSFETS



NFET

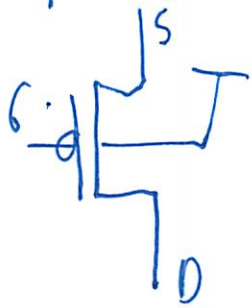
pull down



Conducts V_G high

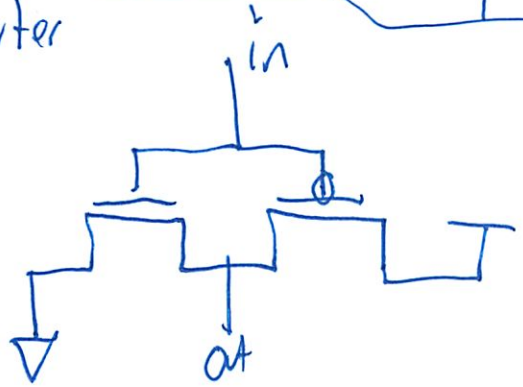
PFET

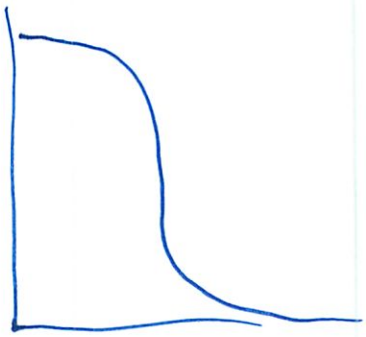
pull up



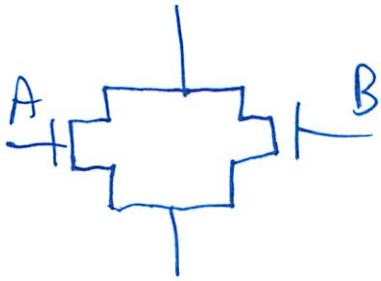
Conducts V_G low

Inverter

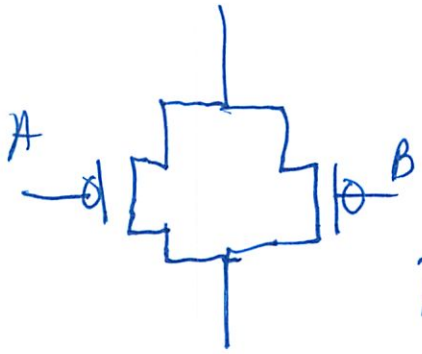




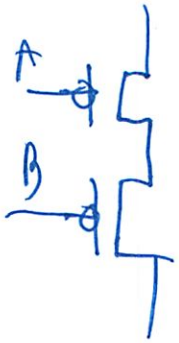
AND



OR



$\bar{A} + \bar{B} = \overline{A \cdot B}$



$\bar{A} \cdot \bar{B} = \overline{A + B}$

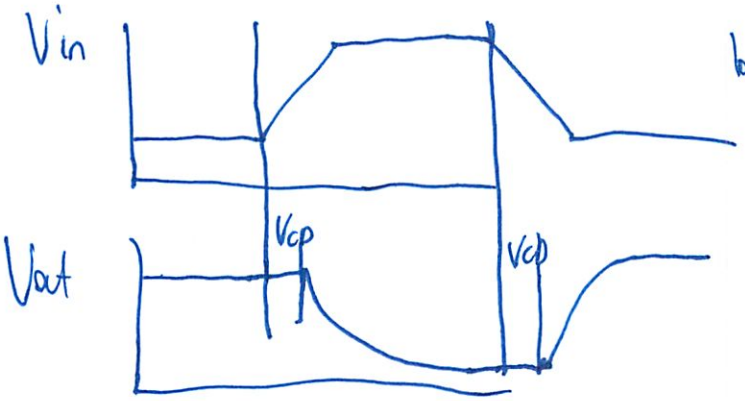
(5)

NAND/NOR Common

1. Make pull down
2. Reverse it so $F = \bar{F}$
3. Combine pull up + pull down

timing

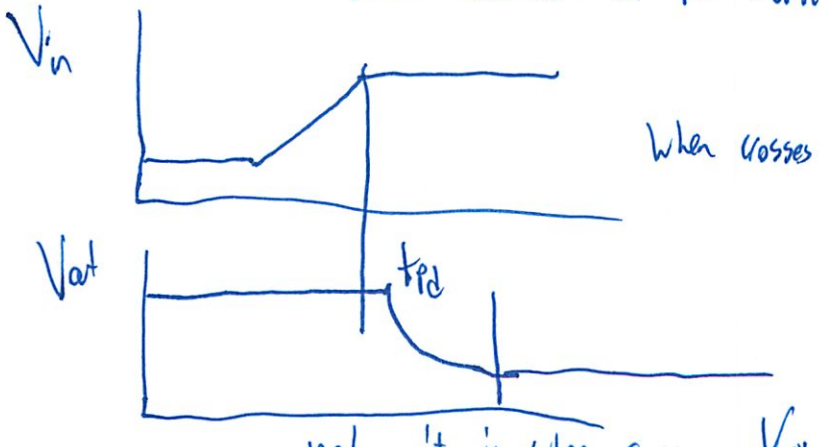
Contamination delay - lower bound on delay from any invalid input to invalid output



not really to inflection pt but crosses V_{IL}

to inflection pt

Propagation delay - upper bound on delay from valid in to valid out



when crosses V_{IH}

note it is when crosses V_{OL}

(6)

Lenient - Output valid when any combo of inputs sufficient to determine output has been valid to t_{pd}

- So tolerates transitions / invalid levels

6.004 Problems

9/23

Basics of info

1. $\log_2\left(\frac{8}{3}\right)$ ✓

2. $\log_2\left(\frac{52}{1}\right)$ next card $\log_2\left(\frac{51}{1}\right)$

   last $\log_2(1)$

   5th card $\log_2\left(\frac{48}{1}\right)$ ✓

3. X 011 _ _ _ _ _

   TWO N bits

   $\log_2\left(\frac{N}{3}\right)$ $\log_2\left(\frac{2^N}{2^{N-3}}\right)$ "did not see this"

   $= \log_2(2^3)$

   $= 3$ bits (duh)

4. X is unknown 8-bit

   Y 8 bit

   HD = 1

   known 7/8 digits of other ~~if~~

   but don't know which

(2)

$\log_2\left(\frac{8}{7}\right)$

2^8 choices

given ~~8~~ narrowed to 8 possible choices

$\log_2\left(\frac{256}{8}\right) = 5$

A lot more tricky than looks.

Top = all possible choices
Bottom = nev possible choices

Huffman trees

1. C more bits to more info
   ie less likely to occur

(These classes are all about tricks...)

2. Decode A B A E C ✓

3. Tree A ✓

Did these in recitation

(3)

F. Prop of dice

$$\sum P_i \log_2\left(\frac{1}{P_i}\right) \qquad P_i = \frac{\text{new}}{\text{old}}$$

$$\frac{1}{36}\log\left(\frac{36}{1}\right) + \frac{2}{36}\log\left(\frac{36}{2}\right) \text{ etc}$$

Remember *1000 because ans is info from 1000 rolls

3.72744 *1000 = 3274.4

Remember shannon limit → can't go smaller than

Unconfined - qv answer more general

Modular Arithmetic

- I forgot we did these qv

Performing math % 2^{32}

How many values

2^{32} ✓

Read the schema...

(9)

Error detection + correction

Did we do this in 6.004?

Not really

Did single bit correction

MD must be 3

Back to 2's complement

Negate by flip each bit + add 1

MSB → LSB

↑ neg or not

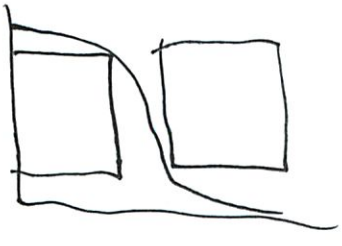
Move to lecture 2

Combo devices

Vol of 0 - No - device can't do ⊘

.5 V ✓

~~Did~~ Did in recitation



(5)

Want largest possible V_{OL}

~~.5~~ 1

Complex math I did not try now

Max noise margins

V_{OL} low as possible

V_{OH} high " "

Hook up devices so No largest noise margin

look at VTC

$4 - 2(W + .5 - 1)$ Gt

$5 - 2W$

So $5 - 3W \geq 3$

If $W \leq \frac{2}{3}$

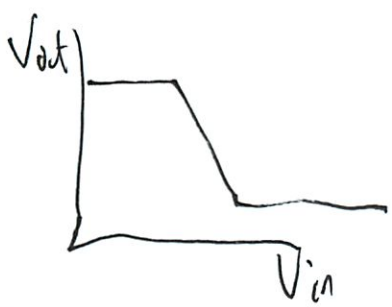
Still don't know why we went through all that trouble!

Never learned this method

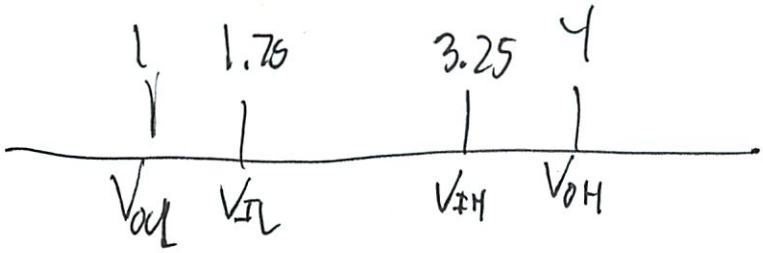
- well that is what studying is for

(6)

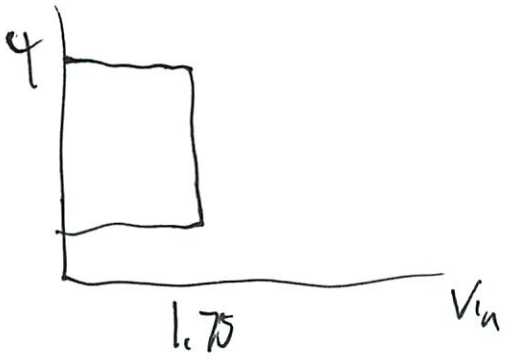
Inverter



.75 noise mgins



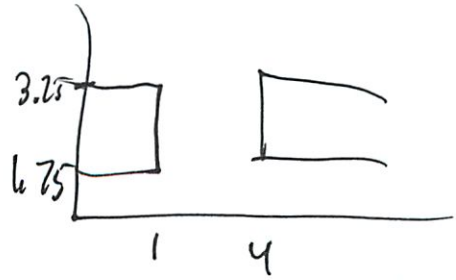
So draw Vat 1st 1
draw box



(X)

But can reduce

$V_{in} = 1$
$V_{out} = 3.25$



- Yes can do - not optimal - but possible (X)

(7)

No! Gain is not > 1

How did I mess that up?

I should have checked gain

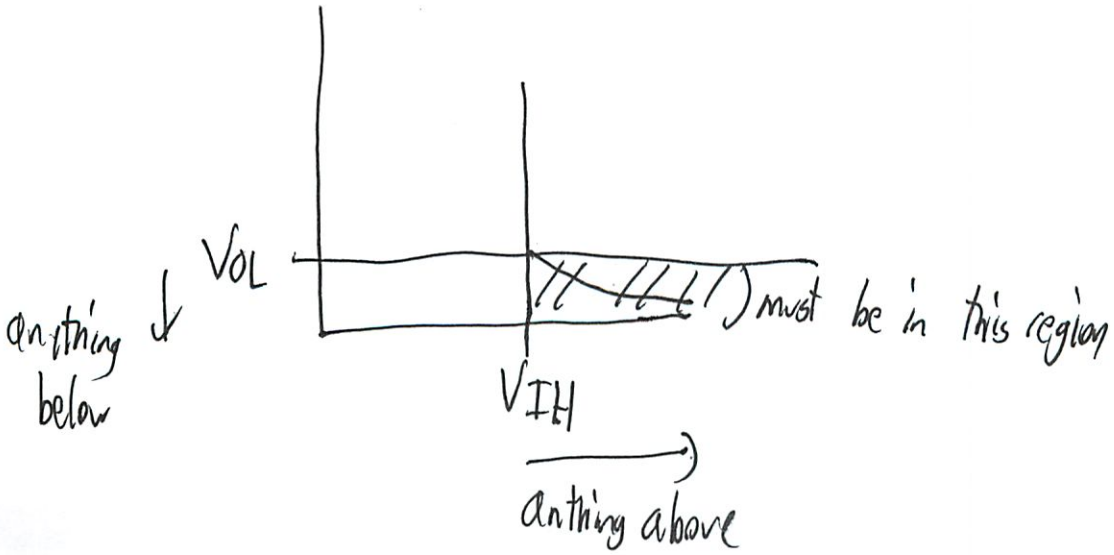
But then I should not have been ablet to draw boxes

- ? does it have to do w/ connecting to another device later???

V_{IH} must be high enough to produce V_{OL}

- can produce V_{OL} or less

Aha! So what does that look like on graph?



(8) So my value

$V_{IL} = 1$ V_{out} 3.25

4 1.75

(X) Not true

Still not positive on this

Look carefully on test

$V_{OH} = V_{IH} + N$

$V_{OL} = V_{OL} + N$

* V_{IH} generates V_{OL} or less at
* V_{IL} of 1 V_{OH} or more of

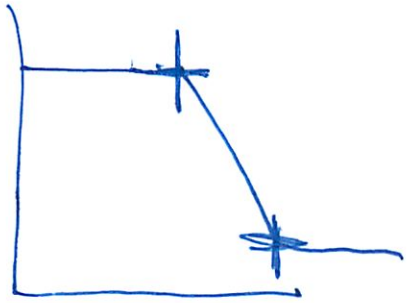
Find N

Think would see better if do example in recitation

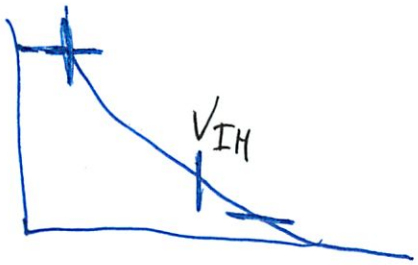
↑ go to 2 in a row fir

9

Like for A just do transitions



* Always check gain



"How did they get that"

"When gain < 1"

Need to move on

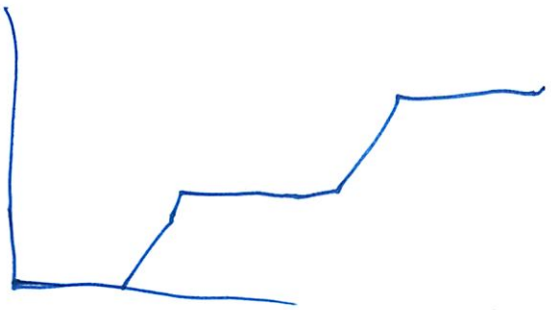
Complicated static discipline qu, tricky

A valid output does not imply a valid input

$$\text{Slope} = \frac{\text{rise}}{\text{run}} > 1$$

(10)

Tertiary output
- 3 stable regions



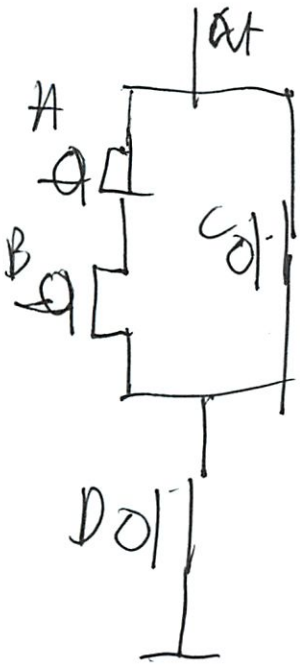
Soldiers
∞ gain since always adjust temp if differs
"I'm confused exactly what gain is/represents
(Not concentrating ...)

WP: Gain > 1 = Amplification

Lecture 3 CMOS

Pulldown - comes to pull up by coveting
- try it

(11)



b) What is fn

So look at 1 and manually do TT

A	B	C	D	Out
0	0	0	0	0
0	0	0	1	1
1	0	0	0	0
1		1		1

nothing closed →

(12)

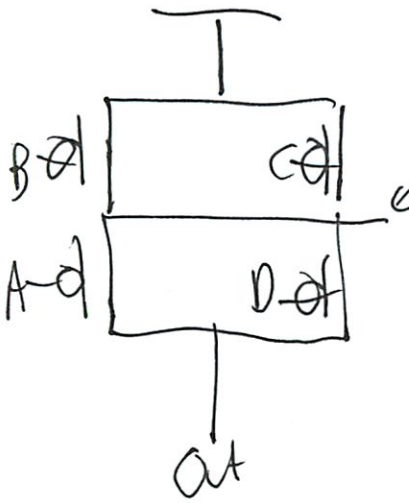
But then how to summarize?

They want ans in form of combo AND ORs

$-(D + (C(A + B)))$

I didn't get what they were asking

Pull up given

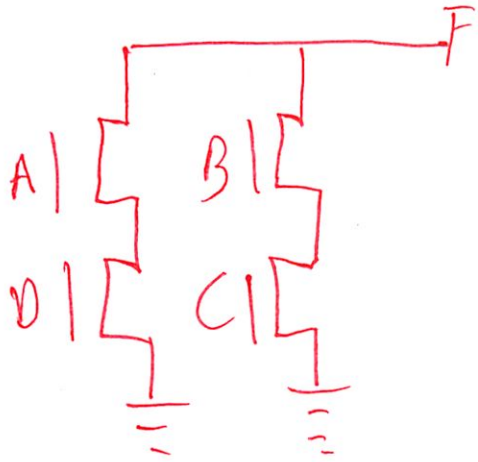


← like complicates things
- no it doesn't
- 2 ors together

=

(13)

No line matters



Oh...
It was 2 ~~AND~~ ORs in AND series
I was thinking that - but did not follow thru

This is $(A \cdot D) + (B + C) = F$

Or $(\bar{B} + \bar{C}) \cdot (\bar{A} + \bar{D}) = \bar{F}$ ← they say this is F

$\overline{(B \cdot C)} \cdot \overline{(A \cdot D)} = \bar{F}$

$\overline{\overline{(B \cdot C)} \cdot \overline{(A \cdot D)}} = F$

Well F for pulldown?

---

3. Can circuit be used as CMOS gate?

Yes - pull down + pull up is complementary

I forgot what the rule was

Make it change more quickly
- less items / levels

(14)

But can it be done here?

↑ width of PFet

↑ target was possible ans

But this also ↑ capacitance

But speed gain from ↑ pullup more than offsets slowdown ↑ Capacitance

Can also ↓ W of 2 NFET pulldowns

will ↓ capacitance + speed up tnx

The lecture slides say nothing about width....

So to speedup ↑ width P

↓ " N

Tend to think N → P

first

Speedup ↓ ↑ width

(15) Out of time quick review

Implement gate

Smoke From ~~voltages~~ current flowing on both part

Have 3 left make inverter

Really complex circuits
- simplify 1st

CMOS Fab
- optional

Quiz 1 Debrief

9/23

After talking

VTC - think did pretty well
CMOS - figured out — figured it out — good studying

0 AND 0 = 0!

Hamming Distance - Very bad at

Timing on circuits - need to review

Bits of info - still tricky, did ok

But reliability/ECC bad

- since skipped in review

Did never study 2s complement

← all things lecture did not focus on

1	2	/4
2	10	/13
3	2	/6
4	7	/7
	21	/30

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Fall 2011

Quiz #1: September 23 2011

Name Michael Plasmeier			*Athena login name* theplaz	*Score*
Brad, 26-322 ☐ WF 10 ☐ WF 11	**Silvina**, 34-303 ☐ WF 11 ☐ WF 12	**Li-Shiuan**, 34-304 ☐ WF 12 ☒ WF 1	**Chris**, 34-303 ☒ WF 1 ☐ WF 2	**David**, 36-155 ☐ WF 2 ☐ WF 3

Notes:

(1) PLEASE FILL IN YOUR NAME, ATHENA NAME, and MARK YOUR SECTION ABOVE before working on the problems on this quiz.

(2) Occasionally, reference materials and/or extra copies of diagrams are provided on the backs of quiz pages. Feel free to use page backs as scratch space, but be sure to mark your answers on the answer lines provided for each question.

Problem 1 (4 points): **Quickies and Trickies**

(A) What decimal integer is represented by the 5-bit two's complement binary number **11100**?

Decimal equivalent of two's complement 11100: 1 X -1

I did not do - should have read!

(B) Is it possible for an inverter to have a contamination delay that is greater than its propagation delay?

not natural w/

Can t_{cd} be greater than t_{pd}? circle one: YES ... CAN'T TELL ... (NO)

(C) In lecture we saw that a 2-input NOR gate might obey the static discipline (i.e., be a valid combinational device) without being a *lenient* combinational device. Is it possible for an inverter to be a valid combinational device without being lenient?

any input → output valid

Might a valid inverter not be lenient? circle one: YES ... CAN'T TELL ... (NO)

(D) A properly-designed CMOS gate has N inputs and one output, and is built using N-1 PFETs (as well as some NFETs). What conclusion can you draw?

pulldown

C1: It uses NFETs in its pullup circuit.
C2: Its output is always 0. useless
C3: Its output is independent of at least one of the N inputs. depends - not always
C4: It is not lenient. don't know - see three ins
(C5): None of the above.

Circle best choice: C1 ... C2 ... X C3 ... C4 ... (C5)

where out is not valid - don't think so

Problem 2. (13 points): **UVWXYZ Coding**

A secretive government agency has a large amount of data that has been encrypted as strings of letters from the alphabet UVWXYZ. They need to transmit this data over a binary channel (i.e., one that communicates a sequence of 0's and 1's), and have asked for your help.

They ask you to design a fixed-length code that transmits each of the U, V, W, X, Y, or Z letters as a message codeword (binary string). $2^3 = 8$ 6

(A) (1 point) Using the shortest possible fixed-length code, how many bits are transmitted for each message codeword (i.e., for each letter from the UVWXYZ alphabet)?

Bits transmitted for each UVWXYZ message: 3 **bits**

You design a fixed-length coding that translates each of the six letters to a minimally-sized codeword, and present it to to the agency. Your contact there asks, perhaps to impress you with his technical acumen, what the minimal Hamming distance is between valid codewords in the fixed-length code you've supplied.

000
111
101
011

(B) (1 point) What is the minimal Hamming distance between valid code words in your fixed-length code?

Minimal Hamming distance between valid codewords: 1

forget exp - was in 6.01

Max Noyes, an agency spokesperson, is concerned about single-bit errors that may be occasionally be introduced during codeword transmission. He probes you about error-detecting characteristics of your simple fixed-length code.

(C) (1 point) Can your code detect all single-bit errors?

forget exact details(detect hamming < D Correct < 2D

Detect **single-bit errors? Circle one: YES ... NO**

Since may be other character

In order to improve its error-handling characteristics, you add a single parity bit to each codeword. You choose to use odd parity, i.e. you set the parity bits to make the total number of 1's in each codeword odd.

(D) (1 point) What is the minimal Hamming distance between valid code words in your fixed-length code with the added parity bit?

Minimal Hamming distance between valid codewords: 2

(E) (1 point) Can your revised code *detect* all single-bit errors? Can it *correct* them?

Single-bit errors can be (Circle all that apply): DETECTED ... CORRECTED

Exploring options, Max asks what the minimum Hamming distance between valid codewords would have to be to allow for correction of both 2-bit and 1-bit errors.

(F) (1 point) What is your answer? What is the necessary Hamming distance?

Minimal Hamming distance to correct 2-bit errors: 4 (−1)

but then 2 - revise better

Les Bitz is suddenly put in charge of the project and is determined to cut costs. He insists on abandoning all worries about error detection and correction, and wants to focus on minimizing the number of bits that must be transmitted. Les provides you with some key additional information: he knows the probability of each of the six letters (that is, the relative frequency at which they appear in strings to be transmitted). Les provides you with the following partially-filled table:

Message	Probability	Info (bits)
U	1/2	ln(2)
V	1/4	ln(4)
W, X, Y, Z	1/16	ln(16)

ln(1/p)

The table gives the probability of each of the six messages. The remaining column, reflecting the number of bits of information communicated by each message, has been left blank.

(G) (3 points) Complete the table by filling in the missing three entries.

(Complete above table)

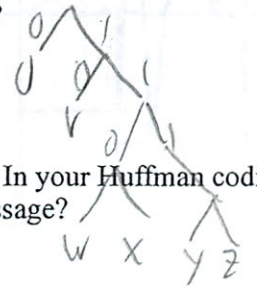
(H) (1 point) What is the average amount of information conveyed by each message?

$\frac{1}{2}\ln(2) + \frac{1}{4}\ln(4) + \frac{1}{16}\ln(16)$ (4.)

Average info per message: (−1) **bits**

Responding to pressure from Les to reduce the number of bits transmitted, you remember the Huffman coding scheme you learned in 6.004 and use it to design an appropriate variable-length code.

(I) (2 points) In your Huffman coding scheme, how many bits are transmitted for a U? For a V? For a W?



Codeword size for U: 1 **; for V:** 2 **; for W:** 4

did not build properly

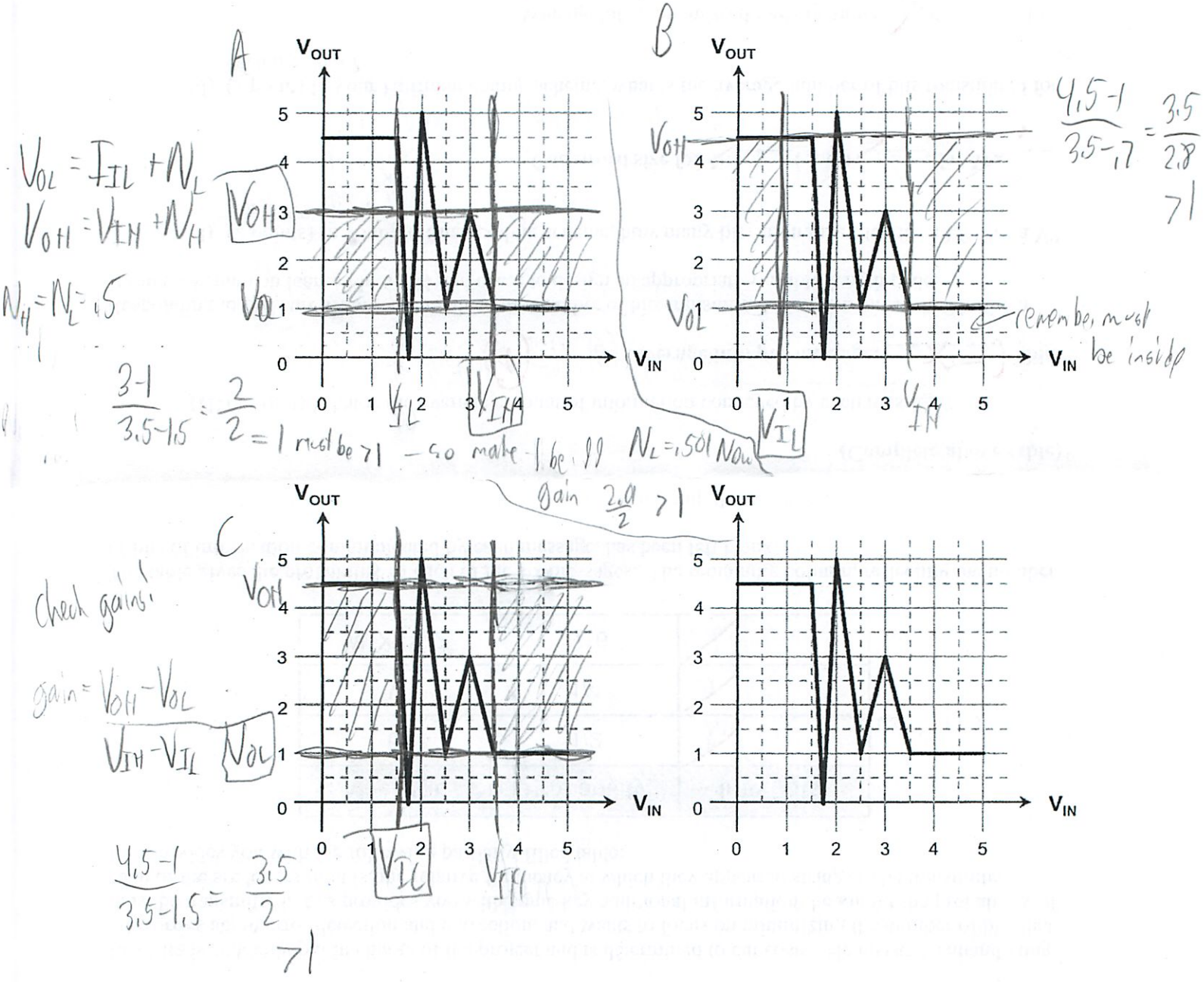
(J) (1 point) In your Huffman coding scheme, what is the average number of bits transmitted for each message?

Average bits transmitted per message: 1.25 (−1) **bits**

$\frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{16}\cdot 4$

$\frac{1}{2} + \frac{1}{2} + \frac{1}{4} = 1.25$

Scratch copies of diagram from problem 3:

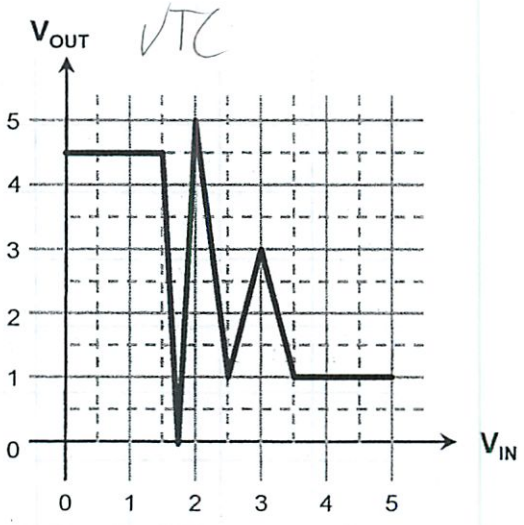


Problem 3. (6 points) **Logical Negativism**

Logical Negativism, Inc is an MIT spinoff whose aim is to market a new logic family whose only component type is an inverter. Their motto, *Logic is NOT everything*, has convinced a venture group that this business plan has merit, and they are now well funded.

Their founder, Ms. Anna Logge, has developed a device to be used as their inverter; its static voltage transfer curve is shown to the right. Anna is considering the choice of parameters by which LNI's logic family will represent logic values, and needs your help.

The voltage transfer curve of a proposed inverter for a new logic family is shown to the right (spare copies of this diagram are on the back of page 3 of this quiz).



Several possible schemes for mapping logic values to voltages are being considered, as summarized in the incomplete table below. **Recall that Noise Immunity (last row) is defined as the lesser of the two noise margins.**

Complete LNI's table, by filling in missing entries. Choose each value you enter so as to maximize the noise margins of the corresponding scheme. **If the numbers in a scheme can't be completed such that the LNI device functions as an inverter with positive noise margins, put an X in the entries for that column.**

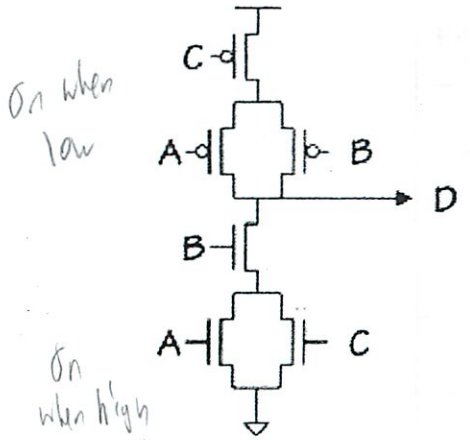
LNI's Possible Logic Mappings:

	Scheme A	Scheme B	Scheme C
V_{OL}	.999	1	1
V_{IL}	1.5	0.7	1.5
V_{IH}	3.5	3.5	3.5
V_{OH}	3	4.5	4.5
Noise Immunity	.5	.7	.5

-2 -2

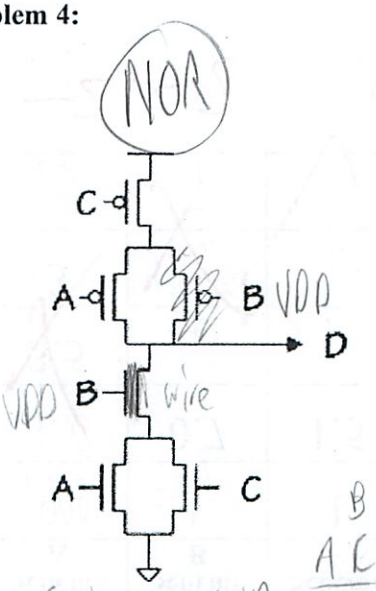
(complete table – 10 entries)

Scratch copies of diagram from problem 4:

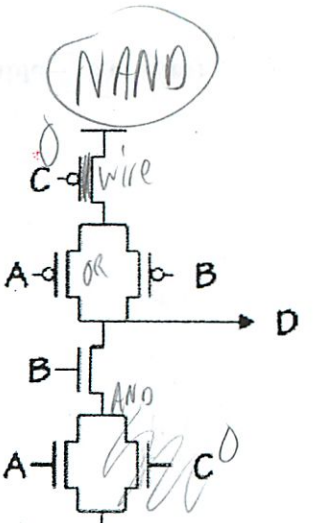


On when low

On when high



Could you NAND NOR OR



A	B	C	T	B	D				
0	0	0	✓	X	1				
0	0	1	X	X					
0	1	0	X	✓	1				
0	1	1	✓	X	0				
1	0	0	X	✓	1				
1	0	1	X	X					
1	1	0	✓	X	0				
1	1	1	✓	X	0				

When both are closed what happens?

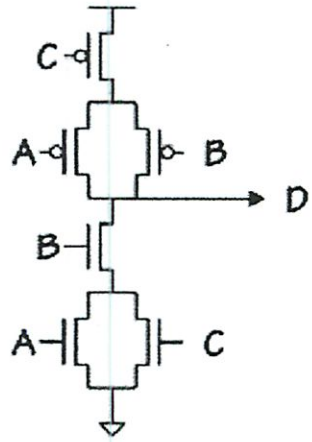
$A \cdot B = F$

$\bar{A} \cdot \bar{B} = \bar{F}$

C, Fast 0 AND 0 = 0 C,

Problem 3. (7 points): **Saga of Helena Handbasket**

Helena Handbasket, who barely passed 6.004, has been hired to design CMOS gates for Hapless Logic, Inc. Remembering something about PFETs in pullups and NFETs in pulldowns, her first design was a 3-input device whose circuit is shown to the right.



Helena's intent was that these devices compute some useful 3-input Boolean function D=F(A,B,C); unfortunately, the devices don't seem to work as planned. To make matters worse, she had 1,000,000,000 of the devices fabricated (thinking that the order to the fab must, of course, be in binary rather than decimal). The defective devices, known within HLI as the Gates of Helena, have become a subject of ridicule.

Helena has brought you in as a consultant. Your first task is to figure out how badly Helena blew the design of her 3-input logic device – in particular, whether it drives the output D to a valid logic level for every combination of the inputs A, B, and C. You may use the scratch diagrams on the back of the previous page for your work if you like.

(A) (2 points) Are there logical (0/1) values of A, B, and C for which D is not driven at all? If so, give values for A, B, and C that leave D undriven; else write NONE.

A, B, C values for undriven D, or "NONE": (001) (101)

(B) (2 points) Are there logical values of A, B, and C for which D is pulled down and up simultaneously? If so, values for A, B, and C that cause such a conflict; else write NONE.

A, B, C values for output conflict, or "NONE": None

good - otherwise could burn them all

Nora Nanda, Helena's assistant, suggests that the devices might be salvaged by using them to compute useful functions of fewer than 3 inputs. She proposes that a two-input function of X and Y, for example, might be computed by connecting each of the three inputs A, B, and C to either **X**, **Y**, or the logical constants **0** (ground) or **1** (vdd), and reading the output on D.

(C) (1 point) Can Nora's approach be used to compute **NAND** of X and Y? If so, choose values (X, Y, 0, or 1) for each of A, B, and C such that D is NAND(X,Y); else circle NO.

Choose A, B, C values or circle NO: A=X; B=Y; C=0; or NO

(D) (1 point) Can Nora's approach be used to compute **NOR** of X and Y? If so, choose values (X, Y, 0, or 1) for each of A, B, and C such that D is NOR(X,Y); else circle NO.

Choose A, B, C values or circle NO: A=X; B=1; C=Y; or NO

(E) (1 point) Can Nora's approach be used to compute **OR** of X and Y? If so, choose values (X, Y, 0, or 1) for each of A, B, and C such that D is OR(X,Y); else circle NO.

Choose A, B, C values or circle NO: A=____; B=____; C=____; or NO

Neg logic only, buy some inverters

END OF QUIZ!

1	/ 4
2	/13
3	/ 6
4	/ 7
	/30

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Fall 2011

Quiz #1: September 23 2011

Name Solutions			*Athena login name*	*Score* 30
Brad, 26-322 ☐ WF 10 ☐ WF 11	**Silvina**, 34-303 ☐ WF 11 ☐ WF 12	**Li-Shiuan**, 34-304 ☐ WF 12 ☐ WF 1	**Chris**, 34-303 ☐ WF 1 ☐ WF 2	**David**, 36-155 ☐ WF 2 ☐ WF 3

Notes:

(1) PLEASE FILL IN YOUR NAME, ATHENA NAME, and MARK YOUR SECTION ABOVE before working on the problems on this quiz.

(2) Occasionally, reference materials and/or extra copies of diagrams are provided on the backs of quiz pages. Feel free to use page backs as scratch space, but be sure to mark your answers on the answer lines provided for each question.

Problem 1 (4 points): **Quickies and Trickies**

(A) What decimal integer is represented by the 5-bit two's complement binary number **11100**?

Decimal equivalent of two's complement 11100: -4

(B) Is it possible for an inverter to have a contamination delay that is greater than its propagation delay?

Can t_{cd} be greater than t_{pd}? circle one: YES ... CAN'T TELL ... (NO)

(C) In lecture we saw that a 2-input NOR gate might obey the static discipline (i.e., be a valid combinational device) without being a *lenient* combinational device. Is it possible for an inverter to be a valid combinational device without being lenient?

Might a valid inverter not be lenient? circle one: YES ... CAN'T TELL ... (NO)

(D) A properly-designed CMOS gate has N inputs and one output, and is built using N-1 PFETs (as well as some NFETs). What conclusion can you draw?

C1: It uses NFETs in its pullup circuit.
C2: Its output is always 0.
C3: Its output is independent of at least one of the N inputs.
C4: It is not lenient.
C5: None of the above.

Circle best choice: C1 ... C2 ... (C3) ... C4 ... C5

Problem 2. (13 points): **UVWXYZ Coding**

A secretive government agency has a large amount of data that has been encrypted as strings of letters from the alphabet UVWXYZ. They need to transmit this data over a binary channel (i.e., one that communicates a sequence of 0's and 1's), and have asked for your help.

They ask you to design a fixed-length code that transmits each of the U, V, W, X, Y, or Z letters as a message codeword (binary string).

(A) (1 point) Using the shortest possible fixed-length code, how many bits are transmitted for each message codeword (i.e., for each letter from the UVWXYZ alphabet)?

Bits transmitted for each UVWXYZ message: 3 **bits**

You design a fixed-length coding that translates each of the six letters to a minimally-sized codeword, and present it to to the agency. Your contact there asks, perhaps to impress you with his technical acumen, what the minimal Hamming distance is between valid codewords in the fixed-length code you've supplied.

(B) (1 point) What is the minimal Hamming distance between valid code words in your fixed-length code?

Minimal Hamming distance between valid codewords: 1

Max Noyes, an agency spokesperson, is concerned about single-bit errors that may be occasionally be introduced during codeword transmission. He probes you about error-detecting characteristics of your simple fixed-length code.

(C) (1 point) Can your code detect all single-bit errors?

***Detect* single-bit errors? Circle one: YES ... (NO)**

In order to improve its error-handling characteristics, you add a single parity bit to each codeword. You choose to use odd parity, i.e. you set the parity bits to make the total number of 1's in each codeword odd.

(D) (1 point) What is the minimal Hamming distance between valid code words in your fixed-length code with the added parity bit?

Minimal Hamming distance between valid codewords: 2

(E) (1 point) Can your revised code *detect* all single-bit errors? Can it *correct* them?

Single-bit errors can be (Circle all that apply): (DETECTED) .. CORRECTED

Exploring options, Max asks what the minimum Hamming distance between valid codewords would have to be to allow for correction of both 2-bit and 1-bit errors.

(F) (1 point) What is your answer? What is the necessary Hamming distance?

Minimal Hamming distance to correct 2-bit errors: 5

Les Bitz is suddenly put in charge of the project and is determined to cut costs. He insists on abandoning all worries about error detection and correction, and wants to focus on minimizing the number of bits that must be transmitted. Les provides you with some key additional information: he knows the probability of each of the six letters (that is, the relative frequency at which they appear in strings to be transmitted). Les provides you with the following partially-filled table:

Message	Probability	Info (bits)
U	1/2	1
V	1/4	2
W, X, Y, Z	1/16	4

The table gives the probability of each of the six messages. The remaining column, reflecting the number of bits of information communicated by each message, has been left blank.

(G) (3 points) Complete the table by filling in the missing three entries.

(Complete above table)

(H) (1 point) What is the average amount of information conveyed by each message?

Average info per message: 2 **bits**

Responding to pressure from Les to reduce the number of bits transmitted, you remember the Huffman coding scheme you learned in 6.004 and use it to design an appropriate variable-length code.

(I) (2 points) In your Huffman coding scheme, how many bits are transmitted for a U? For a V? For a W?

Codeword size for U: 1 **; for V:** 2 **; for W:** 4

(J) (1 point) In your Huffman coding scheme, what is the average number of bits transmitted for each message?

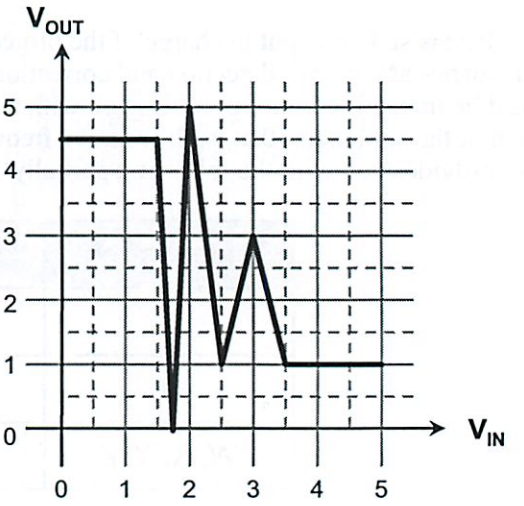
Average bits transmitted per message: 2 **bits**

Problem 3. (6 points) **Logical Negativism**

Logical Negativism, Inc is an MIT spinoff whose aim is to market a new logic family whose only component type is an inverter. Their motto, *Logic is NOT everything*, has convinced a venture group that this business plan has merit, and they are now well funded.

Their founder, Ms. Anna Logge, has developed a device to be used as their inverter; its static voltage transfer curve is shown to the right. Anna is considering the choice of parameters by which LNI's logic family will represent logic values, and needs your help.

The voltage transfer curve of a proposed inverter for a new logic family is shown to the right (spare copies of this diagram are on the back of page 3 of this quiz).



Several possible schemes for mapping logic values to voltages are being considered, as summarized in the incomplete table below. **Recall that Noise Immunity (last row) is defined as the lesser of the two noise margins.**

Complete LNI's table, by filling in missing entries. Choose each value you enter so as to maximize the noise margins of the corresponding scheme. **If the numbers in a scheme can't be completed such that the LNI device functions as an inverter with positive noise margins, put an X in the entries for that column.**

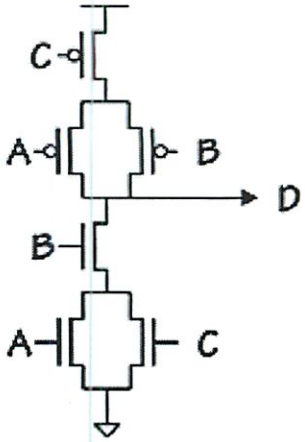
LNI's Possible Logic Mappings:

	Scheme A	**Scheme B**	**Scheme C**
V_{OL}	X	X	1
V_{IL}	X	0.7	1.5
V_{IH}	3.5	X	3.5
V_{OH}	3	X	4.5
Noise Immunity	X	X	0.5

(complete table – 10 entries)

Problem 3. (7 points): **Saga of Helena Handbasket**

Helena Handbasket, who barely passed 6.004, has been hired to design CMOS gates for Hapless Logic, Inc. Remembering something about PFETs in pullups and NFETs in pulldowns, her first design was a 3-input device whose circuit is shown to the right.



Helena's intent was that these devices compute some useful 3-input Boolean function D=F(A,B,C); unfortunately, the devices don't seem to work as planned. To make matters worse, she had 1,000,000,000 of the devices fabricated (thinking that the order to the fab must, of course, be in binary rather than decimal). The defective devices, known within HLI as the Gates of Helena, have become a subject of ridicule.

Helena has brought you in as a consultant. Your first task is to figure out how badly Helena blew the design of her 3-input logic device – in particular, whether it drives the output D to a valid logic level for every combination of the inputs A, B, and C. You may use the scratch diagrams on the back of the previous page for your work if you like.

(A) (2 points) Are there logical (0/1) values of A, B, and C for which D is not driven at all? If so, give values for A, B, and C that leave D undriven; else write NONE.

A, B, C values for undriven D, or "NONE": B=0, C=1 (A=1 or 0)

(B) (2 points) Are there logical values of A, B, and C for which D is pulled down and up simultaneously? If so, values for A, B, and C that cause such a conflict; else write NONE.

A, B, C values for output conflict, or "NONE": None

Nora Nanda, Helena's assistant, suggests that the devices might be salvaged by using them to compute useful functions of fewer than 3 inputs. She proposes that a two-input function of X and Y, for example, might be computed by connecting each of the three inputs A, B, and C to either **X**, **Y**, or the logical constants **0** (ground) or **1** (vdd), and reading the output on D.

(C) (1 point) Can Nora's approach be used to compute **NAND** of X and Y? If so, choose values (X, Y, 0, or 1) for each of A, B, and C such that D is NAND(X,Y); else circle NO.

Choose A, B, C values or circle NO: A=X; B=Y; C=0; **or NO**

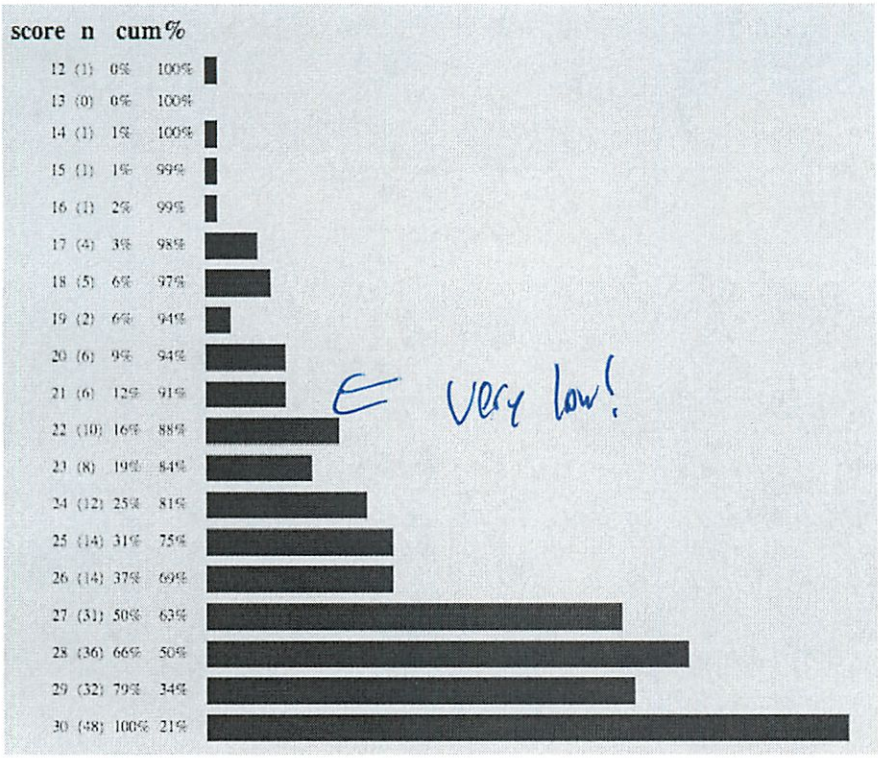
(D) (1 point) Can Nora's approach be used to compute **NOR** of X and Y? If so, choose values (X, Y, 0, or 1) for each of A, B, and C such that D is NOR(X,Y); else circle NO.

Choose A, B, C values or circle NO: A=X; B=1; C=Y; **or NO**

(E) (1 point) Can Nora's approach be used to compute **OR** of X and Y? If so, choose values (X, Y, 0, or 1) for each of A, B, and C such that D is OR(X,Y); else circle NO.

Choose A, B, C values or circle NO: A=____; B=____; C=____; **or NO**

END OF QUIZ!


score n cum%
12 (1) 0% 100%
13 (0) 0% 100%
14 (1) 1% 100%
15 (1) 1% 99%
16 (1) 2% 99%
17 (4) 3% 98%
18 (5) 6% 97%
19 (2) 6% 94%
20 (6) 9% 94%
21 (6) 12% 91%
22 (10) 16% 88%
23 (8) 19% 84%
24 (12) 25% 81%
25 (14) 31% 75%
26 (14) 37% 69%
27 (31) 50% 63%
28 (36) 66% 50%
29 (32) 79% 34%
30 (48) 100% 21%
Very low!