BEE 271 Digital circuits and systems

Spring 2017

Lecture 1: Orientation and intro to logic circuits

Nicole Hamilton

https://faculty.washington.edu/kd1uj

Nicole Hamilton

https://faculty.washington.edu/kd1uj/
kd1uj@uw.edu
H: 425-702-8184
C: 425-765-9574

Office hours by appointment
(I do not have an on-campus office.)

Education

BS & MS EE, Stanford, 1973.

MBA, Boston University, 1987.

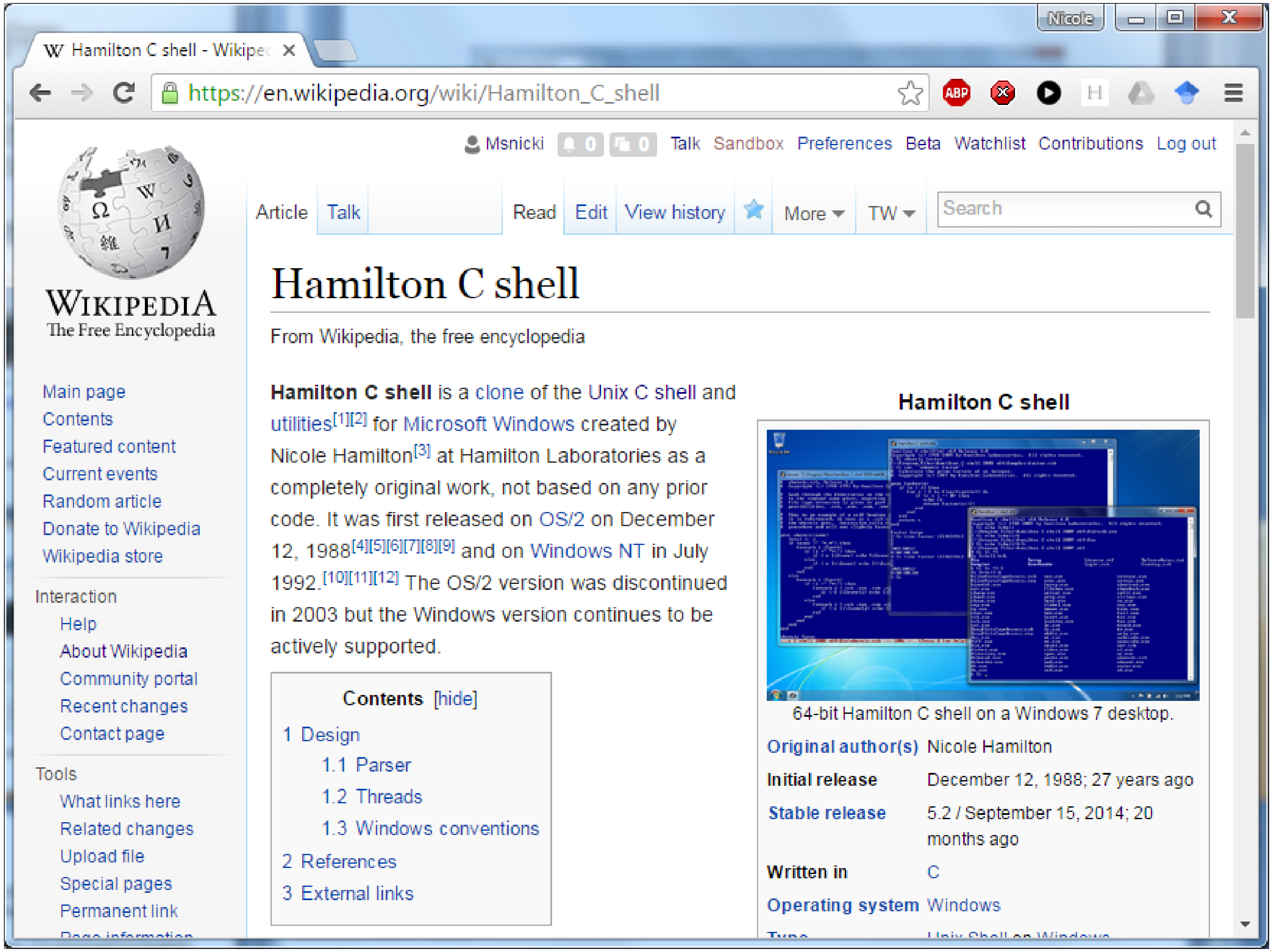
Background

Most of it as an entrepreneur selling a C shell I wrote for Windows.

Also worked at IBM, Microsoft and RealNetworks.

At Microsoft, I wrote the ranker and query language for the first release of what's now Bing.

Here at UWB since 2013, initially as a Capstone advisor.


Hamilton C shell - Wikipedia

https://en.wikipedia.org/wiki/Hamilton_C_shell

Msnicki 0 0 Talk Sandbox Preferences Beta Watchlist Contributions Log out

Article Talk Read Edit View history More TW Search

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link

Hamilton C shell

From Wikipedia, the free encyclopedia

Hamilton C shell is a clone of the Unix C shell and utilities[1][2] for Microsoft Windows created by Nicole Hamilton[3] at Hamilton Laboratories as a completely original work, not based on any prior code. It was first released on OS/2 on December 12, 1988[4][5][6][7][8][9] and on Windows NT in July 1992.[10][11][12] The OS/2 version was discontinued in 2003 but the Windows version continues to be actively supported.

Contents [hide]

Hamilton C shell

64-bit Hamilton C shell on a Windows 7 desktop.

Original author(s)	Nicole Hamilton
Initial release	December 12, 1988; 27 years ago
Stable release	5.2 / September 15, 2014; 20 months ago
Written in	C
Operating system	Windows

Download a free copy from my faculty page.

This is my sixth time teaching this class.
I hope to improve each time.
I do pay attention to my student evaluations.

Lectures

Mondays and Wednesdays
5:45 pm to 7:45 pm
Beardslee 260

Labs

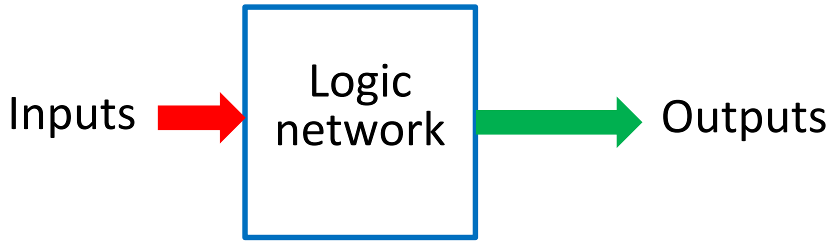
Mondays
3:30 pm to 5:30 pm
Beardslee 220

Topics

1. Combinatorial logic.
2. Synchronous sequential logic and finite state machines.
3. Verilog and FPGAs.

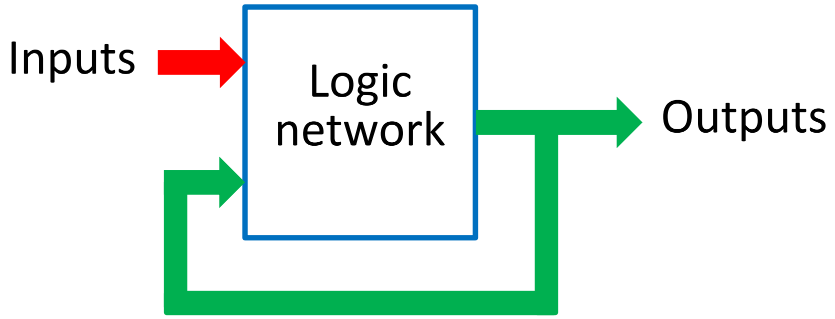
Combinatorial vs. Sequential Logic

Combinatorial or combinational logic



1. No memory elements.
2. No feedback from the outputs to the inputs.
3. Outputs depend only on the inputs.

Sequential logic



1. Contains memory that can remember a present state.
2. Outputs feed back to the inputs.
3. Outputs depend on both inputs and the present state.

Will not cover

1. Asynchronous (non-clocked) logic.
2. Processor architecture, pipelining, etc.

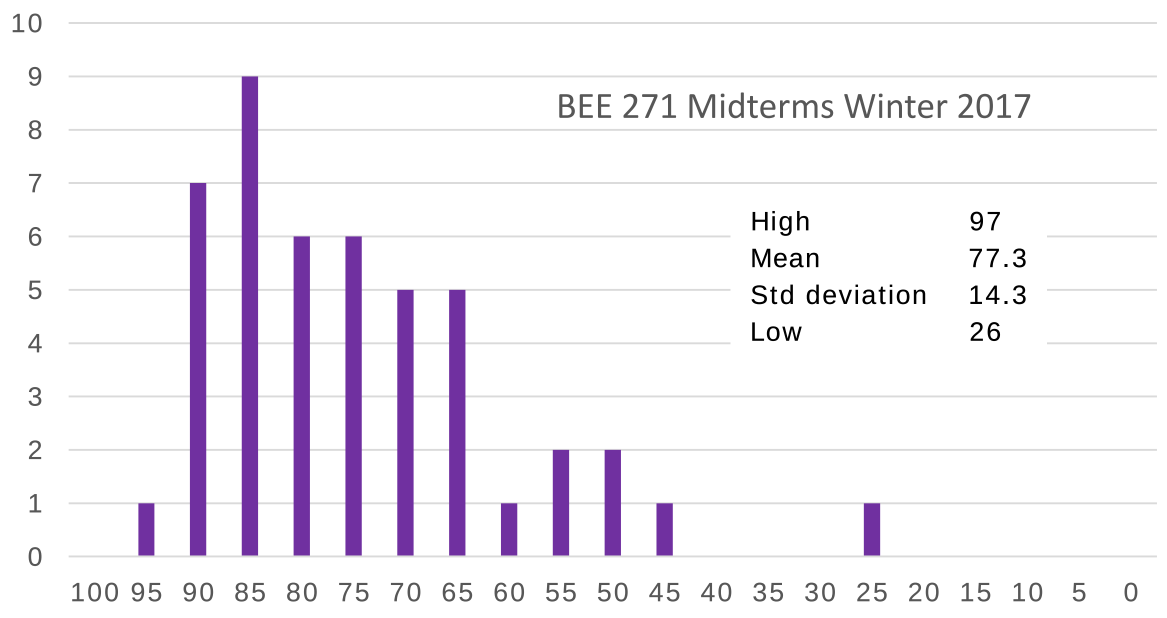
Grading

Homework	10%
Labs	30%
Midterm	30%
Final	30%

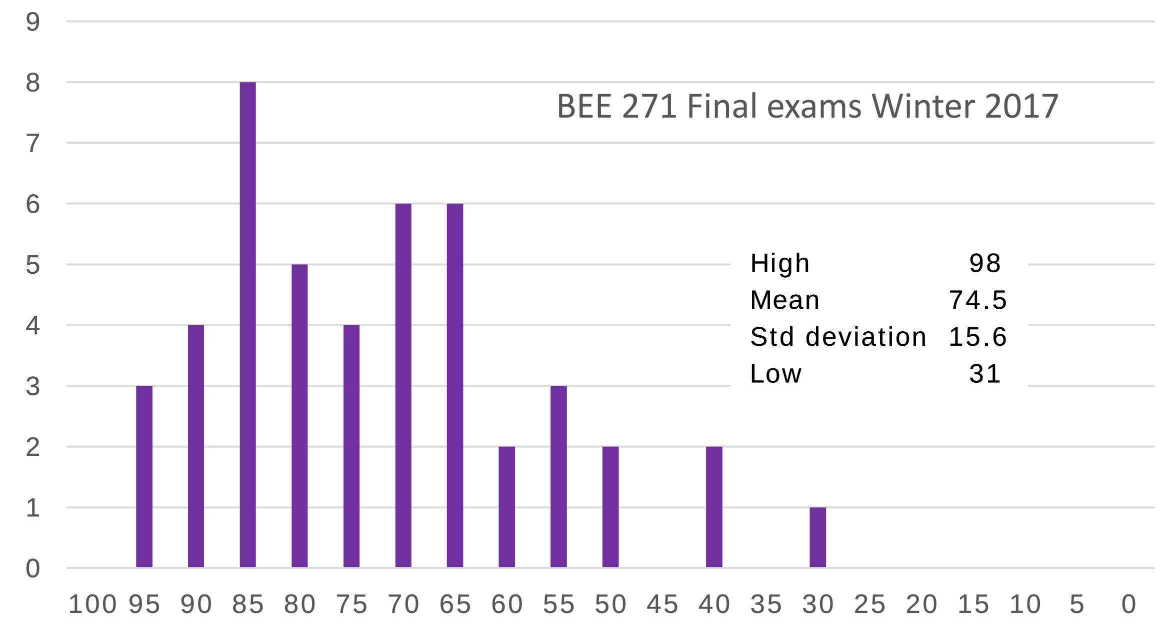
You may do both labs and homework in pairs.

I grade on the curve. I expect most grades should fall between 2.7 and 4.0 with a mean around 3.3 to 3.4, then fit the results.

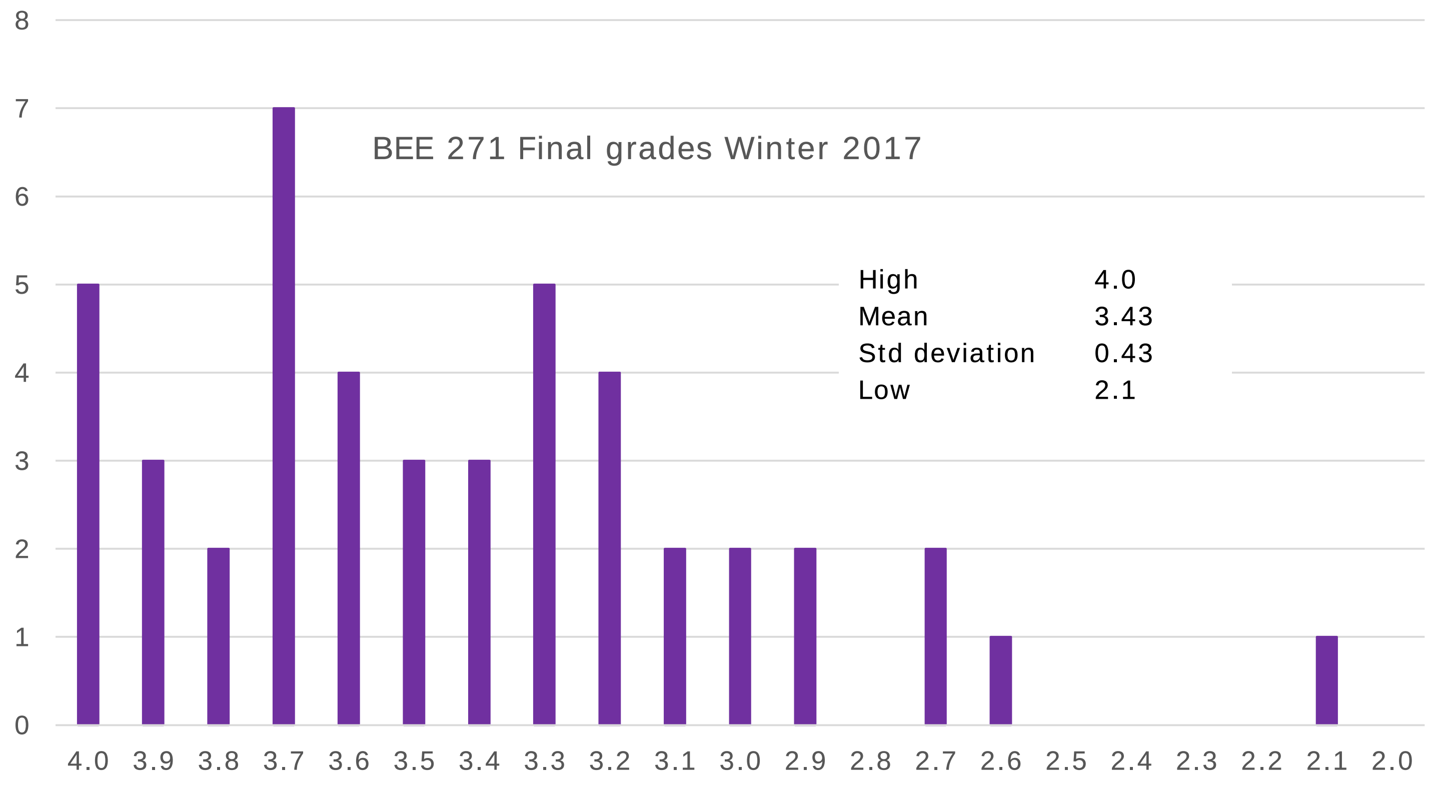
Most people do well on homework and labs, so most of the difference between a 4.0 and a 2.7 is performance on the exams.



High	97
Mean	77.3
Std deviation	14.3
Low	26

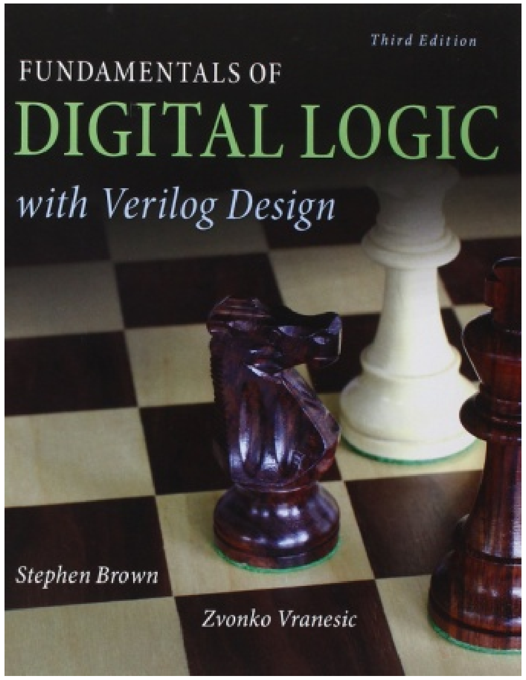


High	98
Mean	74.5
Std deviation	15.6
Low	31



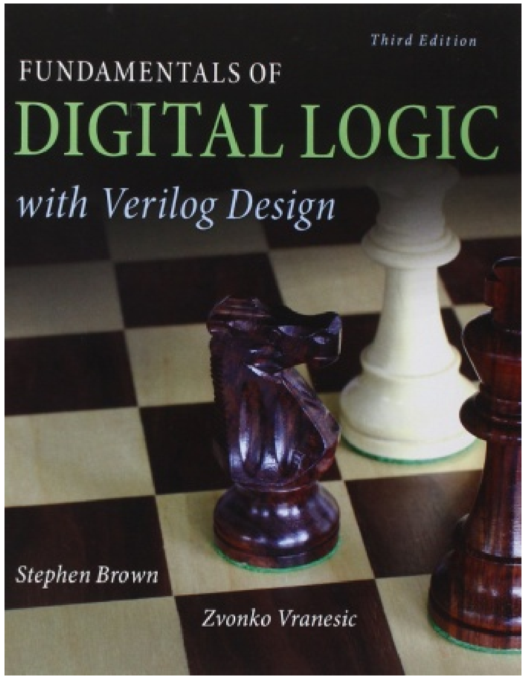
High	4.0
Mean	3.43
Std deviation	0.43
Low	2.1

Required text



Fundamentals of Digital Logic with Verilog Design, Third Edition
Stephen Brown
Zvonko Vranesi
McGraw-Hill Education, 2013
ISBN 978-0073380544

Required text



We will cover chapters 1 through 6 plus section 9.2 on hazards.

You will also need to read Appendix A, a tutorial on Verilog.

All the work must be your own

1. Copying answers from another student or off the internet will get a zero, even if you’re clear about where you got them.
2. If you omit the attribution, submit work that’s not your own or try to deceive me with fabricated results, you will, in addition, find yourself reported for academic misconduct.
3. I’m good at spotting misconduct and very good at reporting it.
4. ***I do not give warnings. I report everything.***

Fundamentals of Digital Logic
Chapter 1. Introduction

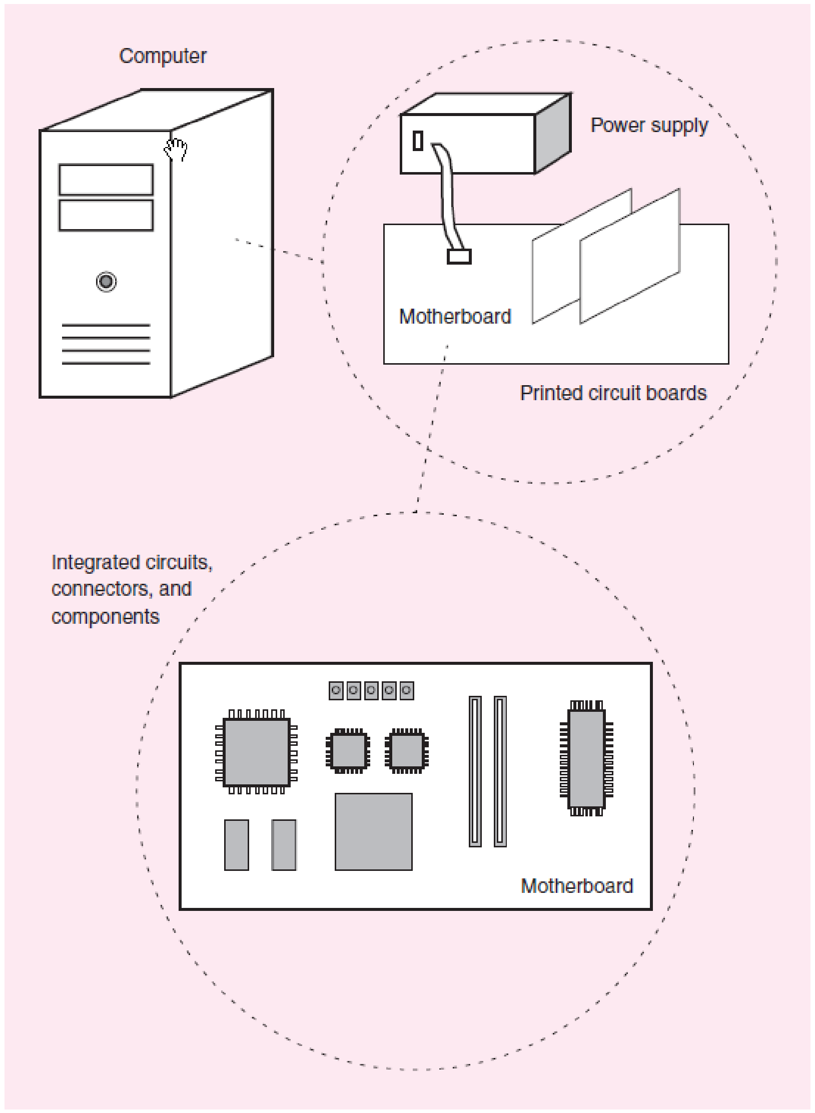


Figure 1.4 A digital hardware system (Part *a*).

Focus of this course

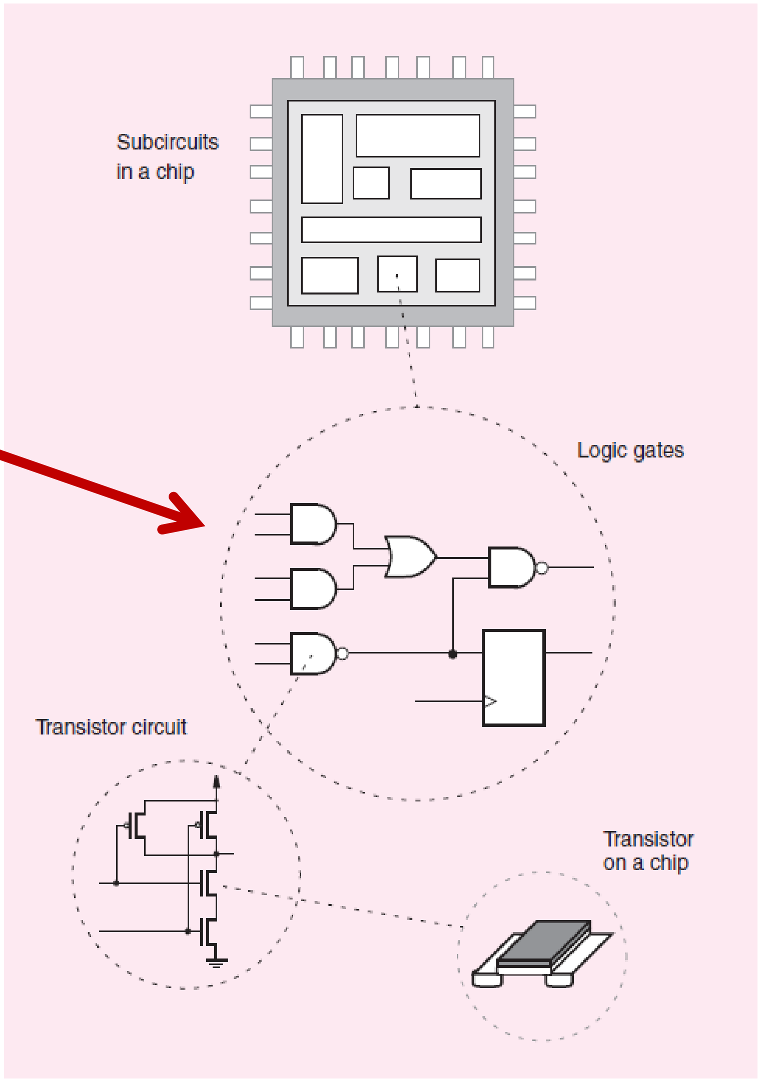


Figure 1.4 A digital hardware system (Part *b*).

Digital hardware devices

Standard chips

7400 series transistor-transistor logic (TTL) small scale integration (SSI) parts.

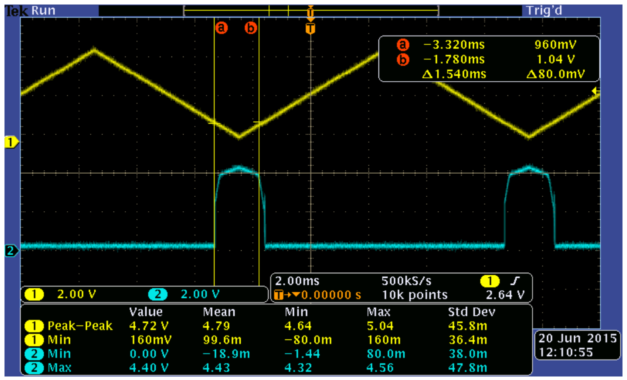
Field programmable logic devices (FPGAs)

Programmed with Verilog or VHDL hardware programming languages or by dragging and dropping logic gates onto a schematic to do anything you want.
Downsides: Volatile and slow.

Application-specific integrated circuits (ASICs)

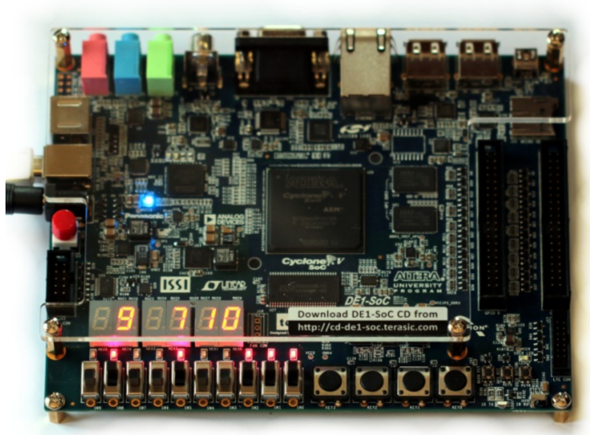
Custom or semi-custom depending on the volume.
Hugely expensive.
Allows a single chip to be used, conserving board space and making a tiny final product possible.

Four labs to support the lectures.

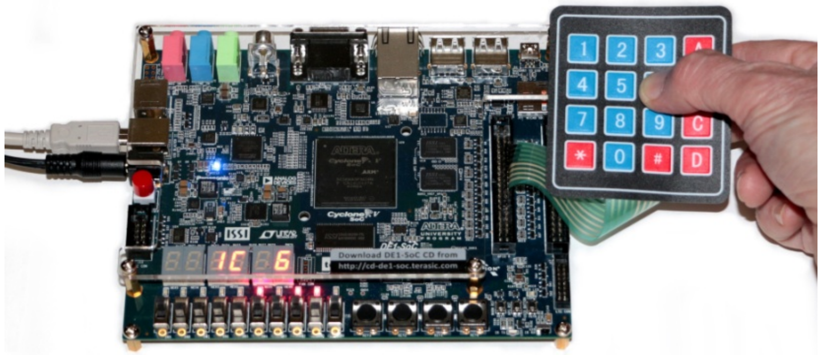


```
Tek Run                                              Trig'd
                                   a  -3.320ms     960mV
                                   b  -1.780ms     1.04 V
                                      Δ1.540ms     Δ80.0mV
1  2.00 V      2  2.00 V      2.00ms       500kS/s      1 ʃ
                              T→▼0.00000 s 10k points   2.64 V
               Value    Mean     Min      Max      Std Dev
1 Peak-Peak    4.72 V   4.79     4.64     5.04     45.8m
1 Min          160mV    99.6m    -80.0m   160m     36.4m
2 Min          0.00 V   -18.9m   -1.44    80.0m    38.0m
2 Max          4.40 V   4.43     4.32     4.56     47.8m
                                                   20 Jun 2015
                                                   12:10:55
```

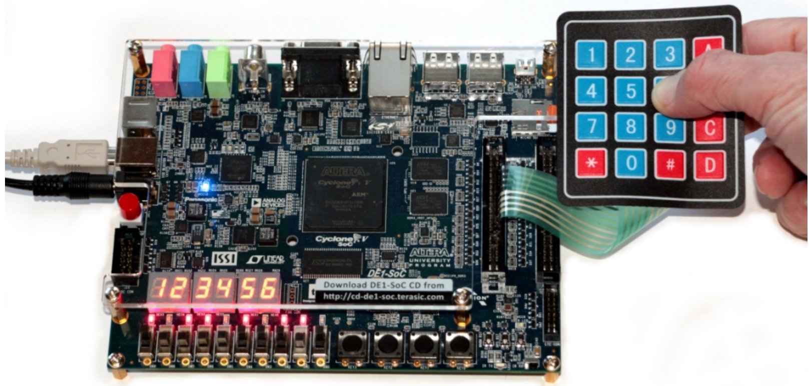
1. Digital logic devices.



2. Hex adding machine.

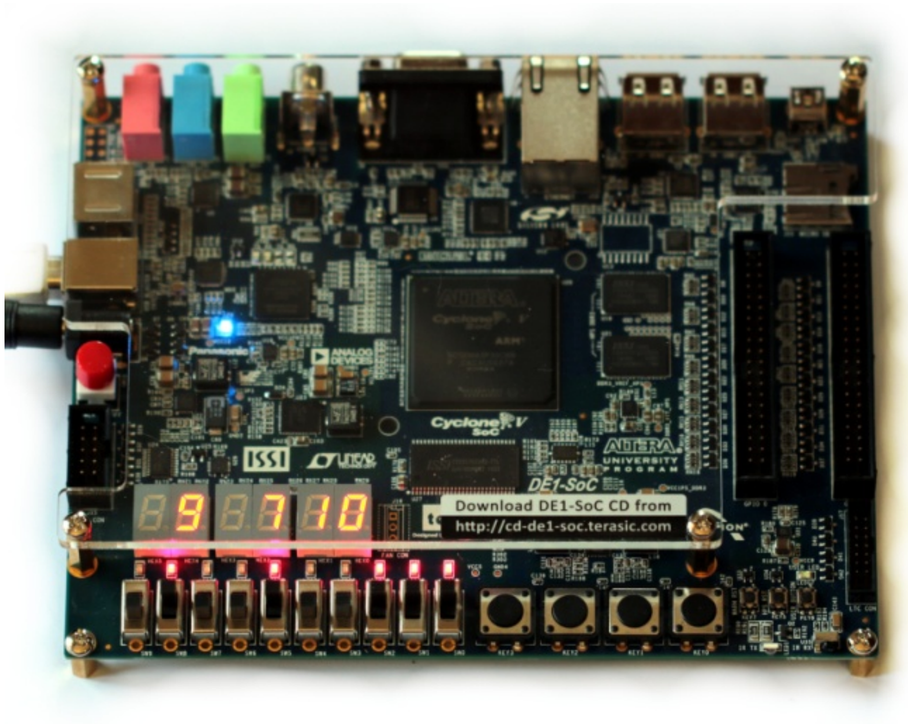


3. Keypad scanner.



4. Keypad debouncer.

Terasic DE1-SoC



$175 academic price

FPGA

Altera Cyclone V SoC

85K programmable logic elements

64 MB SDRAM

Hard processor system (HPS)

Dual-core ARM

2 hard memory controllers

Runs Linux and “bare metal” apps

Board

6 seven segment displays

10 switches

10 LEDs

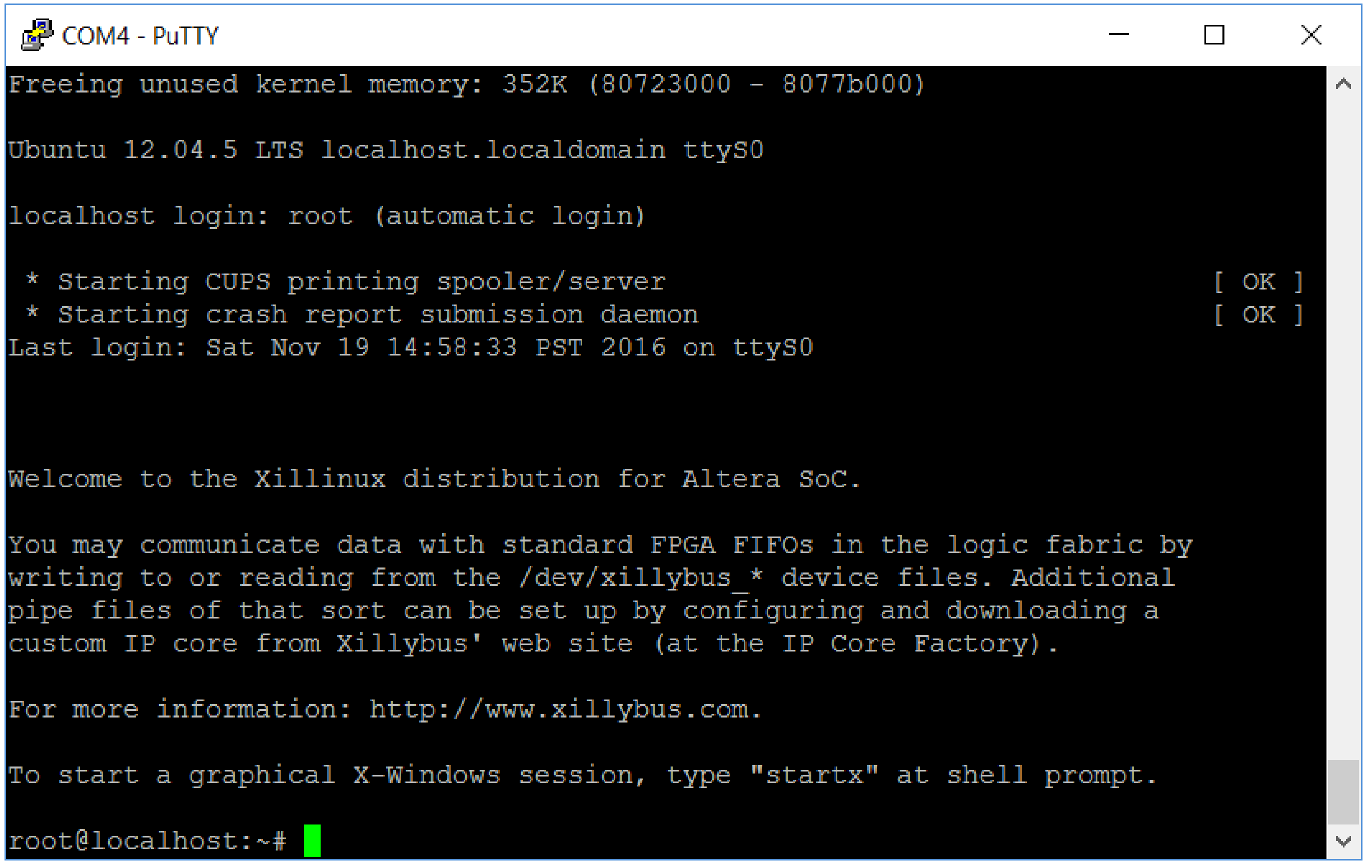
4 pushbuttons

2 40-pin GPIO headers

1 GB DDR3 SDRAM

+ Micro SD, USB, Ethernet, VGA, ADC, keyboard, mouse, audio, video

Able to boot Ubuntu Linux from a Micro SD card as a command window via PuTTY.



```
Freeing unused kernel memory: 352K (80723000 - 8077b000)

Ubuntu 12.04.5 LTS localhost.localdomain ttyS0

localhost login: root (automatic login)

 * Starting CUPS printing spooler/server                                [ OK ]
 * Starting crash report submission daemon                              [ OK ]
Last login: Sat Nov 19 14:58:33 PST 2016 on ttyS0



Welcome to the Xillinux distribution for Altera SoC.

You may communicate data with standard FPGA FIFOs in the logic fabric by
writing to or reading from the /dev/xillybus_* device files. Additional
pipe files of that sort can be set up by configuring and downloading a
custom IP core from Xillybus' web site (at the IP Core Factory).

For more information: http://www.xillybus.com.

To start a graphical X-Windows session, type "startx" at shell prompt.

root@localhost:~#
```

With a keyboard, mouse and VGA display, it runs the Ubuntu desktop.

Software environment

Quartus Prime

The Intel/Altera IDE for compiling Verilog to the FPGA, including SystemBuilder to create a new DE1-SoC project.

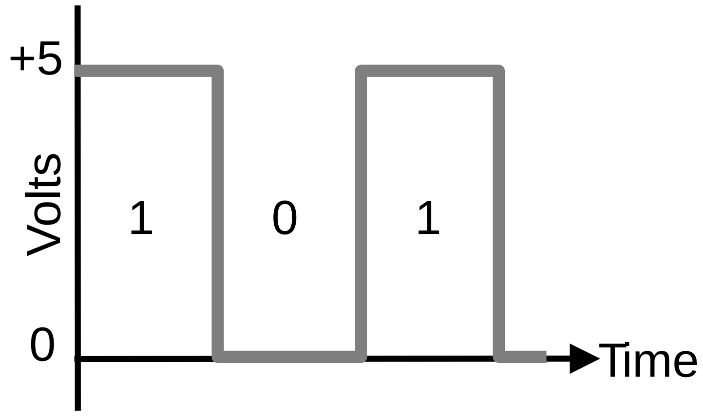
SignalTap II

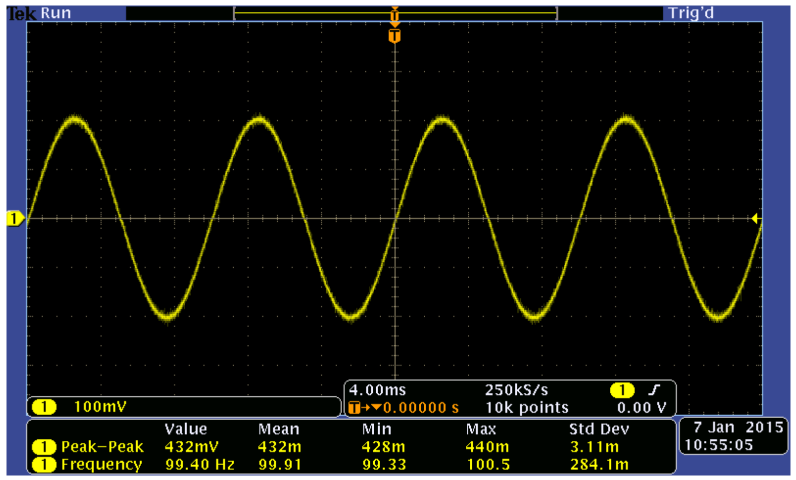
A remote logic analyzer that can be compiled onto the FPGA along with your own code.

ModelSim

The Verilog simulator.

Digital vs. Analog





```
4.00ms              250kS/s        1
T→▼0.00000 s        10k points     0.00 V

                Value      Mean    Min     Max     Std Dev     7 Jan  2015
1 Peak–Peak     432mV      432m    428m    440m    3.11m       10:55:05
1 Frequency     99.40 Hz   99.91   99.33   100.5   284.1m
```

Digital means data is represented by *discrete* values.

Binary/Boolean means we use only *2* values.

1 = Yes, on, 5 V, high, TRUE
0 = No, off, 0 V, low, FALSE

Analog means the values vary *continuously* over a possibly very broad range.

Digital representation of information

Typical kinds of data we might like to represent in a digital form:

Numbers

Characters

Music

Video

Advantages of Digital Circuits

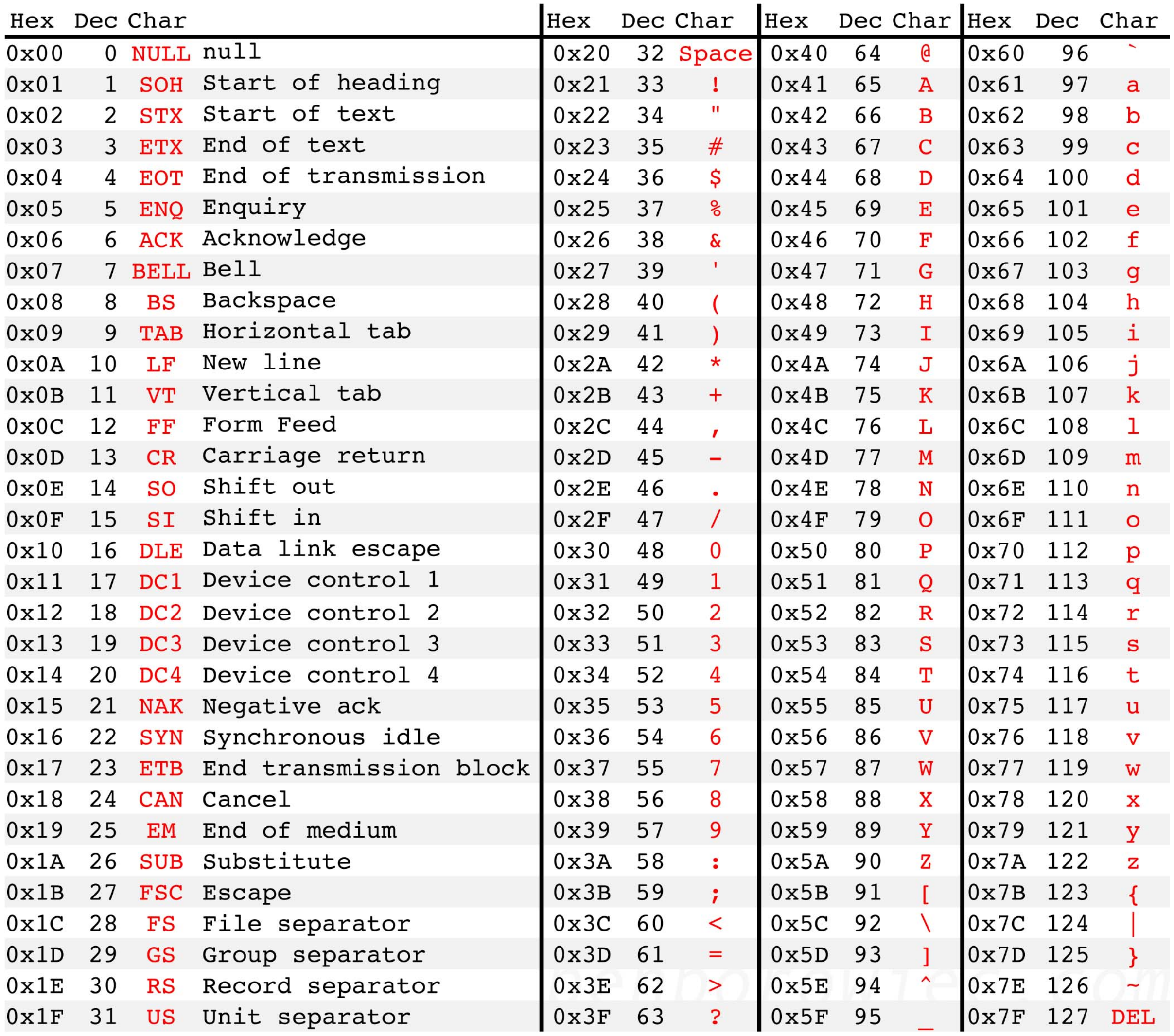
1. Analog systems: slight error in input yields large error in output
2. Digital systems more accurate and reliable

   Readily available as self-contained, easy to cascade building blocks

3. Computers use digital circuits internally
4. Interface circuits (i.e., sensors & actuators) often analog

This course is about logic design, not system design (processor architecture), not circuit design (transistor level)

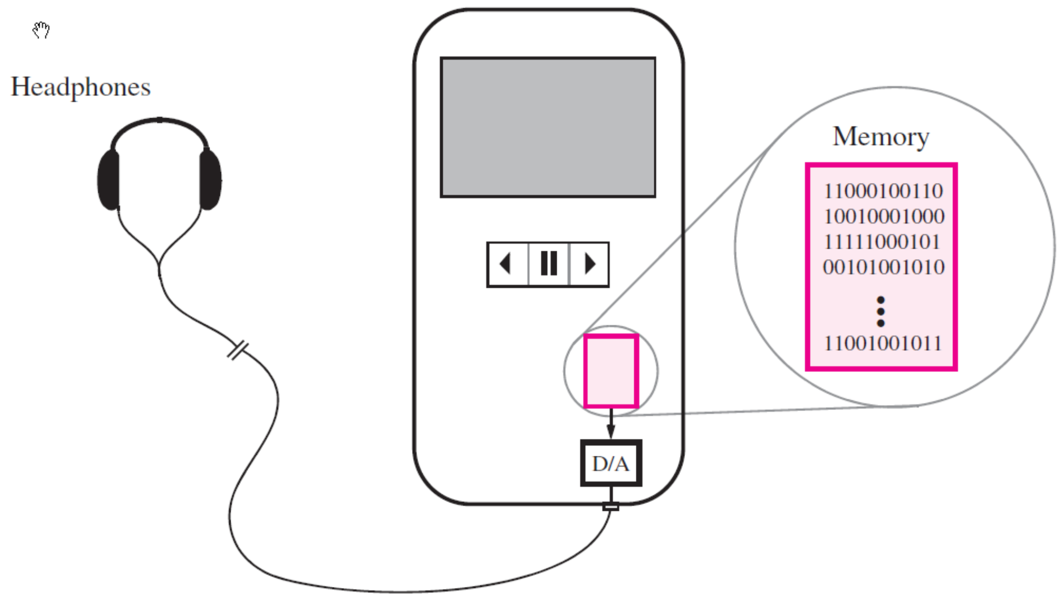
The ASCII character set for representing text



Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Image source: https://www.pinterest.com/pin/569494315354562943/

(Now being supplanted by UNICODE and UTF-8 to support international character sets.)



Numerous audio, image and video file formats.

Specialized hardware to encode or decode the data.

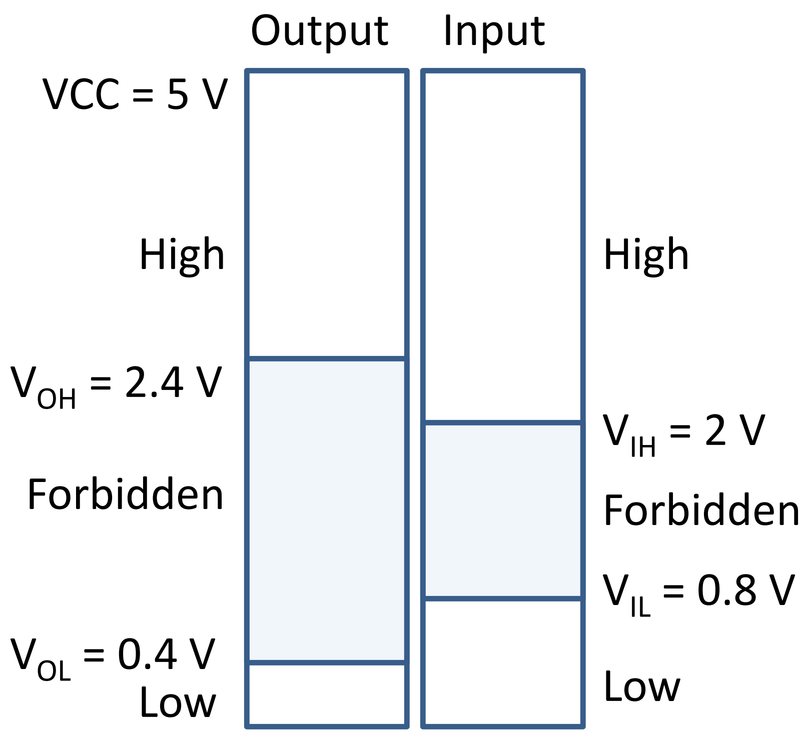
Analog-to-digital (A/D) and digital-to-analog (D/A) converters

Figure 1.7. Using digital technology to represent music.

We only have bits

We represent everything in bits, where each bit can be only a 0 or a 1.

Implemented as voltage levels in a digital circuit, e.g., shown here for TTL.



Binary numbers

The one data format we care about in this class.

Binary numbers

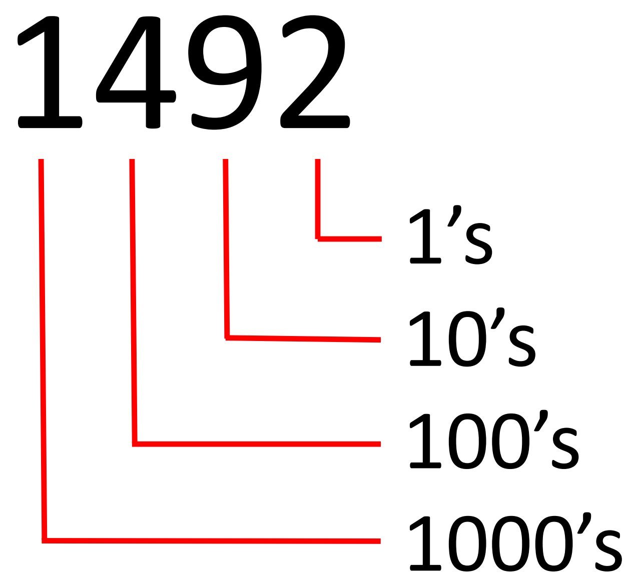
Strings of bits.

Like decimal numbers
but with only 1’s and 0’s.

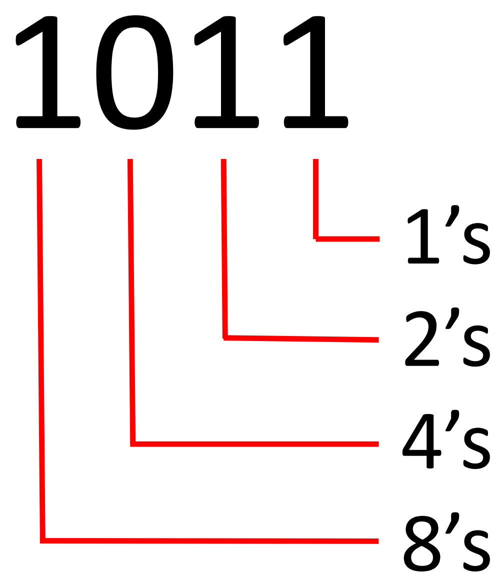
1011

Numbers are positional

In decimal

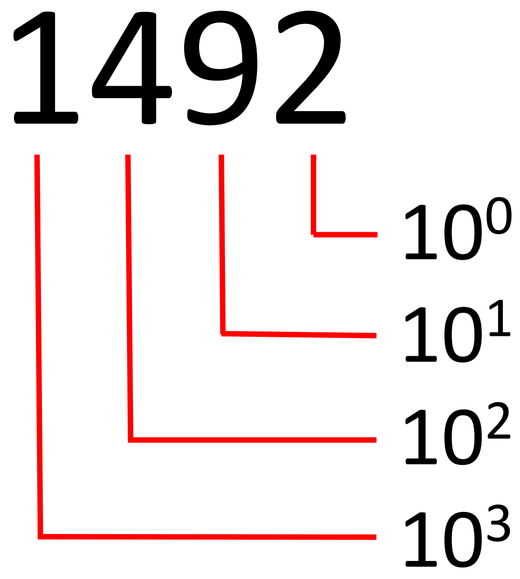


In binary

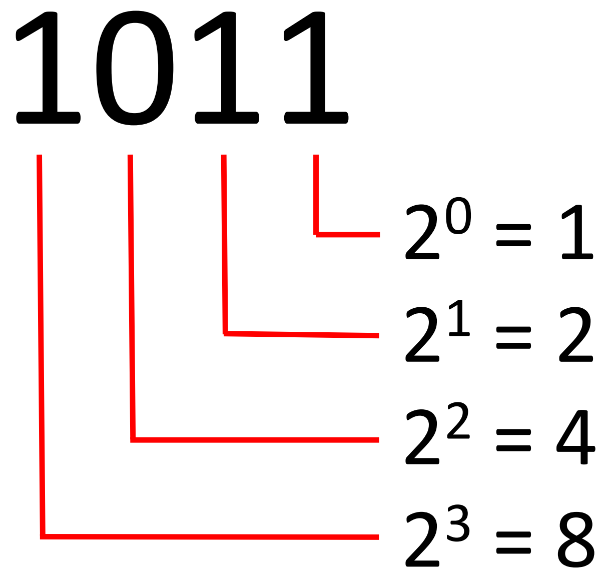


Numbers are positional

In decimal



In binary



Adding zeroes to the left doesn’t change the value.

In decimal

001492 = 1492

In binary

001011 = 1011

When we add numbers we get carries.

In decimal	In binary
110	*011*
1492	1011
+ 525	+ 011
2017	1110

Hex

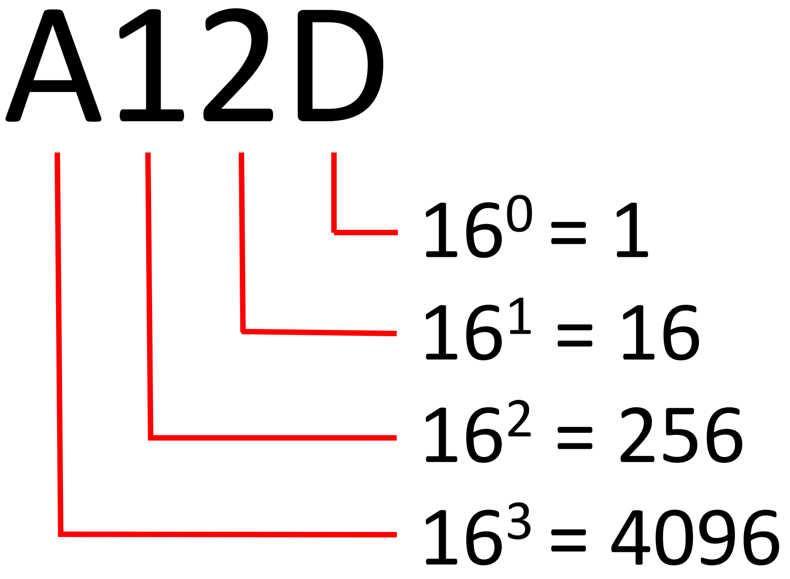
1. Hard to read long strings of nothing but 1's and 0's.
2. So we break it up into groups of 4 bits called *nibbles*, starting *at the LSB*.
3. Take each 4-bit group as a value from 0 to 15.
4. Values 10 to 15 written as A to F.

0111010010011111

0111	0100	1001	1111
7	4	9	F

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

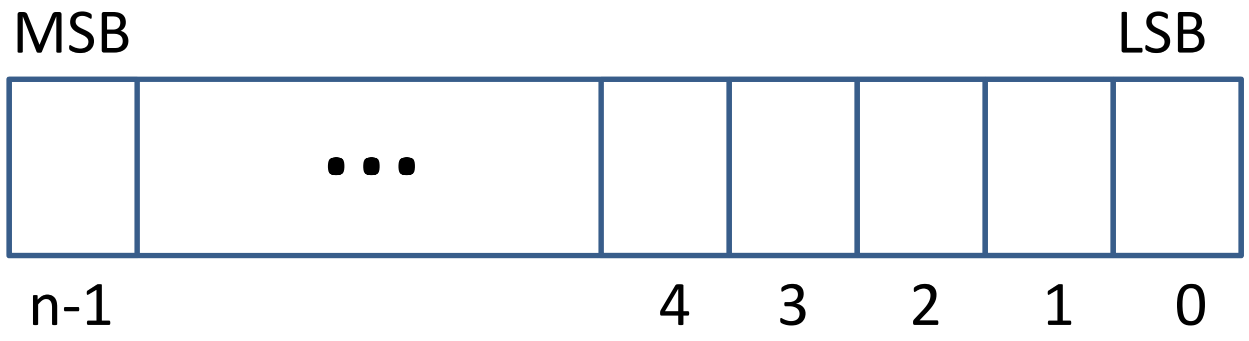
In hex



A12D

$$
\begin{aligned}
13 * 16^0 &= 13 \\
2 * 16^1 &= 32 \\
1 * 16^2 &= 256 \\
10 * 16^3 &= 40960 \\
\hline
& \ 41261
\end{aligned}
$$

Binary numbers

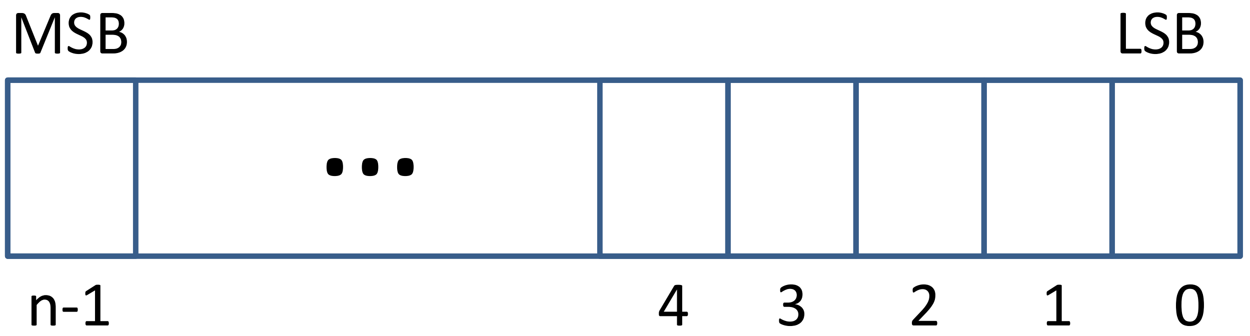


Numbering of the individual bits is from least significant bit (LSB) to most significant bit (MSB).

If b0 = 0, the number is even.
If b0 = 1, the number is odd.

Each bit represents a power of 2.

Value of a binary number



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

Hex

1. Hard to read long strings of nothing but 1's and 0's.
2. So we break it up into groups of 4 bits at a time, starting *at the LSB*.
3. Take each 4-bit group as a value from 0 to 15.
4. Values 10 to 15 written as A to F.

0111010010011111

0111 0100 1001 1111

7 4 9 F

Hexadecimal base 16 notation

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Exercise

What is binary 10100101 in decimal and hex?

Exercise

What is binary 10100101 in decimal and hex?

```
Value       = 1*1 + 0*2 + 1*4 + 0*8 + 0*16
                + 1*32 + 0*64 + 1*128
            = 1 + 4 + 32 + 128
            = 165

10100101  = 1010 0101 = A5 hex
          = A5 = 10*16 + 5 = 165
```

Exercise

What is binary 1111100 in decimal and hex?

Exercise

What is binary 1111100 in decimal and hex?

Value = 0*1 + 0*2 + 1*4 + 1*8 + 1*16 + 1*32 + 1*64
= 4 + 8 + 16 + 32 + 64
= 124

Only 7 bits given, extend with high-order zeros.

10100101 = 111 1100 = 0111 1100 = 7C hex
= 7*16 + 12 = 124

Converting to binary

1. Repeatedly integer divide by 2 until the result is 0.
2. At each step, the remainder is the next bit, starting with the LSB.

(We start at the LSB because the lowest bit is just odd or even.)

Convert 12 to binary

Value	Result	Remainder	
12	6	0	LSB
6	3	0	
3	1	1	
1	0	1	MSB

12 base 10 = 1100 binary = Hex C

Exercise: Convert 957 to binary

Value	Result	Remainder
957		

Exercise: Convert 957 to binary

Value	Result	Remainder	
957	478	1	LSB
478	239	0	
239	119	1	
119	59	1	
59	29	1	
29	14	1	
14	7	0	
7	3	1	
3	1	1	
1	0	1	MSB

957 decimal = 11 1011 1101 binary = 3BD hex

Exercise: Convert 1492 to hex

Value	Result	Remainder
1492		

Exercise: Convert 1492 to hex

Value	Result	Remainder	
1492	746	0	LSB
746	373	0	
373	186	1	
186	93	0	
93	46	1	
46	23	0	
23	11	1	
11	5	1	
5	2	1	
2	1	0	
1	0	1	MSB

1492 decimal = 101 1101 0100 binary = 5D4 hex

Chapter 2

Introduction to Logic Circuits

Example: Warning lights in a car

1. DoorAjar (DriverDoorOpen, PassengerDoorOpen)

2. HighBeamIndicator (LightsOn, HighBeams)

Example: Warning lights in a car

1. DoorAjar (DriverDoorOpen, PassDoorOpen)

   DoorAjar = DriverDoorOpen OR PassengerDoorOpen

2. HighBeamIndicator (LightsOn, HighBeams):

   HighBeamIndicator = LightsOn AND HighBeams

Example: Warning lights in a car

3. SeatBeltLight (DriverBeltIn)

4. SeatBeltLight (DriverBeltIn, PassengerBeltIn, PassengerPresent)

Example: Warning lights in a car

3. SeatBeltLight (DriverBeltIn)

   SeatBeltLight = NOT DriverBeltIn

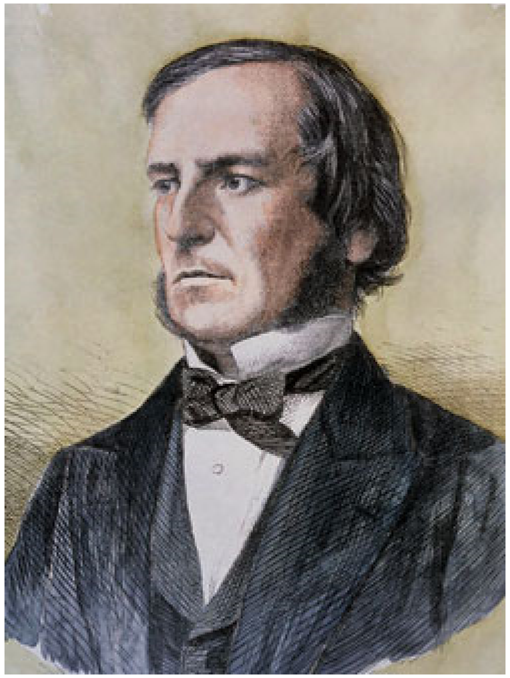
4. SeatBeltLight (DriverBeltIn, PassengerBeltIn, PassengerPresent)

   SeatBeltLight = NOT DriverBeltIn OR PassengerPresent AND NOT PassengerBeltIn

   (We prioritize NOT before AND before OR.)

Boolean algebra

Named after George Boole, who published an algebraic description of the processes involved in logical thought and reasoning in 1849.



https://en.wikipedia.org/wiki/George_Boole

Boolean algebra

In the 1930s, used by Claude Shannon to describe circuits built with switches, and thus with logic circuits.

https://en.wikipedia.org/wiki/Claude_Shannon

ax·i·om

/ˈaksēəm/

noun

a statement or proposition that is regarded as being established, accepted, or self-evidently true.

"the **axiom that** supply equals demand"

synonyms: accepted truth, general truth, dictum, truism, principle; More

- MATHEMATICS

  a statement or proposition on which an abstractly defined structure is based.

Axioms of Boolean Algebra

1a. $0 \bullet 0 = 0$

1b. $1 + 1 = 1$

2a. $1 \bullet 1 = 1$

2b. $0 + 0 = 0$

3a. $0 \bullet 1 = 1 \bullet 0 = 0$

3b. $1 + 0 = 0 + 1 = 1$

4a. If $x = 0$, then $x' = 1$

4b. If $x = 1$, then $x' = 0$

Notice the *duality*:

$$\begin{tikzcd} \bullet \arrow[r, bend left] & + \arrow[l, bend left] \end{tikzcd}$$

$$\begin{tikzcd} 0 \arrow[r, bend left] & 1 \arrow[l, bend left] \end{tikzcd}$$

Duality

Given any logic expression, its dual is obtained by swapping all the + and • operators and ones and zeros.

$$\begin{tikzcd}\bullet \arrow[r, bend left] & + \arrow[l, bend left]\end{tikzcd}$$

$$\begin{tikzcd}0 \arrow[r, bend left] & 1 \arrow[l, bend left]\end{tikzcd}$$

Single-variable theorems

5a.	$x \bullet 0 = 0$	
5b.	$x + 1 = 1$	
6a.	$x \bullet 1 = x$	
6b.	$x + 0 = x$	
7a.	$x \bullet x = x$	Replication
7b.	$x + x = x$	
8a.	$x \bullet x' = 0$	
8b.	$x + x' = 1$	
9.	$(x')' = x$	

Easily proved by *perfect induction,* trying all the possibilities.

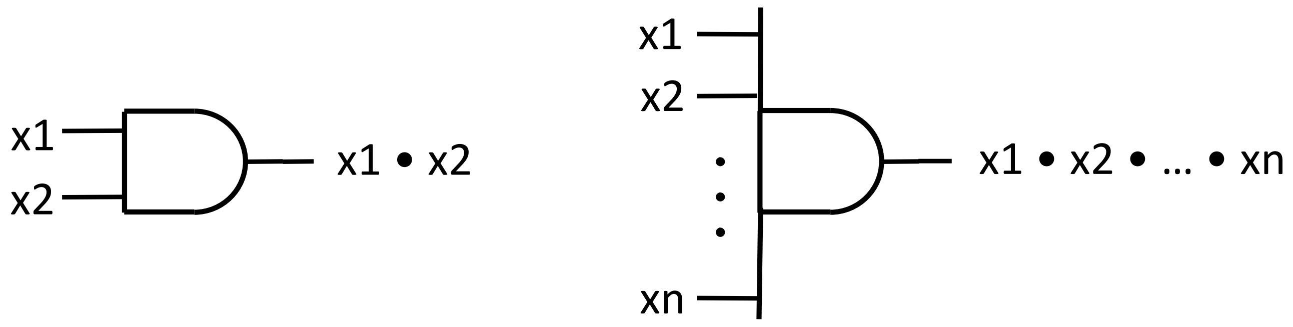
Boolean Algebra

Values	0, 1
Variables	A, B, C, Sum, DoorOpen, ..
Operations	NOT, AND, OR, XOR

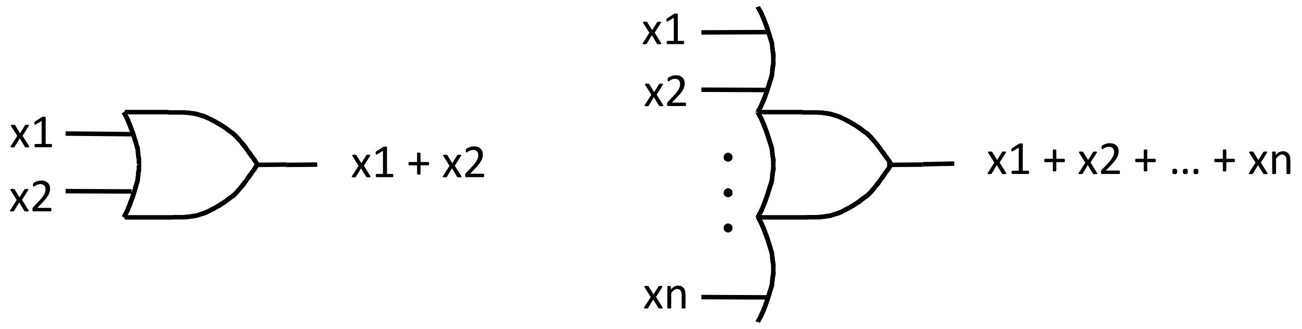
Operation	*Written as*
NOT a	$\overline{a}$ or a’
a AND b	a • b or a b
a OR b	a + b
a XOR b	a ^ b

The basic gates.

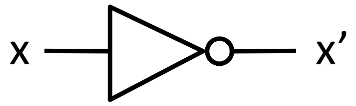
AND If all inputs are true, the output is true.



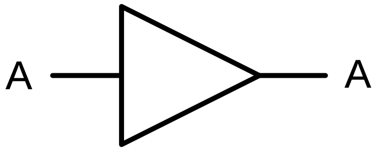
OR If any input is true, the output is true.



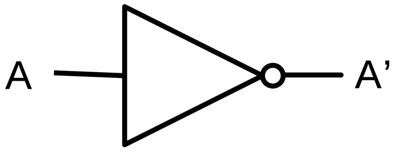
NOT The output is the inverse of the input.



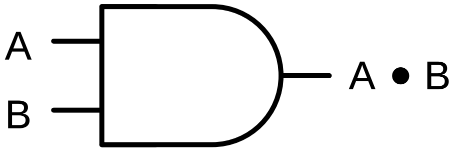
A more complete set of gates



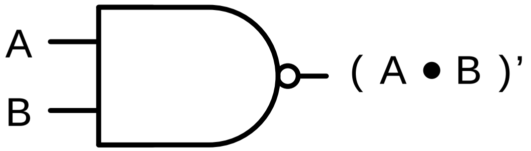
Buffer



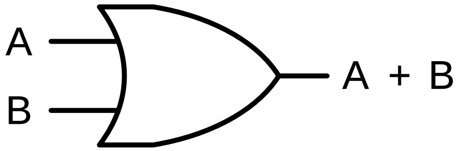
Inverter



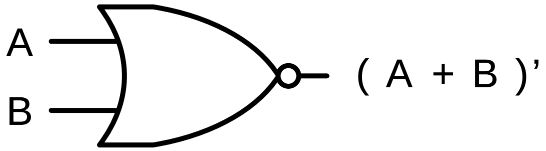
AND



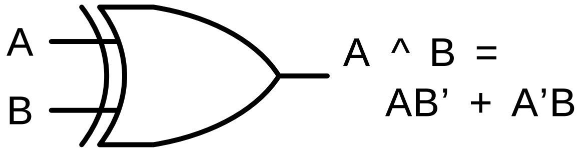
NAND



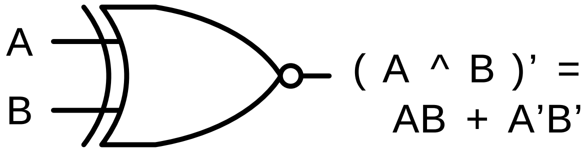
OR



NOR



XOR



XNOR

Truth tables

We describe Boolean functions with truth tables.

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

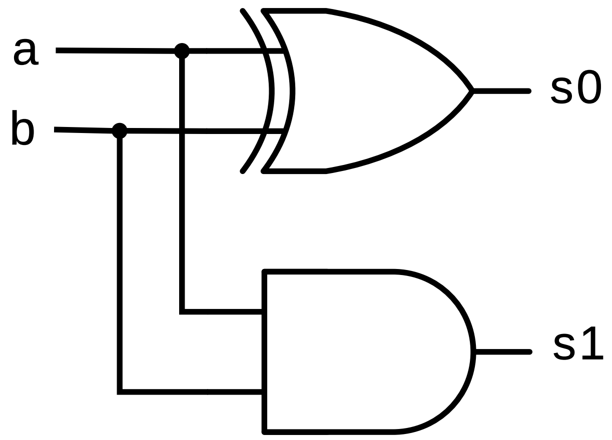
a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

a	NOT a
0	1
1	0

a	0	0	1	1
+b	+0	+1	+0	+1
s1 s0	0 0	0 1	0 1	1 0

a	b	s1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Addition of one-bit binary numbers.

Truth tables

Deriving Boolean equations from truth tables:

a	b	s1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

OR together *product* terms for each truth table row where the function is 1.

If input variable is 0, it appears in complemented form; if 1, it appears uncomplemented.

Truth tables

Deriving Boolean equations from truth tables:

a	b	s1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

s0 = a ^ b

s1 = a b

Example: a full adder

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Sum =

Cout =

Example: a full adder

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Sum = A’ B’ Cin + A’ B Cin’ + A B’ Cin’ + A B Cin

Cout = A’ B Cin + A B’ Cin + A B Cin’ + A B Cin