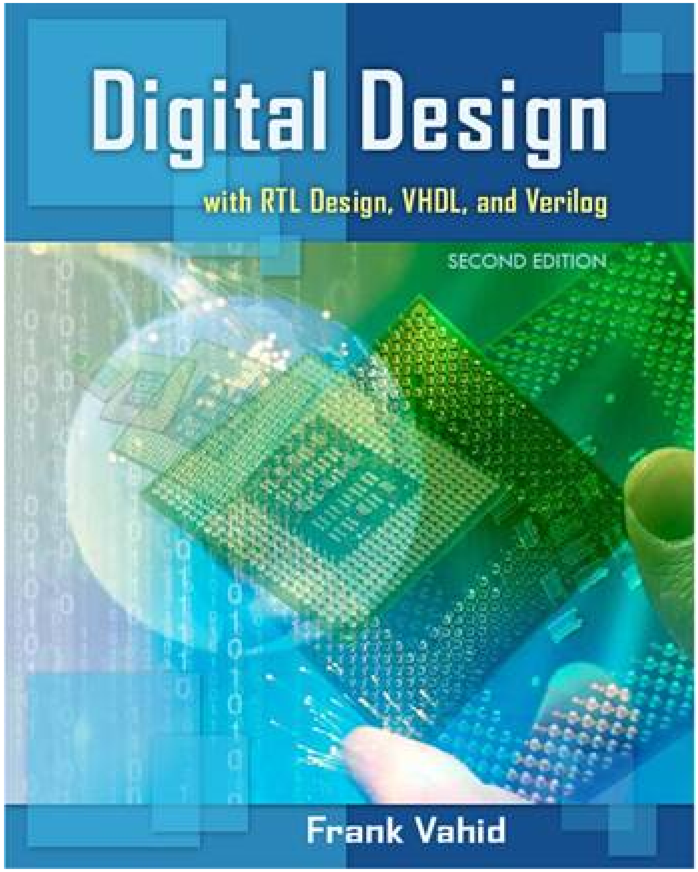


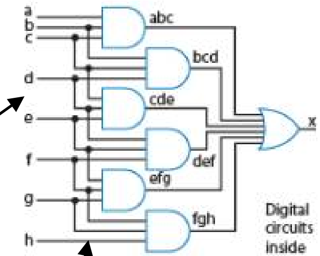
Digital Design

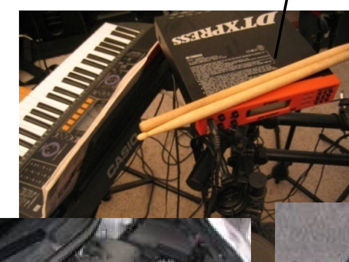
Chapter 1: Introduction

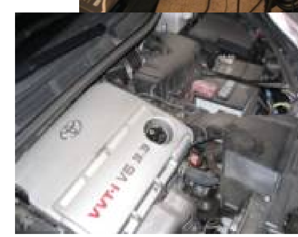
Slides to accompany the textbook *Digital Design, with RTL Design, VHDL, and Verilog*, 2nd Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2010.
http://www.ddvahid.com

Why Study Digital Design?

- Look "under the hood" of computers
  - Solid understanding --> confidence, insight, even better programmer when aware of hardware resource issues
- Electronic devices becoming digital
  - Enabled by shrinking and more capable chips
  - Enables:
    - Better devices: Sound recorders, cameras, cars, cell phones, medical devices,...
    - New devices: Video games, PDAs, ...
  - Known as "embedded systems"
    - Thousands of new devices every year
    - Designers needed: Potential career direction





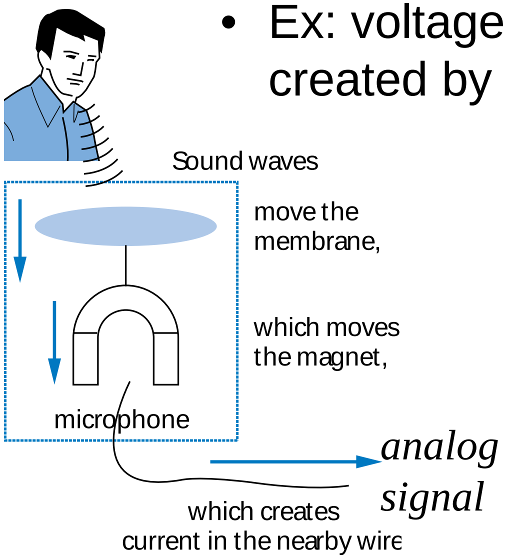


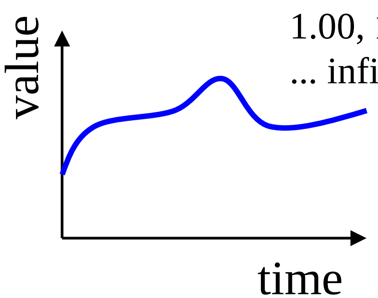
1995	1997	1999	2001	2003	2005	2007	
Portable music players	Satellites	Cell phones	DVD players	Cameras	Video recorders	TVs	Musical instruments ???

- Years shown above indicate when digital version began to *dominate*
  - (*Not* the first year that a digital version appeared)

What Does “Digital” Mean?

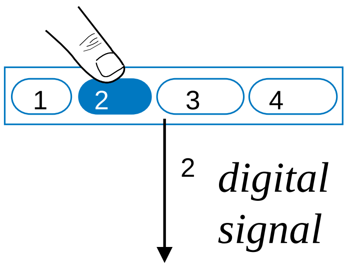
- Analog signal
  - Infinite possible values
    - Ex: voltage on a wire created by microphone

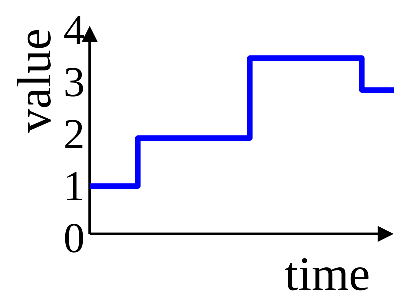




Possible values:
1.00, 1.01, 2.0000009, ... infinite possibilities

- Digital signal
  - Finite possible values
    - Ex: button pressed on a keypad

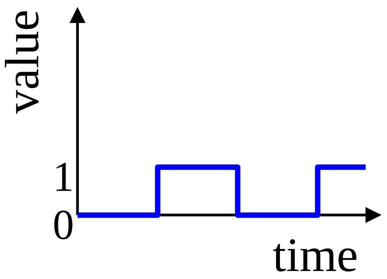




Possible values:
0, 1, 2, 3, or 4.
That’s it.

Digital Signals with Only Two Values: Binary

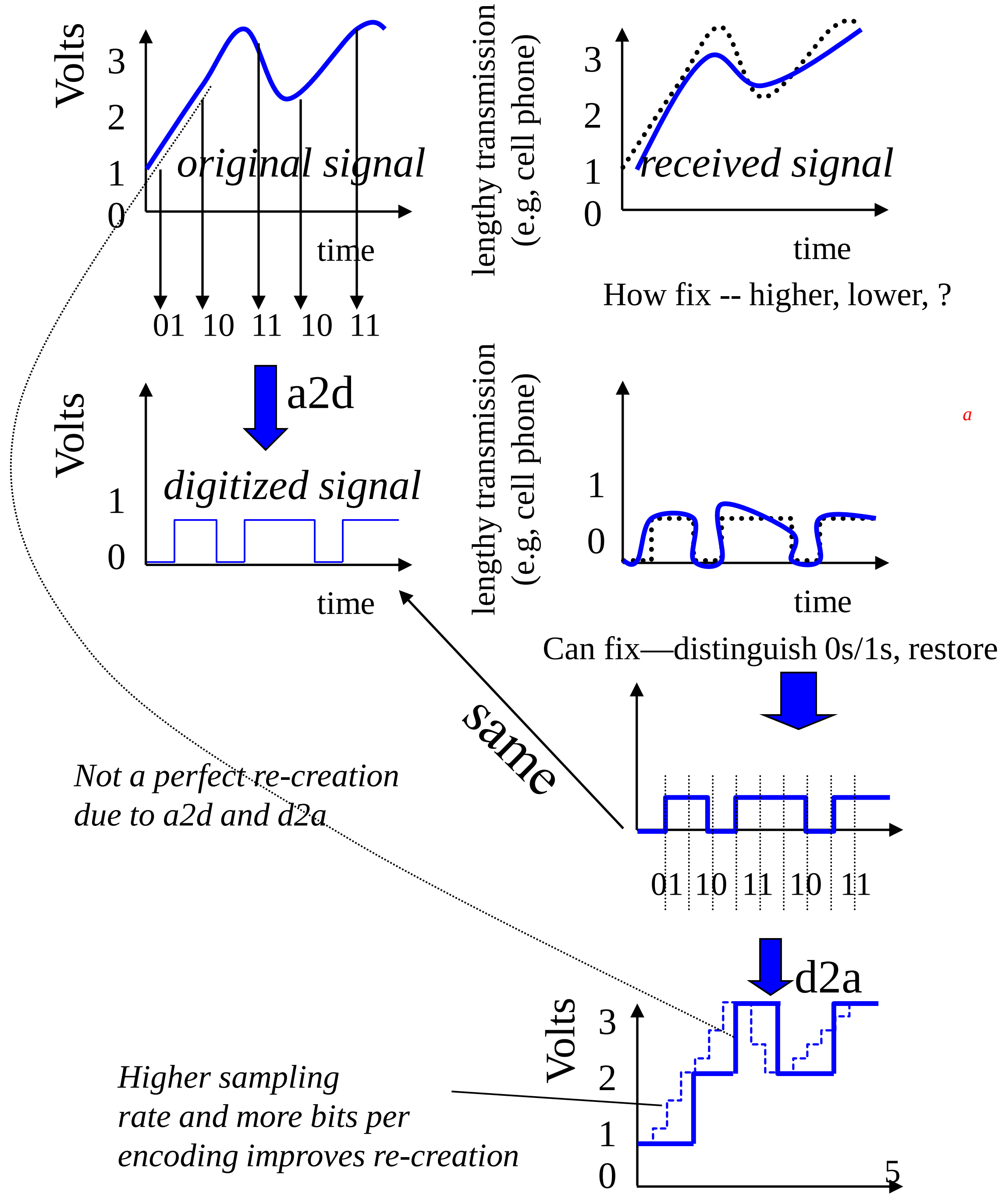
- ***Binary*** digital signal -- only *two* possible values
  - Typically represented as **0** and **1**
  - One *b*inary dig*it* is a ***bit***
  - We'll only consider *binary* digital signals
  - Binary is popular because
    - Transistors, the basic digital electric component, operate using *two* voltages (more in Chpt. 2)
    - Storing/transmitting one of *two* values is easier than three or more (e.g., loud beep or quiet beep, reflection or no reflection)



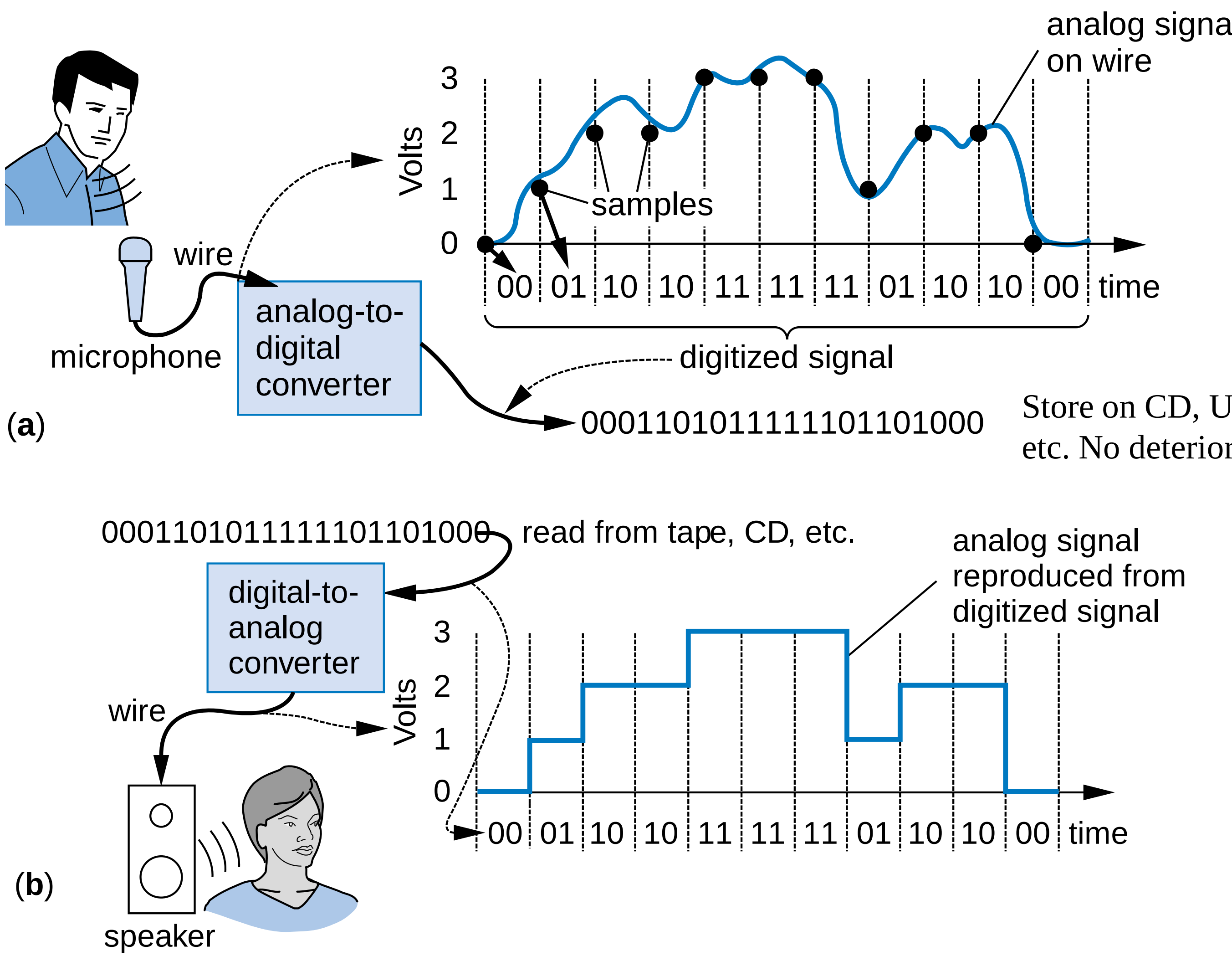
Example of Digitization Benefit

- Analog signal (e.g., audio, video) may lose quality
  - Voltage levels not saved/copied/transmitted perfectly
- Digitized version enables near-perfect save/cpy/tran.
  - “Sample” voltage at particular rate, save sample using bit encoding
  - Voltage levels still not kept perfectly
  - But we can distinguish 0s from 1s

Let bit encoding be:
  1 V: “01”
  2 V: “10”
  3 V: “11”



Digitization Benefit: Can Store on Digital Media



Store on CD, USB drive, etc. No deterioration.

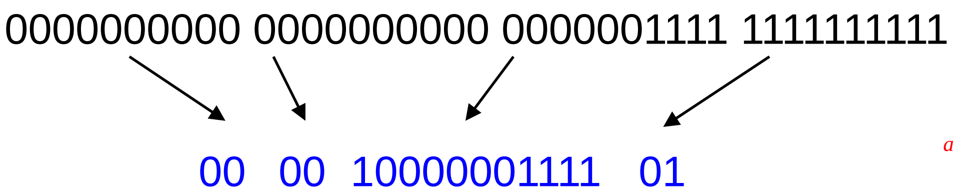
a

Digitized Audio: Compression Benefit

- Digitized audio can be compressed
  - e.g., MP3s
  - A CD can hold about 20 songs uncompressed, but about 200 compressed
- Compression also done on digitized pictures (jpeg), movies (mpeg), and more
- Digitization has many other benefits too

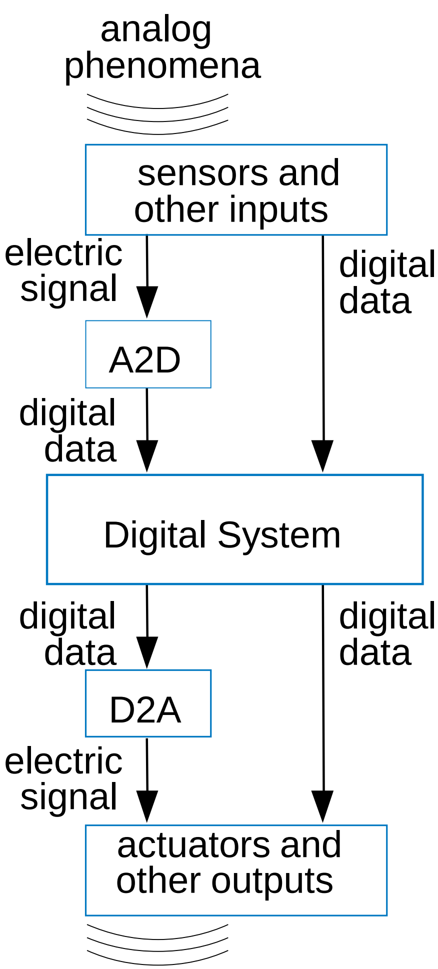
Example compression scheme:

00 means 0000000000
01 means 1111111111
1X means X

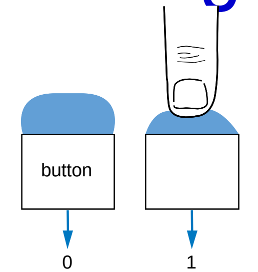


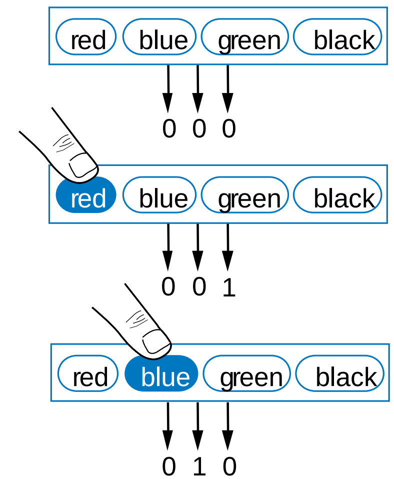
a

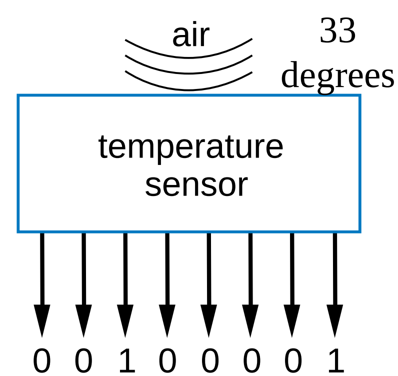
How Do We Encode Data as Binary for Our Digital System?



- Some inputs inherently binary
  - Button: not pressed (0), pressed (1)
- Some inputs inherently digital
  - Just need encoding in binary
  - e.g., multi-button input: encode red=001, blue=010, ...
- Some inputs analog
  - Need analog-to-digital conversion
  - As done in earlier slide -- sample and encode with bits







How to Encode Text: ASCII, Unicode

- ASCII: 7- (or 8-) bit encoding of each letter, number, or symbol
- Unicode: Increasingly popular 16-bit encoding
  - Encodes characters from various world languages

Sample ASCII encodings

Encoding	Symbol
010 0000	<space>
010 0001	!
010 0010	"
010 0011	#
010 0100	$
010 0101	%
010 0110	&
010 0111	'
010 1000	(
010 1001	)
010 1010	*
010 1011	+
010 1100	,
010 1101	-
010 1110	.
010 1111	/

Encoding	Symbol
100 0001	A
100 0010	B
100 0011	C
100 0100	D
100 0101	E
100 0110	F
100 0111	G
100 1000	H
100 1001	I
100 1010	J
100 1011	K
100 1100	L
100 1101	M

Encoding	Symbol
100 1110	N
100 1111	O
101 0000	P
101 0001	Q
101 0010	R
101 0011	S
101 0100	T
101 0101	U
101 0110	V
101 0111	W
101 1000	X
101 1001	Y
101 1010	Z

Encoding	Symbol
110 0001	a
110 0010	b
...	
111 1001	y
111 1010	z
011 0000	0
011 0001	1
011 0010	2
011 0011	3
011 0100	4
011 0101	5
011 0110	6
011 0111	7
011 1000	8
011 1001	9

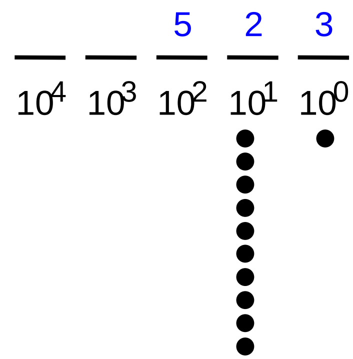
Question:
What does this ASCII bit sequence represent?

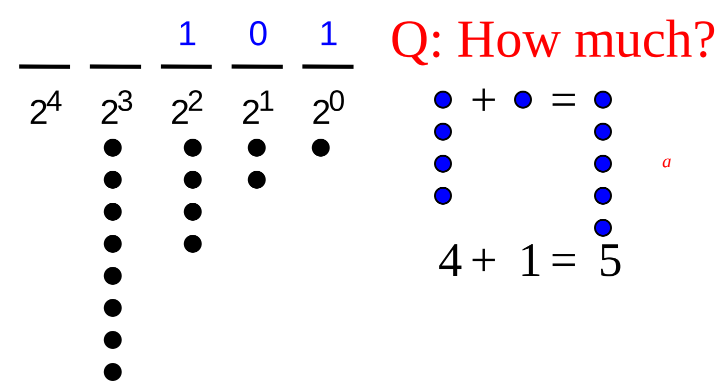
1010010 1000101 1010011 1010100

R E S T

How to Encode Numbers: Binary Numbers

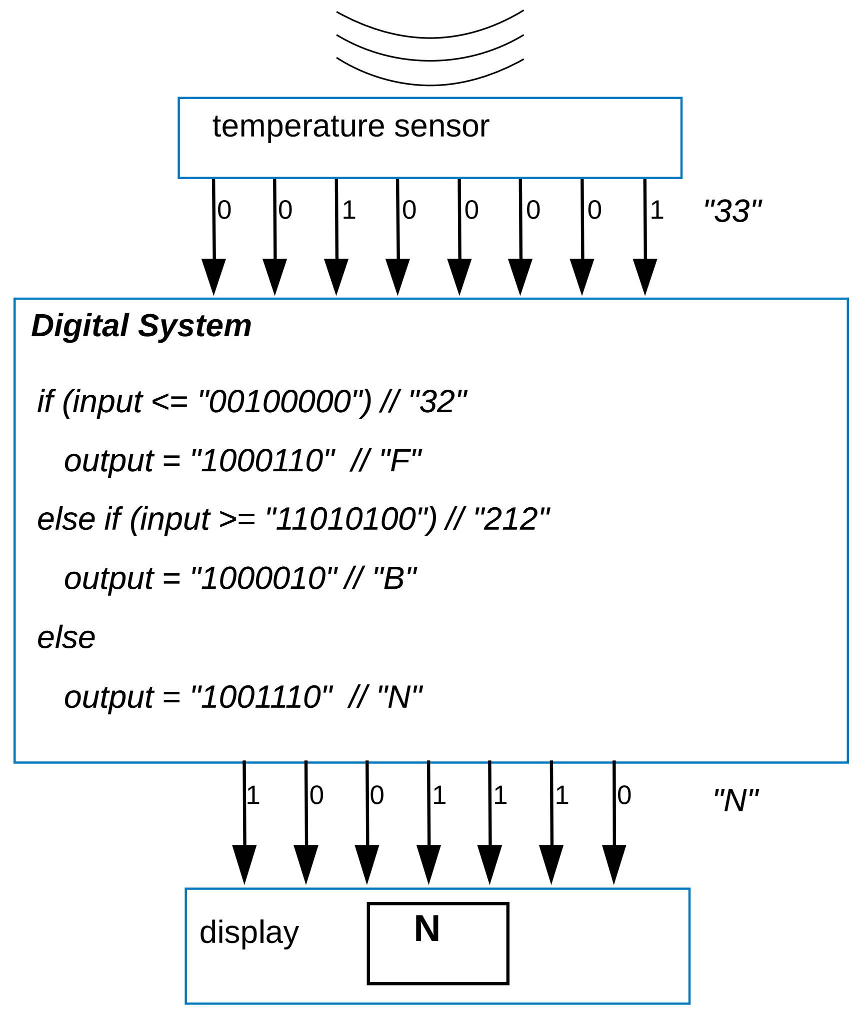
- Each position represents a quantity; symbol in position means how many of that quantity
  - Base ten (***decimal***)
    - Ten symbols: 0, 1, 2, ..., 8, and 9
    - More than 9 -- next position
      - So each position power of 10
    - Nothing special about base 10 -- used because we have 10 fingers
  - Base two (***binary***)
    - Two symbols: 0 and 1
    - More than 1 -- next position
      - So each position power of 2





Using Digital Data in a Digital System

- A temperature sensor outputs temperature in binary
- The system reads the temperature, outputs ASCII code:
  - "F" for freezing (0-32)
  - "B" for boiling (212 or more)
  - "N" for normal
- A display converts its ASCII input to the corresponding letter



Converting from Binary to Decimal

- Just add weights
  - 1_2 is just $1*2^0$, or 1_{10}.
  - 110_2 is $1*2^2 + 1*2^1 + 0*2^0$, or 6_{10}. We might think of this using base ten weights: 1*4 + 1*2 + 0*1, or 6.
  - 10000_2 is 1*16 + 0*8 + 0*4 + 0*2 + 0*1, or 16_{10}.
  - 10000111_2 is 1*128 + 1*4 + 1*2 + 1*1 = 135_{10}. Notice this time that we didn't bother to write the weights having a 0 bit.
  - 00110_2 is the same as 110_2 above — the leading 0's don't change the value.

Useful to know powers of 2:

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
512	256	128	64	32	16	8	4	2	1

Practice counting up by powers of 2: 512 256 128 64 32 16 8 4 2 1

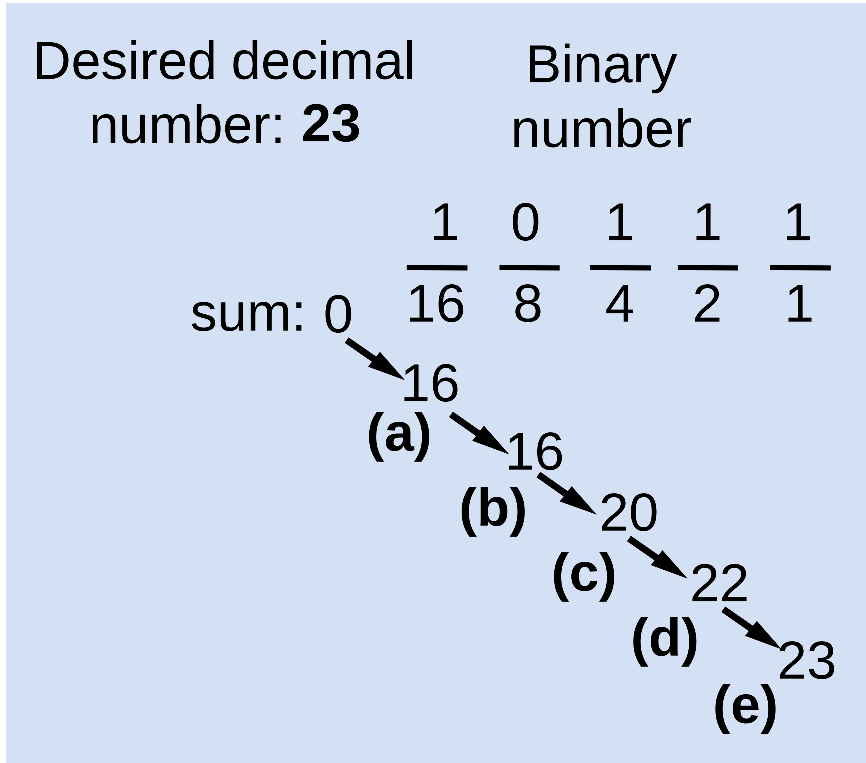
Converting from Decimal to Binary

- Put 1 in leftmost place without sum exceeding number
- Track sum

	Desired decimal number: **12**	Current sum	Binary number: 16	8	4	2	1
(a)	16 > 12, too big; Put 0 in 16’s place	0	**0** (***16***)	_	_	_	_
(b)	8 <= 12, so put 1 in 8’s place, current sum is 8	8	0	**1** (***8***)	_	_	_
(c)	8+4=12 <= 12, so put 1 in 4’s place, current sum is 12	12	0	1	**1** (***4***)	_	_
(d)	Reached desired 12, so put 0s in remaining places	*done*	0	1	1	0	0

Converting from Decimal to Binary

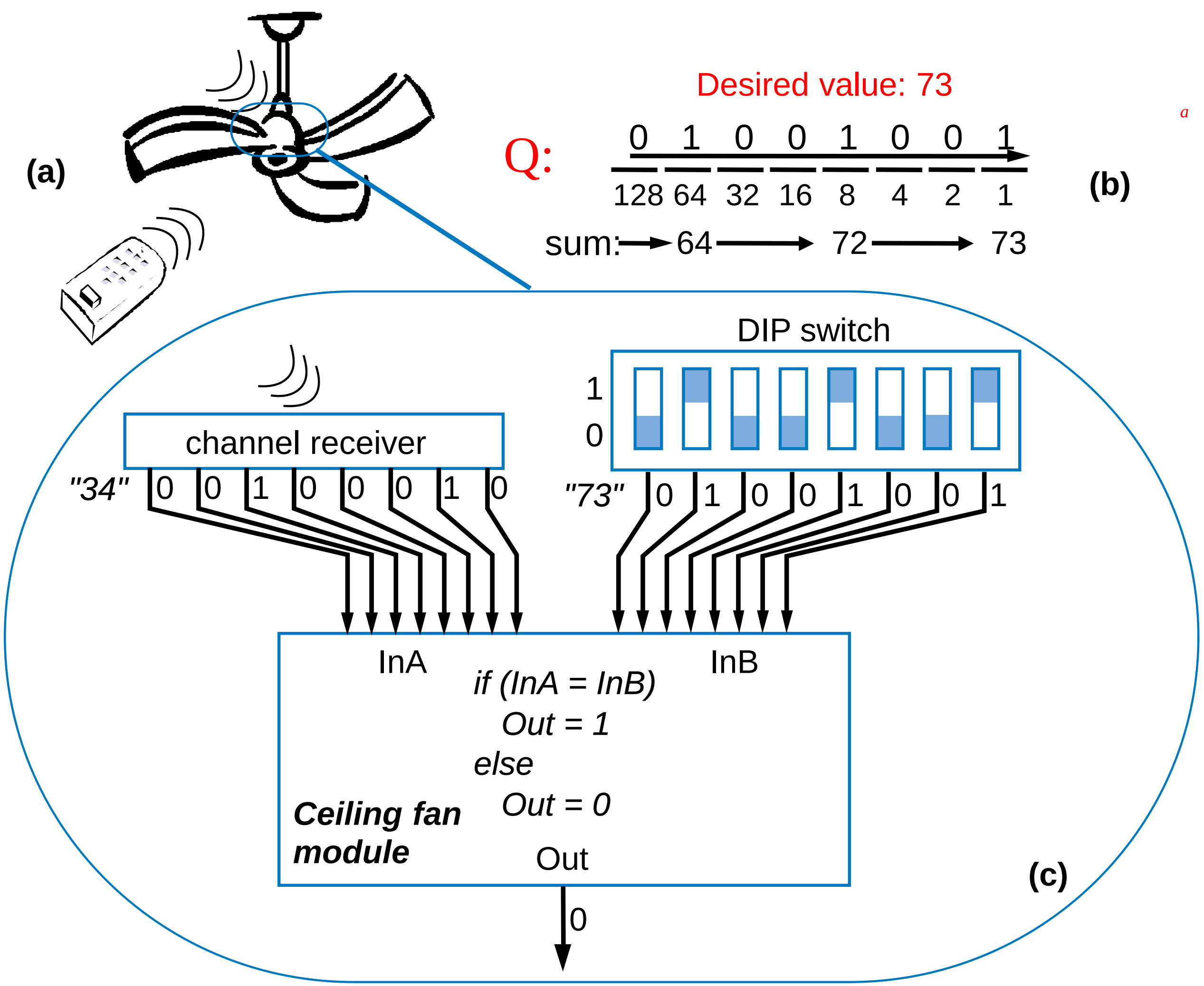
- Example using a more compact notation



a

Example: DIP-Switch Controlled Channel

- Ceiling fan receiver should be set in factory to respond to channel “73”
- Convert 73 to binary, set DIP switch accordingly



Base Sixteen: Another Base Used by Designers

8	A	F
16^4	16^3	16^2	16^1	16^0

8 → 1000, A → 1010, F → 1111

hex	binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

hex	binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

- Nice because each position represents four base-two positions
  - Compact way to write binary numbers
- Known as ***hexadecimal***, or just ***hex***

Q: Write 11110000 in hex

F 0

Q: Convert hex A01 to binary

1010 0000 0001

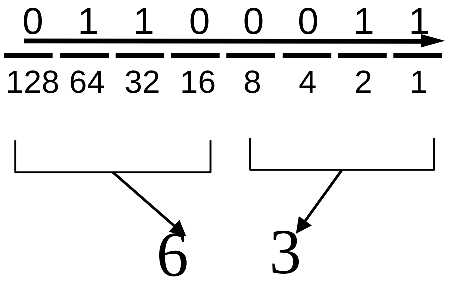
Decimal to Hex

- Easy method: convert to binary first, then binary to hex

Convert 99 base 10 to hex

First convert to binary:

Then binary to hex:

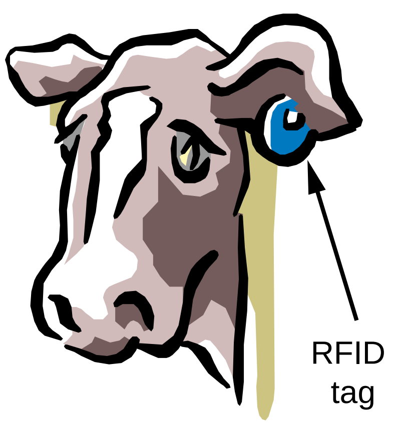


(Quick check: 6*16 + 3*1 = 96+3 = 99)

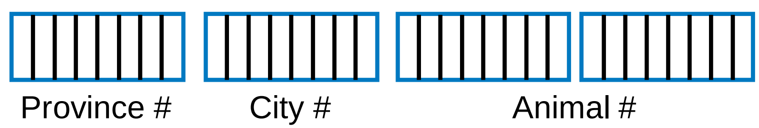
Hex Example: RFID Tag

- Batteryless tag powered by radio field
  - Transmits unique identification number
  - Example: 32 bit id
    - 8-bit province number, 8-bit country number, 16-bit animal number
    - Tag contents are in binary
    - But programmers use hex when writing/reading

(a)



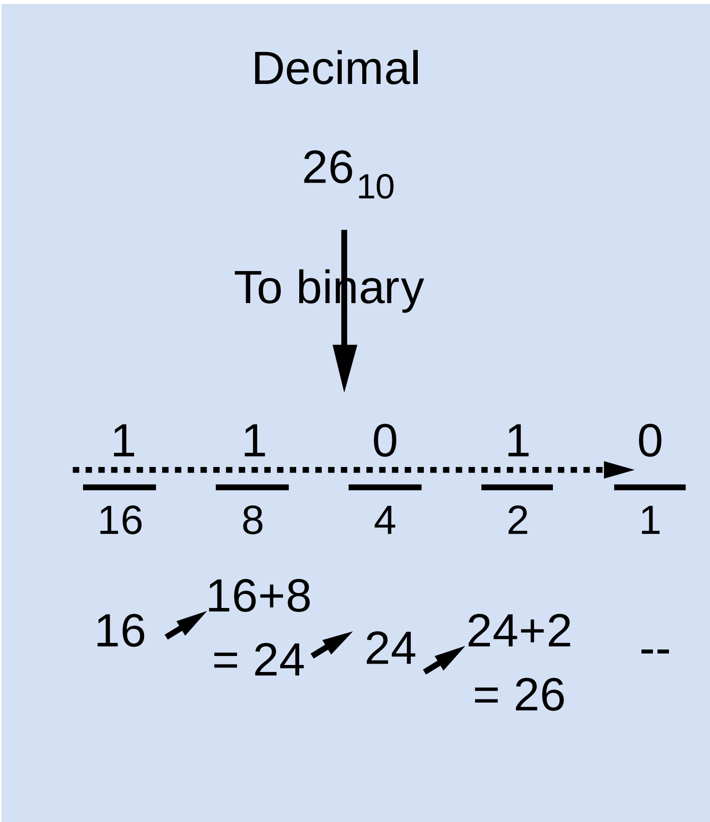
(b)

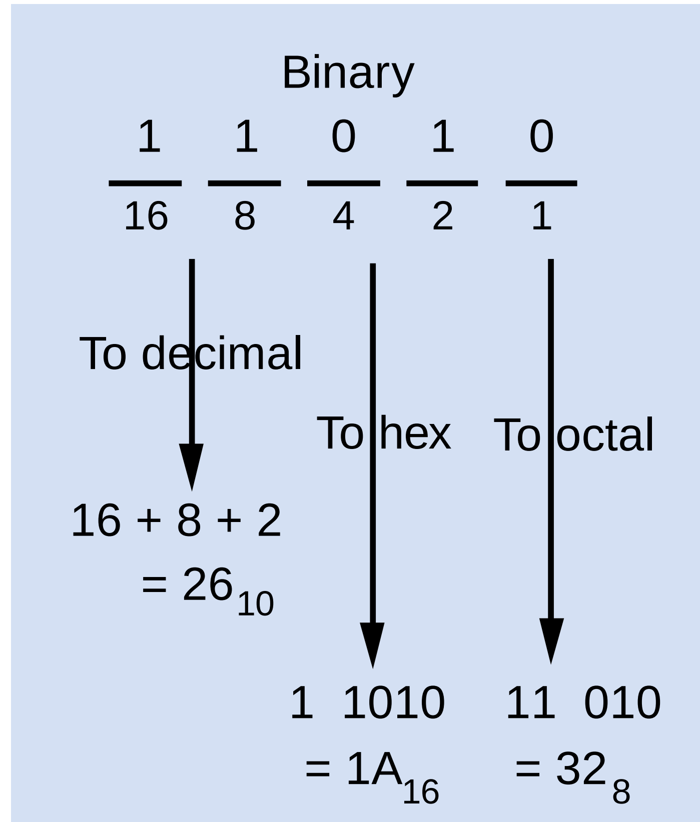


	Province	City	Animal
(c)	Province: 7	City: 160	Animal: 513
(d)	00000111	10100000	00000010 00000001
(e)	07	A0	02 01

(f) **Tag ID in hex: 07A00201**

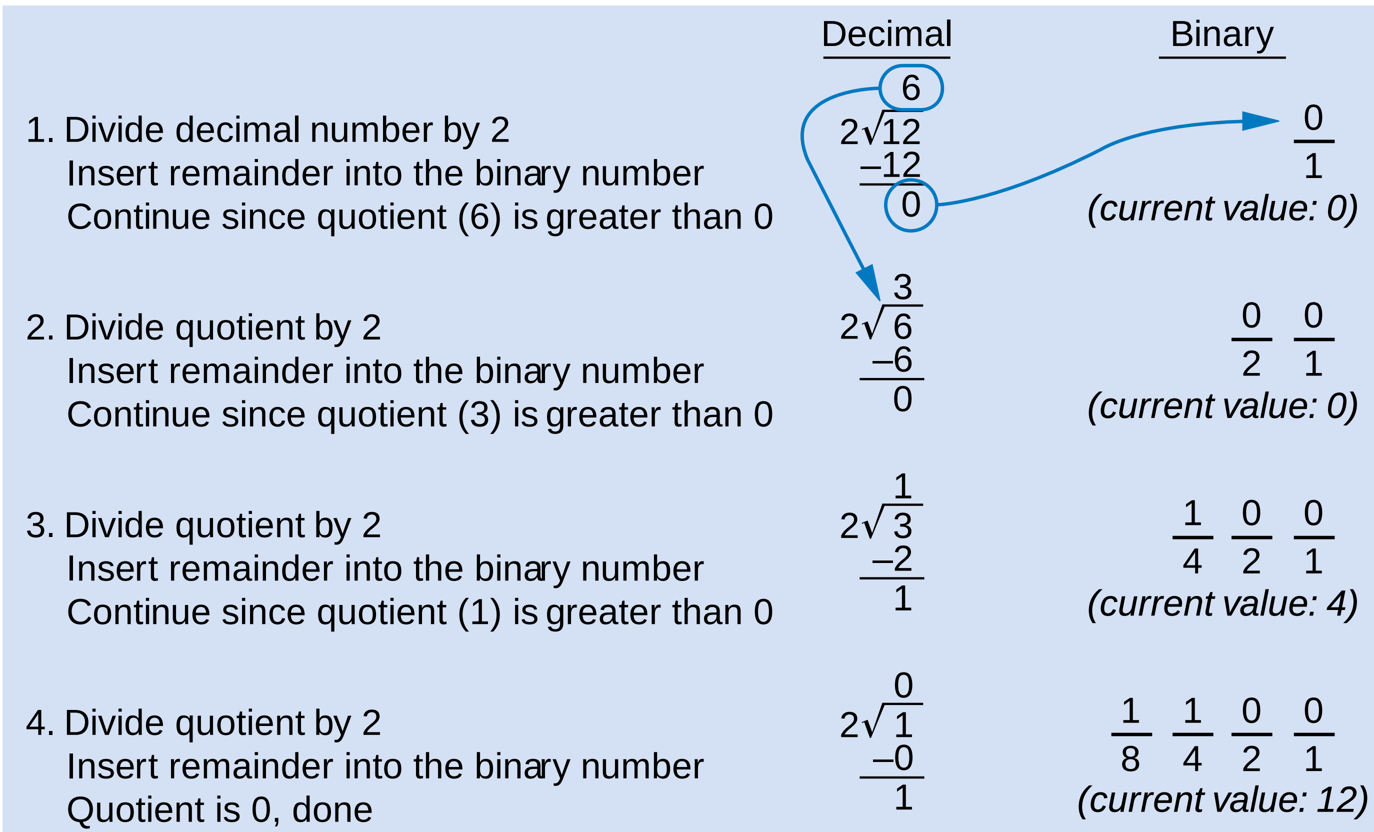
Converting To/From Binary by Hand: Summary





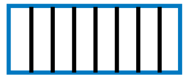
Divide-By-2 Method Common in Automatic Conversion

- Repeatedly divide decimal number by 2, place remainder in current binary digit (starting from 1s column)



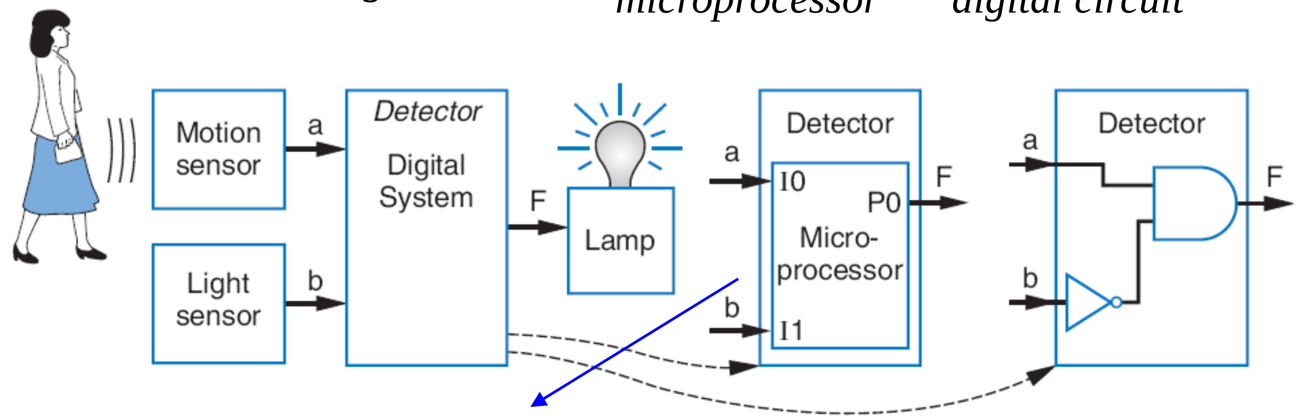
Note: Works for any base N—just divide by N instead

Bytes, Kilobytes, Megabytes, and More

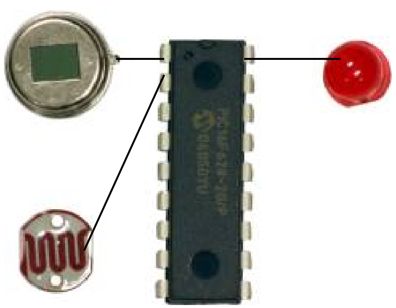
- Byte: 8 bits 
- Common metric prefixes:
  - kilo (thousand, or 10^3), mega (million, or 10^6), giga (billion, or 10^9), and tera (trillion, or 10^{12}), e.g., kilobyte, or KByte
- BUT, metric prefixes also commonly used inaccurately
  - 2^{16} = 65536 commonly written as “64 Kbyte”
  - Typical when describing memory sizes
- Also watch out for “KB” for kilobyte vs. “Kb” for kilobit

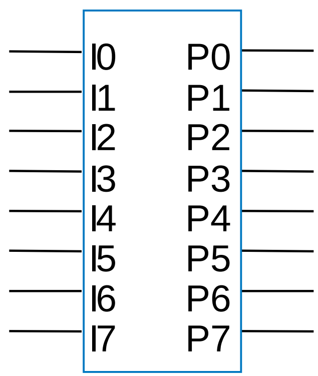
Implementing Digital Systems: Programming Microprocessors Vs. Designing Digital Circuits

Desired motion-at-night detector | *Programmed microprocessor* | *Custom designed digital circuit*

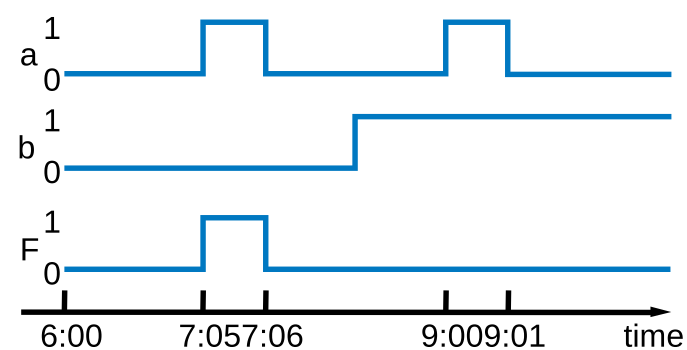


- Microprocessors a common choice to implement a digital system
  - Easy to program
  - Cheap (as low as $1)
  - Readily available



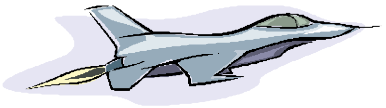



```
void main()
{
   while (1) {
      P0 = I0 && !I1;
      // F = a and !b,
   }
}
```



Digital Design: When Microprocessors Aren't Good Enough

- With microprocessors so easy, cheap, and available, why design a digital circuit?
  - Microprocessor may be too slow
  - Or too big, power hungry, or costly



Wing controller computation task:

- 50 ms on microprocessor
- 5 ms as custom digital circuit

If must execute 100 times per second:

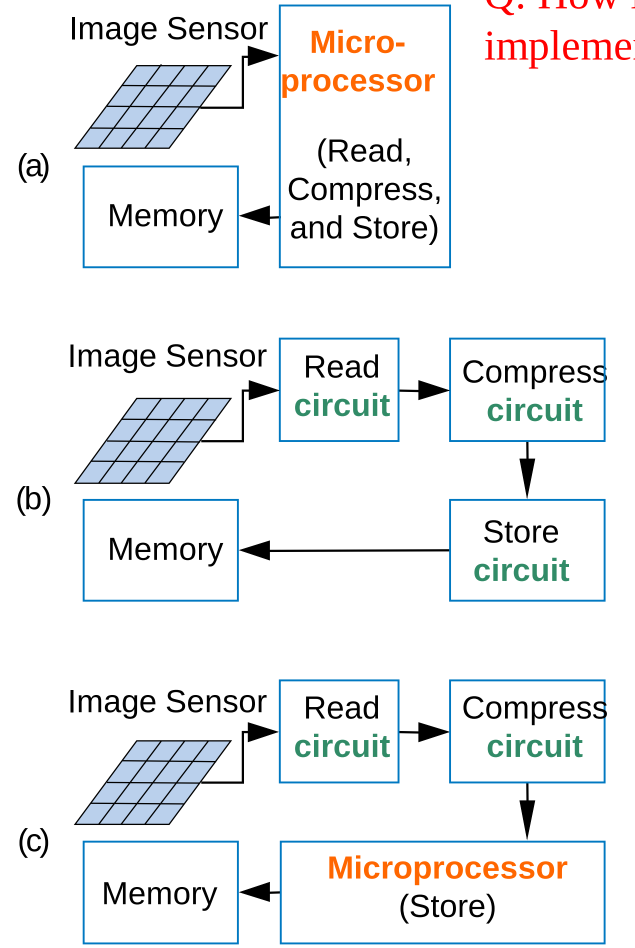
- 100 * 50 ms = 5000 ms = 5 seconds
- 100 * 5 ms = 500 ms = 0.5 seconds

Microprocessor too slow, circuit OK.

Digital Design: When Microprocessors Aren't Good Enough

- Commonly, designers partition a system among a microprocessor and custom digital circuits

Q: How long for each implementation option?



5+8+1
=14 sec

.1+.5+.8
=1.4 sec

.1+.5+1
=1.6 sec

Good compromise

Sample digital camera task execution times (in seconds) on a microprocessor versus a digital circuit:

Task	Microprocessor	Custom Digital Circuit
Read	5	0.1
Compress	8	0.5
Store	1	0.8

Chapter Summary

- Digital systems surround us
  - Inside computers
  - Inside many other electronic devices (embedded systems)
- Digital systems use 0s and 1s
  - Encoding analog signals to digital can provide many benefits
    - e.g., audio—higher-quality storage/transmission, compression, etc.
  - Encoding integers as 0s and 1s: Binary numbers
- Microprocessors (themselves digital) can implement many digital systems easily and inexpensively
  - But often not good enough—need custom digital circuits