

Digital Design

Chapter 1: Introduction

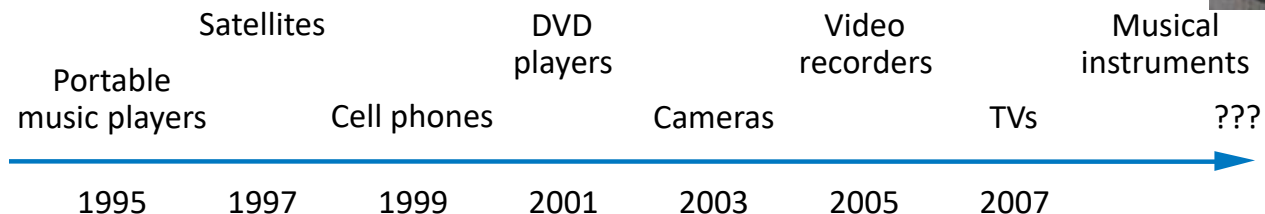
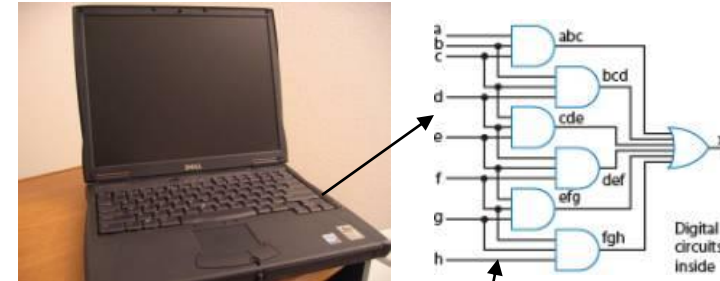
Slides to accompany the textbook *Digital Design, with RTL Design, VHDL, and Verilog*, 2nd Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2010.
<http://www.ddvahid.com>

Copyright © 2010 Frank Vahid

Instructors of courses requiring Vahid's *Digital Design* textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as unanimated pdf versions on publicly-accessible course websites.. PowerPoint source (or pdf with animations) may **not** be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.ddvahid.com> for information.

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



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

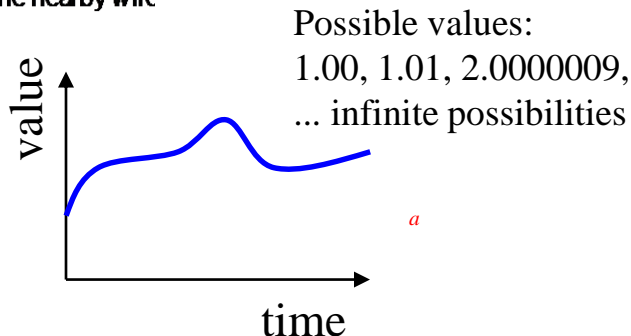
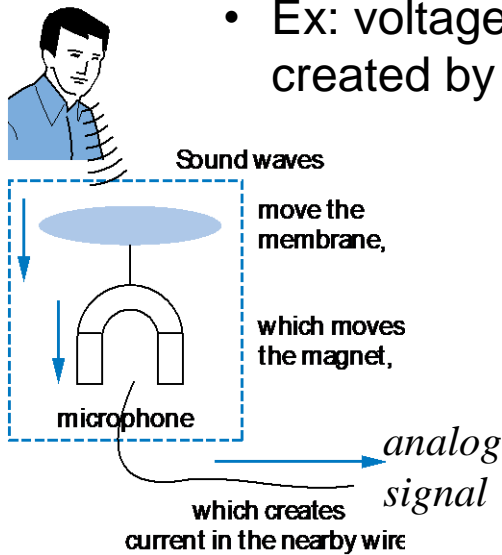


What Does “Digital” Mean?

- Analog signal

- Infinite possible values

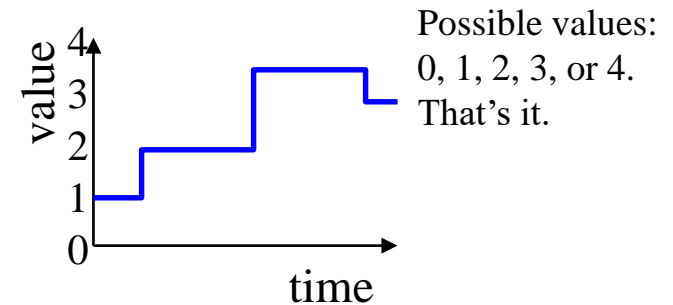
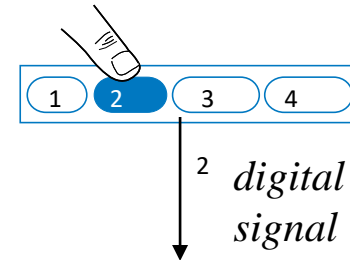
- Ex: voltage on a wire created by microphone



- Digital signal

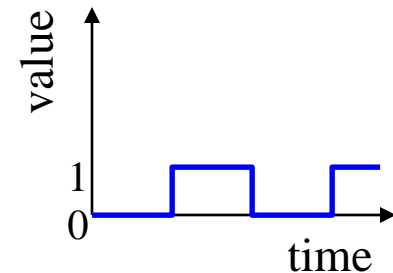
- Finite possible values

- Ex: button pressed on a keypad



Digital Signals with Only Two Values: Binary

- **Binary** digital signal -- only *two* possible values
 - Typically represented as **0** and **1**
 - One *binary digit* 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

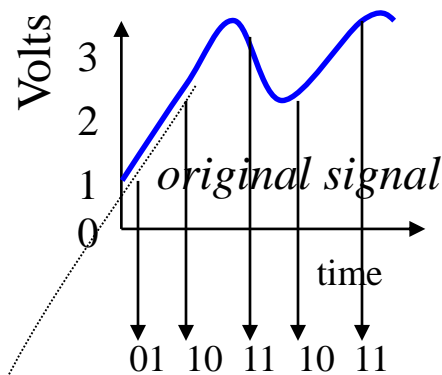
- Analog signal (e.g., audio, video) may lose quality
 - Voltage levels not saved/copied/transmitted perfectly
- Digitized version enables near-perfect save/cpy/tran.
 - “Sample” voltage at particular rate, save sample using bit encoding
 - Voltage levels still not kept perfectly
 - But we can distinguish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

Let bit encoding be:

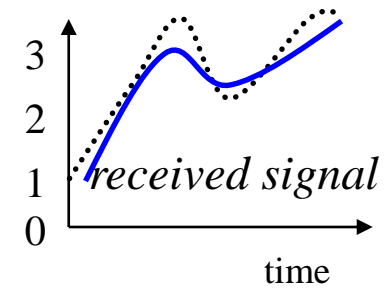
1 V: “01”

2 V: “10”

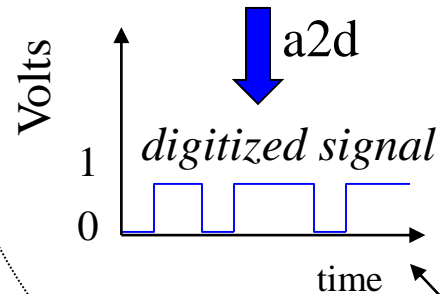
3 V: “11”



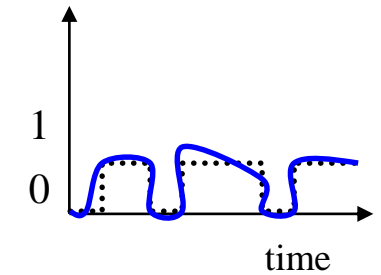
lengthy transmission
(e.g., cell phone)



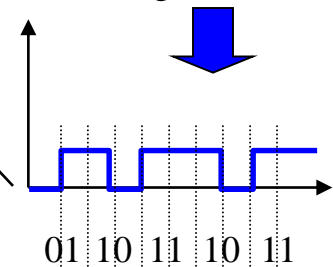
How fix -- higher, lower, ?



lengthy transmission
(e.g., cell phone)



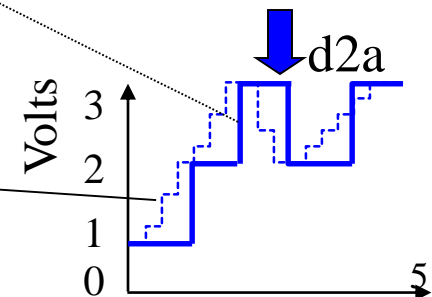
Can fix—distinguish 0s/1s, restore



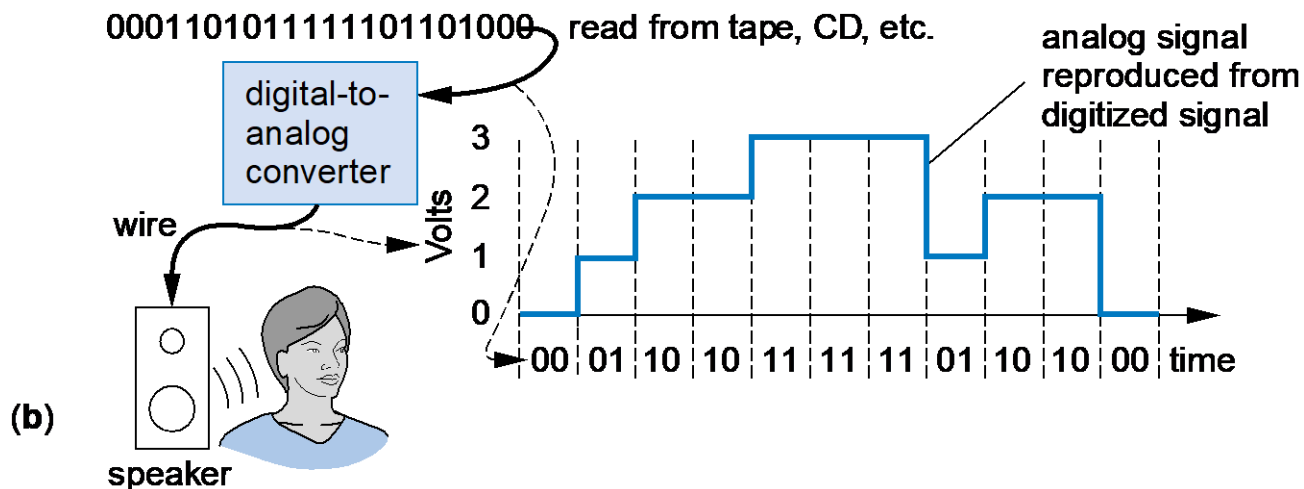
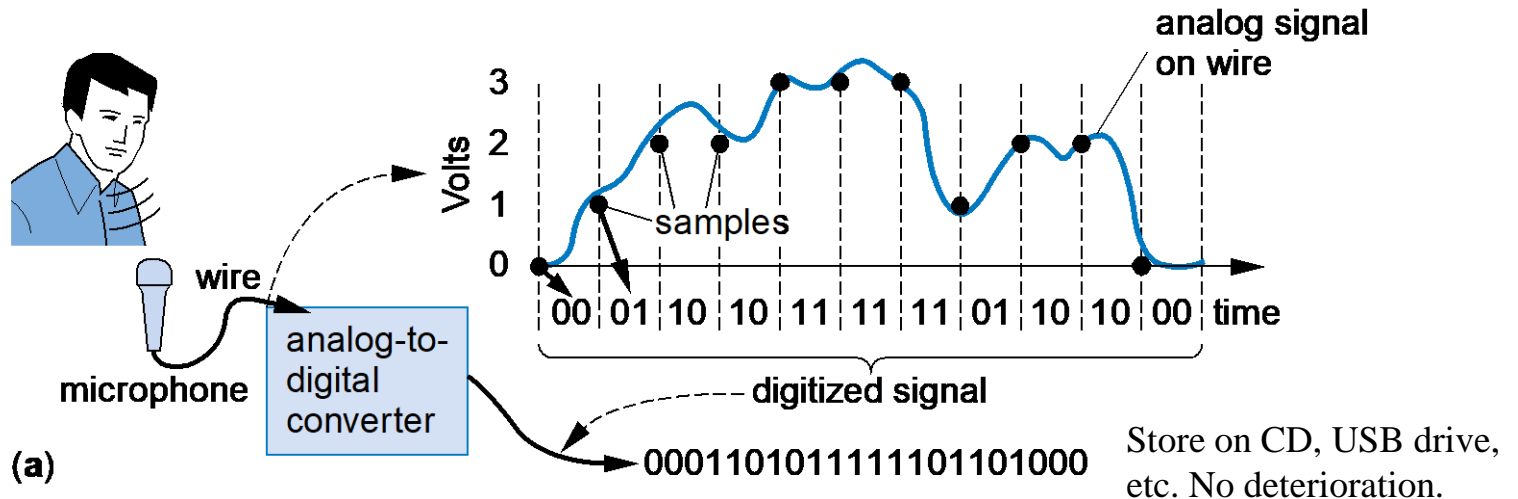
Not a perfect re-creation
due to a2d and d2a

same

Higher sampling
rate and more bits per
encoding improves re-creation



Digitization Benefit: Can Store on Digital Media



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:

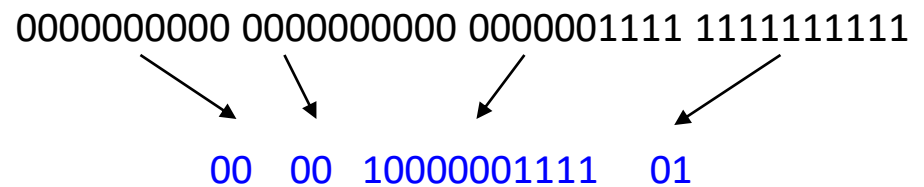
00 means 0000000000

01 means 1111111111

1X means X

0000000000 0000000000 0000001111 1111111111

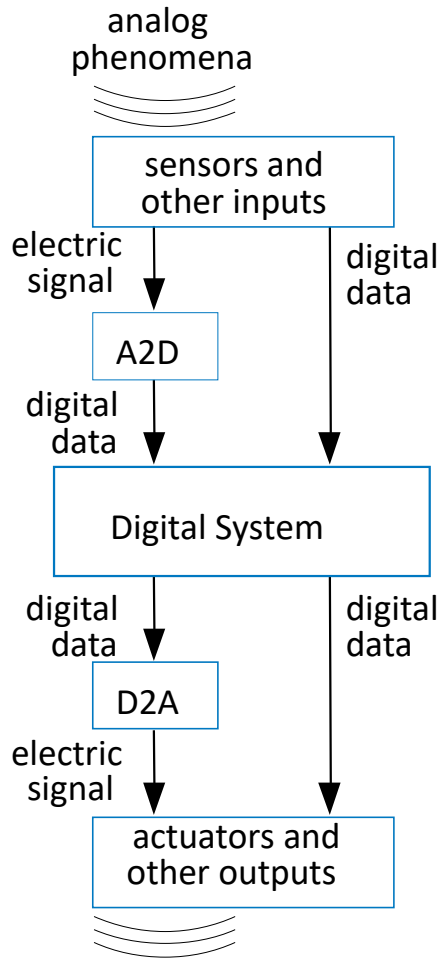
00 00 10000001111 01



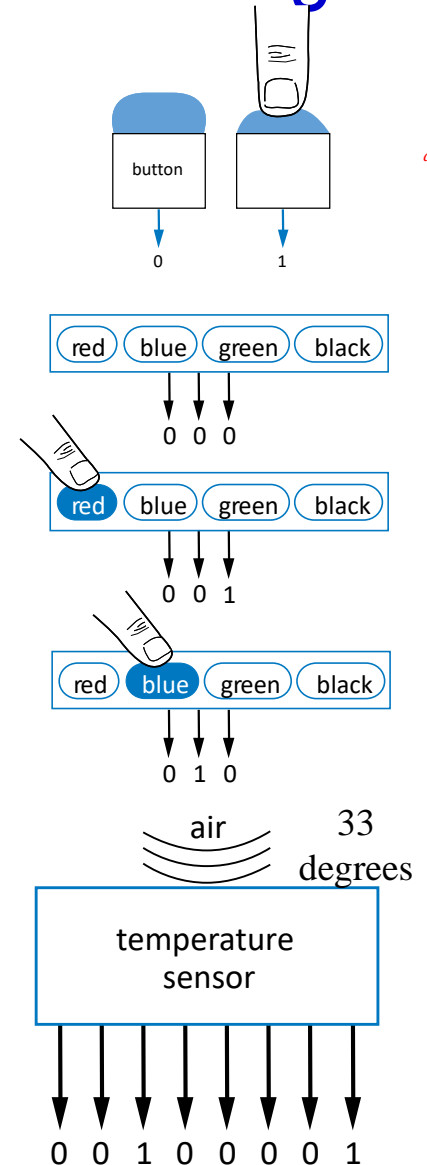
a



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

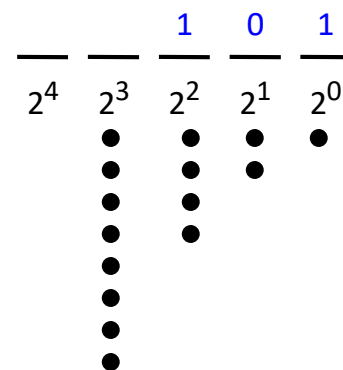
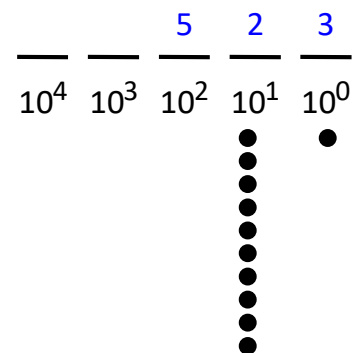
↓ ↓ ↓ ↓
R E S T

a



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



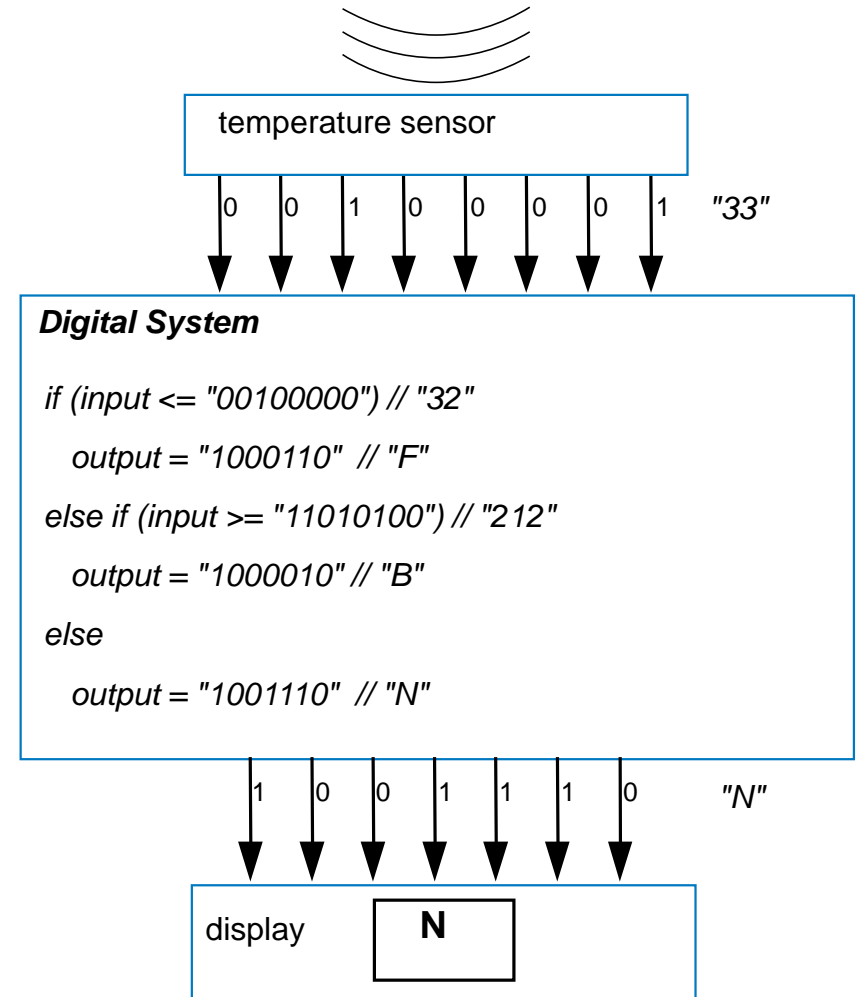
Q: How much?

● + ● = ●
 ● + ● = ●
 ● + ● = ●
 ● + ● = ●
 4 + 1 = 5



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 $1 \cdot 2^0$, or 1_{10} .
 - 110_2 is $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$, or 6_{10} . We might think of this using base ten weights: $1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$, or 6.
 - 10000_2 is $1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 0 \cdot 1$, or 16_{10} .
 - 10000111_2 is $1 \cdot 128 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 135_{10}$. 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
(a)	16 > 12, too big; Put 0 in 16's place	0	$\frac{0}{16}$ $\frac{\quad}{8}$ $\frac{\quad}{4}$ $\frac{\quad}{2}$ $\frac{\quad}{1}$
(b)	8 ≤ 12, so put 1 in 8's place, current sum is 8	8	$\frac{0}{16}$ $\frac{1}{8}$ $\frac{\quad}{4}$ $\frac{\quad}{2}$ $\frac{\quad}{1}$
(c)	8+4=12 ≤ 12, so put 1 in 4's place, current sum is 12	12	$\frac{0}{16}$ $\frac{1}{8}$ $\frac{1}{4}$ $\frac{\quad}{2}$ $\frac{\quad}{1}$
(d)	Reached desired 12, so put 0s in remaining places	done	$\frac{0}{16}$ $\frac{1}{8}$ $\frac{1}{4}$ $\frac{0}{2}$ $\frac{0}{1}$

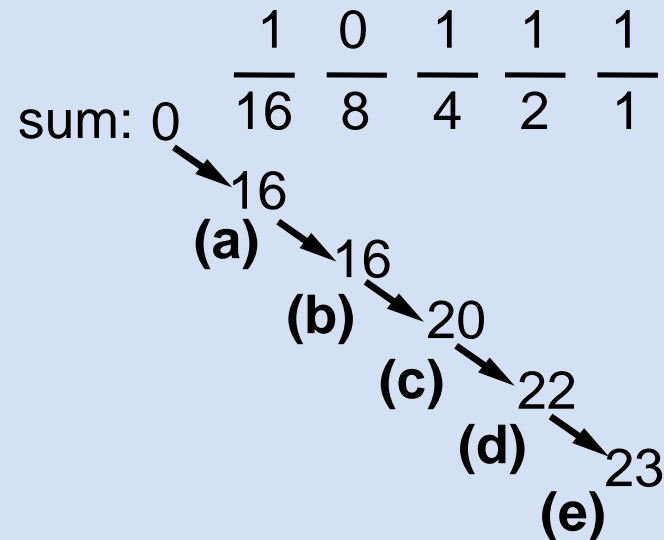


Converting from Decimal to Binary

- Example using a more compact notation

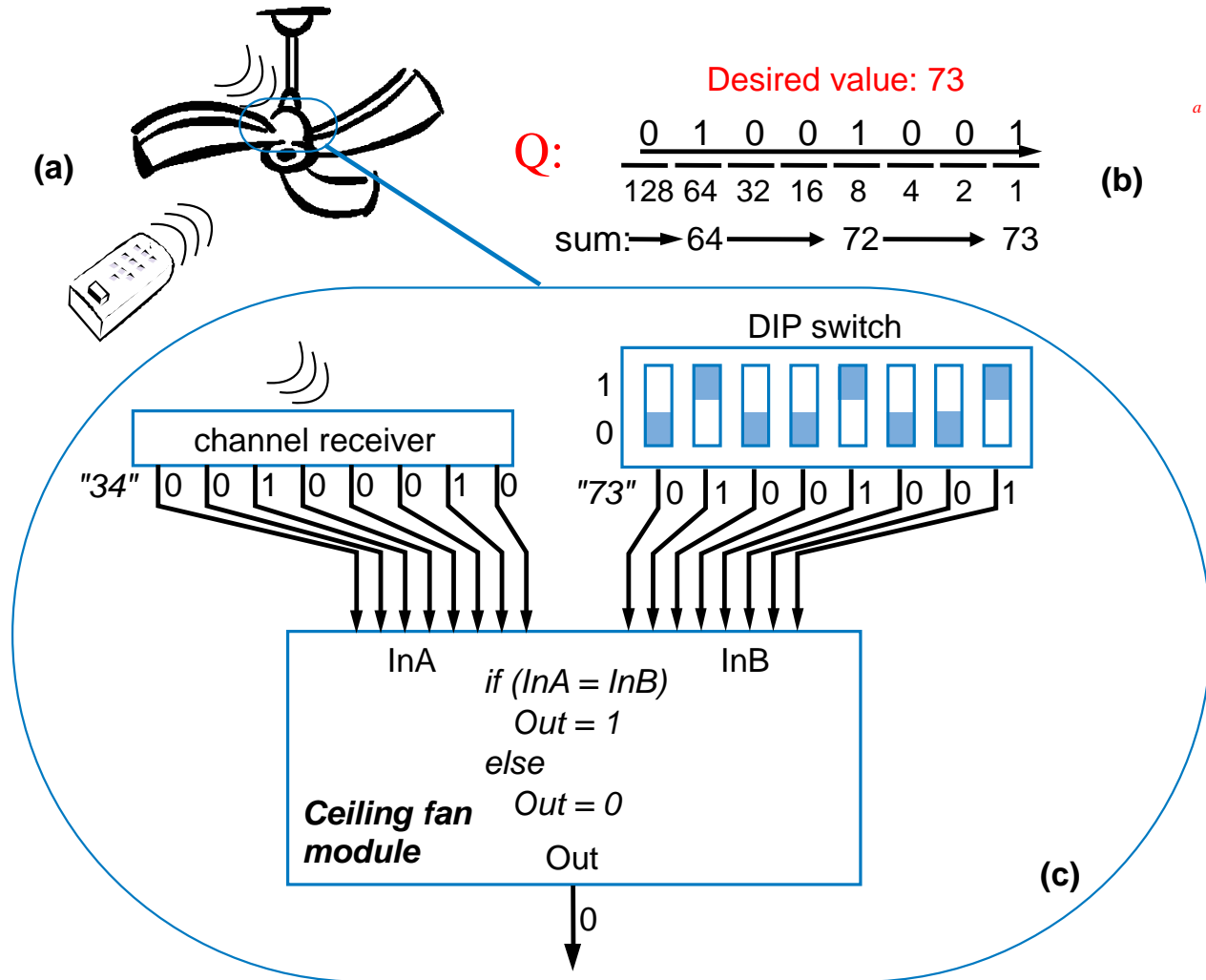
Desired decimal number: **23**

Binary number



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

$$\begin{array}{ccccc}
 \hline & & 8 & A & F \\
 \hline
 16^4 & 16^3 & 16^2 & 16^1 & 16^0 \\
 \\
 & & \downarrow 8 & \downarrow A & \downarrow F \\
 & & 1000 & 1010 & 1111
 \end{array}$$

hex	binary	hex	binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	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

Q: Convert hex A01 to binary



Decimal to Hex

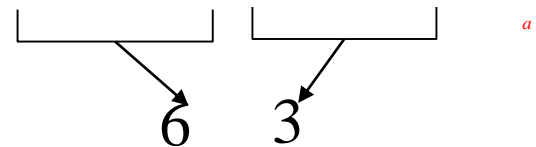
- Easy method: convert to binary first, then binary to hex

Convert 99 base 10 to hex

First convert to binary:

0	1	1	0	0	0	1	1	<i>a</i>
128	64	32	16	8	4	2	1	

Then binary to hex:

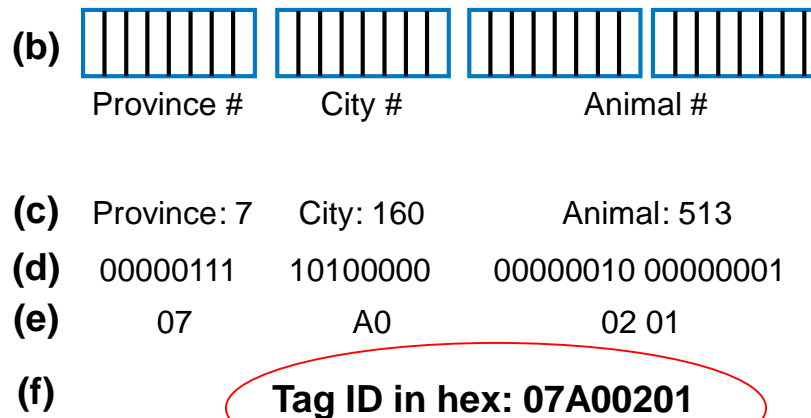
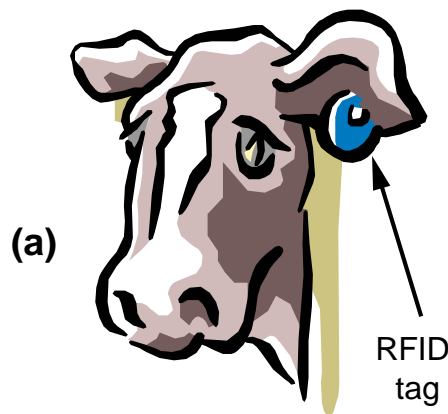


(Quick check: $6*16 + 3*1 = 96+3 = 99$) *a*

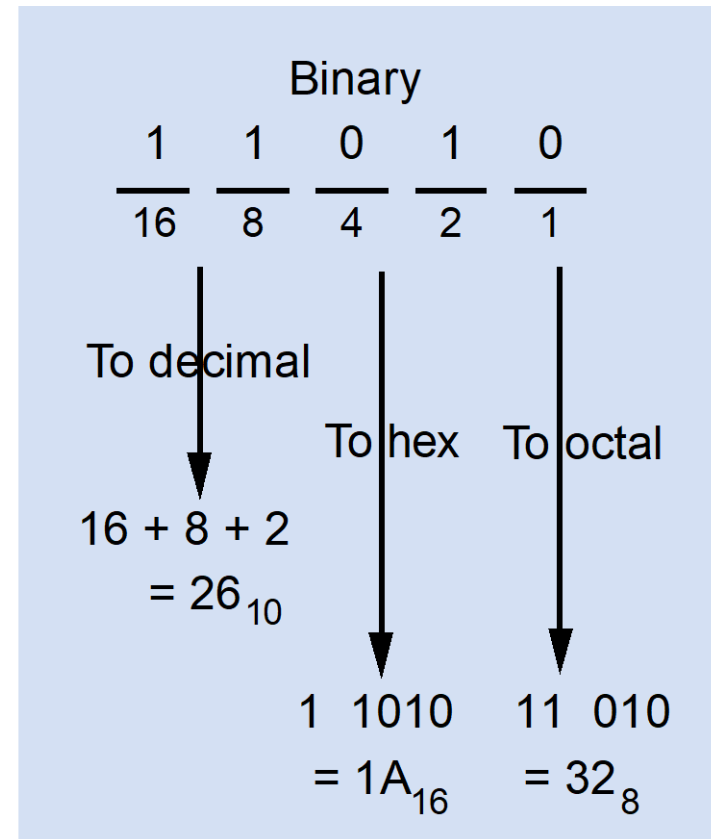
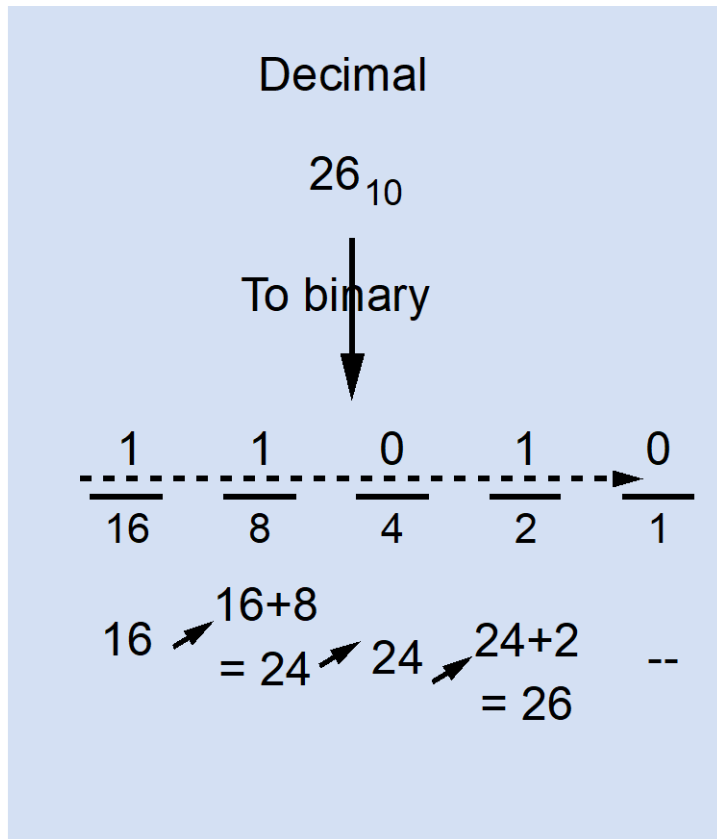


Hex Example: RFID Tag

- 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



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)

	Decimal	Binary
1. Divide decimal number by 2 Insert remainder into the binary number Continue since quotient (6) is greater than 0	$\begin{array}{r} 6 \\ 2\overline{)12} \\ \underline{-12} \\ 0 \end{array}$	$\begin{array}{r} 0 \\ 1 \end{array}$ (current value: 0)
2. Divide quotient by 2 Insert remainder into the binary number Continue since quotient (3) is greater than 0	$\begin{array}{r} 3 \\ 2\overline{)6} \\ \underline{-6} \\ 0 \end{array}$	$\begin{array}{r} 0 \quad 0 \\ 2 \quad 1 \end{array}$ (current value: 0)
3. Divide quotient by 2 Insert remainder into the binary number Continue since quotient (1) is greater than 0	$\begin{array}{r} 1 \\ 2\overline{)3} \\ \underline{-2} \\ 1 \end{array}$	$\begin{array}{r} 1 \quad 0 \quad 0 \\ 4 \quad 2 \quad 1 \end{array}$ (current value: 4)
4. Divide quotient by 2 Insert remainder into the binary number Quotient is 0, done	$\begin{array}{r} 0 \\ 2\overline{)1} \\ \underline{-0} \\ 1 \end{array}$	$\begin{array}{r} 1 \quad 1 \quad 0 \quad 0 \\ 8 \quad 4 \quad 2 \quad 1 \end{array}$ (current value: 12)

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



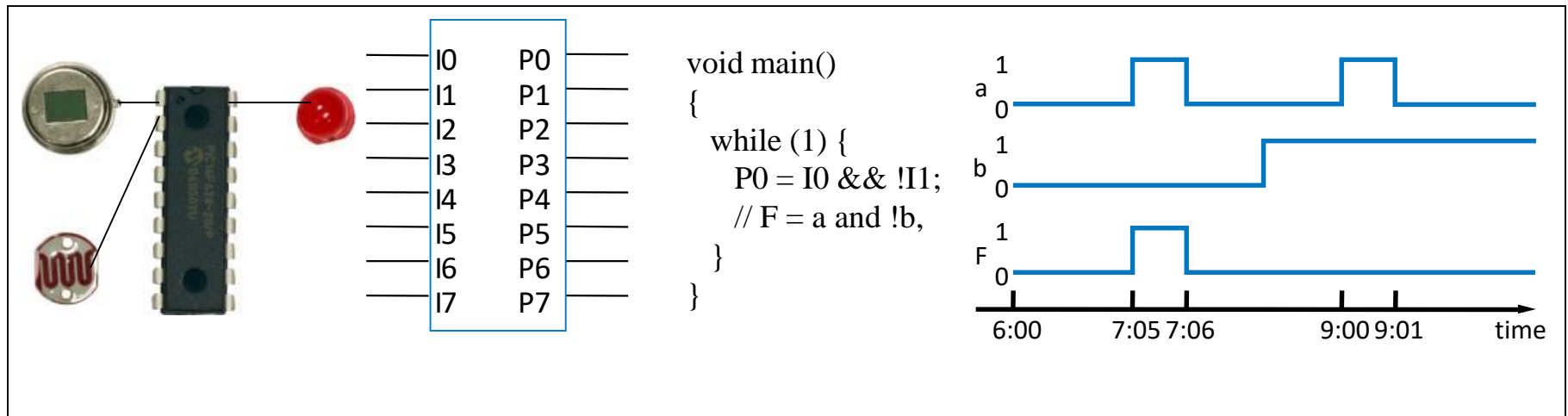
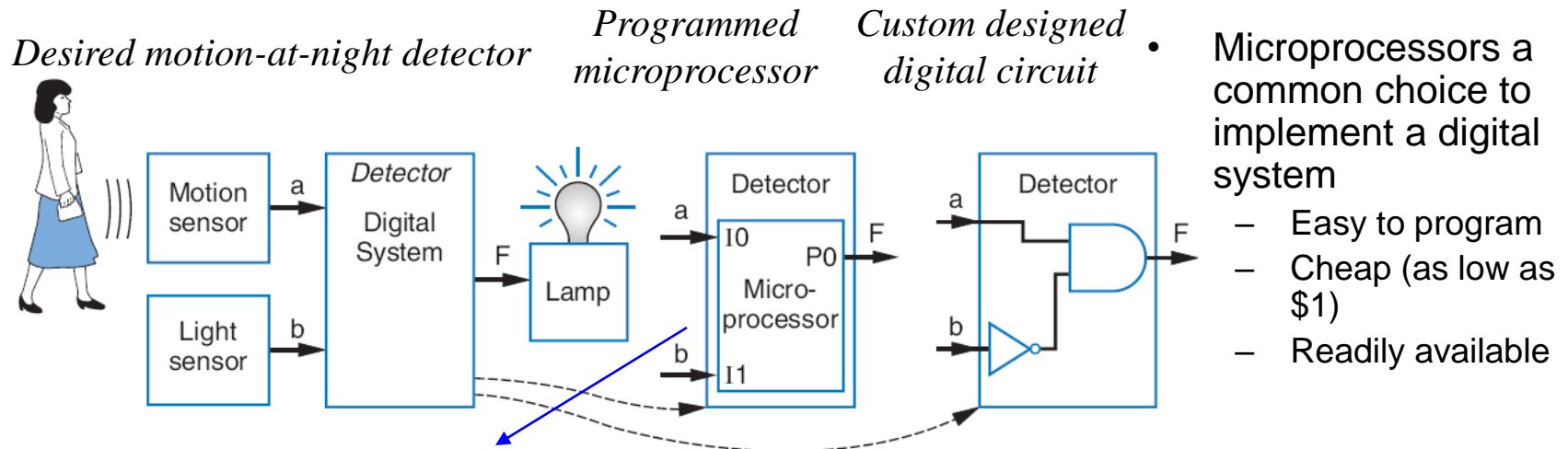
Bytes, Kilobytes, Megabytes, and More

- Byte: 8 bits A diagram showing a horizontal row of eight adjacent rectangular boxes, each representing a bit within a byte.
- 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



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:

- $100 * 50 \text{ ms} = 5000 \text{ ms} = 5 \text{ seconds}$
- $100 * 5 \text{ ms} = 500 \text{ ms} = 0.5 \text{ seconds}$

Microprocessor too slow, circuit OK.

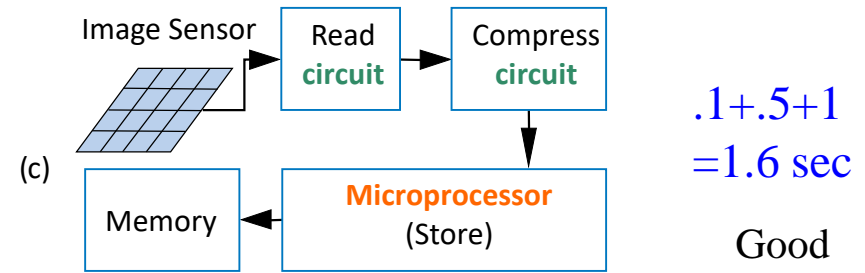
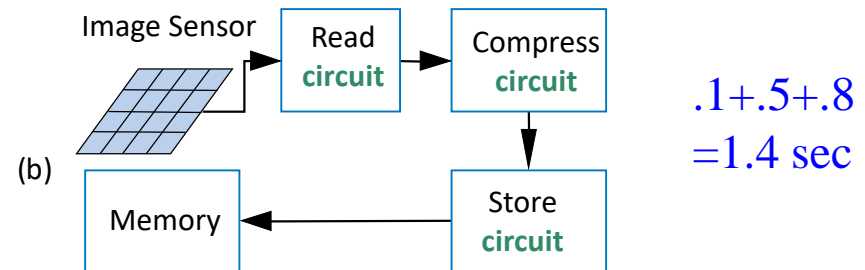
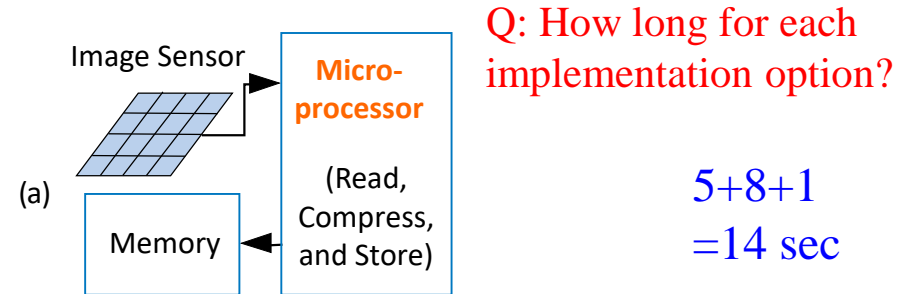


Digital Design: When Microprocessors Aren't Good Enough

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

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



Good compromise



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

