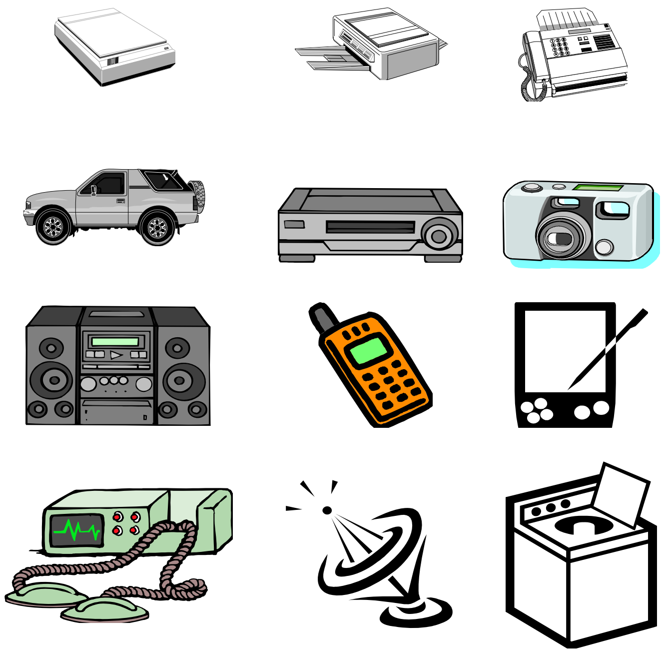
NXT HW
Sensors and Actuators

Brian Nielsen

bnielsen@cs.aau.dk

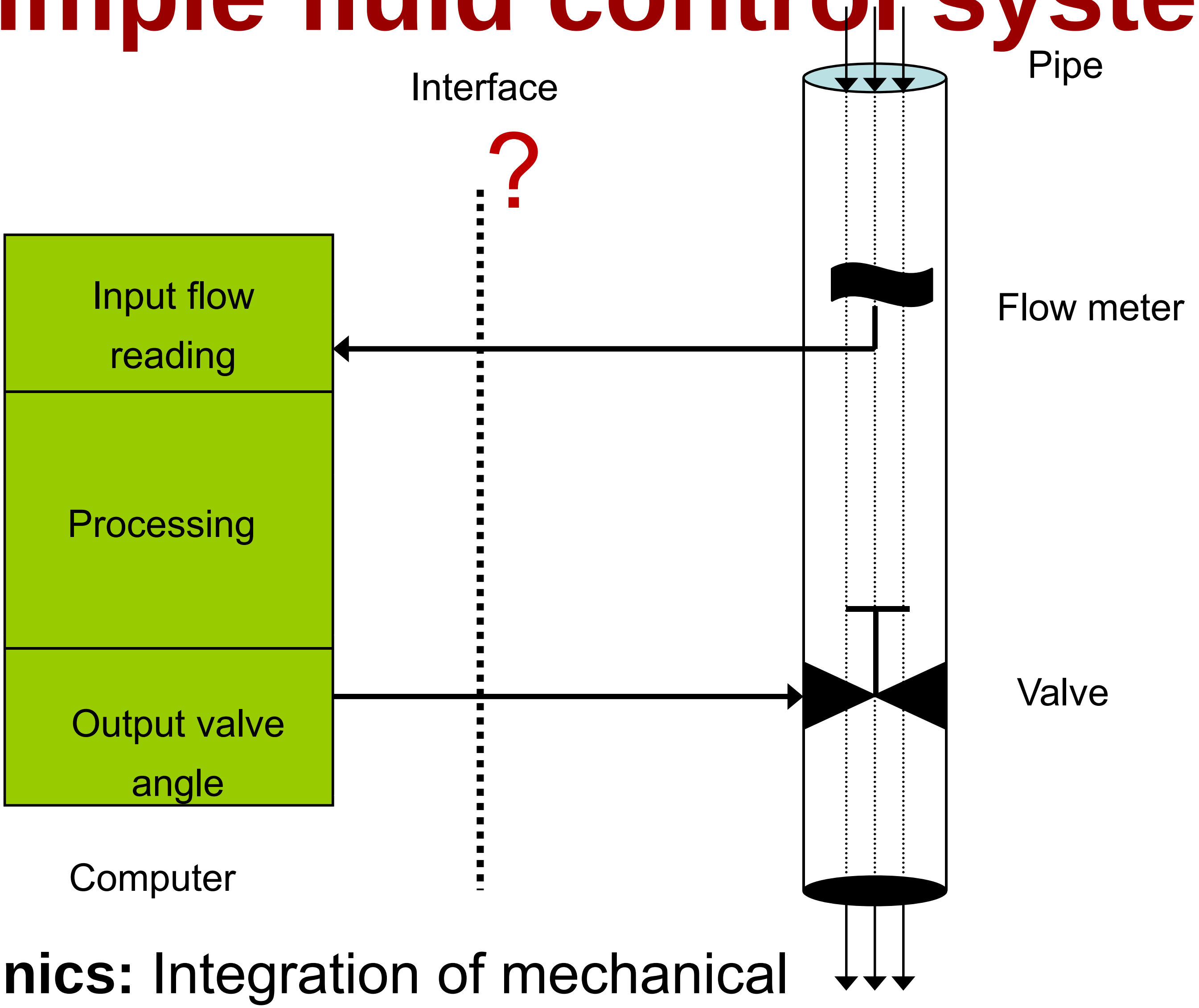
A “short list” of embedded systems

Anti-lock brakes
Auto-focus cameras
Automatic teller machines
Automatic toll systems
Automatic transmission
Avionic systems
Battery chargers
Camcorders
Cell phones
Cell-phone base stations
Cordless phones
Cruise control
Curbside check-in systems
Digital cameras
Disk drives
Electronic card readers
Electronic instruments
Electronic toys/games
Factory control
Fax machines
Fingerprint identifiers
Home security systems
Life-support systems
Medical testing systems
Modems
MPEG decoders
Network cards
Network switches/routers
On-board navigation
Pagers
Photocopiers
Point-of-sale systems
Portable video games
Printers
Satellite phones
Scanners
Smart ovens/dishwashers
Speech recognizers
Stereo systems
Teleconferencing systems
Televisions
Temperature controllers
Theft tracking systems
TV set-top boxes
VCR’s, DVD players
Video game consoles
Video phones
Washers and dryers



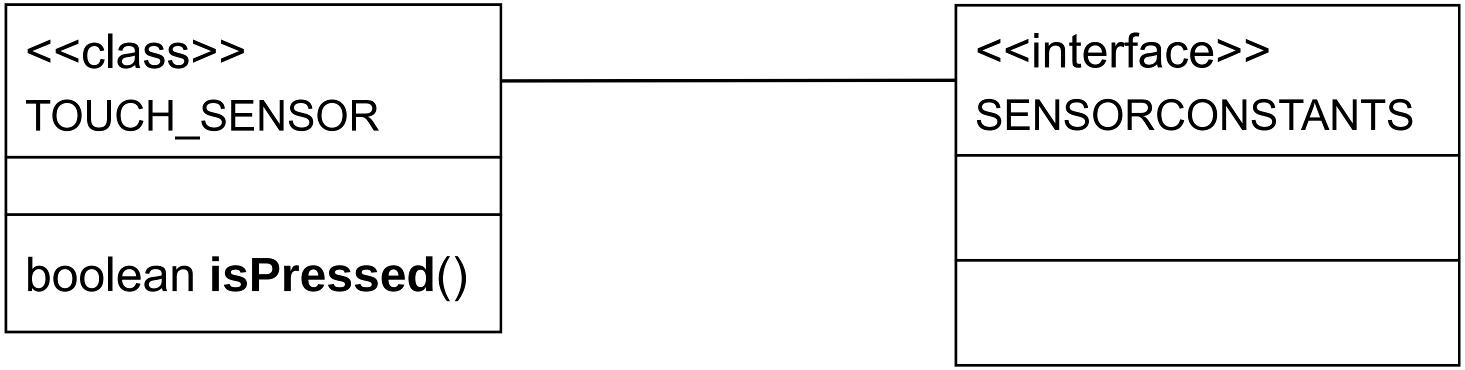
And the list goes on and on

A simple fluid control system



Mechatronics: Integration of mechanical enginnering with electronics and intelligent computer control in the design and manifacturing of industrial products and processes

NXT Sensor API



<<class>>
TOUCH_SENSOR

boolean **isPressed**()

<<interface>>
SENSORCONSTANTS

JAVADOC

Constructor Detail

TouchSensor

`public TouchSensor(ADSensorPort port)`

Create a touch sensor object attached to the specified port.

Parameters:
`port` - port, e.g. Port.S1

Method Detail

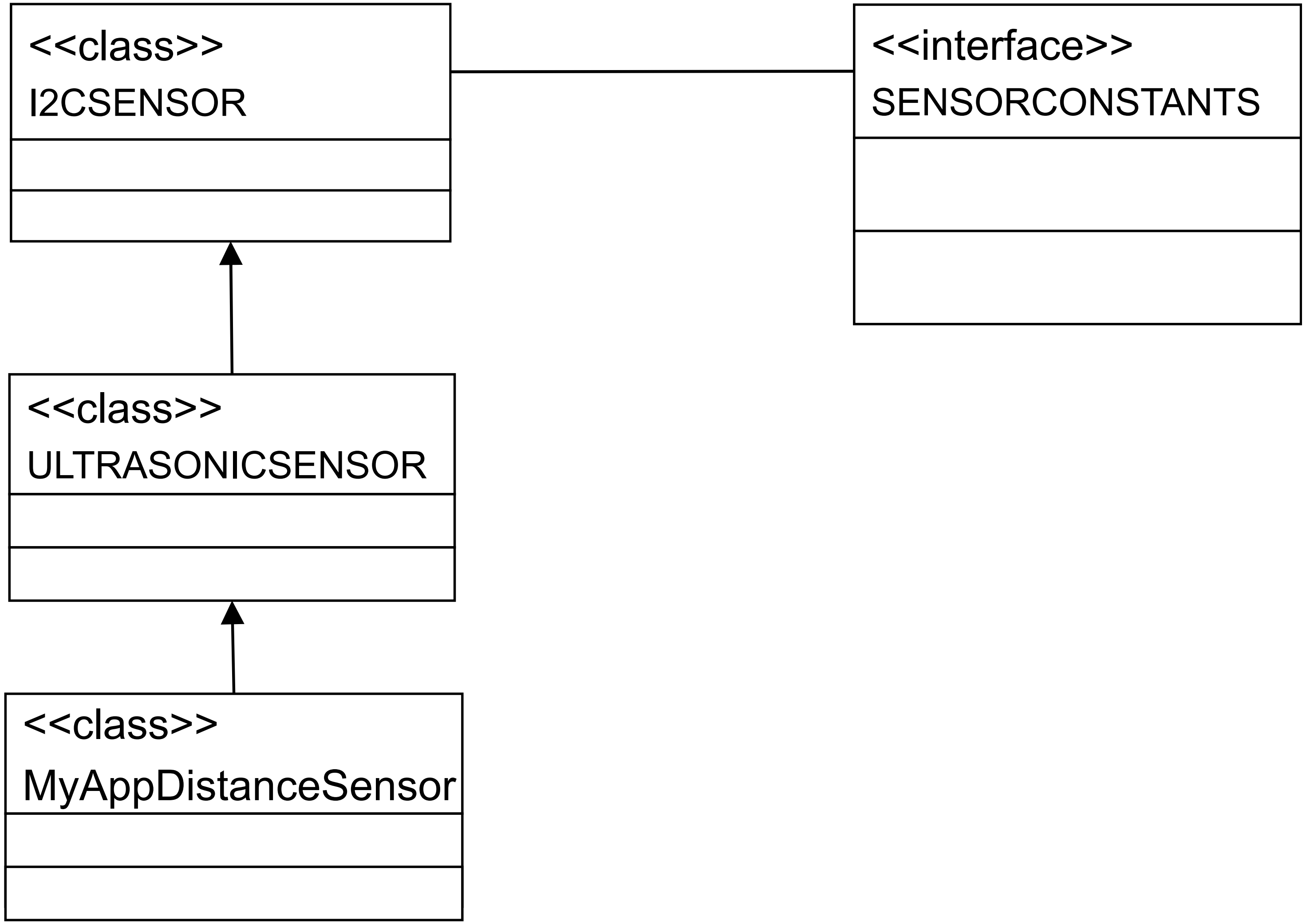
isPressed

`public boolean isPressed()`

Check if the sensor is pressed.

Returns:
`true` if sensor is pressed, `false` otherwise.

NXT Sensor API



<<class>>
I2CSENSOR

<<interface>>
SENSORCONSTANTS

<<class>>
ULTRASONICSENSOR

<<class>>
MyAppDistanceSensor

UltraSonic Methods

- int **capture**() *Set capture mode Set the sensor into capture mode.*
- int **continuous**() *Switch to continuous ping mode.*
- int **getCalibrationData**(byte[] data) *Return 3 bytes of calibration data.*
- byte **getContinuousInterval**() *Return the interval used in continuous mode*.
- int **getData**(int register, byte[] buf, int len) *Executes an I2C read transaction*
- int **getDistance**() *Return distance to an object.*
- int **getDistances**(int[] dist) *Return an array of 8 echo distances.*
- int **getFactoryData**(byte[] data) *Return 10 bytes of factory calibration data*.
- byte **getMode**() *Returns the current operating mode of the sensor.*
- String **getUnits**() *Return a string indicating the type of units in use by the unit.*
- int **off**() *Turn off the sensor*.
- int **ping**() *Send a single ping.*
- int **reset**() *Reset the device Performs a "soft reset" of the device.*
- int **sendData**(int register, byte[] buf, int len) *Executes an I2C write transaction.*
- int **setCalibrationData**(byte[] data) *Set 3 bytes of calibration data.*
- int **setContinuousInterval**(byte interval) *Set the ping interval in continuous mode.*

Sensors & Actuators

- Acknowledgements

Hardware of the sensor network
-- Sensors and peripheral hardware

-- Lin Gu

Sept 8, 2003

Sensors and Actuators

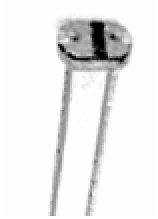
- Transducer: “A device which transforms energy from one domain (magnetic, thermal, mechanical, optical, chemical, electrical) into another”
- **Sensors:** “devices which monitor a parameter of a system, hopefully without disturbing that parameter.”
- **Actuators:** “devices which impose a state on a system, hopefully independent of the load applied to them”

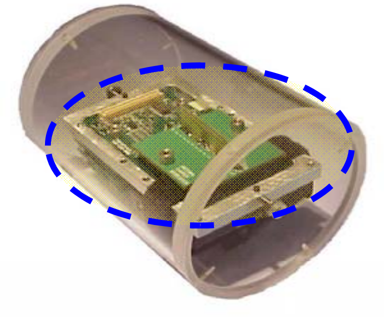
Sensor?

Actuator?

Sensors Examples

- Example of sensors
  - Magnetic sensors
    - Honeywell's HMC/HMR magnetometers
  - Photo sensors
    - Clairex: CL9P4L
  - Temperature sensors
    - Panasonic ERT-J1VR103J
  - Accelerometers
    - Analog Devices: ADXL202JE
  - Motion sensors
    - Advantaca's MIR sensors





- "Without disturbing that parameter" implies that the sensors must be small and low-power devices in order to reduce energy exchange.

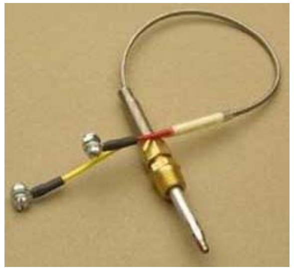
» Sensors: "devices which monitor a parameter of a system, hopefully without disturbing that parameter."

Sensors Types

- Motion / Rotation
- Acceleration
- Force, Torque, Pressure
- Flow
- Temperature
- Proximity
- Light
- Image
- …

Sensor Technology

- EG. Temperature Sensor



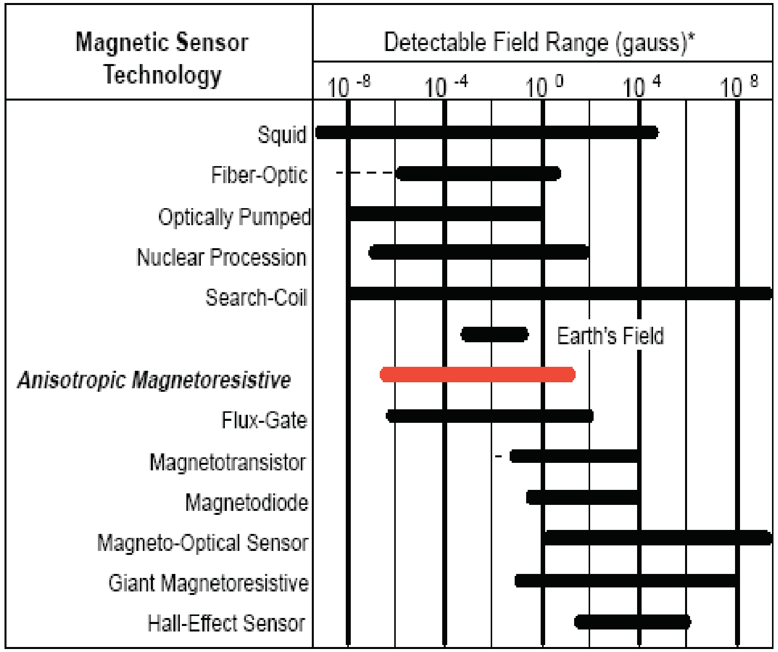
Temperature sensors

Thermocouples	This is the cheapest and the most versatile sensor Applicable over wide temperature ranges (−200°C to 1200°C typical)
Thermistors	Very high sensitivity in medium ranges (up to 100°C typical) Compact but nonlinear in nature
Thermodiodes, thermo transistors	Ideally suited for chip temperature measurements Minimized self heating
RTD—resistance temperature detector	More stable over a long period of time compared to thermocouple Linear over a wide range

REMARK: Properties!

Sensors Properties

- Range: Min to Max value
  - Example
    - HMC1053: +/-6 Gauss
  - What decides range?
    - Saturated point
    - Noise
- Accuracy / Error
  - Diff. Actual and measured value
  - HMC1002: 0.05% (Hysteresis)
- Repeatability
  - HMC1002: 0.05%
- Linearity
  - HMC1002: 0.1% (Best fit straight line +/- 1 Gauss)



* Note: 1gauss = 10^{-4} Tesla = 10^{5} gamma

Table 1–Magnetic sensor technology field ranges

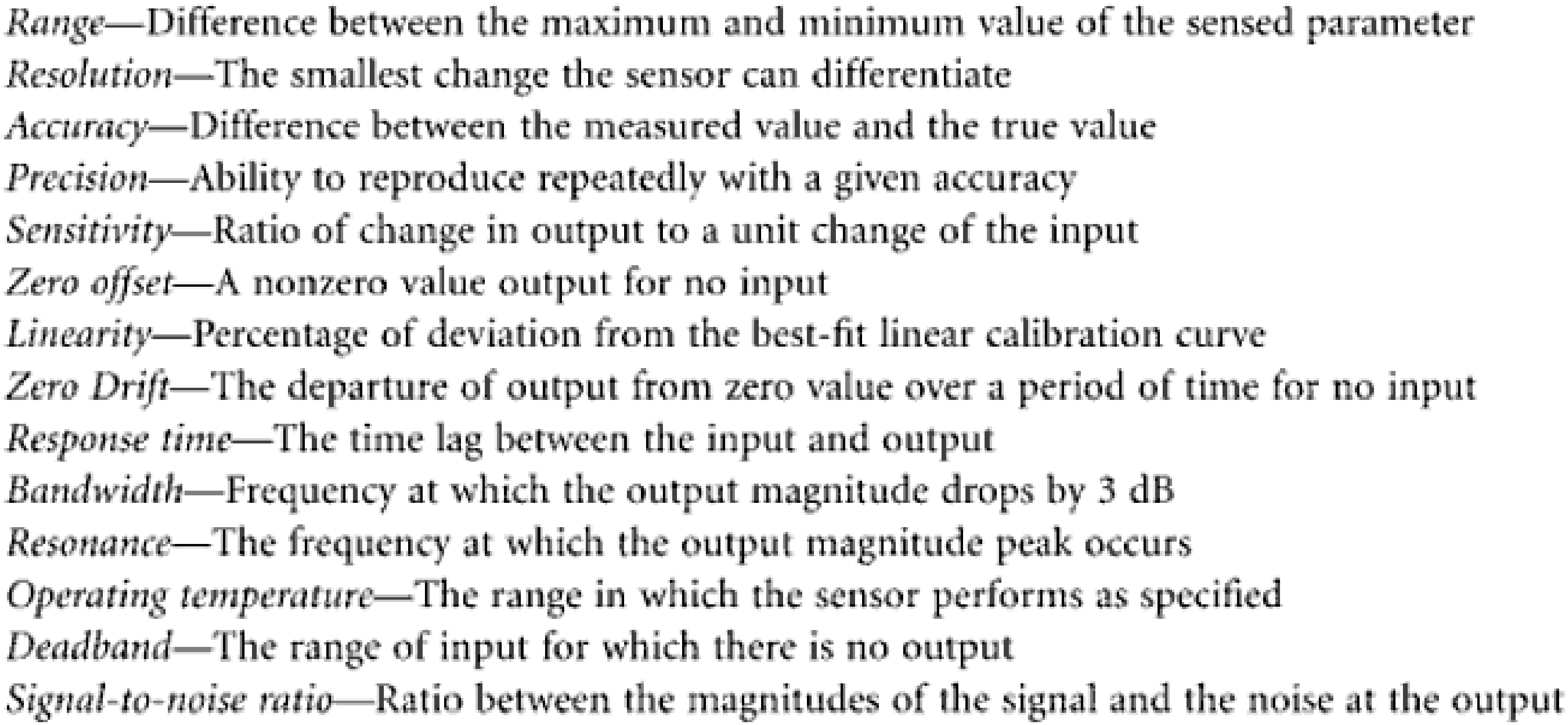
Sensors Properties

- Sensitivity
  - How output reflects input?
  - HMC1053: 1mV/V/gauss
- Efficiency
  - Ratio of the output power to the input power
  - Important for actuators
- Resolution
  - Determined by sensitivity and noise level
  - Measuring noise level
    - SNR
    - Noise floor (High noise floor does not mean “useless”)
  - HMC1002: 27uGauss

Sensors Properties

- Response time
  - How fast the output reaches a fraction of the expected signal level
- Overshoot
  - How much does the output signal go beyond the expected signal level
- Drift and stability
  - How the output signal varies slowly compared to time
- Offset
  - The output when there is no input

Sensor Properties



Range—Difference between the maximum and minimum value of the sensed parameter
Resolution—The smallest change the sensor can differentiate
Accuracy—Difference between the measured value and the true value
Precision—Ability to reproduce repeatedly with a given accuracy
Sensitivity—Ratio of change in output to a unit change of the input
Zero offset—A nonzero value output for no input
Linearity—Percentage of deviation from the best-fit linear calibration curve
Zero Drift—The departure of output from zero value over a period of time for no input
Response time—The time lag between the input and output
Bandwidth—Frequency at which the output magnitude drops by 3 dB
Resonance—The frequency at which the output magnitude peak occurs
Operating temperature—The range in which the sensor performs as specified
Deadband—The range of input for which there is no output
Signal-to-noise ratio—Ratio between the magnitudes of the signal and the noise at the output

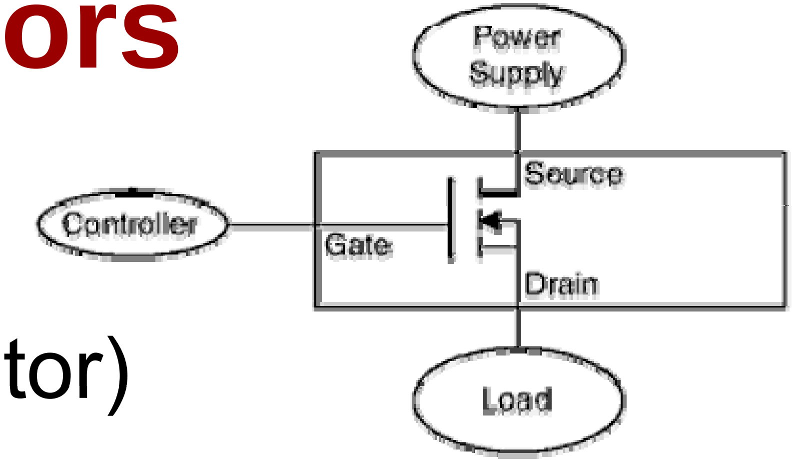
Actuators

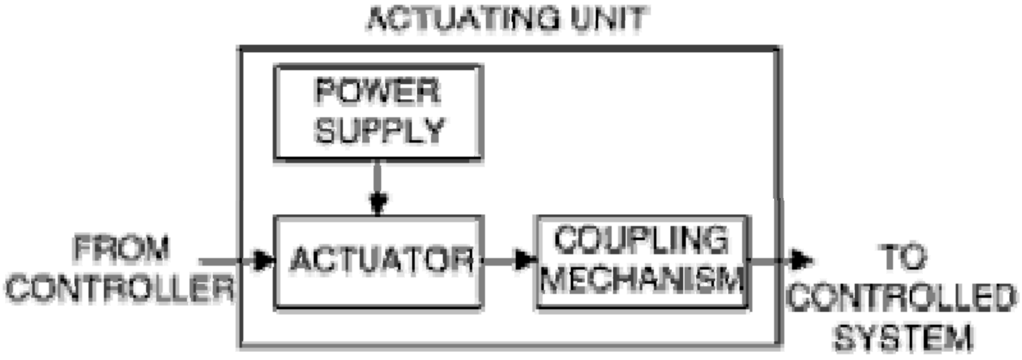
- Examples of Actuators
  - Motor (impose a torque)
  - Pumps (impose pressure or fluid velocity)
- Actuators may be powerful, large, and complicated

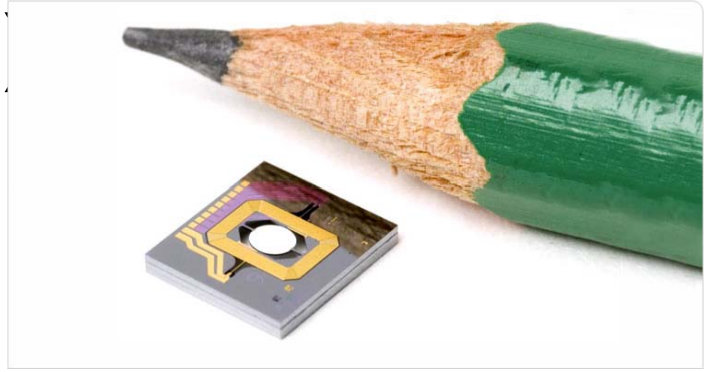
      - » Actuators: “devices which impose a state on a system, hopefully independent of the load applied to them”

Actuators

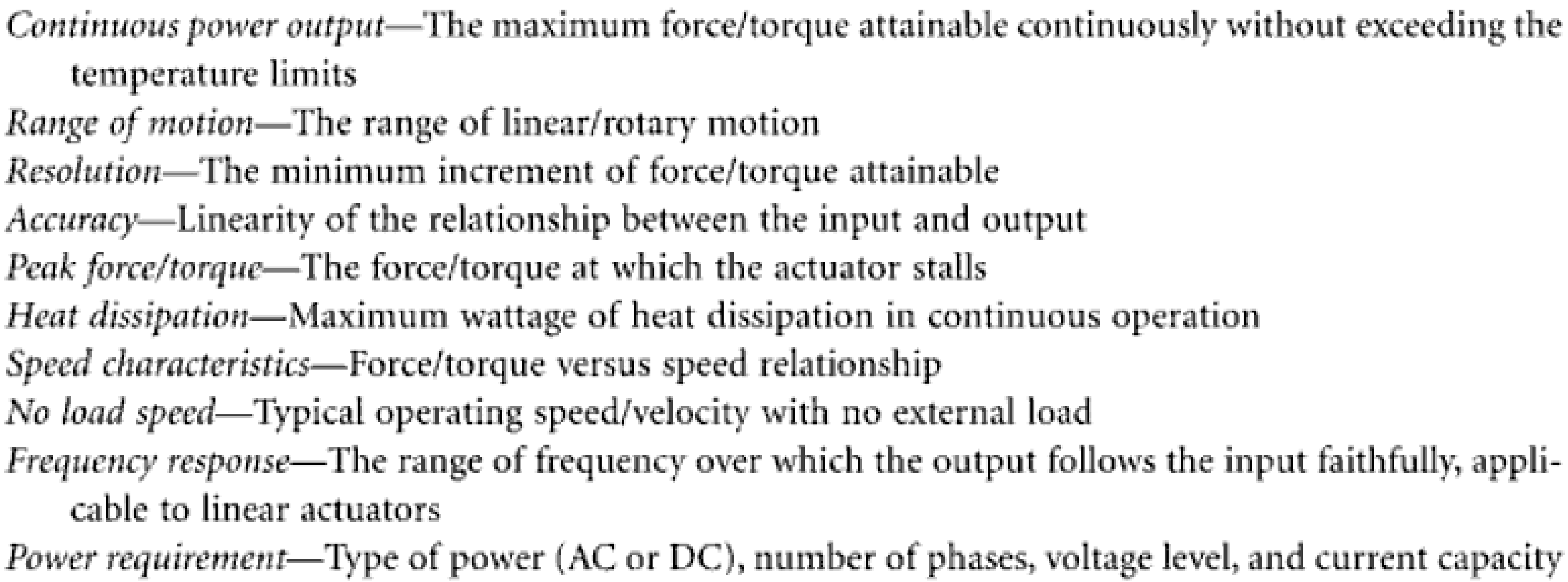
- Electrical
- Electromechanical (motor)
- Electromagnetic
- Hydraulic
- Pneumatic
- Nano/Micro (MEMS)
- …







Actuator Properties



Continuous power output—The maximum force/torque attainable continuously without exceeding the temperature limits

Range of motion—The range of linear/rotary motion

Resolution—The minimum increment of force/torque attainable

Accuracy—Linearity of the relationship between the input and output

Peak force/torque—The force/torque at which the actuator stalls

Heat dissipation—Maximum wattage of heat dissipation in continuous operation

Speed characteristics—Force/torque versus speed relationship

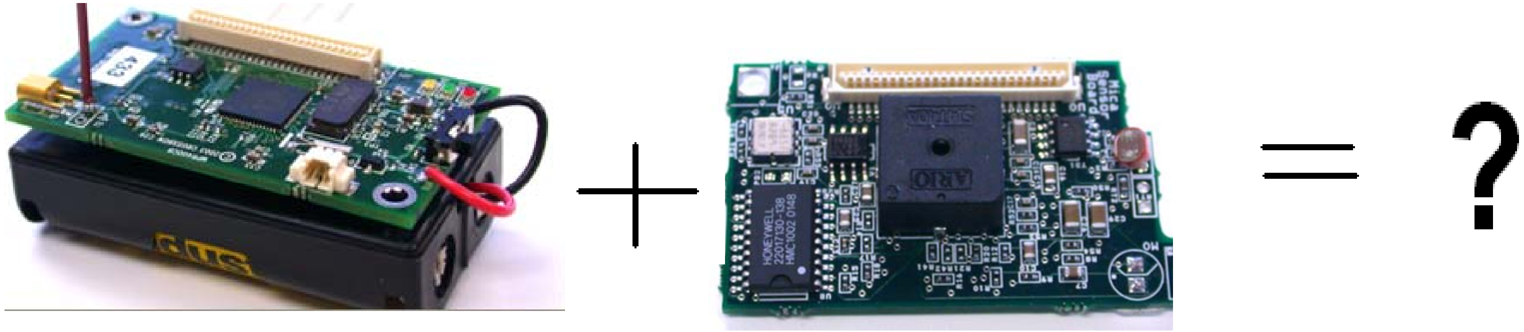
No load speed—Typical operating speed/velocity with no external load

Frequency response—The range of frequency over which the output follows the input faithfully, applicable to linear actuators

Power requirement—Type of power (AC or DC), number of phases, voltage level, and current capacity

Application Requirements

- What's the implication to the application/middleware?
  - Select the suitable sensors for the target application
  - Imposing three general requirements to the application/middleware



Application Requirements

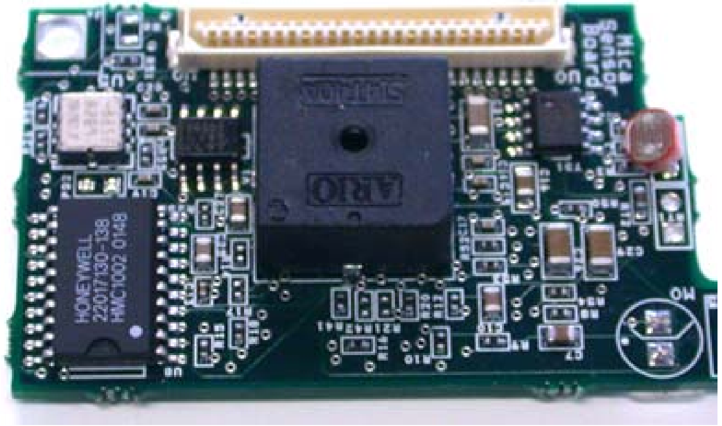
- Requirement 1: sensor part
  - Application designer must be aware of the properties of sensors
    - How to handle imperfect sensor devices
      - Error, offset, drift, …
      - Repeatability
      - Sensors vary
- Requirement 2: sensor reading
  - Application designers must be aware of the errors introduced by the mote hardware?
    - The effect of AD converting
    - The effect of signal amplification/distortion

Application Requirements

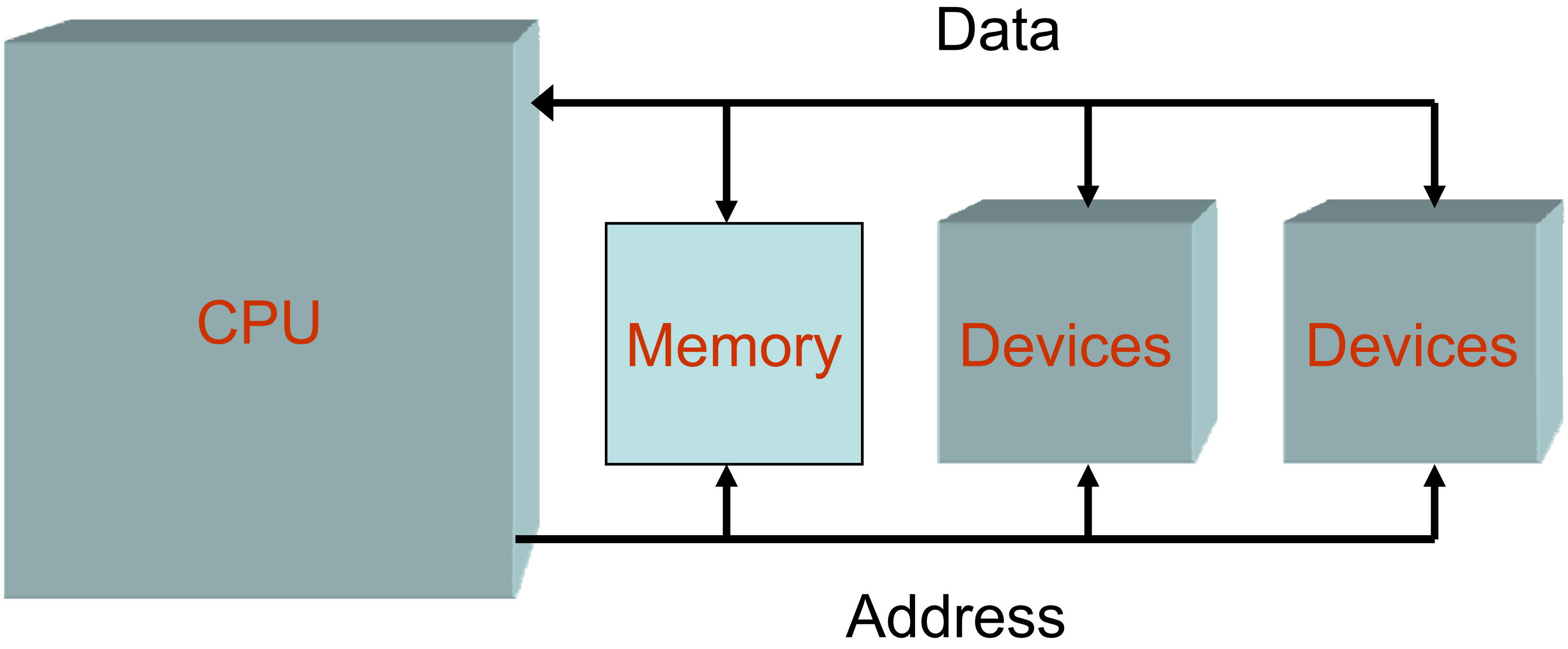
- Requirement 3: interaction
  - The application designer must be aware of the interaction of multiple sensors and the mote hardware
    - How to avoid race conditions on hardware wires and software event handlers?
    - How to control the mutual interaction of various hardware components?
      - Example: radio component increases the noise floor of the motion sensor
    - Can we make the sensors complement with each other to achieve better sensing?

Supporting circuit

- Sensors may need supporting circuit to integrate with other sensors and the target application platform
  - Makes the electrical features of the computer and the I/O device compatible
  - Provides control and data transfer interface to the I/O device
    - PORT / Memory map
    - BUS
    - Interrupts
- Signal conditioning
  - Filtering
  - Amplification



Eg. Memory Mapped Architecture



Data

CPU

Memory

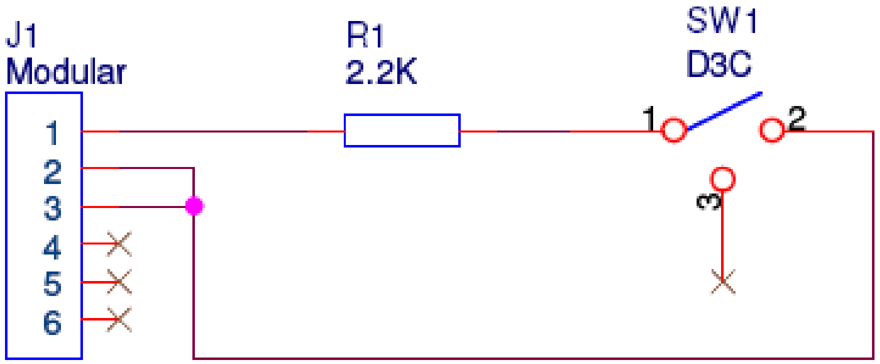
Devices

Devices

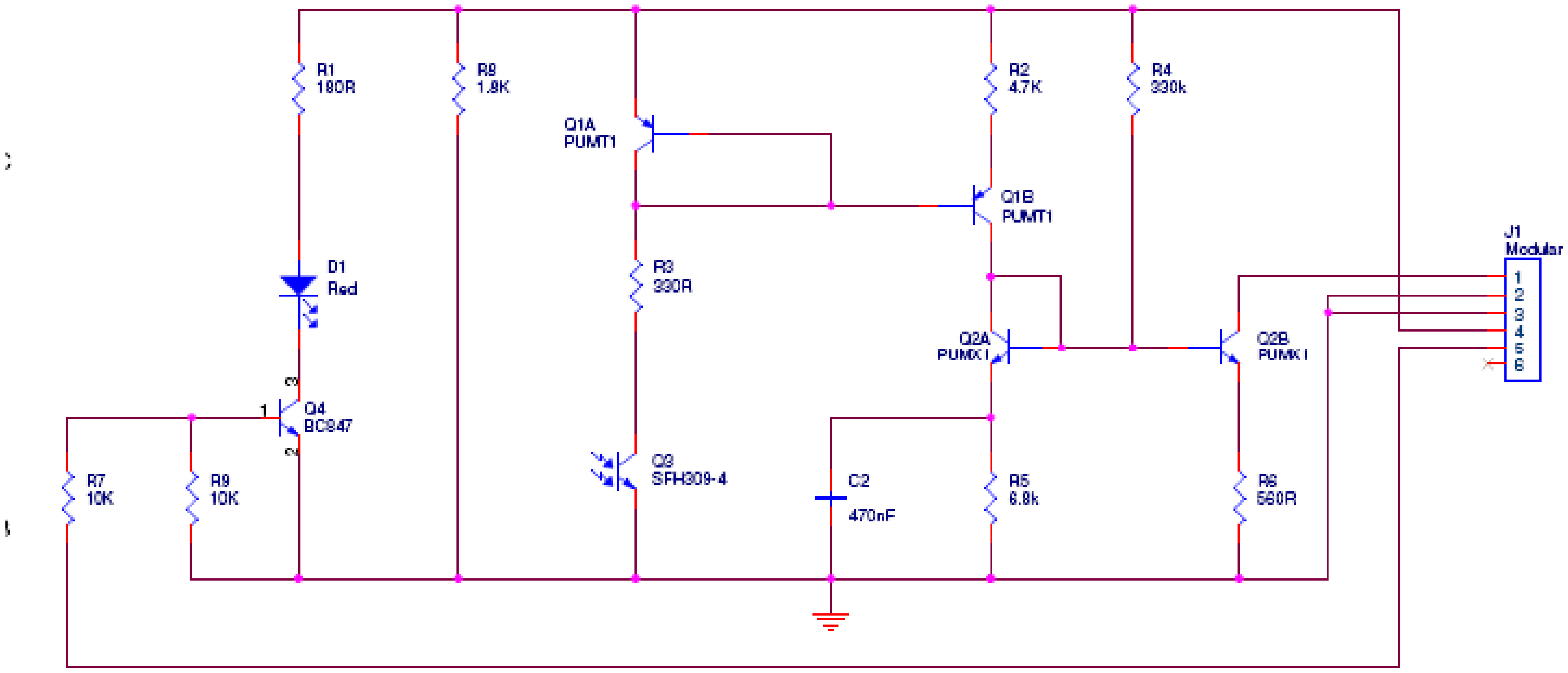
Address

NXT Touch Sensor

BN Cable:
1: White
2: Black
3: Red
4: Green
5: Yellow
6: Blue



NXT Light Sensor

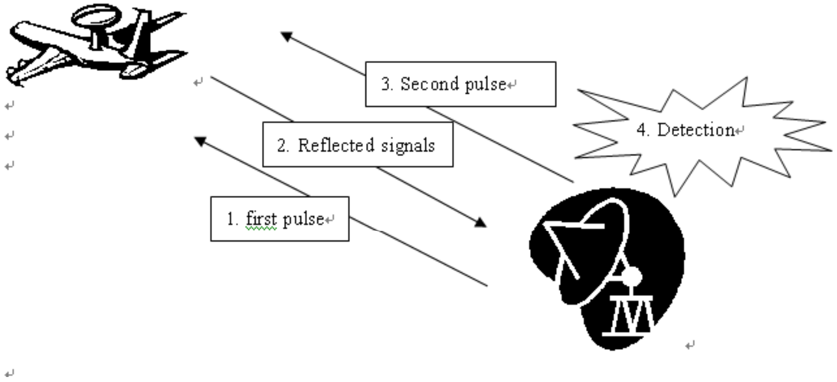


Device Drivers

- Software that controls the operation of an I/O device
  - Uses port registers or memory map to control (read/write) the electronics of the device
  - Polling vs. Interrupt driven
  - Hardware, device and OS dependent
- http://en.wikipedia.org/wiki/Device_driver

Sensors Data Processing Example

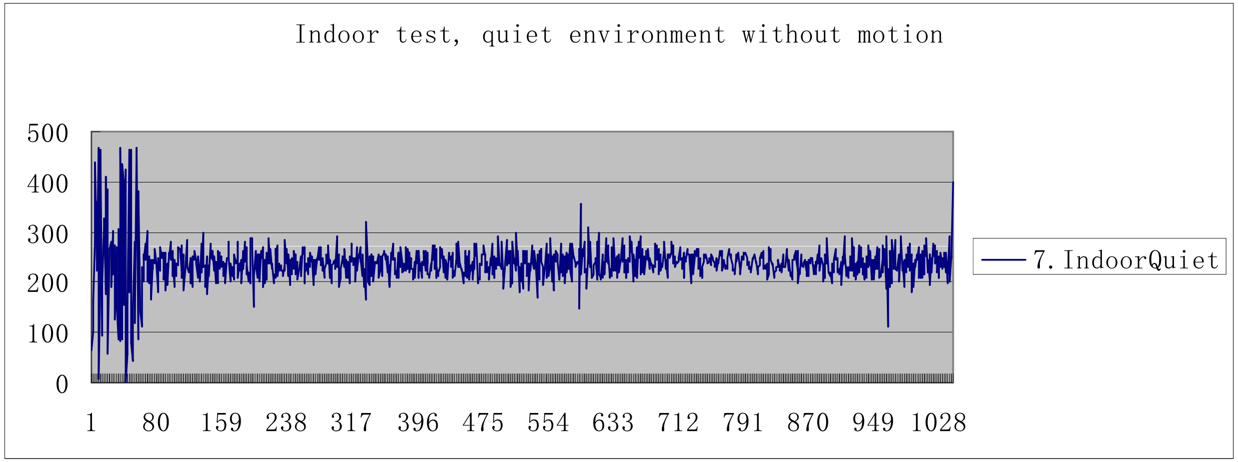
- Motion sensor using MIR
- Micro Impulse Radar
  - TWR-ISM-002
- Output (Advantaca's)
  - Analog
  - Digital
- Packaging
  - 51-pin connector
- Fine tuned receiving gate can potentially detect moving objects at a certain distance
- Is it a typical sensor?



Post-processing

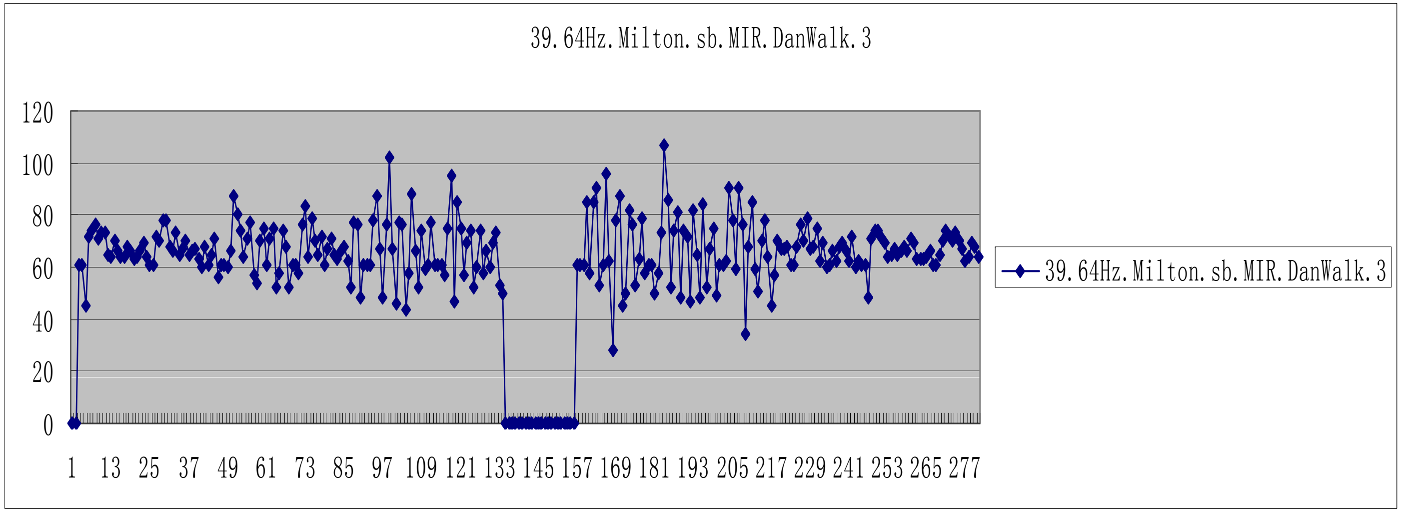
- Post-processing ("POST" ~after raw data has been collected)
  - Process the sensor reading to make it useful to the application
  - The complexity varies from simple threshold algorithm to full-fledged signal processing and pattern recognition
- (but pre – before application decides on actions)

Post Processing MIR Data



- Raw reading of an MIR sensor in a quiet environment
  - The beginning period represents some unknown noise, possibly due to the positioning of the sensor

Post Processing MIR Data



- Raw reading of an MIR sensor as a person walked by
  - The all-zero period is due to unreliable UART interface used to collect the reading and can be ignored.

Post Processing MIR Data

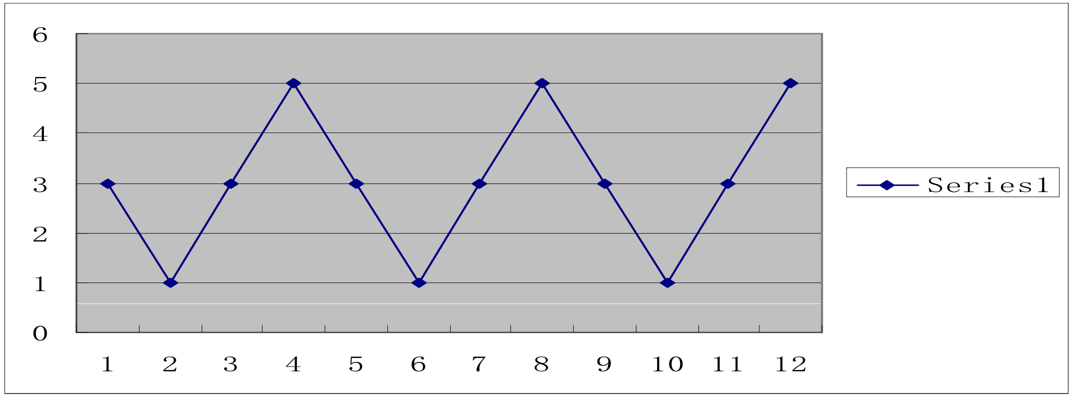
- Use a post-processing algorithm to transform the raw reading to what the application needs
  - The application needs to know whether the motion of interest is detected
  - The post processing needs to filter out noise whenever possible

Post Processing MIR Data

- Post-processing algorithms
  - “Moving variance” algorithm
    - Adapt to the environment dynamically but requires more computation
    - Designed by OSU
    - The basic idea is to track the changes of a statistic variable
    - To avoid the complexity of moving variance computation, another statistics variable was used for mote-based moving object detection
    - If “adapting” feature is not required, offline modeling and online detection can be combined

Post Processing MIR Data

- More on “Moving variance” algorithm
  - Calculate the variance of the samples
  - Example: Suppose the sensor data in a “quiet” environment is as follows

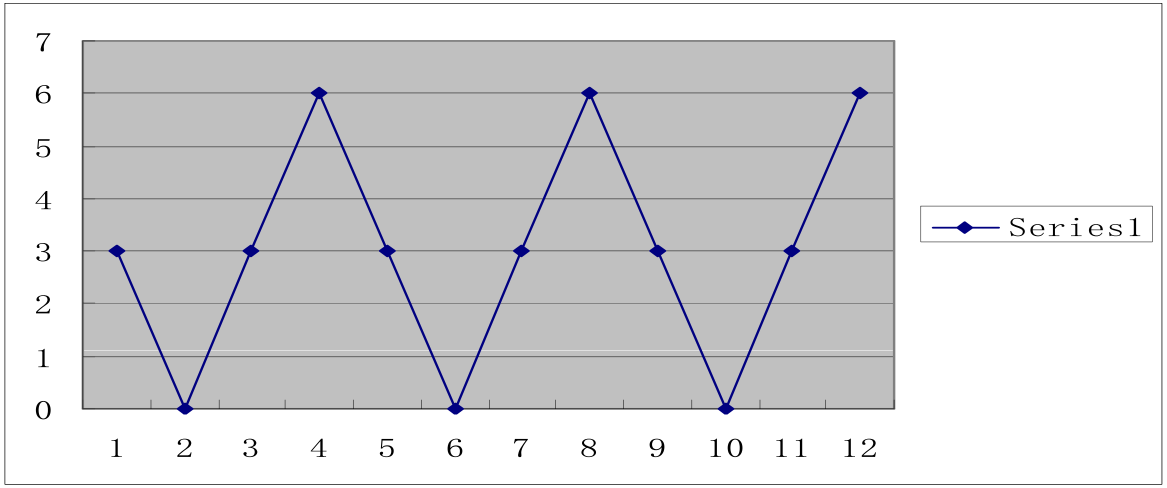


    - Mean: 3
    - Variance: 2.18

      » This is my interpretation of OSU’s algorithm. I have not seen their code or detailed description of it.

Post Processing MIR Data

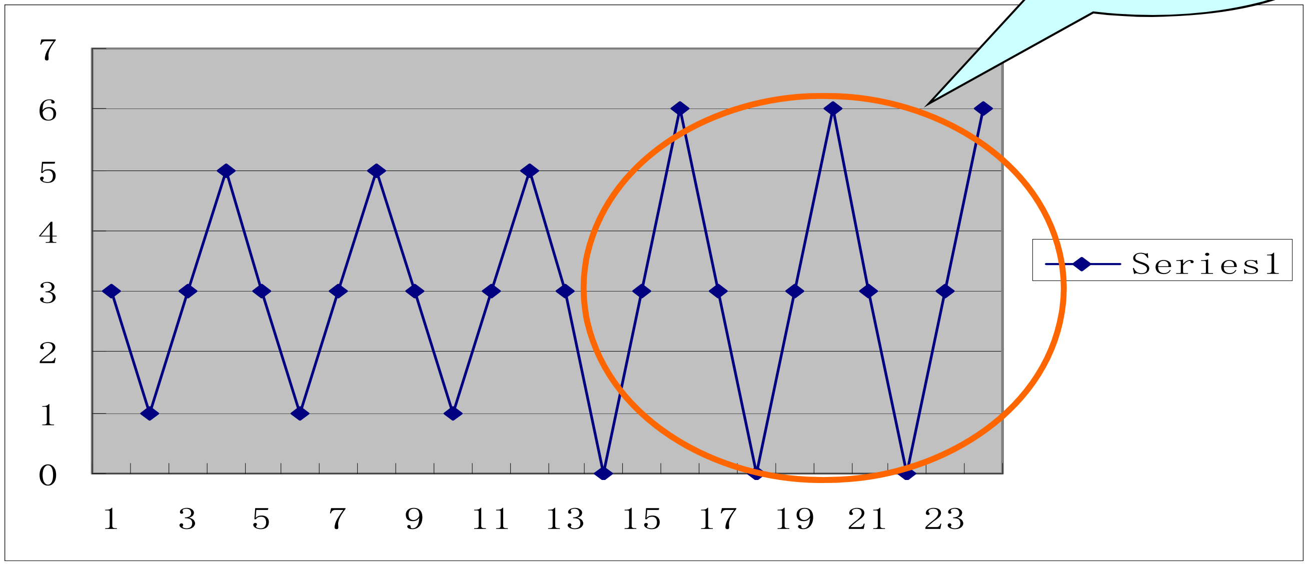
- More on “Moving variance” algorithm
  - Continuously calculate the variance of the recent sampling period
  - When the variance changes, fire a “positive detection” event



- Mean: 3
- Variance: 4.9
  - » This is my interpretation of OSU’s algorithm. I have not seen their code or detailed description of it.

Post Processing MIR Data

- More on “Moving variance” algorithm
  - Overall, the waveform looks like



Motion!

  - On the right half, a “positive” detection event is fired

Post Processing MIR Data

- More on “Moving variance” algorithm
  - This technique can be applied to other statistical variables
    - Mean
    - Standard deviation
    - MIN, MAX
  - The main idea is to use the statistics in a recent sampling period to
    - detect “phase change”
    - filter out burst noise reading
- Change in waveform
- **SIGNAL PROCESSING**

Embedded Systems HW

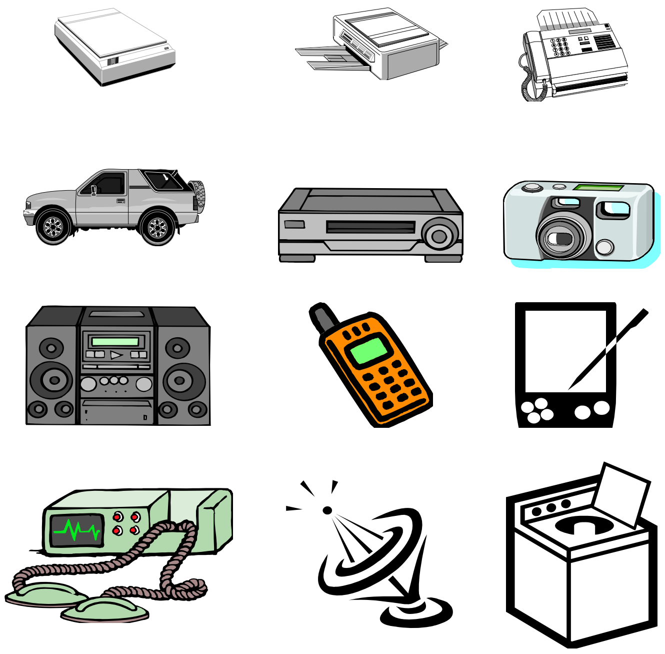
Brian Nielsen

bnielsen@cs.aau.dk

A "short list" of embedded systems

Anti-lock brakes
Auto-focus cameras
Automatic teller machines
Automatic toll systems
Automatic transmission
Avionic systems
Battery chargers
Camcorders
Cell phones
Cell-phone base stations
Cordless phones
Cruise control
Curbside check-in systems
Digital cameras
Disk drives
Electronic card readers
Electronic instruments
Electronic toys/games
Factory control
Fax machines
Fingerprint identifiers
Home security systems
Life-support systems
Medical testing systems

Modems
MPEG decoders
Network cards
Network switches/routers
On-board navigation
Pagers
Photocopiers
Point-of-sale systems
Portable video games
Printers
Satellite phones
Scanners
Smart ovens/dishwashers
Speech recognizers
Stereo systems
Teleconferencing systems
Televisions
Temperature controllers
Theft tracking systems
TV set-top boxes
VCR's, DVD players
Video game consoles
Video phones
Washers and dryers



And the list goes on and on

Processor technology

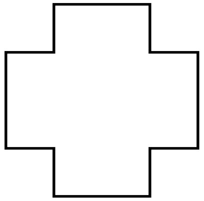
- Processors vary in their customization for the problem at hand

Desired functionality

```
total = 0
for i = 1 to N  loop
   total += M[i]
end loop
```

General-purpose processor

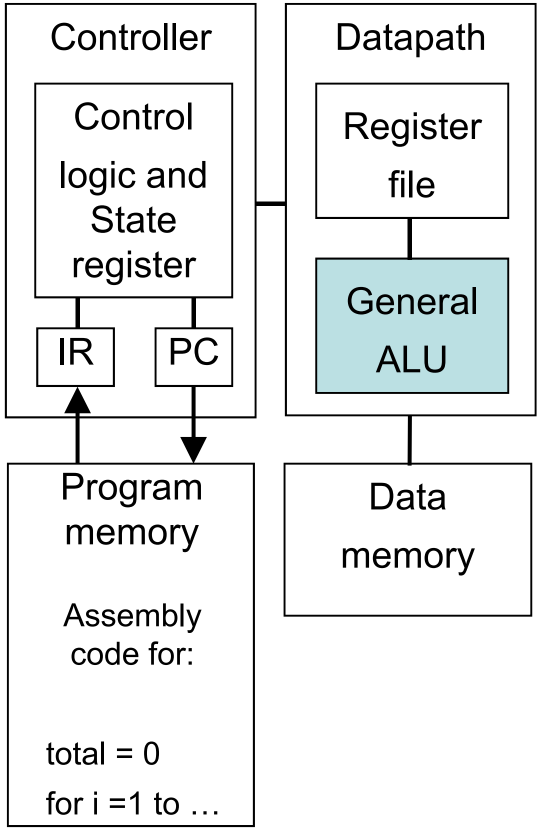
Application-specific processor



Single-purpose processor

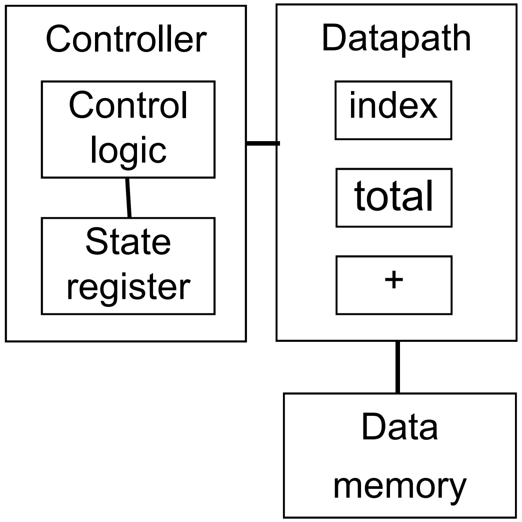
General-purpose processors

- Programmable device used in a variety of applications
  - Also known as “microprocessor”
- Features
  - Program memory
  - General datapath with large register file and general ALU
- User benefits
  - Low time-to-market and NRE costs
  - High flexibility
- “Pentium” the most well-known, but there are hundreds of others



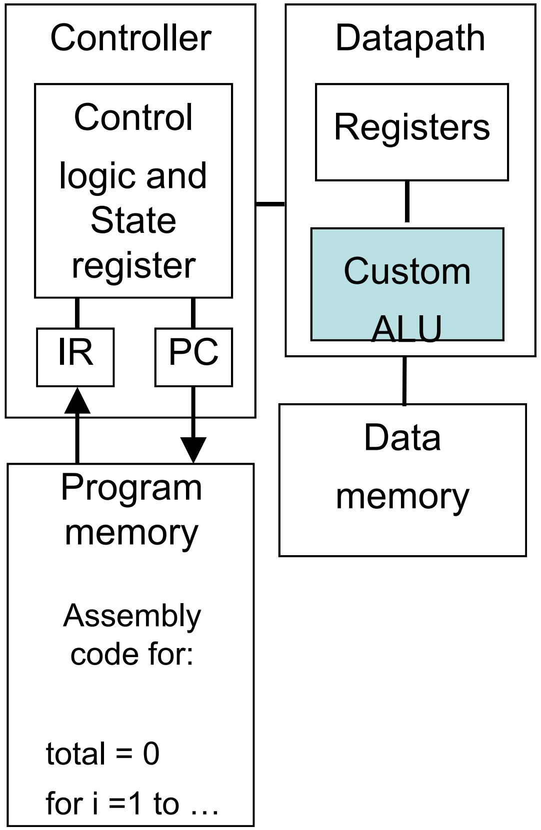
Single-purpose processors

- Digital circuit designed to execute exactly one program
  - a.k.a. coprocessor, accelerator or peripheral
  - JPEG codec
- Features
  - Contains only the components needed to execute a single program
  - No program memory
- Benefits
  - Fast
  - Low power
  - Small size



Application-specific processors

- Programmable processor optimized for a particular class of applications having common characteristics
  - Compromise between general-purpose and single-purpose processors
  - EG microController, DSP
- Features
  - Program memory
  - Optimized datapath
  - Special functional units
- Benefits
  - Some flexibility, good performance, size and power

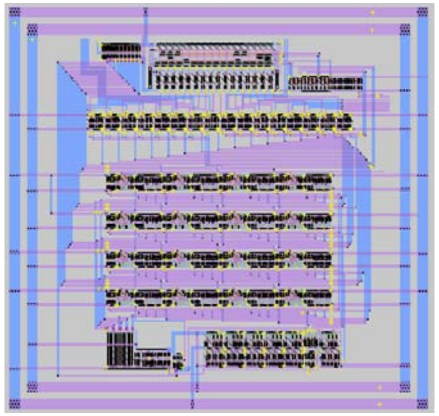


IC technology

- Three types of IC technologies
  - Full-custom/VLSI
  - Semi-custom ASIC (Application Specific Integrated Circuit
  - PLD (Programmable Logic Device)

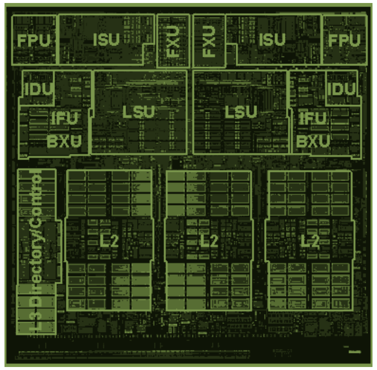
Full-custom/VLSI

- All layers are optimized for an embedded system’s particular digital implementation
  - Placing transistors
  - Sizing transistors
  - Routing wires



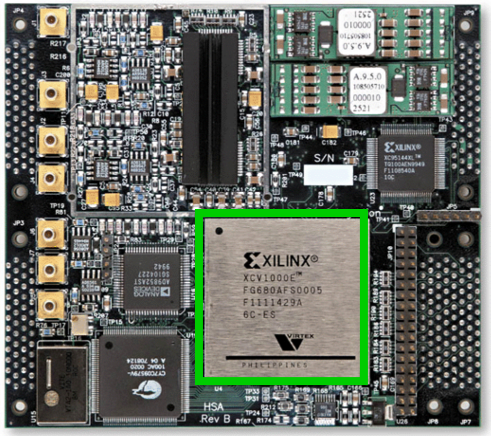
- Benefits
  - Excellent performance, small size, low power
- Drawbacks
  - High NRE cost (e.g., $300k), long time-to-market
  - NRE=Non Recurring Engineering (design)

Semi-custom



- Lower layers are fully or partially built
  - – Designers are left with routing of wires and maybe placing some blocks
- Benefits
  - – Good performance, good size, less NRE cost than a full-custom implementation (perhaps $10k to $100k)
- Drawbacks
  - – Still require weeks to months to develop

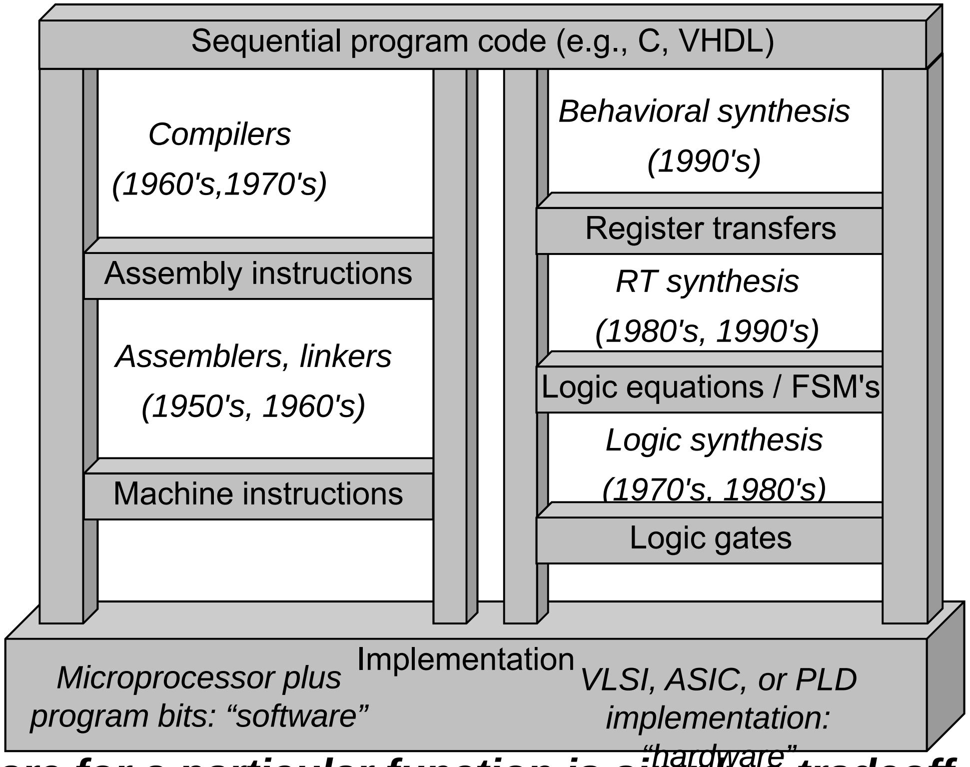
PLD (Programmable Logic Device)



- All layers already exist
  - Designers can purchase an IC
  - Connections on the IC are either created or destroyed to implement desired functionality
  - Field-Programmable Gate Array (FPGA) very popular
- Benefits
  - Low NRE costs, almost instant IC availability
- Drawbacks
  - Bigger, expensive (perhaps $30 per unit), power hungry, slower

The co-design ladder

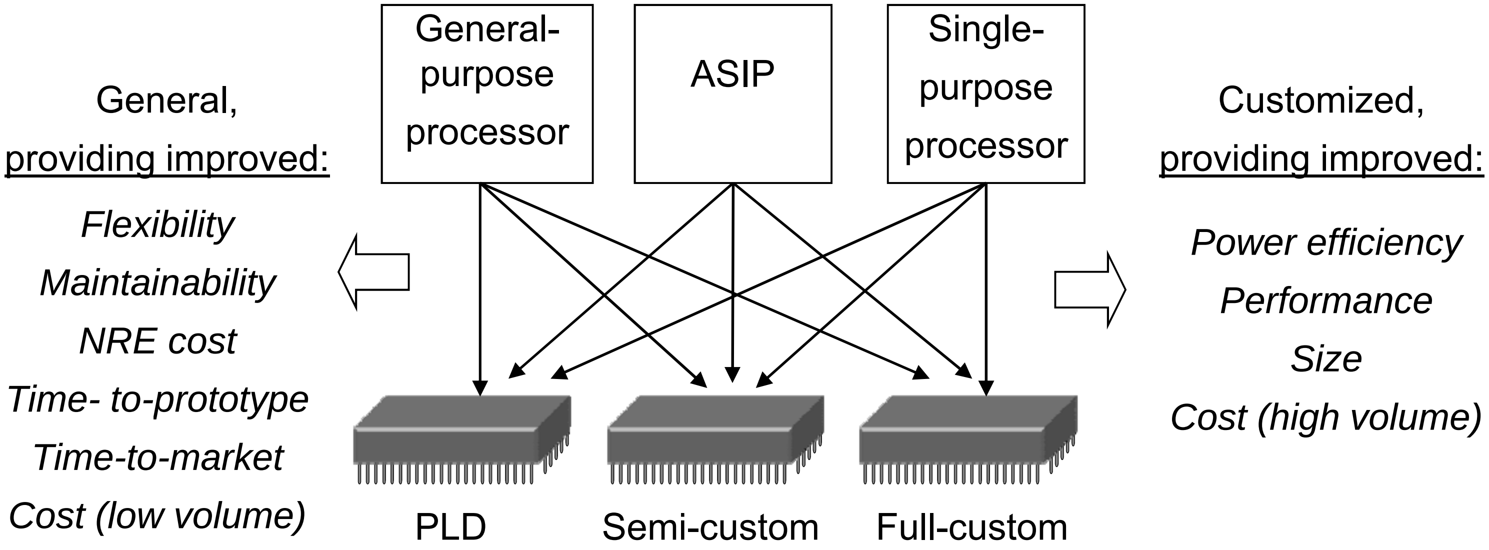
- In the past:
  - Hardware and software design technologies were very different
  - Recent maturation of synthesis enables a unified view of hardware and software
- Hardware/software "codesign"



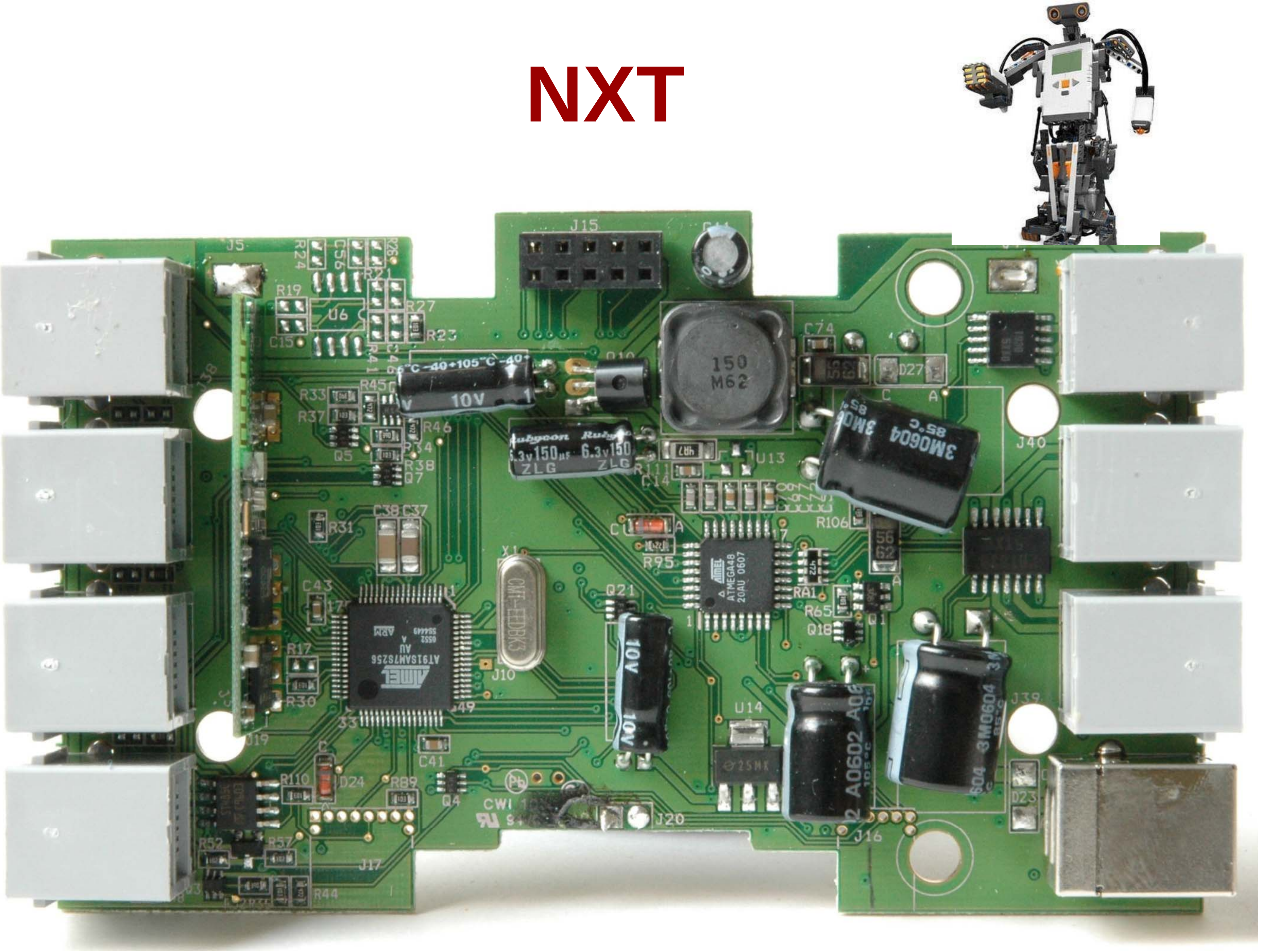
The choice of hardware versus software for a particular function is simply a tradeoff among various design metrics, like performance, power, size, NRE cost, and especially flexibility; there is no fundamental difference between what hardware or software can implement.

Independence of processor and IC technologies

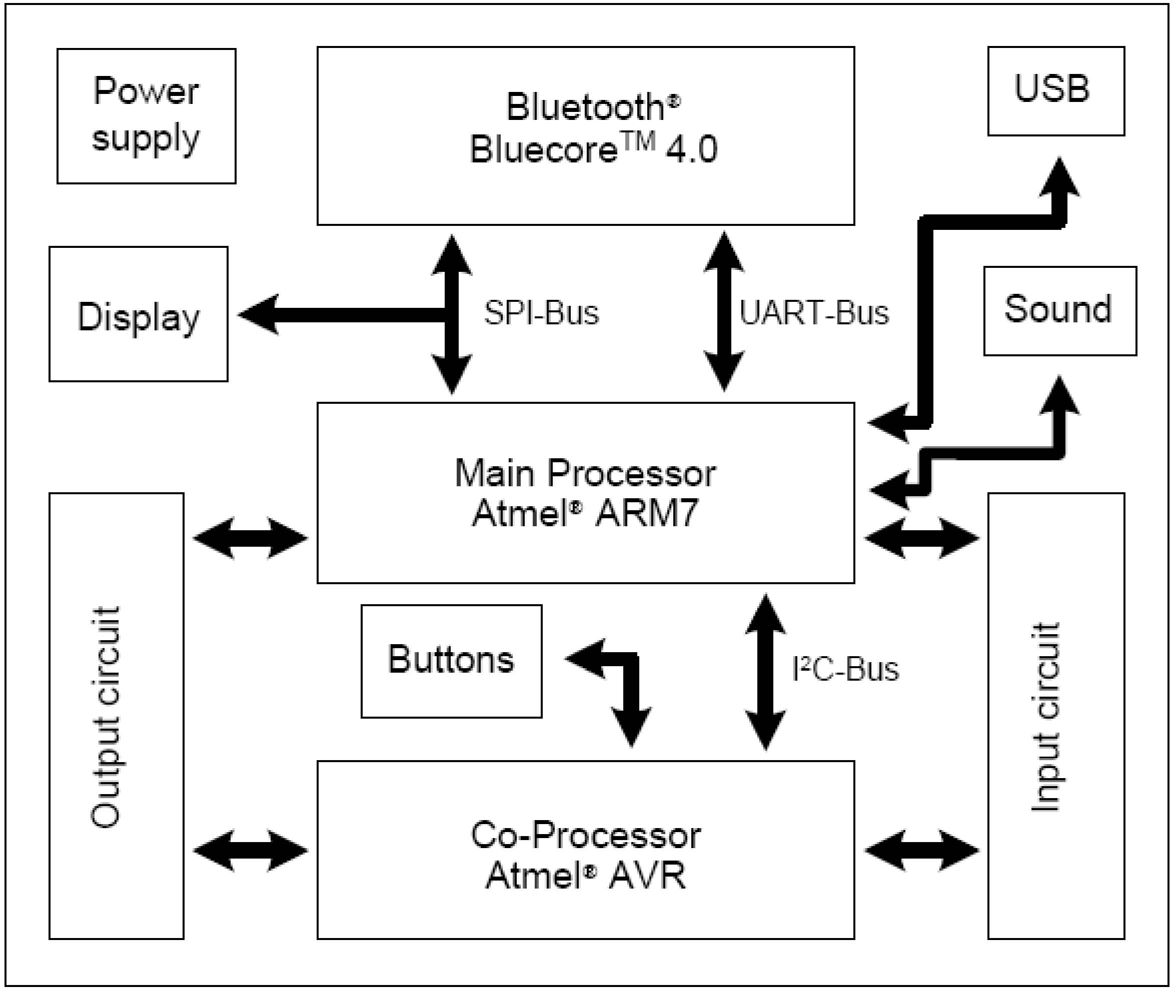
- Basic tradeoff
  - General vs. custom
  - With respect to processor technology or IC technology
  - The two technologies are independent



NXT


J15
J5
ATMEL
AT91SAM7S256
ATMEGA48
20AU 0607
150
M62
X1
J10
J16
J17
J20
J39
J40
U13
U14

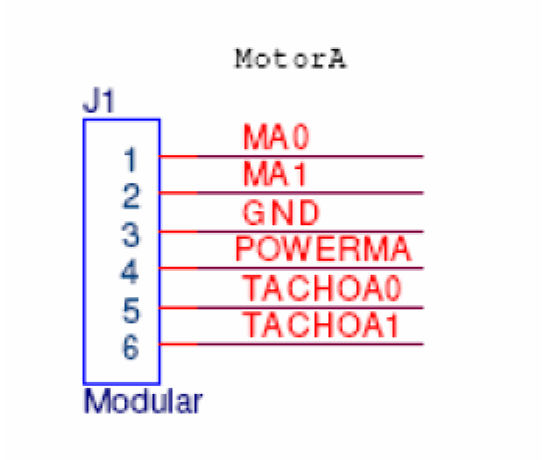
NXT HW Block Diagram



- **Atmel 32 bit ARM**
- 48 MHz
- 256 KB Flash
- 64 KB RAM

- **8 bit AVR** (ATmega48)
- 4KB FLASH
- 512 B RAM
- 8 MHz

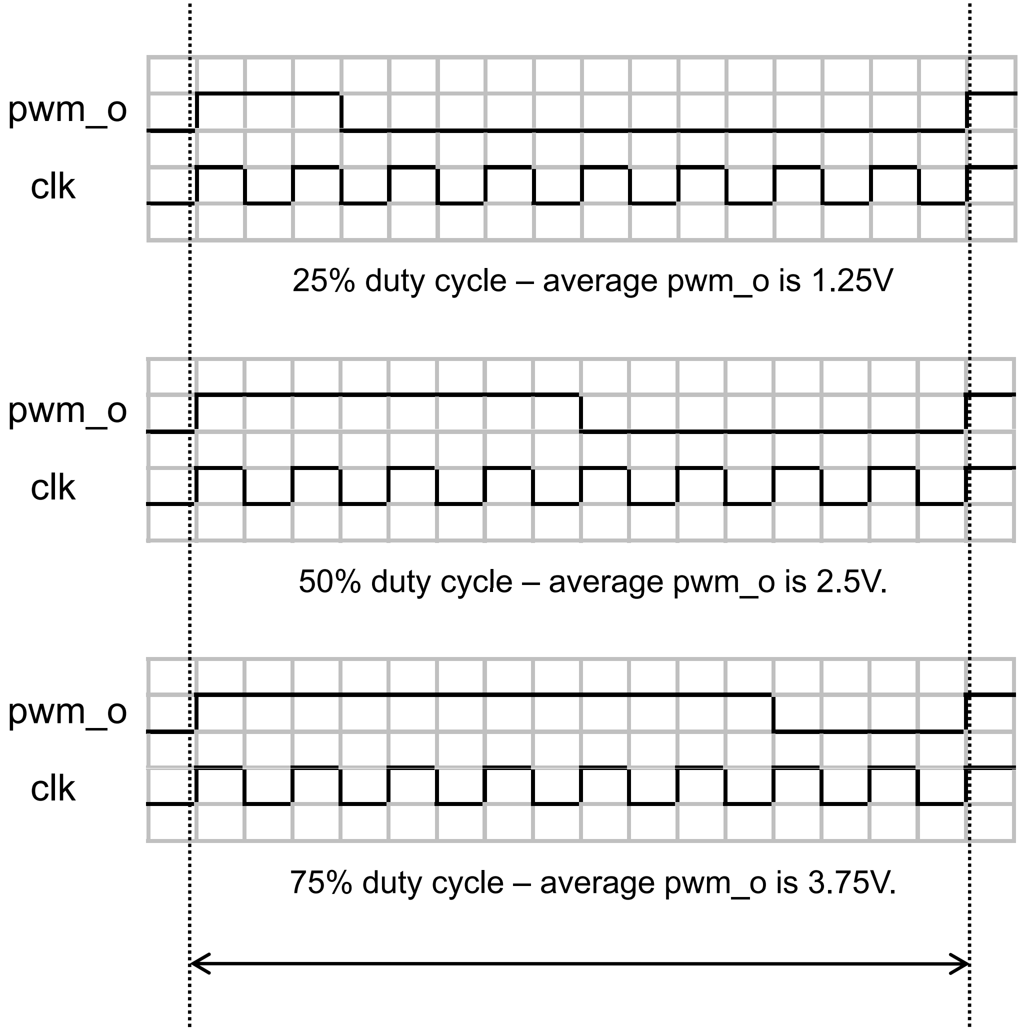
Output Ports



Pin 1, MA0	PWM output signal for the actuators
Pin 2, MA1	PWM output signal for the actuators
Pin 3, GND	Ground signal related to the output supply
Pin 4, POWERMA	4.3 Volt output supply
Pin 5, TACHOA0	Input value that includes Schmitt trigger functionality
Pin 6, TACHOA1	Input value that includes Schmitt trigger functionality

Pulse width modulator

- Generates pulses with specific high/low times
- Duty cycle: % time high
  - Square wave: 50% duty cycle
- Common use: control average voltage to electric device
  - Simpler than DC-DC converter or digital-analog converter
  - DC motor speed, dimmer lights
- Another use: encode commands, receiver uses timer to decode



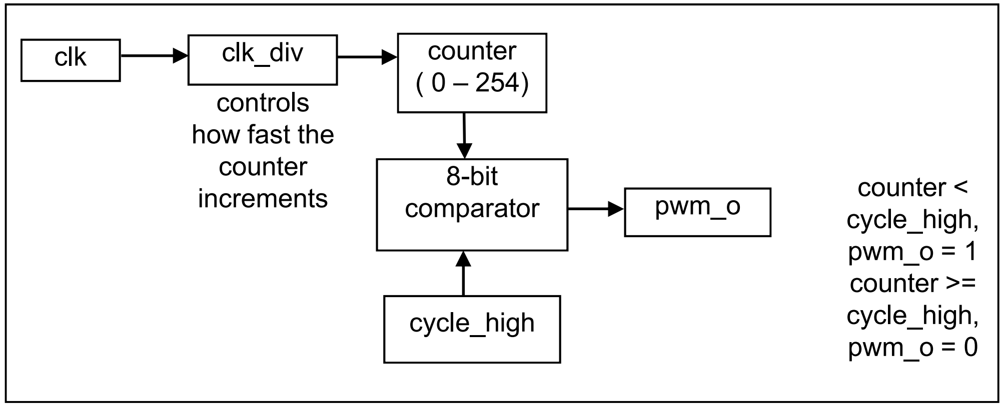
25% duty cycle – average pwm_o is 1.25V

50% duty cycle – average pwm_o is 2.5V.

75% duty cycle – average pwm_o is 3.75V.

Period ~16 clock tics

Controlling a DC motor with a PWM



clk

clk_div

controls how fast the counter increments

counter (0 – 254)

8-bit comparator

pwm_o

cycle_high

counter < cycle_high, pwm_o = 1
counter >= cycle_high, pwm_o = 0

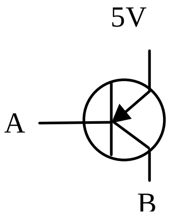
Internal Structure of PWM

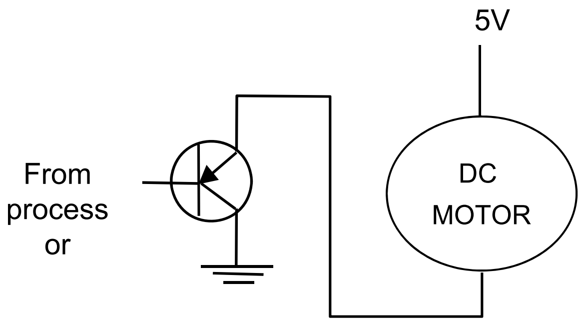
Input Voltage	% of Maximum Voltage Applied	RPM of DC Motor
0	0	0
2.5	50	1840
3.75	75	6900
5.0	100	9200

Relationship between applied voltage and speed of the DC Motor

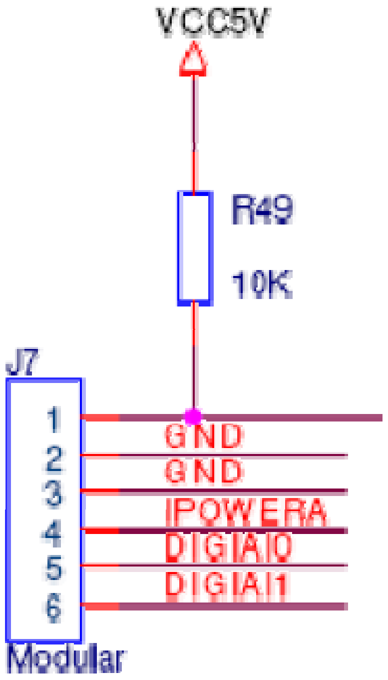
```
void main(void){

   /* controls period */
      PWMP = 0xff;
 /* controls duty cycle */
      PWM1 = 0x7f;

     while(1){};
    }
```

The PWM alone cannot drive the DC motor, a possible way to implement a driver is shown below using an MJE3055T NPN transistor.





Input Ports



Pin 1, ANA	Analog input and possible current output signal
Pin 2, GND	Ground signal
Pin 3, GND	Ground signal
Pin 4, IPOWERA	4.3 Volt output supply
Pin 5, DIGIAI0	Digital I/O pin connected to the ARM7 processor
Pin 6, DIGIAI1	Digital I/O pin connected to the ARM7 processor

- 10 bit AD, 333 Hz (By AVR processor)
- Dig I/O (I2C bus communication -9600bit/s)
- Port 4 - RS484 (921.6 Kbit/s)

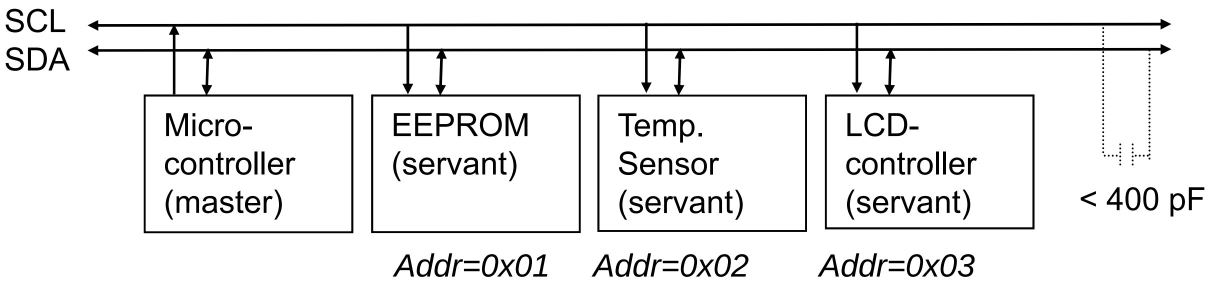
Input Sensors

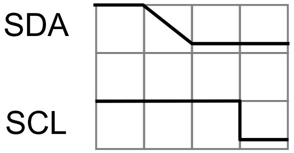
- Passive
  - Light, Touch, Sound, Temp
- Digital
  - UltraSonic
  - I2C
- => Port configuration depends on sensor

Serial protocols: I^2C

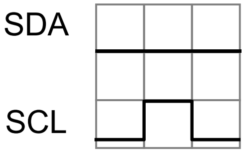
- I^2C (Inter-IC)
  - Two-wire serial bus protocol developed by Philips Semiconductors nearly 20 years ago
  - Enables peripheral ICs to communicate using simple communication hardware
  - Data transfer rates up to 100 kbits/s and 7-bit addressing possible in normal mode
  - 3.4 Mbits/s and 10-bit addressing in fast-mode
  - Common devices capable of interfacing to I^2C bus:
    - EPROMS, Flash, and some RAM memory, real-time clocks, watchdog timers, and microcontrollers

I2C bus structure

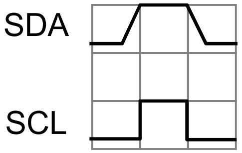




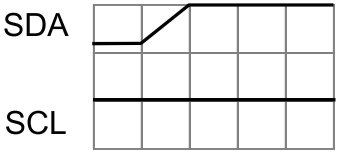
Start condition



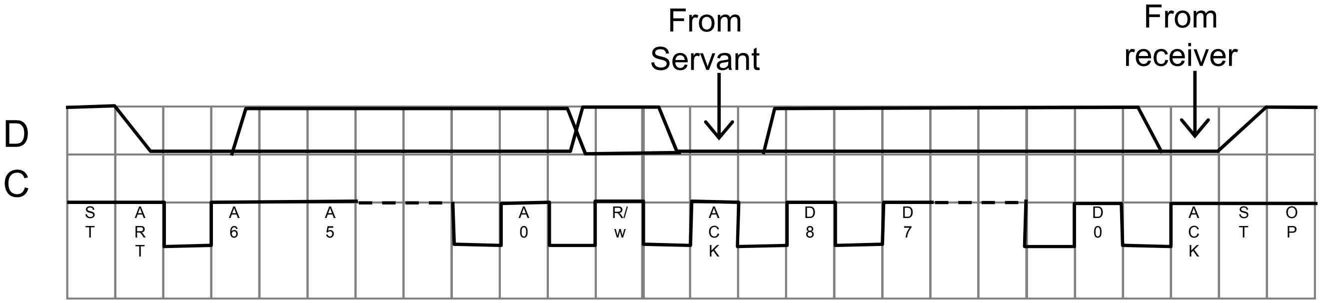
Sending 0



Sending 1



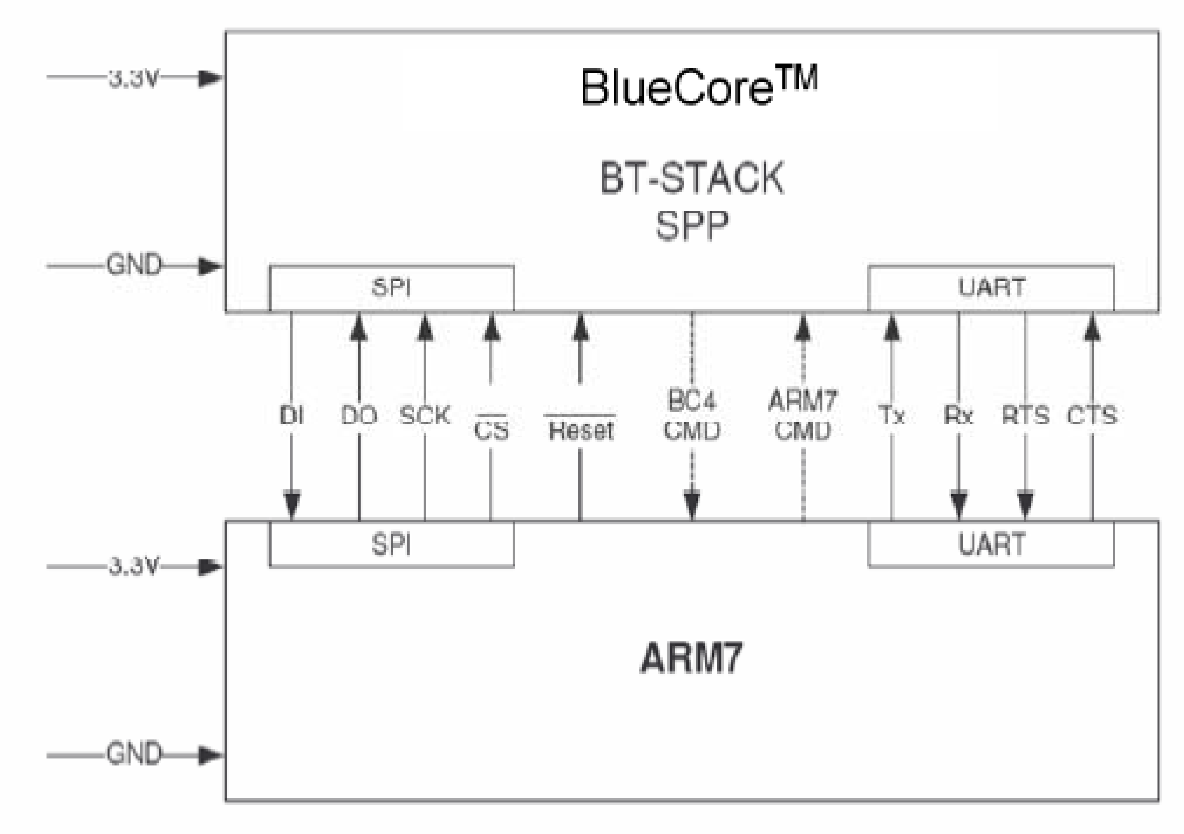
Stop condition



Typical read/write cycle

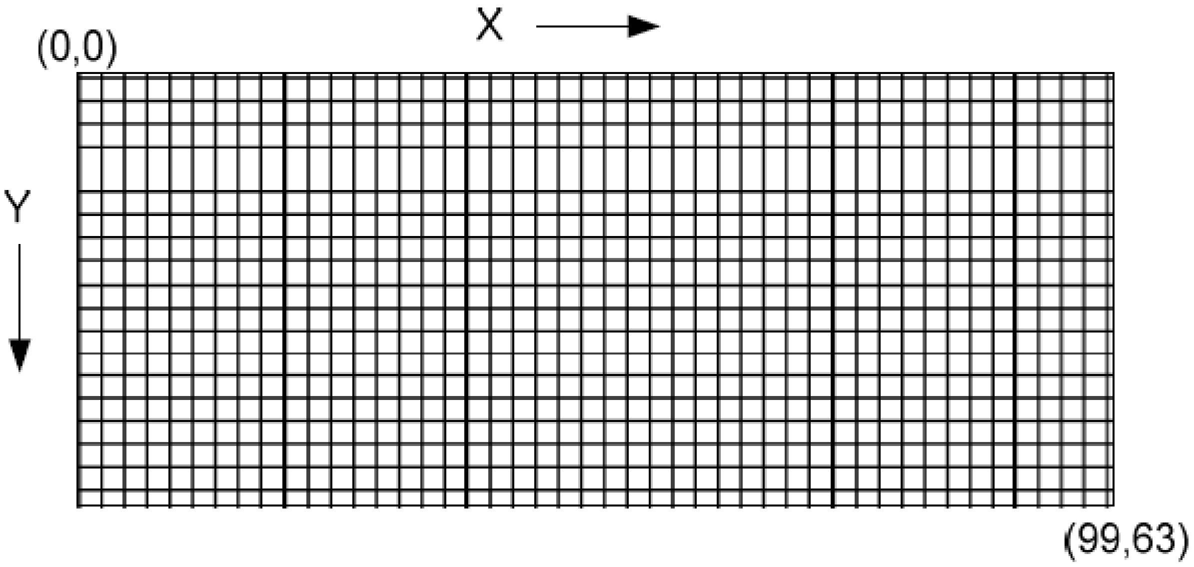
BlueTooth (classII)

- Serial Port Profile



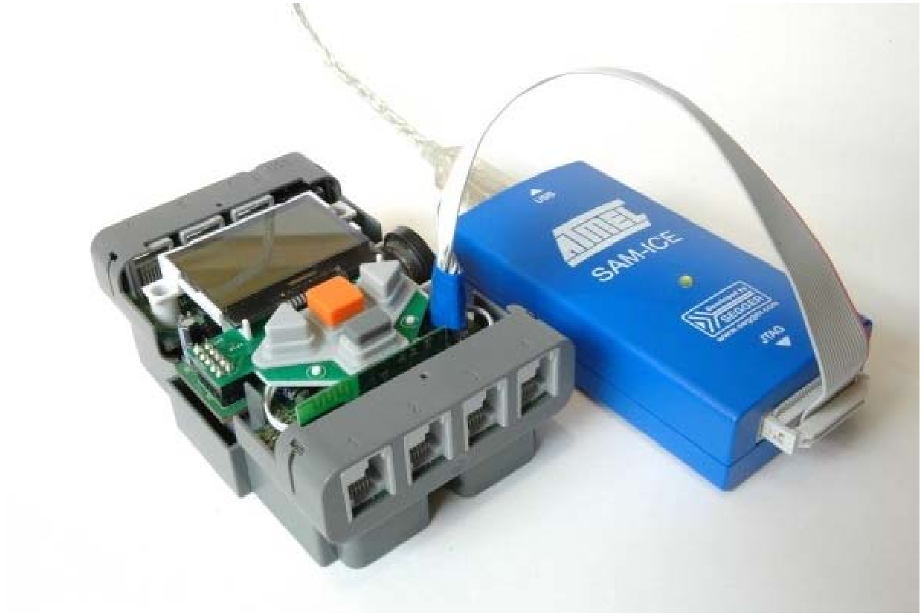
Display

- 100x64 pixel
- ARM 7 via SPI (2MHZ)
- Double Buffering in Firmware



Other

- Sound (PWM by ARM7)
- USB
- Buttons
- JTAG debug (not mounted) for ARM&AVR



AVR <-> ARM

- AVR
  - Power management
  - PWM modulation for engines
  - AD conversion for analogue input ports
  - Buttons
- Exchanged info via internal i2c every 2 ms

ARM to AVR

```
typedef   struct
{
  UBYTE    Power;
  UBYTE    PwmFreq;
  SBYTE    PwmValue[NOS_OF_AVR_OUTPUTS];
  UBYTE    OutputMode;
  UBYTE    InputPower;
}IOTOAVR;
```

AVR to ARM

```
typedef   struct
{
  UWORD    AdValue[NOS_OF_AVR_INPUTS];
  UWORD    Buttons;
  UWORD    Battery;
}IOFROMAVR;
```

Basic Sampling

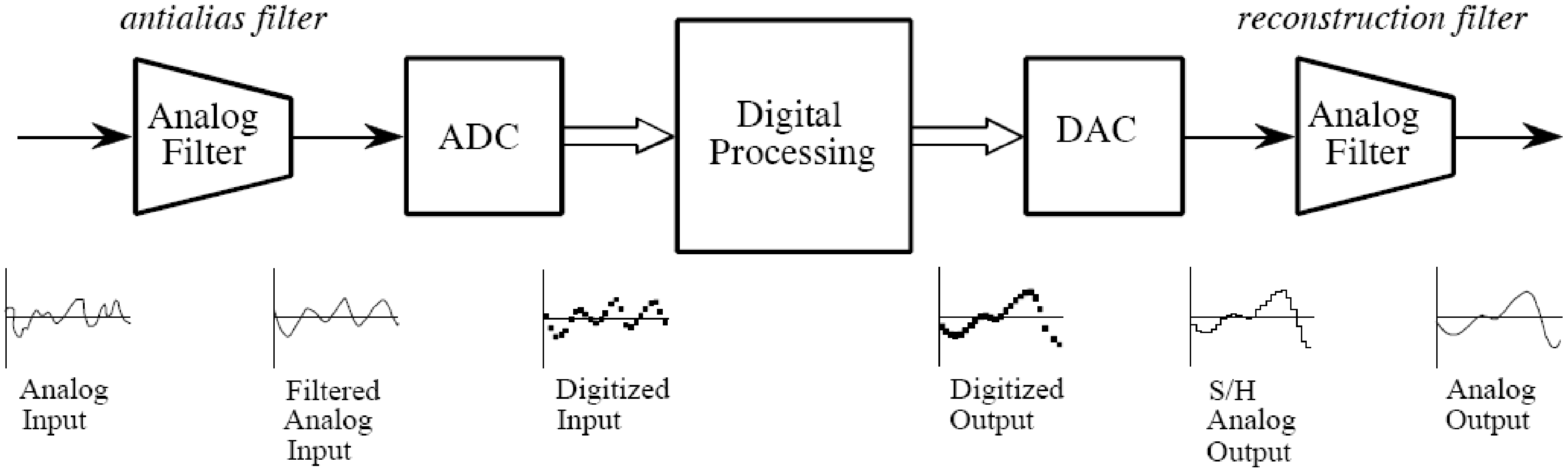
Brian Nielsen

bnielsen@cs.aau.dk

Based on Chapter 3:

"The Scientist and Engineer's Guide to Digital Signal Processing, copyright ©1997-1998 by Steven W. Smith. For more information visit the book's website at: www.DSPguide.com"

AD and DC



antialias filter

Analog Filter

ADC

Digital Processing

DAC

reconstruction filter

Analog Filter

Analog Input

Filtered Analog Input

Digitized Input

Digitized Output

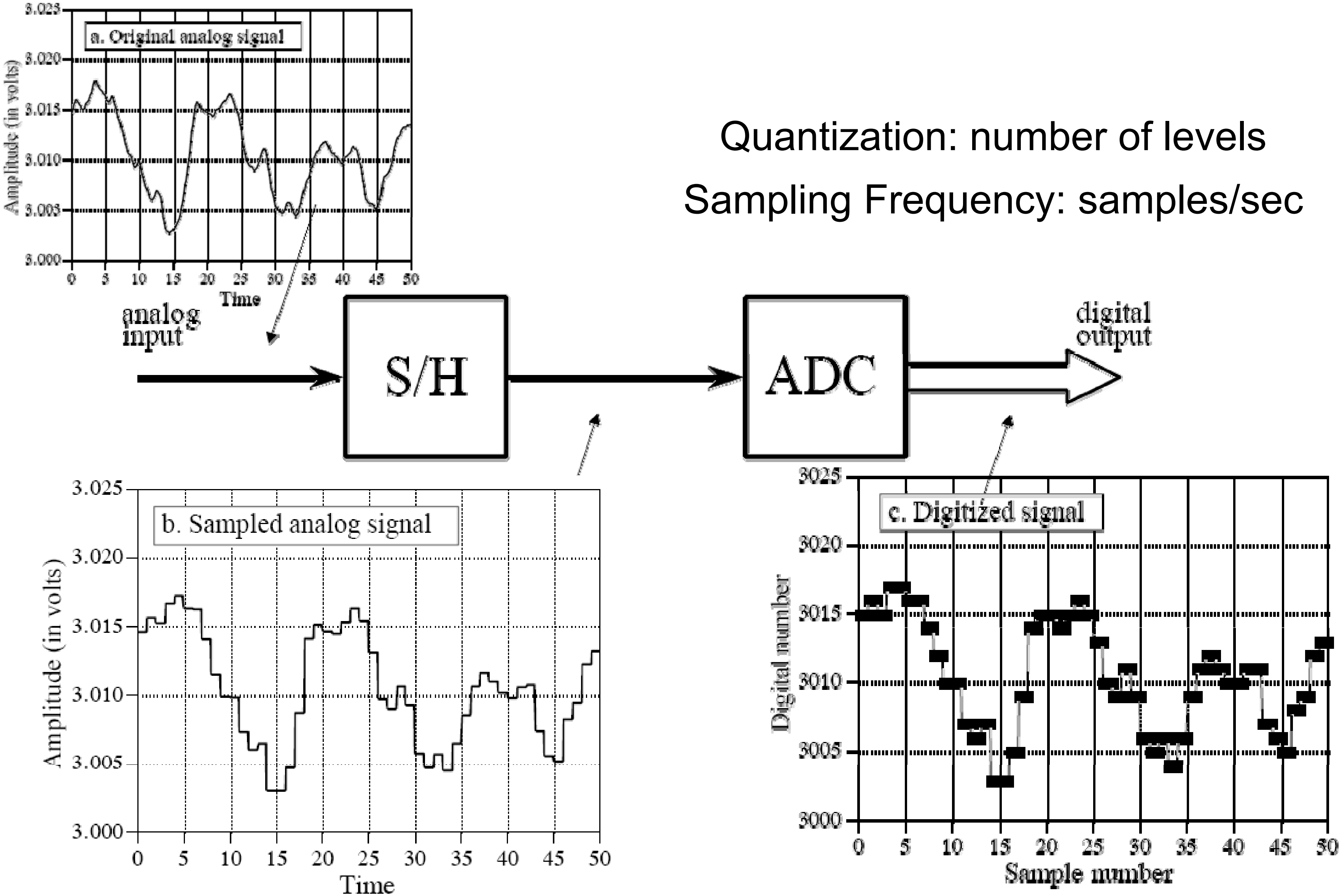
S/H Analog Output

Analog Output

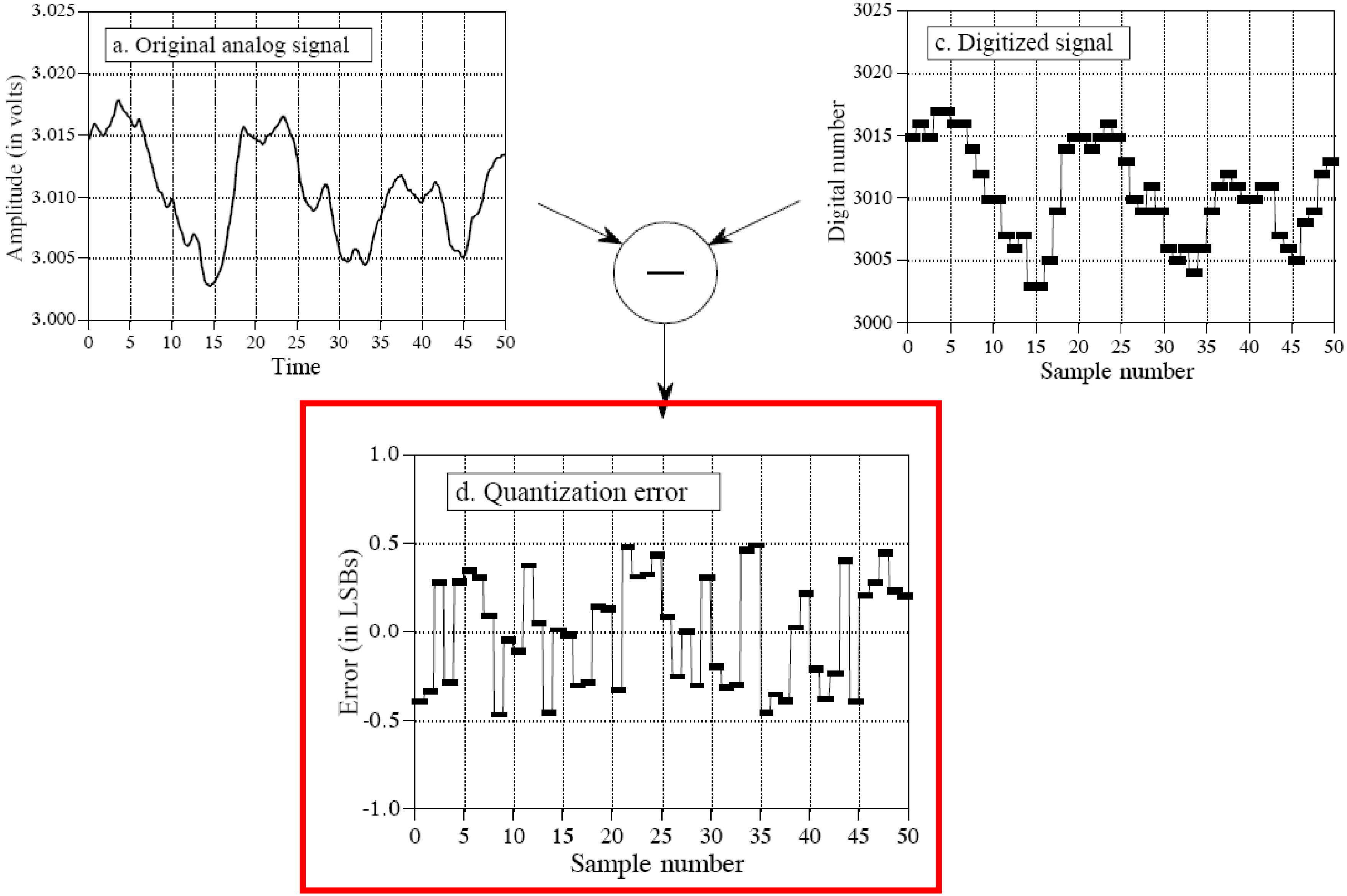
AD Conversion

Quantization: number of levels

Sampling Frequency: samples/sec



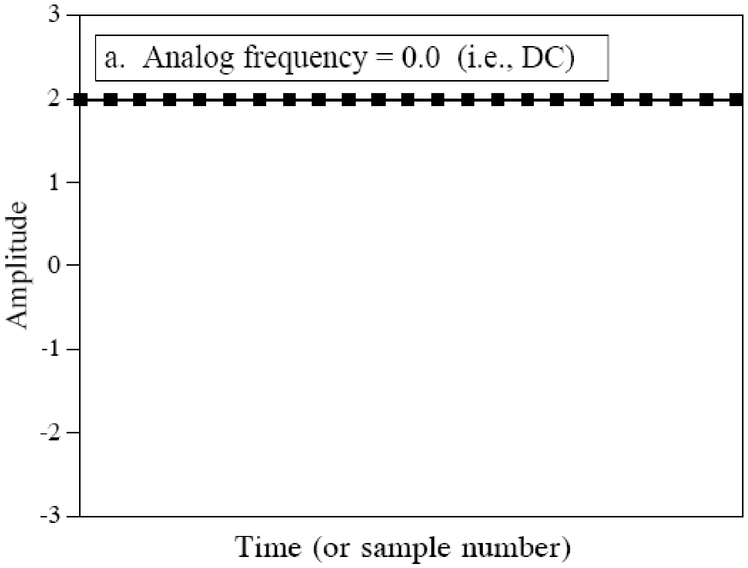
Quantization Error

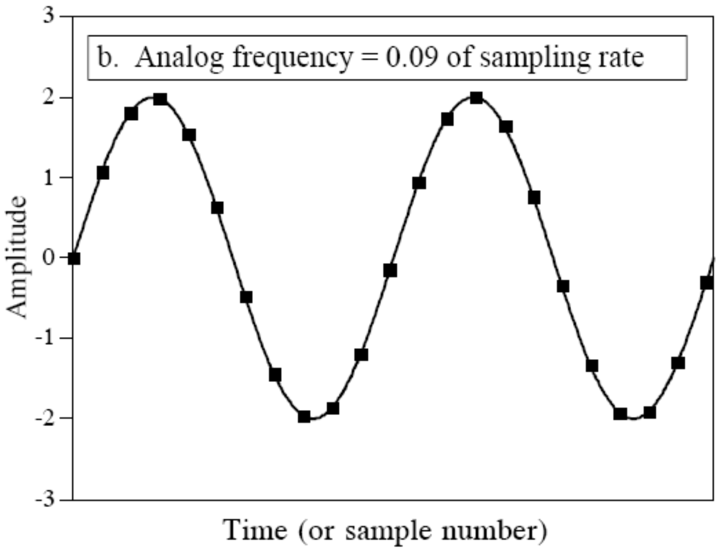


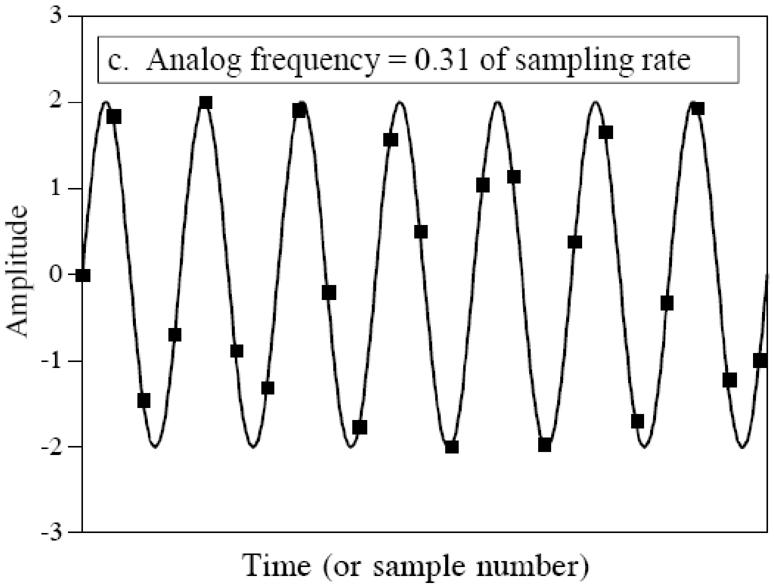
Precision

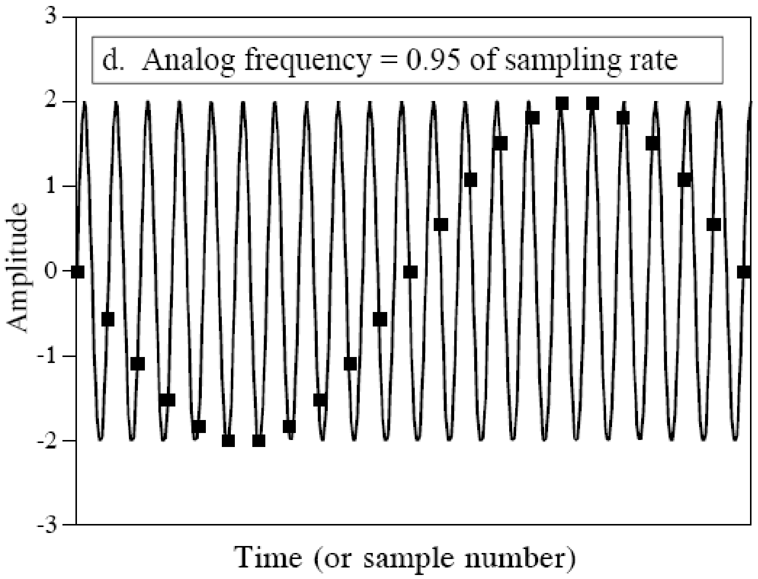
- Q= E_{FSR} / N (if linear)
  - Q is resolution in volts per step (volts per output code),
  - E_{FSR} is the full scale voltage range = $V_H - V_L$,
  - M is the ADC's resolution in bits
  - N is the number of steps (output codes):
    $N = 2^M$
- E.G.
  - Q=(10-0)/2^{12} V/code = 2.44 mV/code

Aliasing


a. Analog frequency = 0.0 (i.e., DC)
3
2
1
0
-1
-2
-3
Amplitude
Time (or sample number)


b. Analog frequency = 0.09 of sampling rate
3
2
1
0
-1
-2
-3
Amplitude
Time (or sample number)


c. Analog frequency = 0.31 of sampling rate
3
2
1
0
-1
-2
-3
Amplitude
Time (or sample number)


d. Analog frequency = 0.95 of sampling rate
3
2
1
0
-1
-2
-3
Amplitude
Time (or sample number)

Sampling Theorem

- To reconstruct the frequency content of a measured signal accurately, the sample rate must be more than twice the highest frequency contained in the measured signal

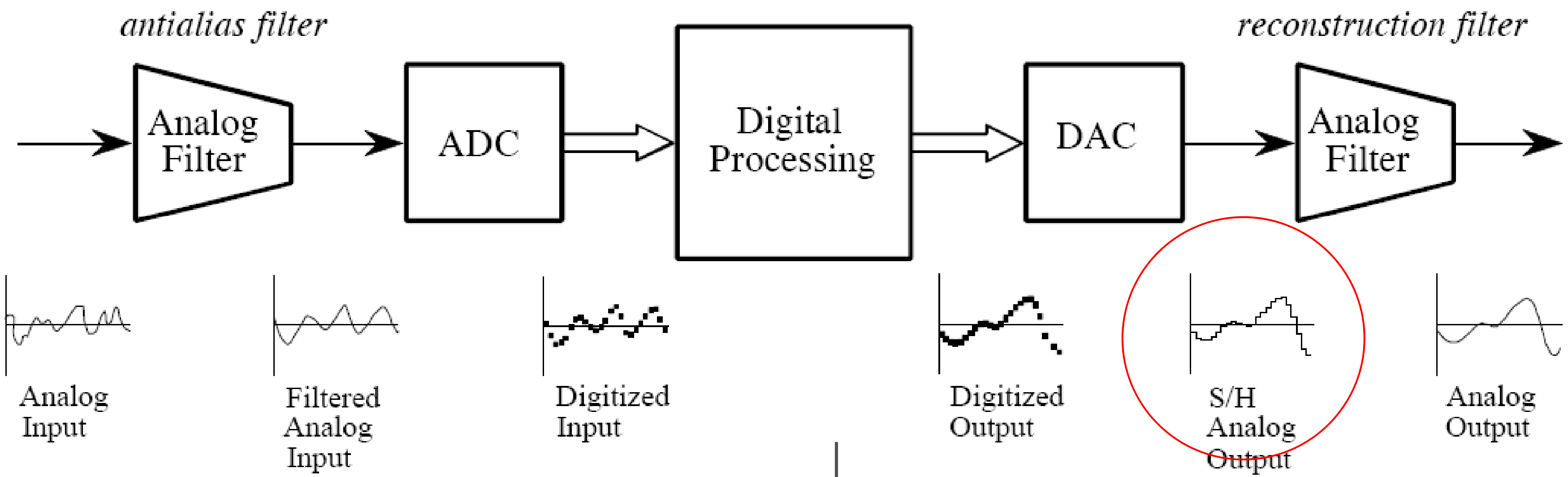
$$f_s > 2f_m$$

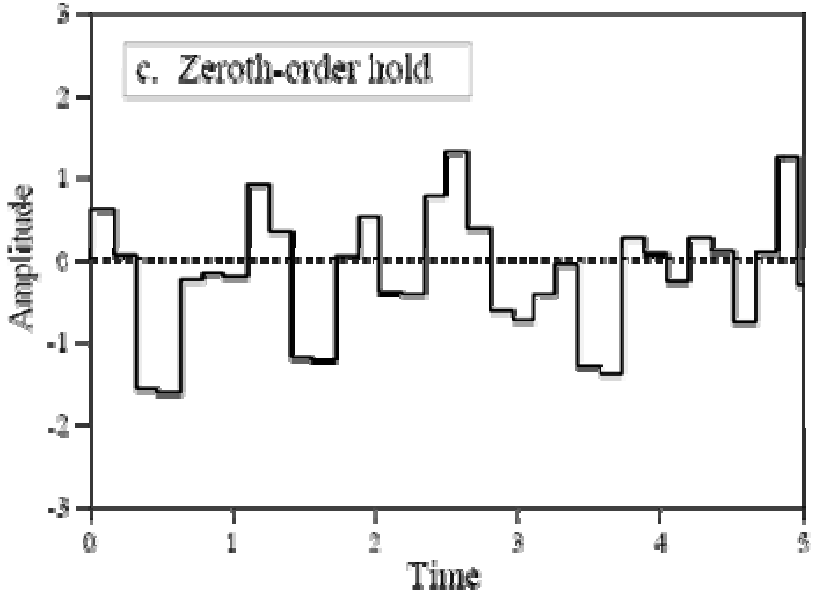
$$\delta t < \frac{1}{2f_m}$$

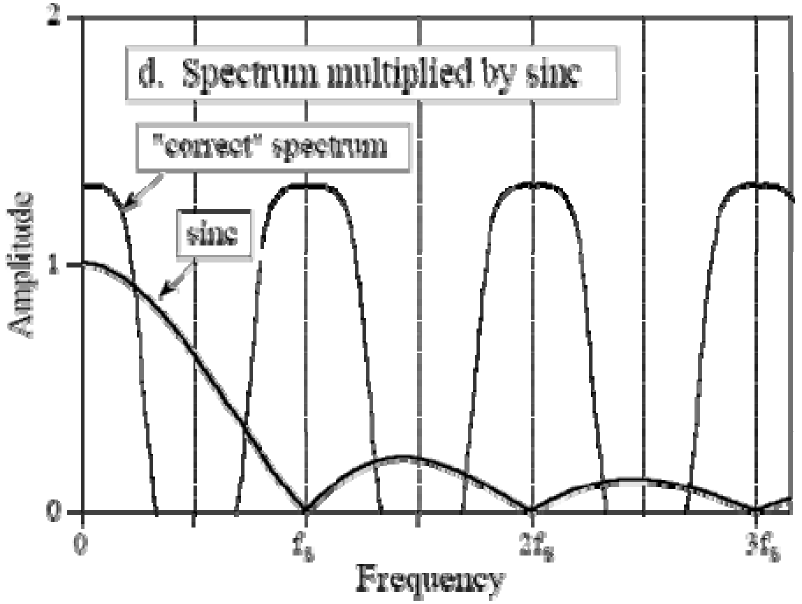
- Nyquist Frequency (half sampling frequency)

$$f_N = \frac{f_s}{2} = \frac{1}{2\delta t}$$

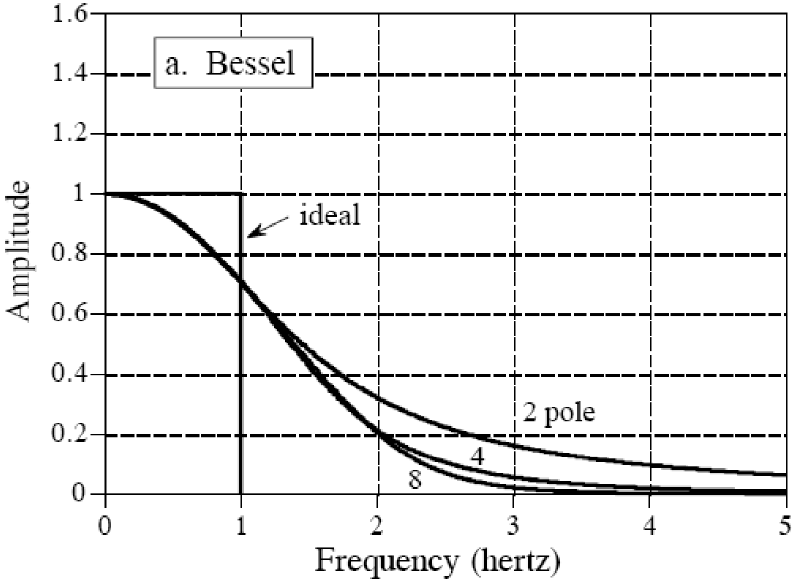
AD/ DC

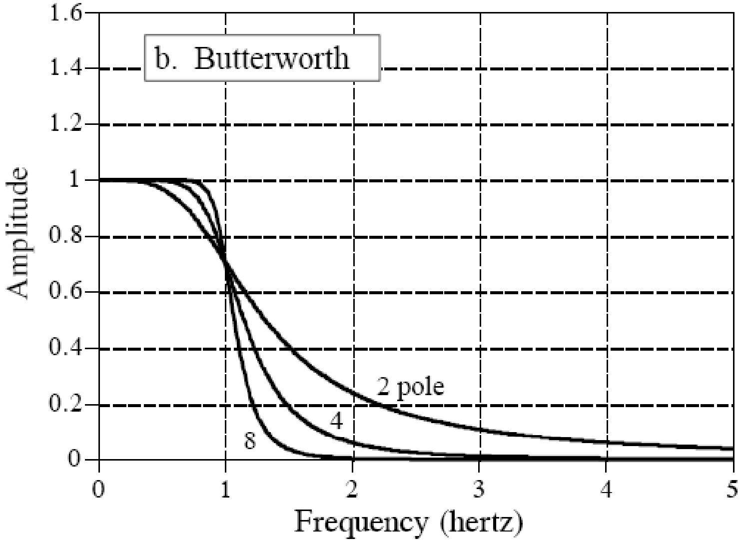

antialias filter
reconstruction filter
Analog Filter
ADC
Digital Processing
DAC
Analog Filter
Analog Input
Filtered Analog Input
Digitized Input
Digitized Output
S/H Analog Output
Analog Output


c. Zeroth-order hold
Amplitude
Time
3
2
1
0
-1
-2
-3
0
1
2
3
4
5


d. Spectrum multiplied by sinc
"correct" spectrum
sinc
Amplitude
Frequency
2
1
0
0
f_s
$2f_s$
$3f_s$

Low-pass filters





- Sharpness
- Attenuation
- Ripple / Over-undershoot