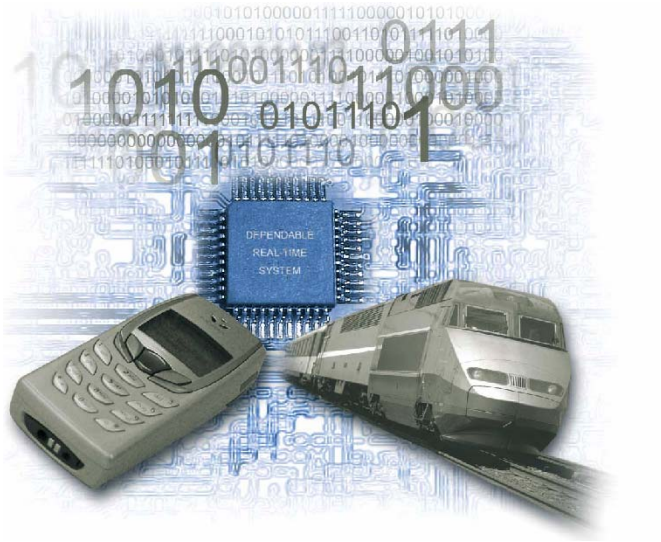
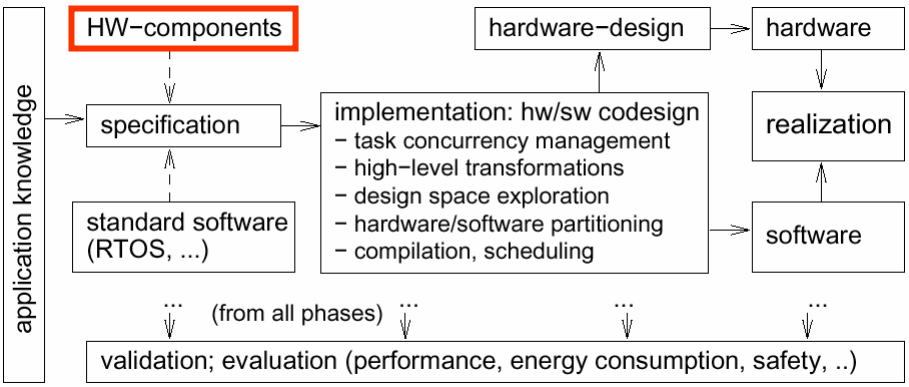
Embedded Systems 11

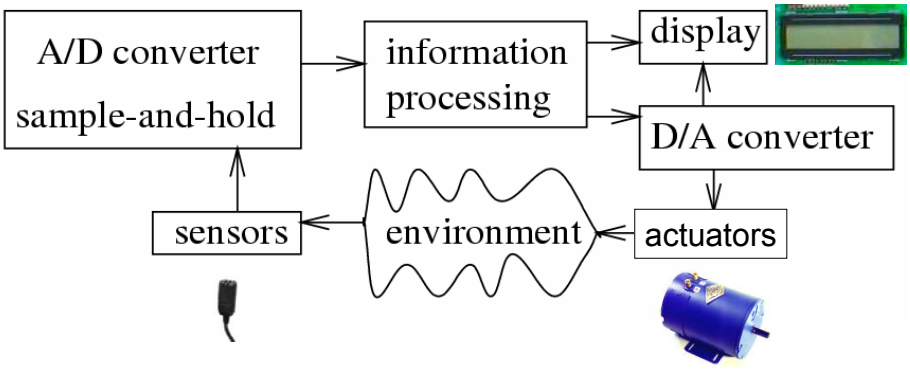


Overview of embedded systems design



Embedded System Hardware

- Embedded system hardware is frequently used in a loop (***„hardware in a loop“***):



Many examples of such loops

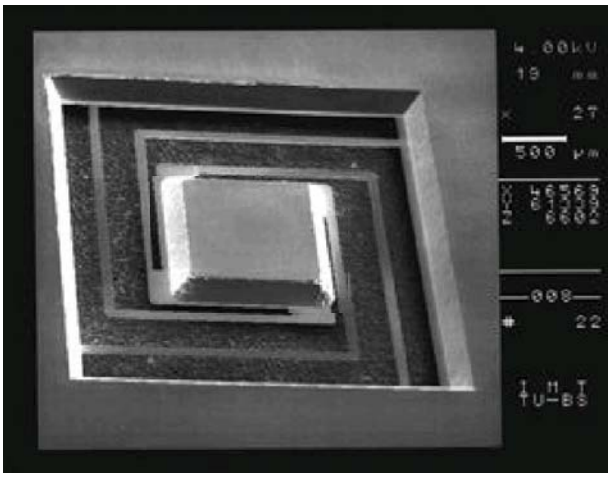
- Heating
- Lights
- Engine control
- Power supply
- …
- Robots

Heating: www.masonsplumbing.co.uk/images/heating.jpg
Robot:: Courtesy and ©: H.Ulbrich, F. Pfeiffer, TU München

Sensors

- Processing of physical data starts with capturing this data.
- Sensors can be designed for virtually every physical and
- chemical quantity
  - including weight, velocity, acceleration, electrical current, voltage, temperatures etc.
  - chemical compounds.
- Many physical effects used for constructing sensors.
- Examples:
  - law of induction (generation of voltages in an electric field),
  - light-electric effects.
- Huge amount of sensors designed in recent years.

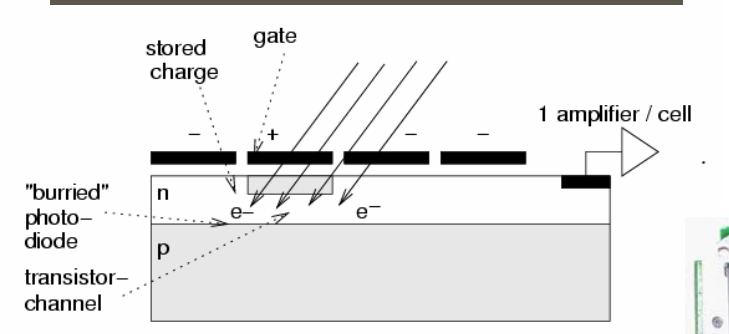
Example: Acceleration Sensor



Courtesy & ©: S. Bütgenbach, TU Braunschweig

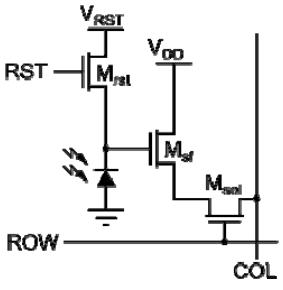
Charge-coupled devices (CCD) image sensors

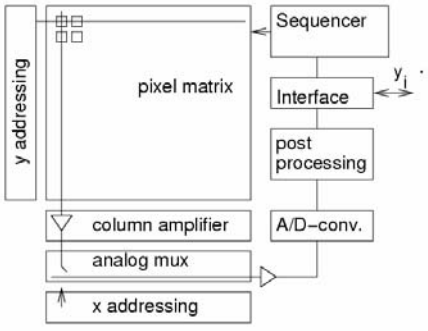
Based on charge transfer to next pixel cell



- Mature technology
- Medium to high-end compact digital cameras

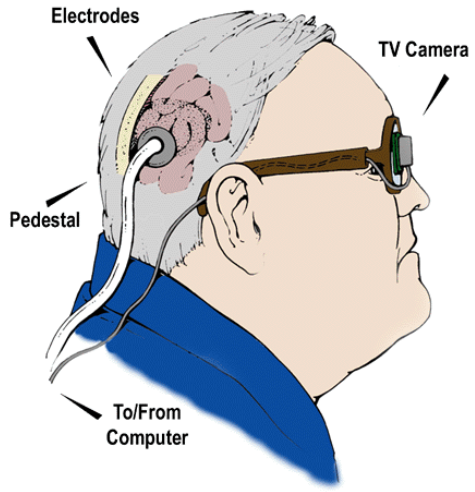
CMOS image sensors



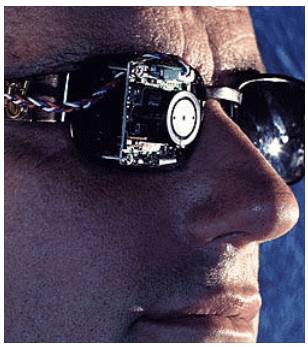


- Lower power consumption
- Lower cost
- Based on standard production process for CMOS chips, allows integration with other components.
- low cost devices
- Automotive
- medical

Artificial eyes

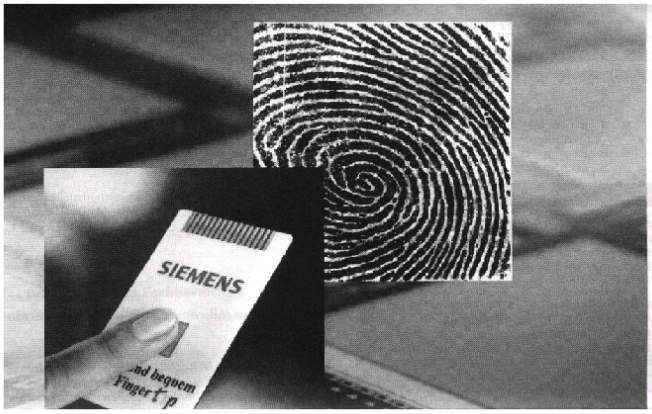


Artificial eyes (2)



Example: Biometrical Sensors

Example: Fingerprint sensor (© Siemens, VDE):



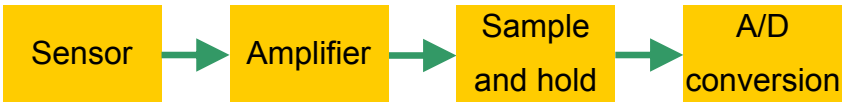
Matrix of 256 x 256 elem. Voltage ~ distance. Resistance also computed. No fooling by photos and wax copies. Carbon dust?

Integrated into ID mouse.

Other sensors

- Rain sensors for wiper control
  („Sensors multiply like rabbits“ [ITT automotive])
- Pressure sensors
- Proximity sensors
- Engine control sensors
- Hall effect sensors

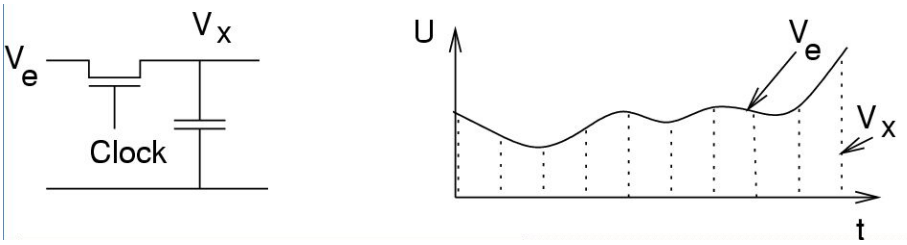
Standard layout of sensor systems for contin. entities



- Sensor: detects/measures entity and converts it to electrical domain
  - May entail ES-controllable actuation: e.g. charge transfer in CCD
- Amplifier: adjusts signal to the dynamic range of the A/D conversion
  - Often dynamically adjustable gain: e.g. ISO settings at digital cameras, input gain for microphones (sound or ultrasound), extremely wide dynamic ranges in seismic data logging
- Sample + hold: samples signal at discrete time instants
- A/D conversion: converts samples to digital domain

Discretization of time

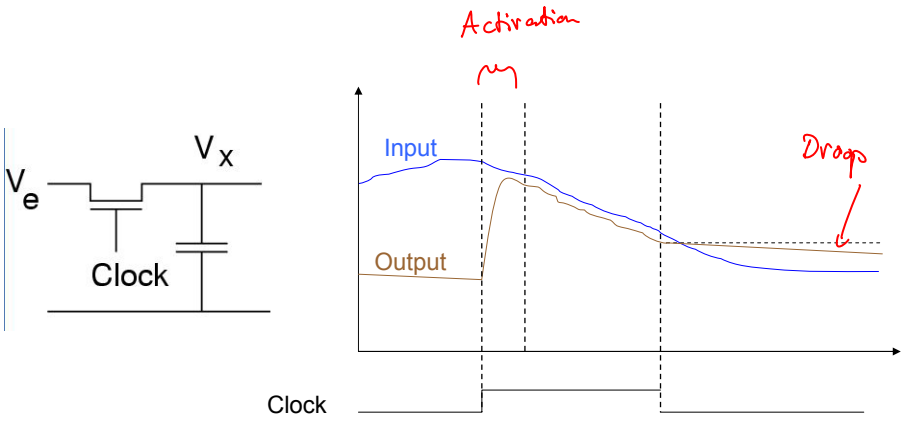
V_e is a mapping $\mathbb{R} \rightarrow \mathbb{R}$



V_x is a **sequence** of values or a mapping $\mathbb{Z} \rightarrow \mathbb{R}$

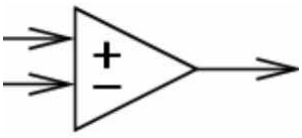
Discrete time: sample and hold-devices.
Ideally: width of clock pulse -> 0

Sample and Hold



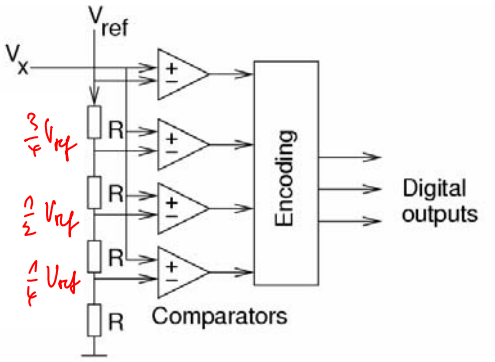
Discretization of values: A/D-converters
1. Flash A/D converter (1)

- Basic element: analog comparator



  - Output = ´1´ if voltage at input + exceeds that at input -.
  - Output = ´0´ if voltage at input - exceeds that at input +.
- Idea:
  - Generate *n* different voltages by voltage divider (resistors), e.g. V_{ref}, ¾ V_{ref}, ½ V_{ref}, ¼ V_{ref}.
  - Use *n* comparators for parallel comparison of input voltage V_x to these voltages.
  - Encoder to compute digital output.

Discretization of values: A/D-converters
1. Flash A/D converter (2)

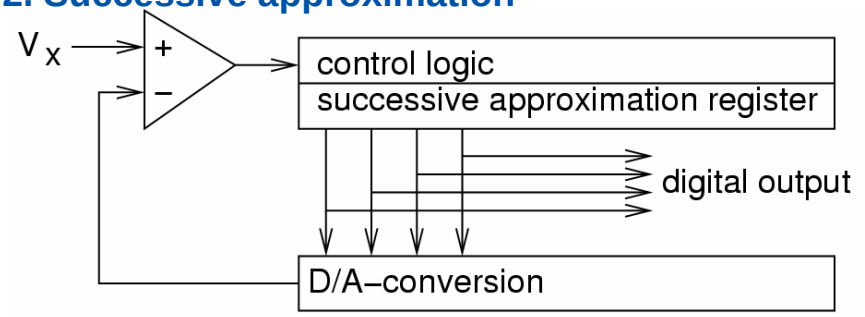


- Parallel comparison with reference voltage
- **Applications**: e.g. in video processing

Speed: O(1)
+ Delay for encoding: O(log m)

Hardware: O(m)
m: # distinguished voltage levels

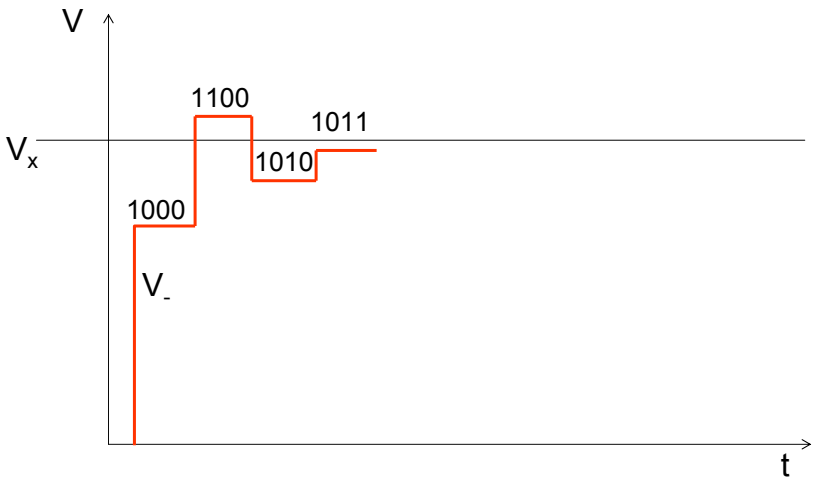
Discretization of values
2. Successive approximation



Key idea: binary search:
- Set MSB='1'
- if too large: reset MSB
- Set MSB-1='1'
- if too large: reset MSB-1

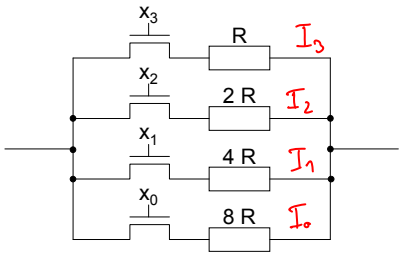
Speed: O(log(m))
Hardware: O(log(m))

Successive approximation (2)



Digital-to-Analog (D/A) Converters

- Convert digital value to conductivity proportional to the digital value



$$I_i = \frac{V_{ref}}{2^{n-1-i} \cdot R} \cdot x_i$$

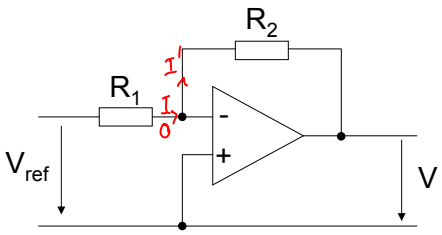
$$I_T = \sum_{i=0}^{n-1} \frac{x_i}{2^{n-1-i} \cdot R} \cdot V_{ref}$$

$$R_T = \frac{1}{\sum_{i=0}^{n-1} \frac{x_i}{2^{n-1-i} \cdot R}}$$

$$\frac{1}{R_T} = \frac{1}{2^{n-1} \cdot R} \underbrace{\sum_{i=0}^{n-1} x_i \cdot 2^i}_{\langle x \rangle} = \frac{1}{2^{n-1} \cdot R} \langle x \rangle$$

Operational amplifier

- Use operational amplifier to convert conductivity to voltage: $V = - V_{ref} R_2 / R_1$



$V + R_2 \cdot I = 0$

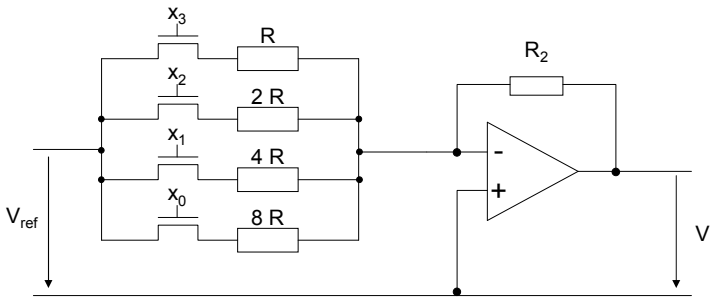
$R_1 \cdot I = V_{ref}$

$\Rightarrow I = \frac{V_{ref}}{R_1}$

$\Rightarrow V + \frac{R_2}{R_1} V_{ref} = 0$

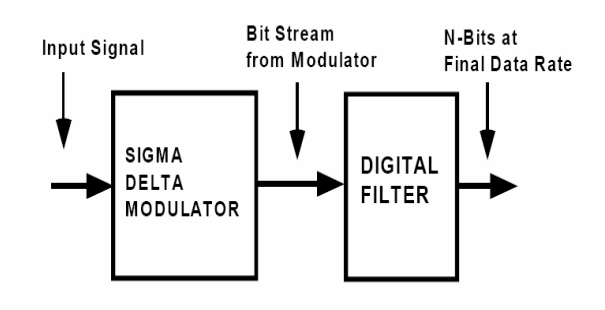
$\Rightarrow V = - V_{ref} \frac{R_2}{R_1}$

Digital-to-Analog (D/A) Converters (3)



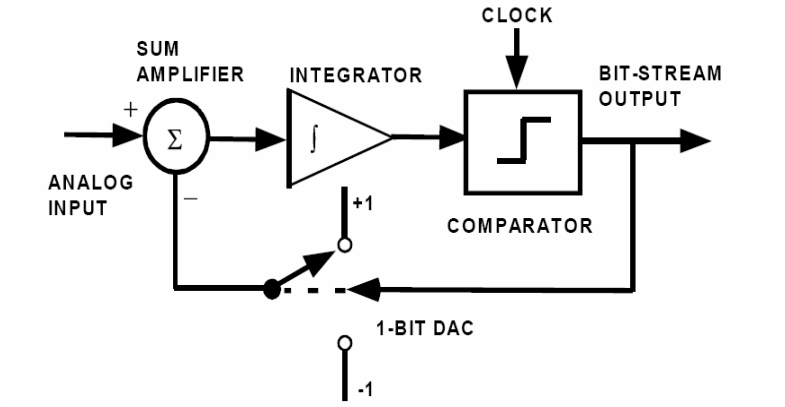
$V = - V_{ref} \cdot \frac{R_2}{R_T} = - V_{ref} \cdot \frac{R_2}{2^{4-1} R}$ <x>

sigma-delta A/D converter



- Modulator generates bit stream whose density of ones (= sliding average value) matches the analog input
- Bit rate many times higher than the final data rate
- Digital filter essentially pursues averaging over sufficiently wide window

1st order sigma-delta modulator



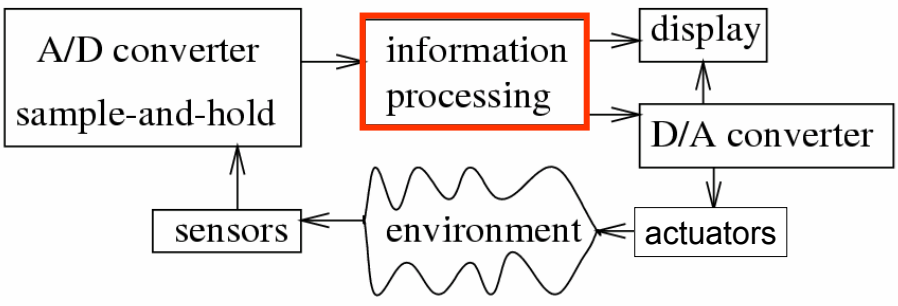
- Generates bit stream whose density of ones (= sliding average) matches the analog input

Actuators and output

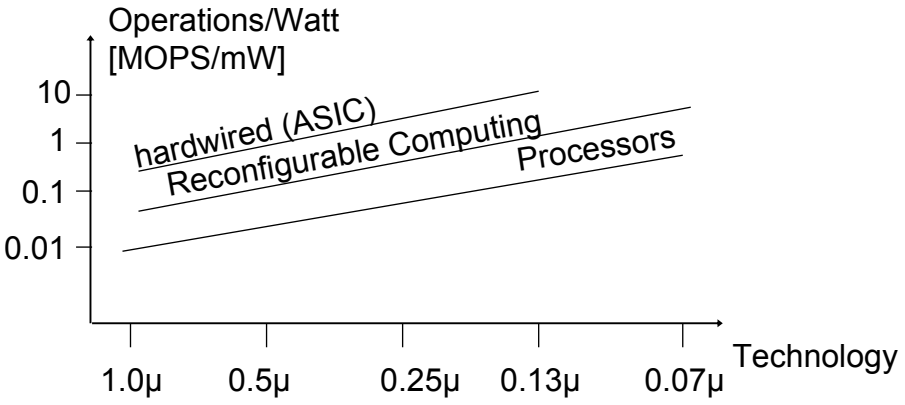
- Huge variety of actuators and outputs, impossible to represent
- Two base types:
  - analogue drive (requires D/A conversion, unless on/off sufficient)
    - CRTs, speakers, electrical motors with collector
    - electromagnetic (e.g., coils) or electrostatic drives
    - piezo drives
  - digital drive (requires amplification only)
    - LEDs
    - stepper motors
    - relais, electromagnetic valve (if actuation slope irrelevant)

Embedded System Hardware

▪ Embedded system hardware is frequently used in a loop (***„hardware in a loop“***):

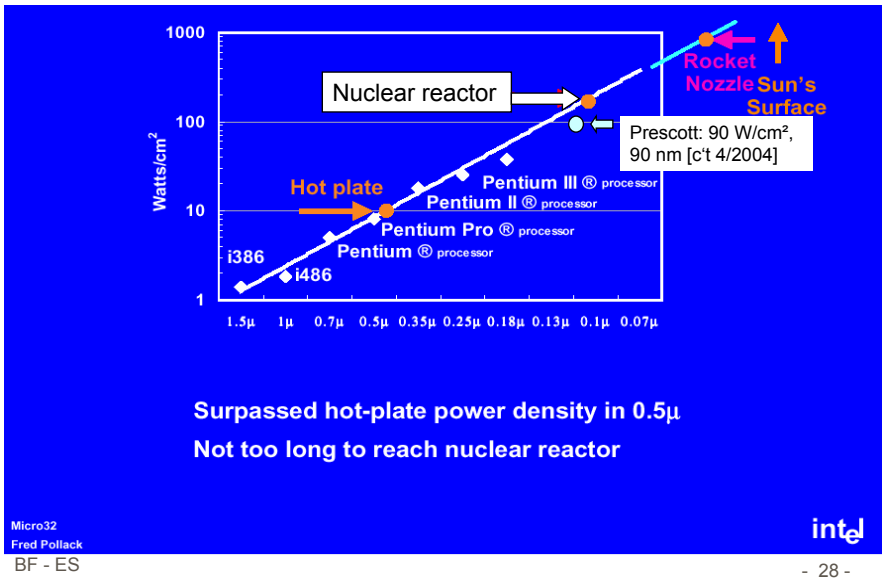


Hardware Efficiency

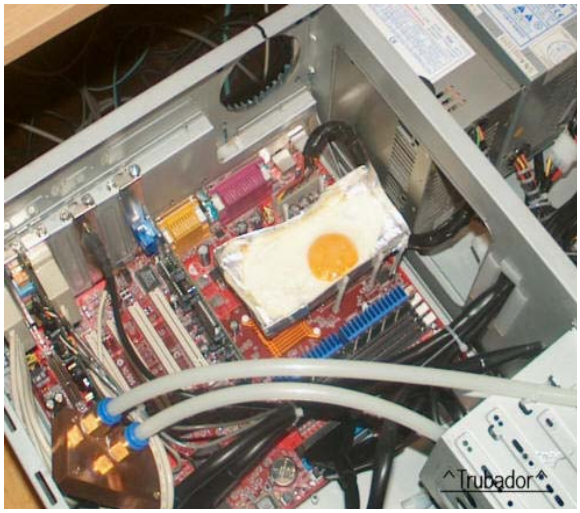


[H. de Man, Keynote, DATE'02;
T. Claasen, ISSCC99]

Power density continues to get worse



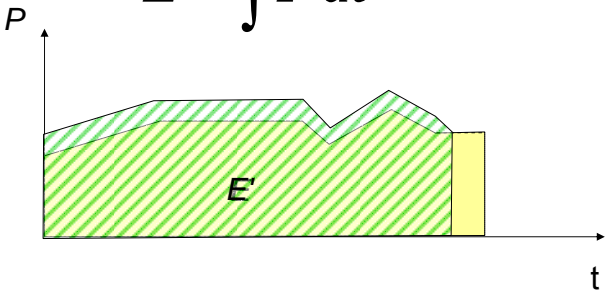
Surpassed hot (kitchen) plate …? Why not use it?



http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html

Power and energy are related to each other

$$E = \int P\,dt$$



In many cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow faster execution.

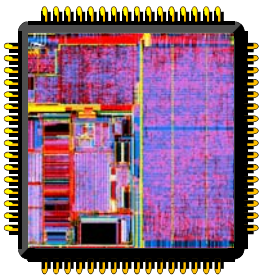
Low Power vs. Low Energy Consumption

- Minimizing the **power consumption** is important for
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - short term cooling

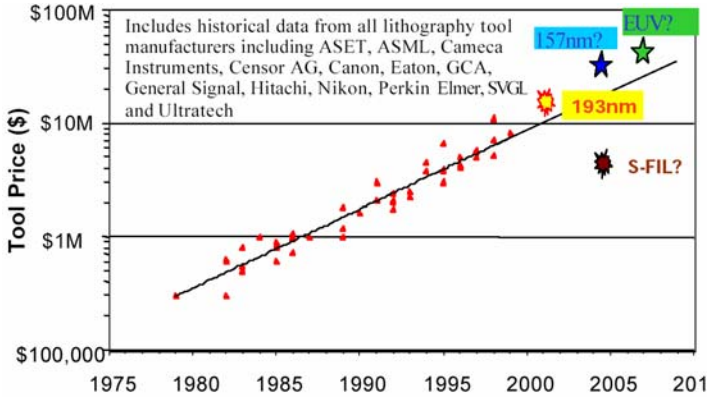
- Minimizing the **energy consumption** is important due to
  - restricted availability of energy (mobile systems)
    - limited battery capacities (only slowly improving)
    - very high costs of energy (solar panels, in space)
  - cooling
    - high costs
    - limited space
  - reliability
  - long lifetimes, low temperatures

Application Specific Circuits (ASICS) or Full Custom Circuits

- Custom-designed circuits necessary
  - if ultimate speed or
  - energy efficiency is the goal and
  - large numbers can be sold.
- Approach suffers from
  - long design times,
  - lack of flexibility (changing standards) and
  - high costs (e.g. Mill. $ mask costs).



Mask cost for specialized HW becomes very expensive



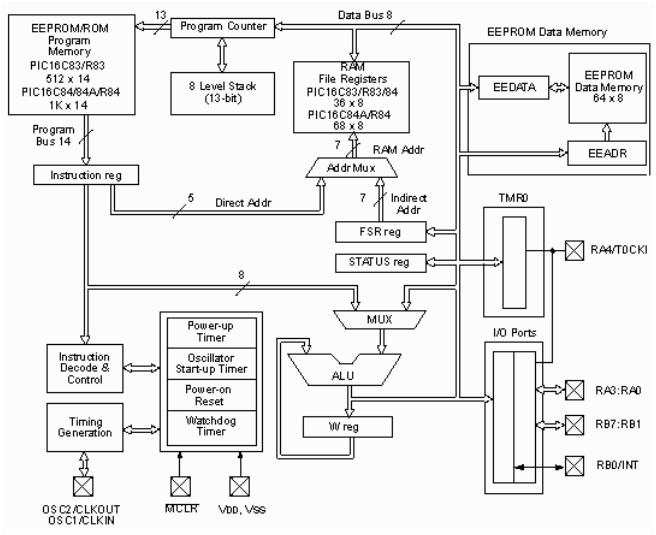
☞Trend towards implementation in Software

[http://www.molecularimprints.com/Technology/tech_articles/MII_COO_NIST_2001.PDF9]

Micro-controllers

- Integrate several components of a microprocessor system onto one chip
  - CPU, Memory, Timer, IO
- Low cost, small packaging
- Easy integration with circuits
- Single-Purpose

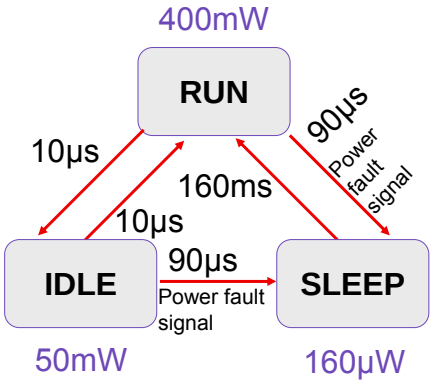
Example: PIC16C8X



Dynamic power management (DPM)

Example: STRONGARM SA1100

- RUN: operational
- IDLE: a sw routine may stop the CPU when not in use, while monitoring interrupts
- SLEEP: Shutdown of on-chip activity



Fundamentals of dynamic voltage scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

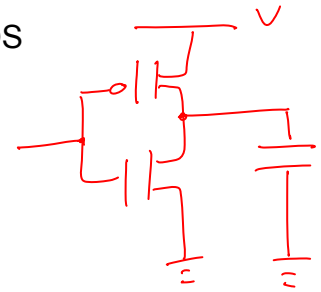
$P = \alpha\, C_L\, V_{dd}^2\, f$ with

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency



Charge at capacitor: $q = V_{dd} \cdot C$

Move q through potential difference ΔV

$\Rightarrow$ Energy potential change by $q \cdot \Delta V$

Fundamentals of dynamic voltage scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

$P = \alpha\, C_L\, V_{dd}^2\, f$ with

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS circuits:

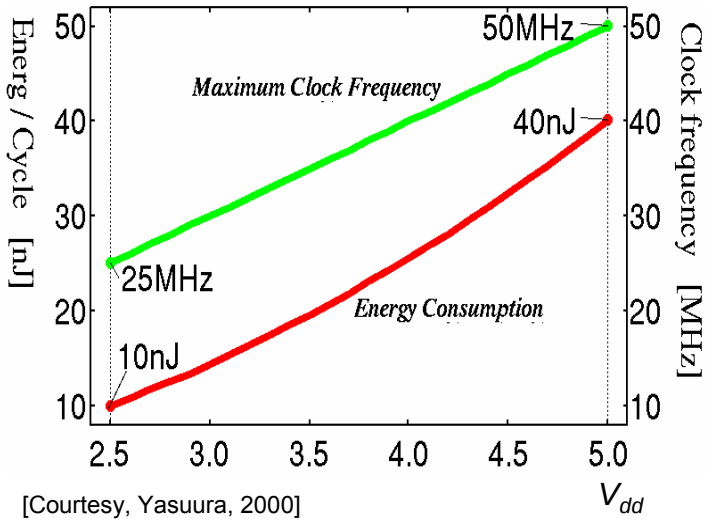
$$\tau = k\, C_L \frac{V_{dd}}{(V_{dd} - V_t)^2} \text{ with}$$

V_t : threshhold voltage

(V_t < than V_{dd})

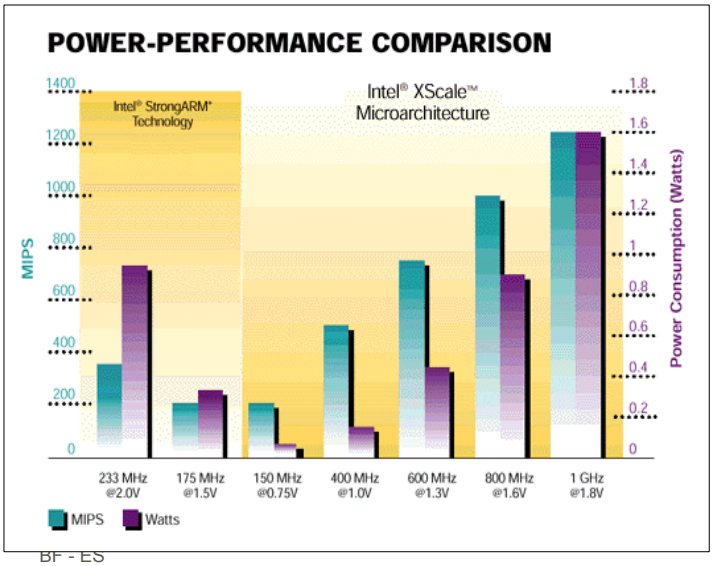
☞Decreasing V_{dd} reduces P quadratically,
while the run-time of algorithms is only linearly increased
E=P x t decreases linearly
(ignoring the effects of the memory system and V_t)

Voltage scaling: Example



[Courtesy, Yasuura, 2000]

Variable-voltage/frequency example: INTEL Xscale

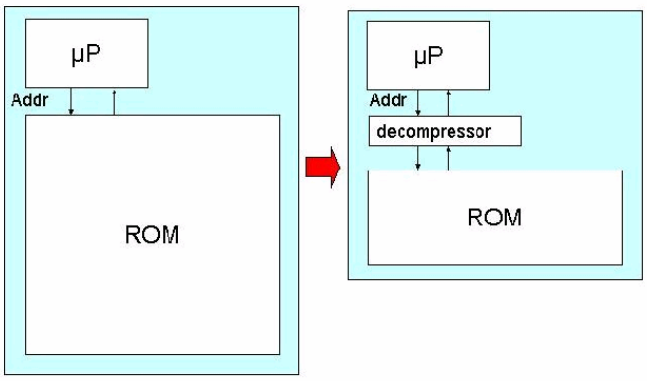


OS should schedule distribution of the energy budget.

From Intel's Web Site

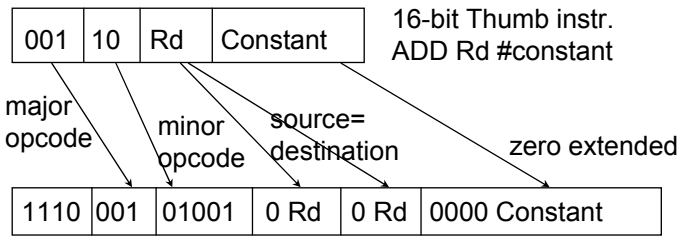
Key requirement #2: Code-size efficiency

- **CISC machines**: RISC machines designed for run-time-, not for code-size-efficiency
- **Compression techniques:** key idea



Code-size efficiency

- **Compression techniques (continued):**
  - 2nd instruction set, e.g. ARM Thumb instruction set:



- Reduction to 65-70 % of original code size
- 130% of ARM performance with 8/16 bit memory
- 85% of ARM performance with 32-bit memory

[ARM, R. Gupta]

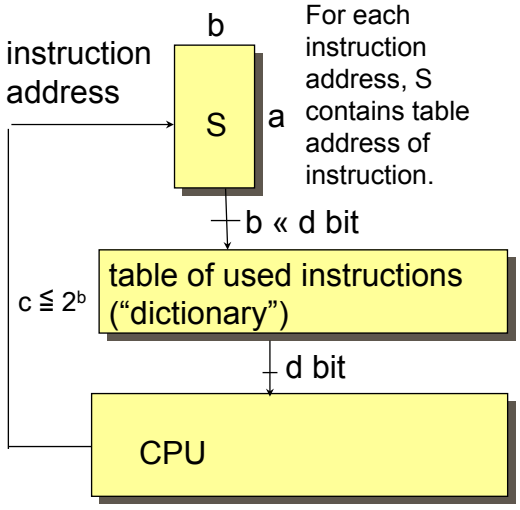
Same approach for LSI TinyRisc, …
Requires support by compiler, assembler etc.

Dictionary approach, two level control store (indirect addressing of instructions)

"*Dictionary-based coding schemes cover a wide range of various coders and compressors.*
Their common feature is that the methods use some kind of a dictionary that contains parts of the input sequence which frequently appear.
The encoded sequence in turn contains references to the dictionary elements rather than containing these over and over."

[Á. Beszédes et al.: Survey of Code size Reduction Methods, Survey of Code-Size Reduction Methods, *ACM Computing Surveys*, Vol. 35, Sept. 2003, pp 223-267]

Key idea (for *d* bit instructions)



Uncompressed storage of *a d*-bit-wide instructions requires *axd* bits.

In compressed code, each instruction pattern is stored only once.

small

Hopefully, *axb+cxd < axd.*

Called nanoprogramming in the Motorola 68000.

Cache-based decompression

- Main idea: decompression whenever cache-lines are fetched from memory.
- Cache lines ↔ variable-sized blocks in memory
  ☞ line address tables (LATs) for translation of instruction addresses into memory addresses.
- Tables may become large and have to be bypassed by a line address translation buffer.

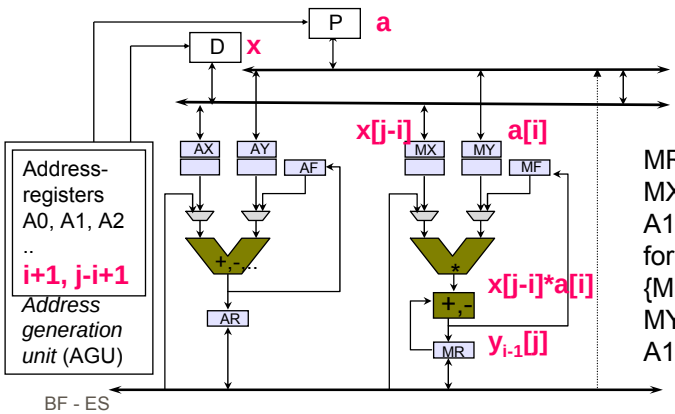
[A. Wolfe, A. Chanin, MICRO-92]

Key requirement #3: Run-time efficiency
- Domain-oriented architectures -

Application: $y[j] = \sum_{i=0}^{n-1} x[j-i]*a[i]$

$\forall i: 0 \le i \le n-1: y_i[j] = y_{i-1}[j] + x[j-i]*a[i]$

Architecture: **Example: Data path ADSP210x**



Application maps nicely onto architecture

```
MR:=0;
MX:=x[n-1]; MY:=a[0];
A1:=1; A2:=n-2;
for ( j:=1 to n)
{MR:=MR+MX*MY;
MY:=a[A1]; MX:=x[A2];
A1++; A2--}
```

Digital Filter:

input: $x = (x_0, x_1, \ldots)$

output: $y = (y_0, y_1, \ldots)$

$$y_i = \sum_{j=0}^{n-1} x_{i-j} * a_j$$

weighted average over the last n sequence elements

Recursively:

$$y_{i,j} = y_{i,j-1} + x_{i-j} * a_j$$

where

$y_{i,-1} = 0$

$y_i = y_{i,n-1}$

DSP-Processors: multiply/accumulate (MAC) and zero-overhead loop (ZOL) instructions

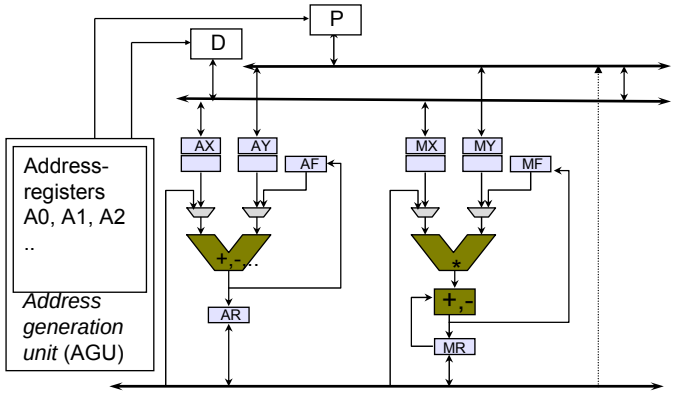
```
MR:=0; A1:=1; A2:=n-2; MX:=x[n-1]; MY:=a[0];
for ( j:=1 to n)
  {MR:=MR+MX*MY; MY:=a[A1]; MX:=x[A2]; A1++; A2--}
```

Multiply/accumulate (MAC) instruction

Zero-overhead loop (ZOL) instruction preceding MAC instruction.
Loop testing done in parallel to MAC operations.

Heterogeneous registers

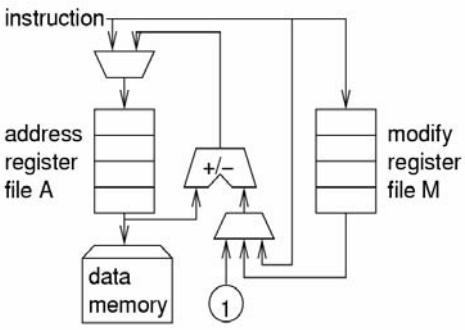
Example (ADSP 210x):



Different functionality of registers An, AX, AY, AF,MX, MY, MF, MR

Separate address generation units (AGUs)

Example (ADSP 210x):



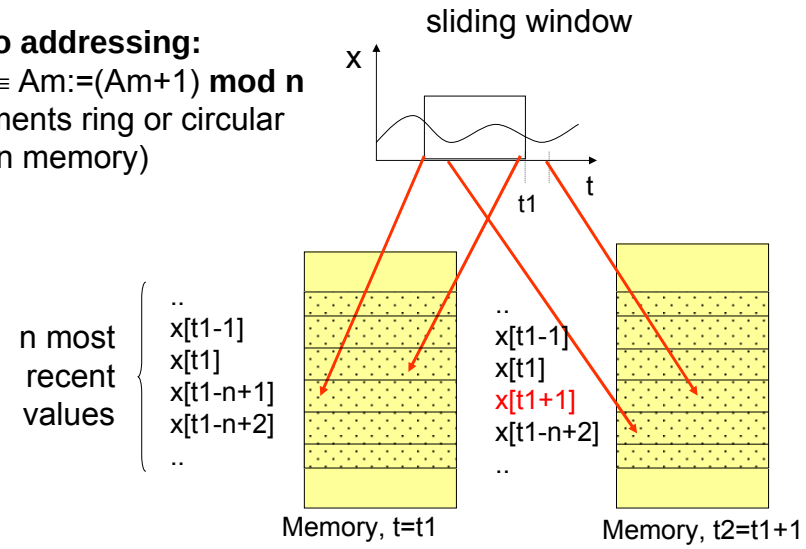
- Data memory can only be fetched with address contained in A,
- but this can be done in parallel with operation in main data path (takes effectively 0 time).
- A := A ± 1 also takes 0 time,
- same for A := A ± M;
- A := <immediate in instruction> requires extra instruction

☞ Minimize load immediates

☞ Optimization in optimization chapter

Modulo addressing

Modulo addressing:
Am++ ≡ Am:=(Am+1) **mod n**
(implements ring or circular buffer in memory)



Saturating arithmetic

- Returns largest/smallest number in case of over/underflows
- Example:

a			0111
b		+	1001
standard wrap around arithmetic			(1)0000
saturating arithmetic			1111
(a+b)/2:	correct		1000
	wrap around arithmetic		0000
	saturating arithmetic + shifted		0111 „almost correct“

- Appropriate for DSP/multimedia applications:
  - No timeliness of results if interrupts are generated for overflows
  - Precise values less important
  - Wrap around arithmetic would be worse.