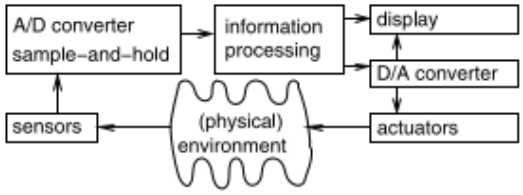
Functional Safety of Embedded Systems - Part 3 - Embedded Systems Hardware

Introduction

- software and hardware important
- reuse of available hard-/software components => platform-based design methodology
- much less standardized than PC-hardware
- hardware-loop (especially fit for control systems, other systems need to evolve)



- discrete (input/ sensor) values can be created by special circuits:
  - *sample-and-hold*
  - *analog-to-digital* (A/D)
- actuators generally analog => need for conversion

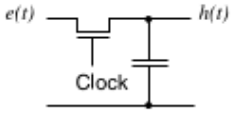
Input

- Sensors for any quantity (weight, velocity, acceleration, electrical current, voltage, temperature, ...), physical phenomena can be used to advantage
  - *Acceleration Sensors*: small mass, that is displaced from default position, changing the resistance of tiny wires
  - *Rain Sensors*: control speed of wipers automatically
  - *Image Sensors* (small arrays of light sensors)
    - *CCD* (charged-coupled device): optimized for optical applications
      - charges are passed from one pixel to the next and read at array boundary (sequential charge) => interface rather complex
    - *CMOS*: use standard CMOS technology for integrated circuits => sensors and logic can be on the same chip => cheap
    - Selecting optimal type not easy, CMOS quality improving, but still less power efficient.
    - Smart sensors => CMOS (integrated logic) => compact cameras
  - *Biometric Sensors*: => to protect mobile/ removable equipment (smart cards, biometric/ biomedical authentication) => authenticate a person *or* check, if the person is who it claims to be
    - iris scan, fingerprint, face recognition
    - exact matches are not possible, also real person might be denied
  - *Artificial Eyes*: affect the eye/ provide vision indirectly
  - *RFID* (Radio Frequency Identification): response of a *tag* to radio frequency signals
    - *tag*: consists of integrated circuit + antenna
    - *RFID readers*: maximum distance to tag depends on type of tag
  - Other sensors: pressure, proximity, engine control, Hall effect, ...

Sensors generate **signals**

Discretization of time: Sample-and-hold circuits

- Computers work in discrete time domain D_T
- Computers can process *streams* of values (discrete sequences)



  - Every time the clock closes the swith, the capacitor is charged to the incoming voltage

- Only the times, at which a sample is taken, can be used later.
    - Capacitor needs some time to charge up => voltage at time x will correspond to time windows x
    - => need to be able to reconstruct original signal (p is period [Oppenheim et al, 2009]:

$$e'_k(t) = \sum_{k=1,3,5,\ldots}^{K} \left(\frac{4}{\pi k} \sin\left(\frac{2\pi k t}{p} \right) \right)$$

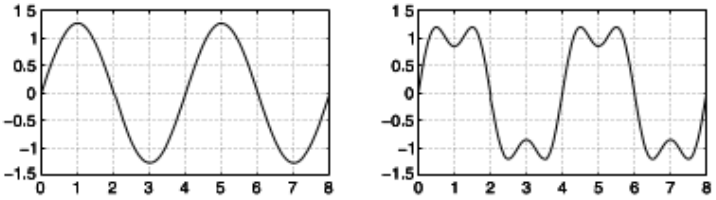


Figure 3.5. Approximation of a square wave by sine waves for K=1 (left) and K=3 (right)

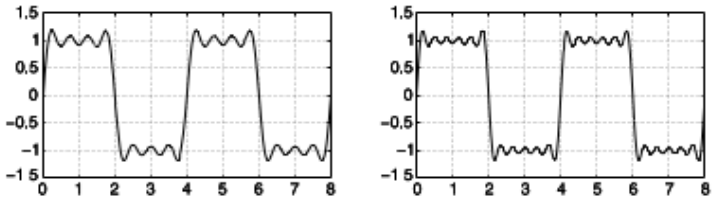
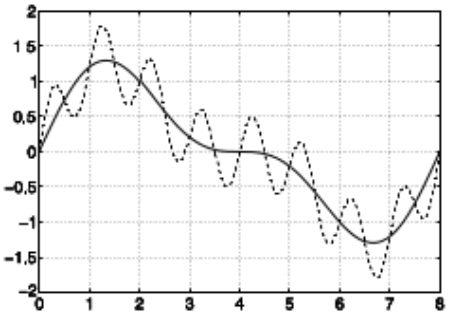


Figure 3.6. Approximation of a square wave by sine waves for K=7 (left) and K=11 (right)

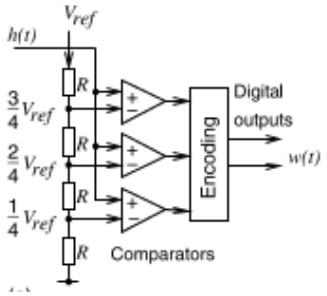
- transformation T_r is *linear*, if for two signals $e_1(t), e_2(t)$: $T_r(e_1 + e_2) = T_r(e_1) + T_r(e_2)$
    - functions may have the same approximation, if measured at the wrong times (aliasing):
      $e_3(t) = \sin\left(\frac{2\pi t}{8}\right) + 0.5 \sin(frac2\pi t4)$
    - $e_4(t) = \sin\left(\frac{2\pi t}{8}\right) + 0.5 \sin(frac2\pi t4) + 0.5 \sin\left(\frac{2\pi t}{1}\right)$

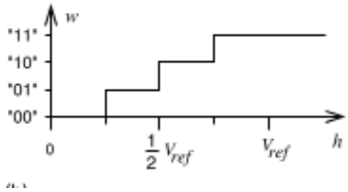


    - => reconstruction of original unsampled signal is not possible, unless additional knowledge is given
    - aliasing is avoided, if incoming frequencies is less than half of the sampling frequency (=> **Nyquist sampling criterion**)
    - anti-aliasing filter: low-pass-filter

Discretization of values: A/D-converter

- map from continuous value domain to discrete value domain => A/D-converters
- large range of A/D converters, presented are two extreme cases:
    - *Flash A/D converter*: large number of comparators
        - *comparator:* two inputs (+, -), if the voltage at + exceeds that at - the ouput is a logical 1, else it's a 0

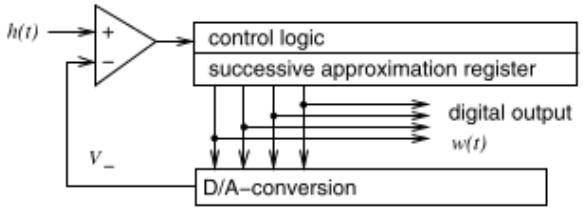




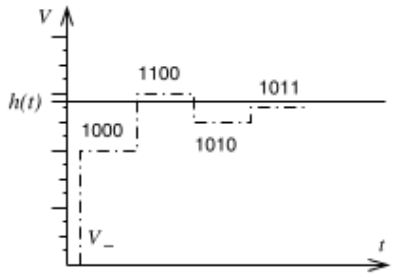
- converting negative numbers (two's complement) requires extensions
- *resolution*: characterization for A/D-converters:
  - bits produced by a converter
  - resolution in volts per step

    $Q = \frac{V_{FSR}}{n}$, V_{FSR} : difference between largest and smallest voltage, n : number of voltage intervals
- advantage of flash-converter: speed (good for video)
- disadvantage: complex hardware ($n-1$ comparators needed)

- *Successive Approximation*: Distinguish between a large number of digital values

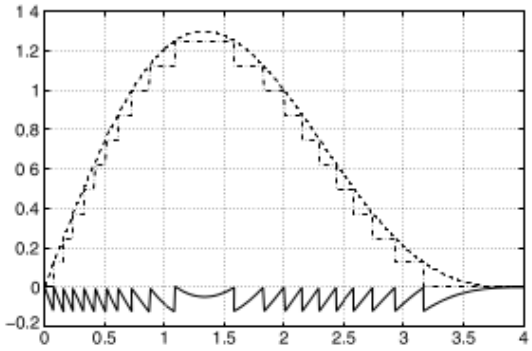


- uses binary search, most significant bit is set to 1 until signal is exceeded, current bit is set to 0, set next to 1, ...



- measured value needs to be constant during operation => Sample-and-Hold-circuit
- advantage: hardware efficiency
- disadvantage: slow log(n) steps

- quantization noise: $noise(t) = w(t) - h(t)$, $h(t)$: original, $w(t)$ (square): digitalization



- signal to noise ratio

$$SNR = 10 \cdot \log \frac{\text{power of "'useful"' signal}}{\text{power of the noise}} = 20 \cdot \log \frac{\text{voltage of "'useful"' signal}}{\text{voltage of the noise}}$$

- measured in dB (decibel), logarithmic, dimensionless

- several type of other A/D-converters, differ by speed and precision
- Techniques for automatically selecting the most appropriate converter exist.

Processing Units

Overview

- embedded systems use electrical energy => *consumed energy*
- power and energy efficiency mucho importante (=> also generally => green computing)
- *ASIC* (application-specific integrated circuits): hardwired multiplexed design, reconfigurable logic, programmable processors
- energy efficiency => cannot be flexible, flexibility => cannot be energy efficient
- power and energy: $E = \int_0^t P(t)dt$
    - systems with more power consumption may be more energy efficient, if they are also faster
    - more power consumptions means more heating => need for cooling

Application Specific Integrated Circuit (ASIC)

- Cost of designing and manufacturing ASICs is quite high => may only be feasible for large quantities
- Design of ASICs is time consuming. Correcting faulty design is hard and therefore expensive
- => ASICs only, if maximum energy efficiency is needed & large number of systems can be sold (or the price is not as important)

Processors

- Processors are programmable => very flexible. Behavior of whole system can be changed with change of software.
- Embedded processors don't need instruction set compatibility like PCs do.

Energy Efficiency

- optimize architectures for energy-efficiency => need to take care, that efficiency of software (generation) is not compromised
- *gate clocking*: parts of processors are disconnected from clock during idle periods
- maybe rid parts of the processor from the clock altogether
    - *globally synchronous, locally asynchronous processors*
    - *globally asynchronous, locally synchronous processors* (GALS)
- *Dynamic Power Management* (DPM): processors have power saving states in addition to operating states (e.g. *run*, *idle*, *sleep*, sleep is powered down, long time returning to run)
- *Dynamic Voltage Scaling* (DVS):
    - Power consumption of CMOS changes quadratically with supplied voltage. (maximum clock frequency is function of supply voltage)
    - Run-time of algorithms is linearly dependent on supply voltage.

Code-Size Efficiency

- Memory very limited => take as little space as possible. (Even more important for *System on a Chip* (SoC), memory => embedded memory, more expensive than regular memory)
- RISC ("normal processors") => designed for speed
- CISC ("older processors", early complex instruction set processors) => designed for code-efficiency (slow memory, no/little cache)
- *Compression Techniques* => reduce amount of silicon + reduce energy needed to fetch information => faster transmission
    - hardware decoder (hopefully small, fast and efficient)
    - save ROM and RAM
    - no run-time penalty
    - decoding of limited context should be possible (don't decode everything to find something)
    - branching into arbitrary addresses must be possible
- *second instruction set*: (e.g. ARM) smaller instruction set called *THUMB* (may increase software development cost, as compiler need to know second instruction set)
- *dictionaries*: each instruction is only stored once, lookup-table for reference of instructions (*nanoprogramming*, *procedure ex-lining*)

Run-Time Efficiency

- Architectures can be customized for certain application domains. => more efficient processors without need for higher clock frequencies.
- *digital signal processing* (DSP): Add filter operations as hardware blocks. Single instruction triggers the chip to compute something more complex. => sinlge instruction realizations of loop bodies

Digital Signal Processing

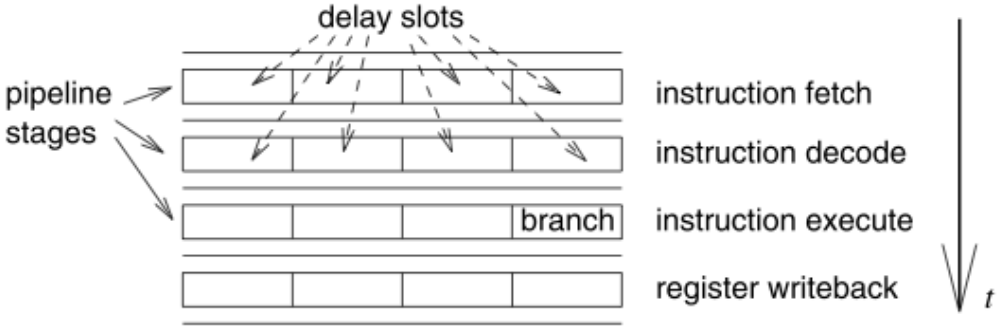
- *Specialized Addressing Modes*: ring buffers, modulo addressing
- *Separate Address Generation Units* (AGU): directly connected to address input of memory
- *Saturation Arithmetic*: changes the way over- and underflow are managed: try to return a result, which is as close as possible to actual result
  - *overflow*: largest possible result
  - *underflow*: slowest possible result
- *Fixed-Point Arithmetic*:
  - 80% of DSP processors don't include Floating Point hardware
- *Real-Time Capability*: Some features in PCs improve average execution time, but at the same time worsen the worst-case execution time. E.g. embedded processors often don't have caches, virtual addressing, demand paging
- *Multiple Memory Banks or Memories*: DSP processors often have two memory banks.
- *Heterogeneous Register Files*: see filter application
- *Multiply/ Accumulate Instructions*: instructions, that can perform multiplication followed by addition.

Multimedia Processors/ Instruction Sets

- Registers/Arithmetic Units often 64 bits => two doubles, four words, 8 bytes can be stored
- *SIMD* (single instruction, multiple data): suppress carry bits at boundaries, write into next object => more efficient overflow handling
  - Has been added to several processors (streaming SIMD extension SSE, short vector instructions)

Very Long Instruction Word (VLIW) processors

- Computational demands increase (especially multimedia, advanced coding techniques, cryptography)
  - No instruction set compatibility needed => instructions explicitly identify parallelism => *explicitly parallel instruction computing* (EPIC)
    - detection of parallelism moved to compiler
    - *very long instruction set words* (VLIW):
      - several operations in instruction word/*instruction packet* => execution in parallel
      - each instruction encoded in separate field => each field controls one hardware component
      - fields may be unused, if not enough parallelism
      - => flexible packet size => note really VLIW, but still EPIC
  - many parallel register accesses due to many parallel processes => partitioned register files => functional unit is only connected to subset of register files
  - potential problem: *delay penalty* (due to branch instructions: additional instructions have already entered the pipeline, before branch instructions enter it)



  - 1. fill *delay slots* with useful stuff, that needs to be executed before branch, some must be filled with *no-operation instructions* (NOP), *branch delay penalty* denotes the loss of performance from NOPs
    2. pipeline is stalled until branch has finished
- *predication* is used to predict, if small if-statements are true and thus executed or NOPs

Micro-Controllers

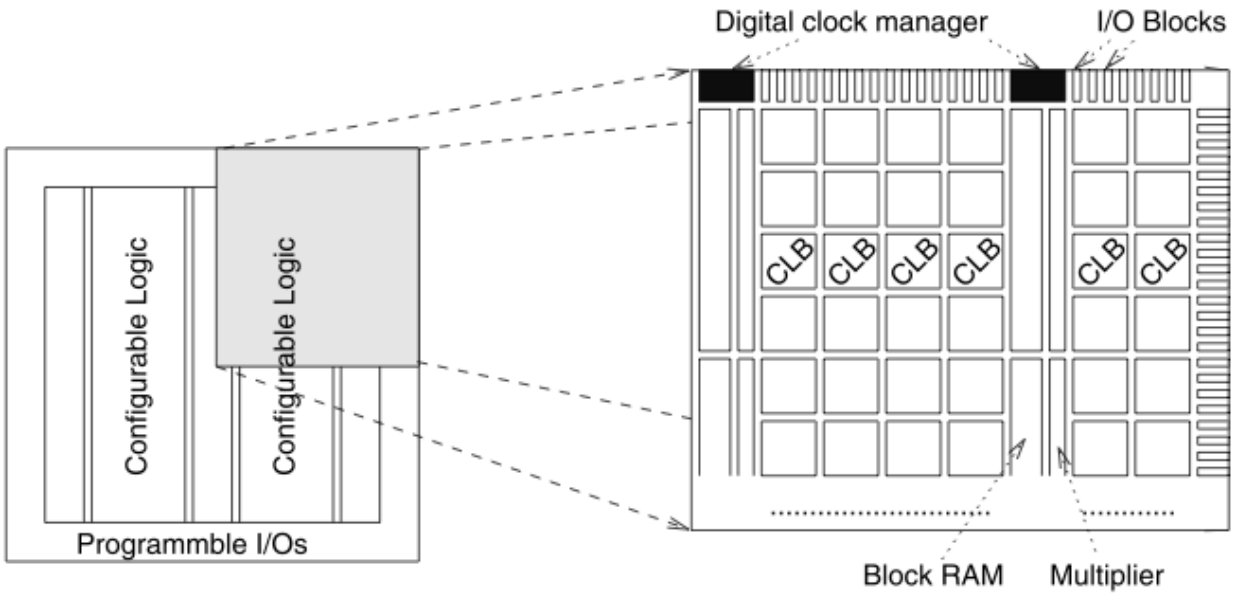
- not complex, example: Intel 8051 (quite typical)
- 8 bit cpu, optimized for control applications
- large set of operations on boolean data types
- program address space of 64kb
- separate data space of 64kb
- 4 kb of program memory (on chip), 128 b of data memory (on chip)
- 31 I/O lines, each addressable individually
- 2 counters (on chip)
- universal asynchronous receiver/ transmitter for serial lines available (on chip)
- clock generation (on chip)
- many variations

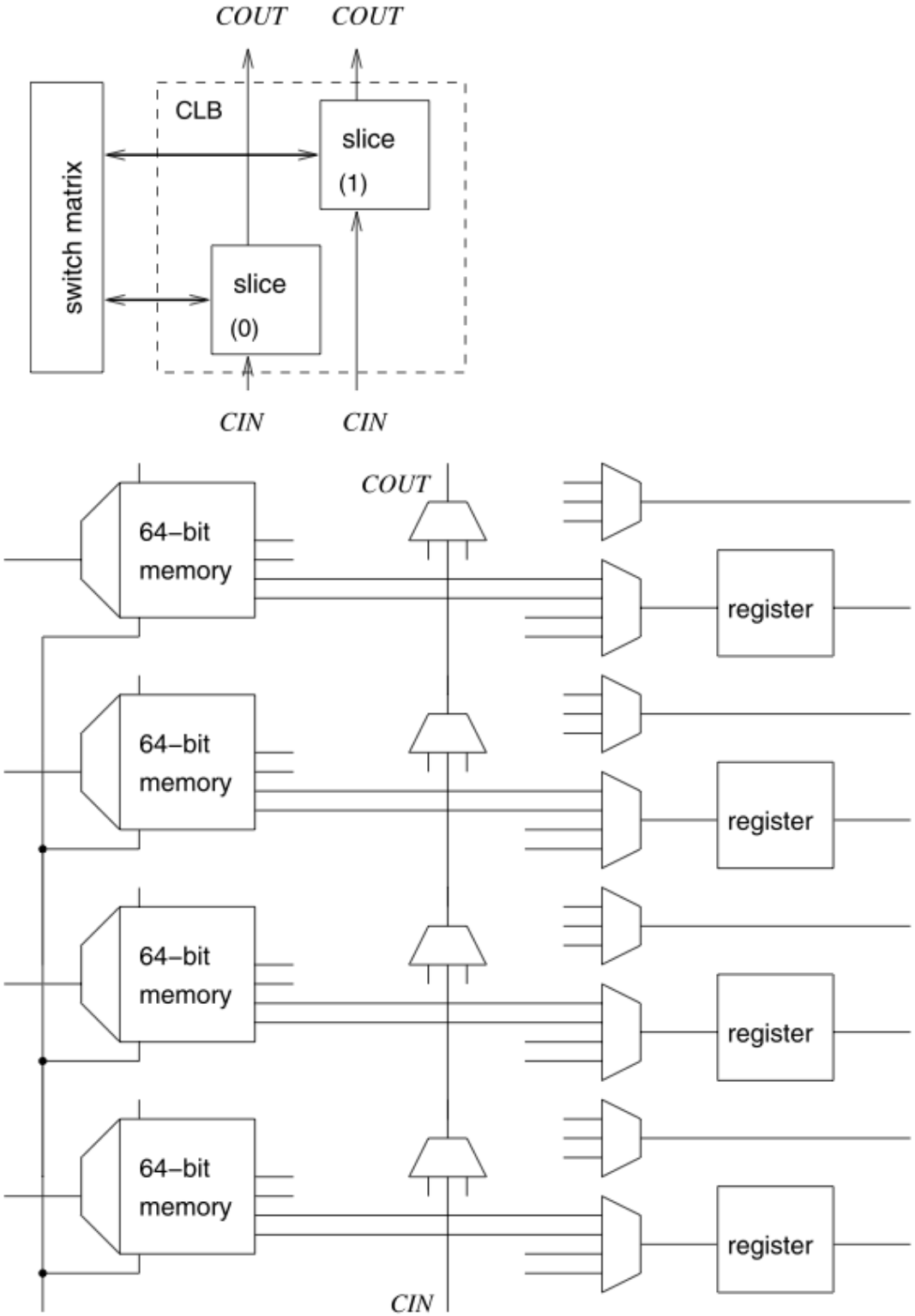
Multiprocessor System-On-a-Chip (MPSoC)

- increase in clock speed => more energy consumption
- PC => homogeneous *multi-core* systems (all processors of same type)
- embedded systems => highly specialized processors => energy efficiency close to ASIC

Reconfigurable Logic

- ASICs often too expensive, software too slow/ energy consuming => *Reconfigurable Logic*: almost as fast as special purpose hardware, reconfigurable by configuration data
- *fast prototyping*: prototype behaves like final system, but can be changed, essential functions need to be available
- *low volume applications*: ASICs too expensive, use reconfigurable logic
- *real-time systems*: timing of FPGA is precisely predictable
- reconfigurable hardware:
  - often includes RAM (volatile), configuration needs to be copied at startup
  - configuration data may be stored on ROM (non-volatile)
- *FPGA* (field programmable gate array): most common reconfigurable logic
  - contain configurable logic blocks
  -





  - each slice contains 4 memories, each memory can implement single 6-input or two 5-input logic functions
  - configuration data determines setting of multiplexers, clocking of registers and RAM, content of RAM components, connections between CLBs
- processors may be part of FPGA
  - *hard core*: physical processor in a special area (efficient)
  - *soft core*: synthesize processor from CLBs (flexible)

Memories

- efficient storage
  - run-time => memory hierarchies (large memory is slower)
  - code-size => compiler, compression
  - energy => memory hierarchies (large memory needs more energy)

- *Moore's Law*: Speed of processors doubles every 18 - 36 months, memory only gets faster by about 1.07
- => use small and fast memory as buffers between main memory and processor
  - *Cache* was introduced to improve run-time efficiency => also works for energy efficiency.
  - predictability may be low
- *Scratch Pad Memories* (SPM): small, fast memory, stores small things for rapid retrieval

Communication

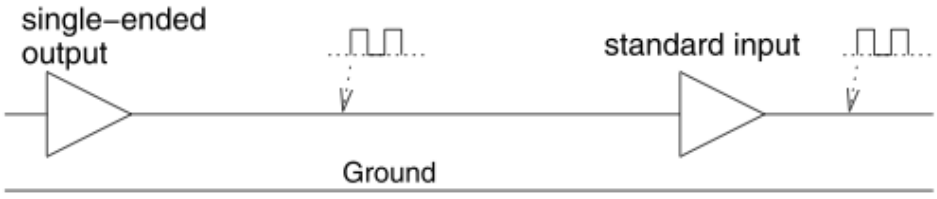
- *Channel*: abstract entity characterized by essential properties of communication
  - maximum information transfer capacity
  - noise parameters
  - probability of errors
- *Communication Media*: physical entities (wireless (radio frequency, infrared), optical (fiber), wires)

Common Communication Requirements

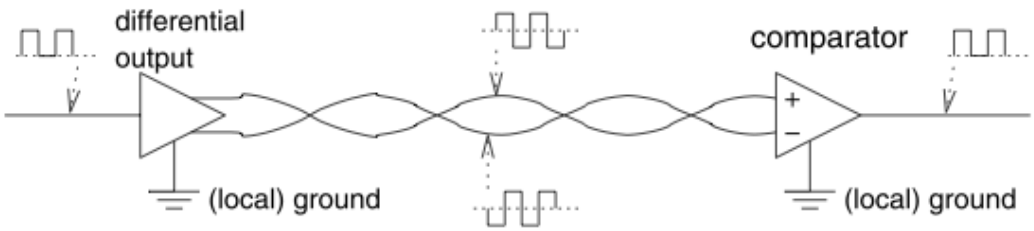
- *Real-Time Behavior*: (standard Ethernet is too slow)
- *Efficiency*: point-to-point often unfeasible (weight, ...)
- *Appropriate Bandwidth and Communication Delay*: enough bandwidth, not to large
- *Support for Event-Driven Communication*: Polling => very predictable real time behavior but delay may be large => emergencies need to immediately be communicated
- *Robustness*: extreme conditions (temperatures, radiation) may affect system (voltage, ...) => communication must be maintained
- *Fault Tolerance*: Design in a way, that faults don't crash the system => retries may invalidate real-time requirements
- *Maintainability, Diagnosability*: obvious
- *Privacy*: may require encryption

Electrical Robustness

- *Single-Ended Signaling*: signals travel a single wire
  - represented by voltage over common ground
  - susceptible to noise => message corruption
  - common ground not easy to achieve



- *Differential Signaling*: signal needs two wires
  - represented:
    - first -> second = positive voltage => 1
    - otherwise => 0
  - Noise added to both wires => no effect
  - magnitude of voltage irrelevant => reflection, resistance, ... have no effect
  - signals don't generate currents => quality of ground wires less important
  - no common ground wire required
  - => larger throughput than single-ended signaling
  - requires two wires for every signal



Guaranteeing Real-Time Behavior

- point-to-point => good real-time behavior
- *common, shared buses* more common, as they don't need a wire for every system
- *priority based arbitration* => poor timing predictability (conflicts difficult to predict at design time), can lead to starvation
- *TDMA* (time division multiple access): each party gets a time slot
  - reduces maximum amount of data
  - guaranteed bandwidth for all partners => no starvation
- *Ethernet*: collisions may occur => all partners must stop sending, wait and retry (*carrier-sense multiple access/ collision detect* (CSMA/CD)) => conflicts can impose great time penalty, but not likely
- solution: *carrier-sense multiple access/ collision avoidance* (CSMA/CA)
  - priorities assigned to partners
  - communication media is assigned to partners during *arbitration phase*
  - during arbitration, partners wanting to communicate say so, higher priority wins
  - highest priority partner has guaranteed real-time behavior, others don't (starvation possible)

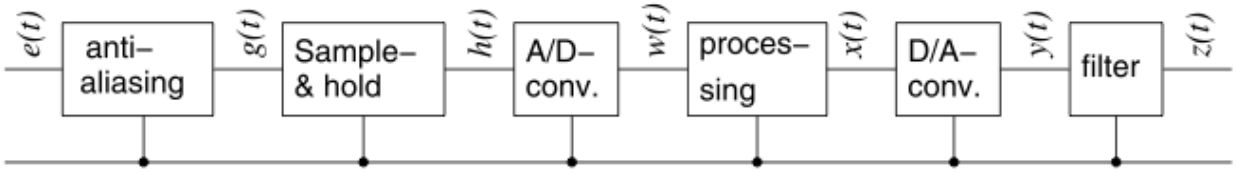
Examples

- *Sensor/ Actuator Bus*: communication between simples devices
- *Field Bus*: similar to sensor/ actuator bus, support larger data rates:
  - *Controller Area Network* (CAN): automotive industry, very cheap (huge quantities) => also in smart phones/ home fabrication
    - differential signaling with twisted pairs
    - arbitration with CSMA/CA
    - 10kbit/s - 1Mbit/s
    - low/ high priority signals
    - max latency of 134 µs (high priority)
  - *Time-Triggered-Protocol* (TTP): fault-tolerant safety systems like airbags
  - *FlexRay(TM)*: TDMA
    - static + dynamic arbitration => real-time and no starvation
    - *Bus Guardians* protect against flooding parties (*babbling idiots*)
  - *LIN* (Local Interconnect Network): low cost, standard for connecting sensors and actuators in automotive
  - *MAP*: bus designed for car factories
  - *EIB* (European Installation BUS): bus designed for smart homes
- *Wired Multimedia Communication*: large data rates
  - *MOST* (Media Oriented System Transport): standard for multimedia in automotive
  - *IEEE 1394* ("FireWire")
- *Wireless Communication*
  - *HSPA* (High Speed Packet Access): 7 Mbit/s
  - *LTE* (long term evolution): even higher rates
  - *Bluetooth* high rate of security
  - *IEEE 802.11* => Ethernet
  - *DECT*: standard for wireless phones in Europe

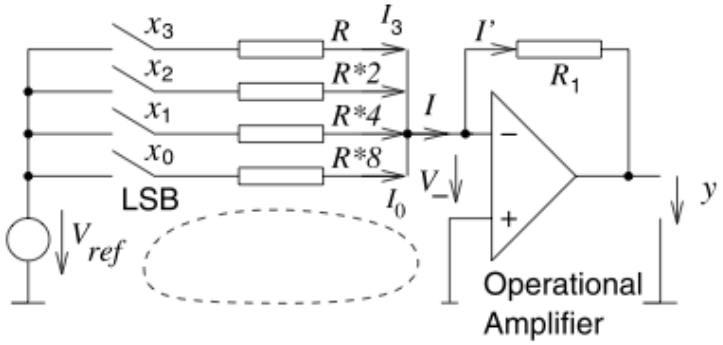
Output

- *Displays*: very wide field, many technologies (LCD, organic displays, ...)
- *Electro-Mechanical Devices*: motors, gauges, ...

Conversion Pipeline:



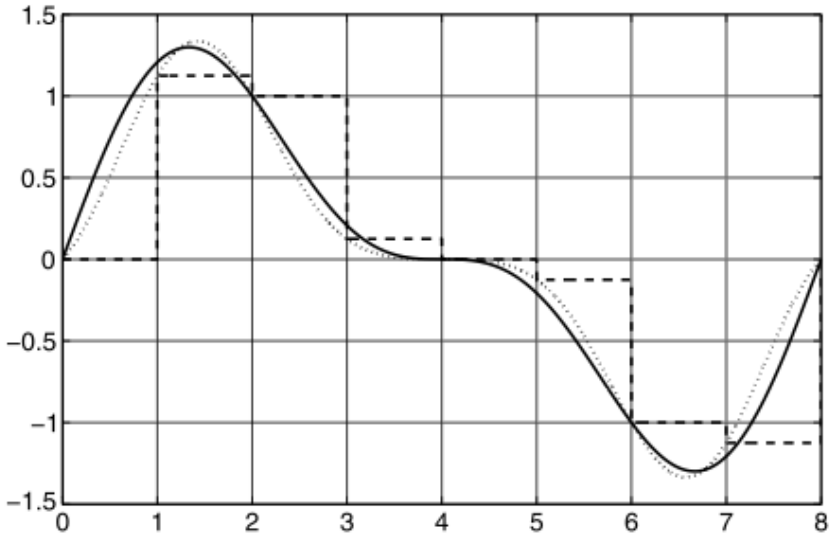
D/A Converter



- idea:
  - generate a current proportional to value of digital signal x
  - convert current into voltage y
- convert time- and value discrete signals to continuous time and value domain

Sampling Theorem

- A/D -> D/A => signal is not reproducible (aliasing)
- When sampling rate is larger than twice the highest frequency => Interpolation is possible:
  - $z(t) = \sum_{s=-\infty}^{\infty} \frac{y(t_s) sin \frac{pi}{p_s}(t-t_s)}{\frac{\pi}{p_s}(t-t_s)} = \sum_{s=-\infty}^{\infty} y(t_s) sinc(t-t_s)$ (Shannon-Whittaker interpolation)



  - not possible in digital domain => time-continuous signal needed
  - Fourier-Transformation: folding operation in time is multiplication with frequency dependent filter in frequency => Low-pass filter does the job!
    - no ideal low-pass filter => approximations

Actuators

- wide range => move large masses or move tiny distance?
  - *Microsystem Technology*: move in the µm area

Secure Hardware

- Might demand special hardware, cryptography, ...
- Might need to resists attacks (like fault injection, radiation)