Embedded Systems 11

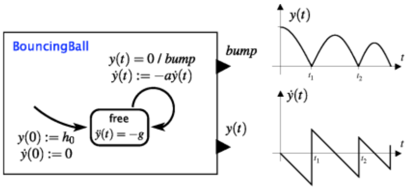
MIDTERM REVIEW: computational models

Communication/ local computations	Shared memory	Asynchronous message passing
Communicating finite state machines	Hybrid automata, statecharts, synchronous composition	
Data flow		Petri nets, Kahn process networks, SDF
Discrete event model	VHDL	

Hybrid automata

- Motivation
  - The design of an embedded system must consider both the continuous evolution of the environment and the discrete computation of the controller
- Major points
  - Modeling with hybrid automata
  - Special cases:
    - (extended) FSMs
    - ODEs
    - Timed Automata
  - Semantics (hybrid time sets, hybrid trajectories)
  - Zenoness

Hybrid Automata



- Q: set of modes
- S: set of state variables, partitioned into
  - C={c_1, c_2, ..., c_n}: continuous signals (with range $\mathfrak{R}$)
  - D={d_1, d_2, ..., d_m}: discrete signals (with range {absent} $\cup$ X)
- U={u_1, u_2, ..., u_k}: set of input signals,
- Init $\subseteq$ Q $\times$ $\mathfrak{R}^n$ $\times$ ({absent} $\cup$ X)^m : initial condition
- F: flows, defining differential equations for each continuous state variable in each mode
- J: Q $\times$ Guards $\rightarrow$ Q $\times$ Resets: jumps, where Guards is a constraint over C and U and Resets is a set of assignments of the form x_i := expr(X,U) for the state variables

Execution of a Hybrid Automaton

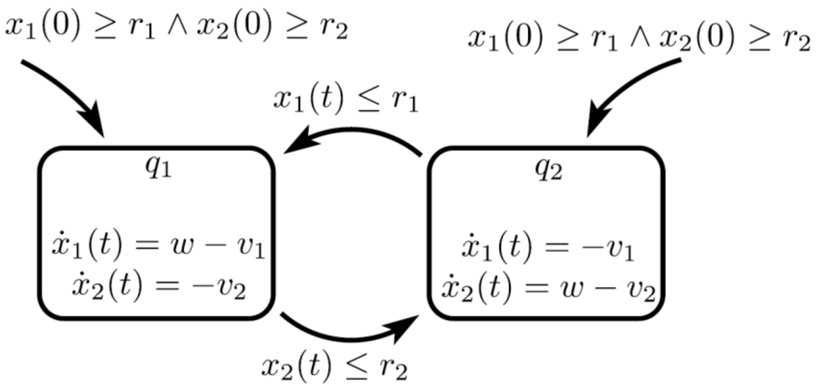
time

$$\begin{array}{ll}
\tau_0 & (q_0, \mathbf{x}_0) \\
\tau_0' & \rightsquigarrow (q_0, \mathbf{x}_0') \\
= & \downarrow \\
\tau_1 & (q_1, \mathbf{x}_1) \\
\tau_1' & \rightsquigarrow (q_1, \mathbf{x}_1') \\
 & \downarrow \\
 & \vdots \\
 & \downarrow \\
\tau_N & (q_N, \mathbf{x}_N) \\
\tau_N' & \rightsquigarrow (q_N, \mathbf{x}_N') \\
 & \downarrow \\
 & \vdots
\end{array}$$

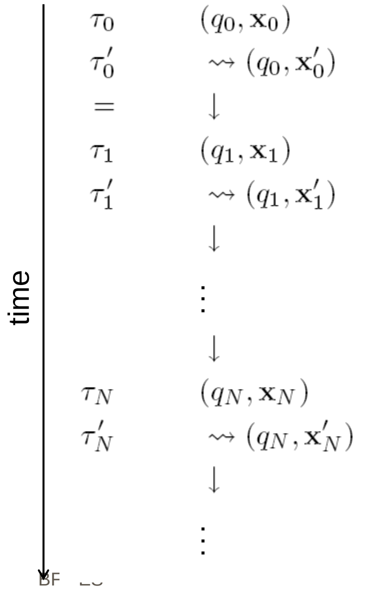
An **execution** of a hybrid automaton is a hybrid trajectory (τ, q, x) that statisfies the following conditions

- Initial condition: $(q_0, x_0) \in$ Init
- Discrete evolution: the pair $((q_i(\tau'_i),x_i(\tau'_i)), (q_{i+1}(\tau_{i+1}),x_i(\tau_{i+1}))$ satisfies J
- Continuous evolution: for all i,
  1. $q_i(\cdot)$ is constant over I_i
  2. $c_i(\cdot)$ is the solution to the differential equations in $F(q(\tau_i))$
  3. $d_i(\cdot)$ are absent during (τ_i,τ'_i)
  4. All jumps in J are disabled during (τ_i,τ'_i)

Q: What‘s wrong with this hybrid automaton?



Zeno Behavior



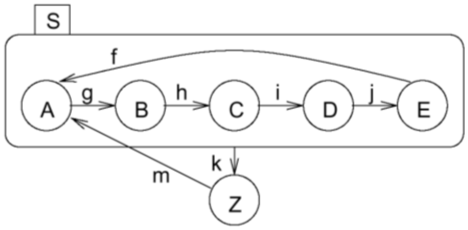
$$\begin{array}{ll}
\tau_0 & (q_0, \mathbf{x}_0) \\
\tau_0' & \quad \rightsquigarrow (q_0, \mathbf{x}_0') \\
= & \quad \downarrow \\
\tau_1 & (q_1, \mathbf{x}_1) \\
\tau_1' & \quad \rightsquigarrow (q_1, \mathbf{x}_1') \\
 & \quad \downarrow \\
 & \vdots \\
 & \quad \downarrow \\
\tau_N & (q_N, \mathbf{x}_N) \\
\tau_N' & \quad \rightsquigarrow (q_N, \mathbf{x}_N') \\
 & \quad \downarrow \\
 & \vdots
\end{array}$$

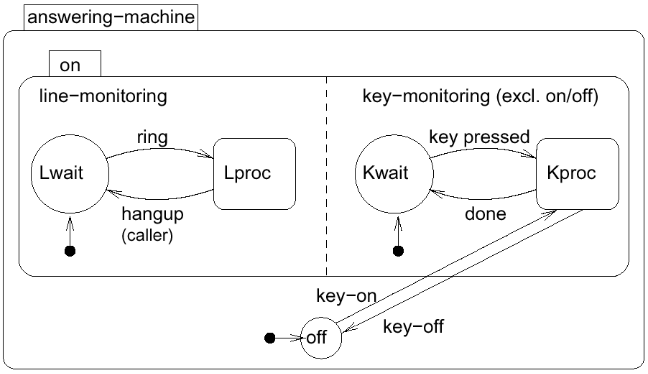
An execution of a hybrid automaton with time set τ is $\mathbf{zeno}$ iff $\langle \tau \rangle = \infty$ but $|\tau| < \infty$.

Statecharts

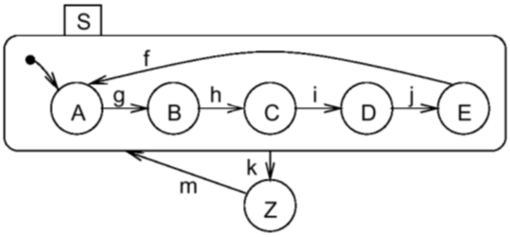
- Motivation
  - Concise models of complex systems:
    Statecharts = FSMs + hierarchy + orthogonality (concurrency)
  - Commercial tools (StateMate, StateFlow, …)
- Major points
  - Semantics
  - Virtual Prototyping (→ Stateflow)

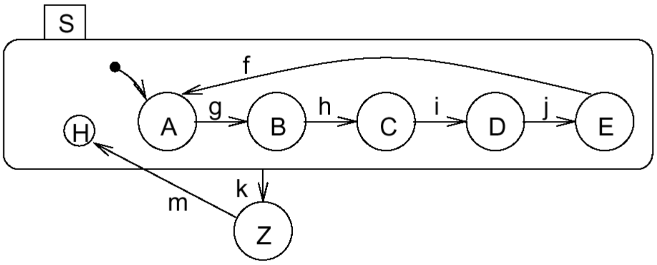
Statecharts: Hierarchy





Statecharts: Default-state mechanism





StateMate semantics

Three phases

1. Effect of external changes on events and conditions is evaluated
2. The set of transitions to be made in the current step and right-hand side of assignments are computed
3. Transitions become effective, variables obtain new values

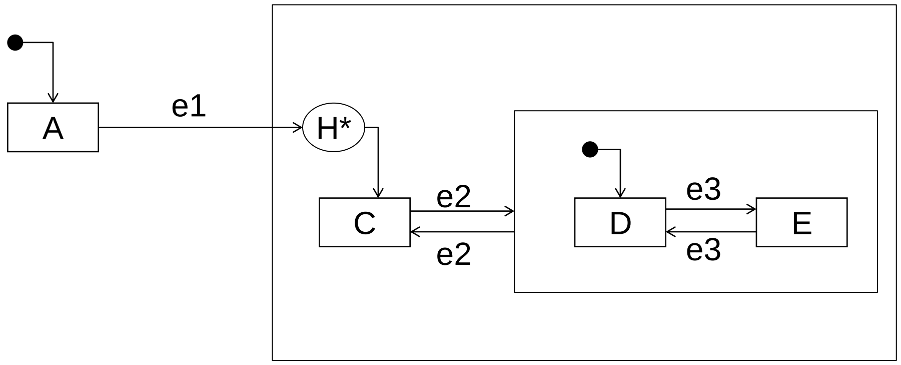
Broadcast mechanism

- Values of variables are visible to all parts of the StateChart model.
- New values become effective in part 3 of the execution stage for the current step and are obtained by all parts of the model in the following step.

☞ StateCharts implicitly assumes a **broadcast** mechanism for variables.

☞ StateCharts is appropriate for local control systems (☺), but not for distributed applications for which updating variables might take some time (☹).

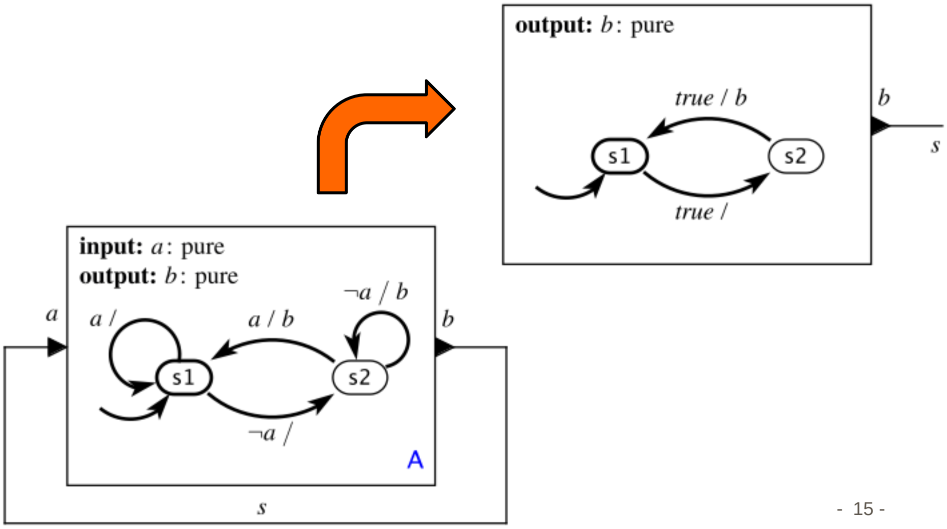
Q: Which states are active after e1,e2,e3,e2,e2?



Synchronous composition

- Motivation
  - Concurrent composition assuming a global clock and instant communication
  - Ensures deterministic system behavior
  - Semantic foundation of synchronous programming languages like Esterel, Lustre, Scade
- Major points
  - Fixed point semantics
  - Wellformedness
  - Constructive semantics

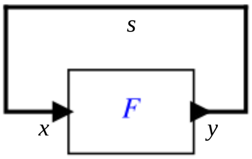
Composite machine



Well-formed feedback

At the n-th reaction, we seek $s(n) \in V_y \cup \{absent\}$ such that

$$s(n) = (f(n))(s(n))$$

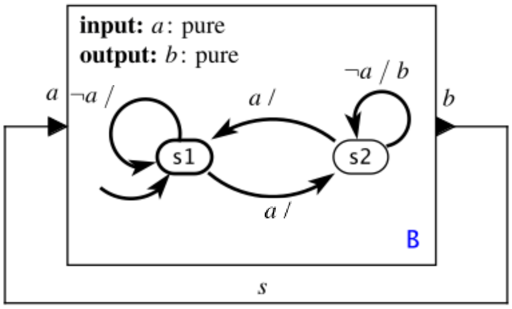


There are two potential problems:

1. It does not exist.
2. It is not unique.

In either case, we call the system **ill formed**. Otherwise, it is **well formed**.

Q: Well-formed?



Petri Nets

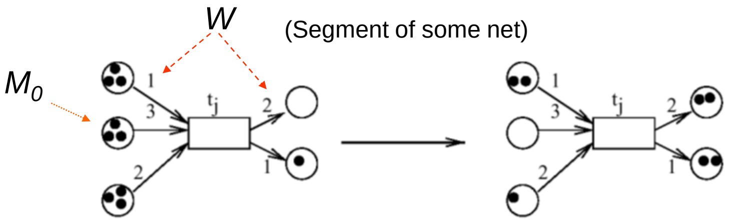
- Motivation
  - Modeling causal dependencies
  - Distributed systems
- Major points
  - Boundedness, coverability graph
  - Liveness, deadlock
  - Place invariants

Place/transition nets

multiple tokens per place

Def.: (P, T, F, K, W, M_0) is called a **place/transition net (P/T net)** iff

1. N=(P,T,F) is a **net** with places P and transitions T
2. K: $P \rightarrow (\mathbb{N}_0 \cup \{\omega\}) \setminus \{0\}$ denotes the **capacity** of places
   (ω symbolizes infinite capacity)
3. W: $F \rightarrow (\mathbb{N}_0 \setminus \{0\})$ denotes the **weight of graph edges**
4. M_0: $P \rightarrow \mathbb{N}_0 \cup \{\omega\}$ represents the **initial marking** of places

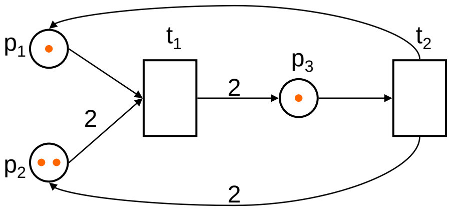


default:
K = ω
W = 1

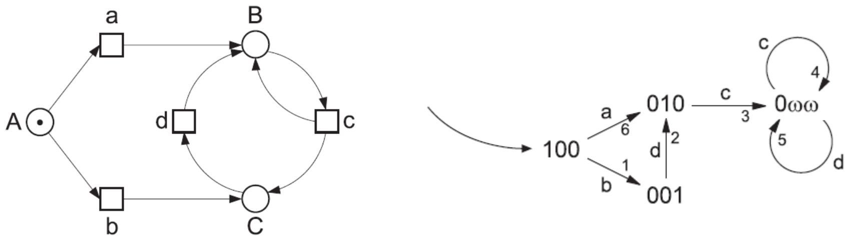
Boundedness

- A place is called **k-bounded** or **k-safe** if it contains in all reachable markings at most k tokens
- A net is **bounded** if each place is bounded

Application: places represent buffers and registers
→ avoid buffer overflow



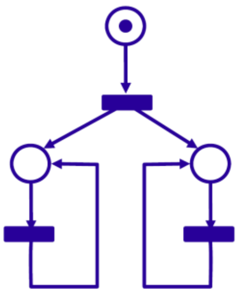
Coverability graph



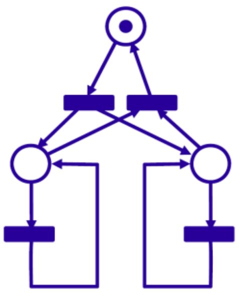
Example from Wolfgang Reisig: Petrinetze, Springer 2010

Liveness

- A transition is **live** if in every reachable marking there exists a firing sequence such that the transition becomes enabled
- A net is **live** if all its transitions are live

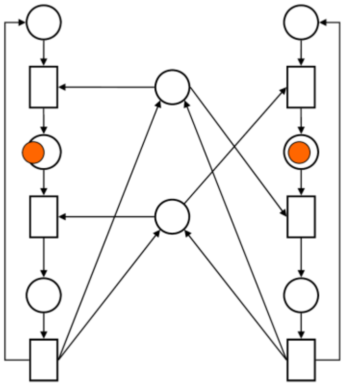


Live
?

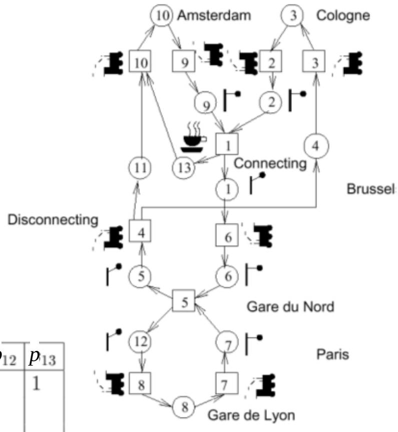


Deadlock

- A **dead marking** (**deadlock**) is a marking where no transition can fire
- A net is **deadlock-free** if no dead marking is reachable



Place invariants



N

$\underline{\boldsymbol{N}}^{T}\ \underline{\boldsymbol{c}}_{R} = \mathbf{0}$, with

$\underline{\boldsymbol{N}}^{T}$=

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}
t_1	1	-1							-1				1
t_2		1	-1										
t_3			1	-1									
t_4				1	-1						1		
t_5					1	-1	-1					1	
t_6	-1					1							
t_7							1	-1					
t_8								1				-1	
t_9									1	-1			
t_{10}										1	-1		-1

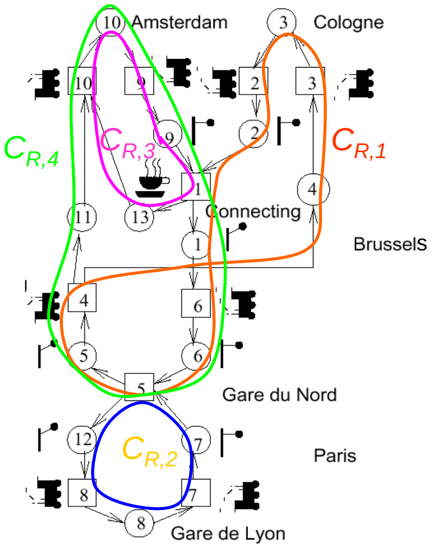
Q: Is this net bounded?

$$c_{R,1} = \begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix}$$

$$c_{R,2} = \begin{pmatrix}0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0\end{pmatrix}$$

$$c_{R,3} = \begin{pmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1\end{pmatrix}$$

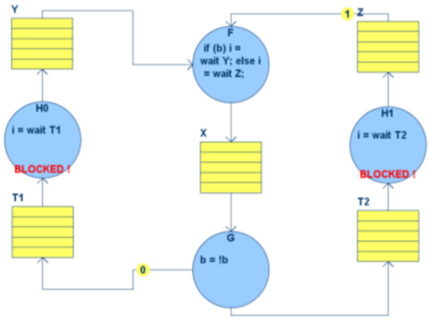
$$c_{R,4} = \begin{pmatrix}1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0\end{pmatrix}$$



Kahn process networks & SDF

- Motivation
  - Many applications can be specified in the form of a set of communicating processes
  - Communication exclusively through FIFOs
  - Describe local behavior + dependencies without worrying about global control
- Major points
  - Kahn process networks
    - Park‘s runtime scheduling algorithm
  - Synchronous data flow (SDF)
    - Lee/Messerschmitt‘s static scheduling algorithm

Kahn process networks

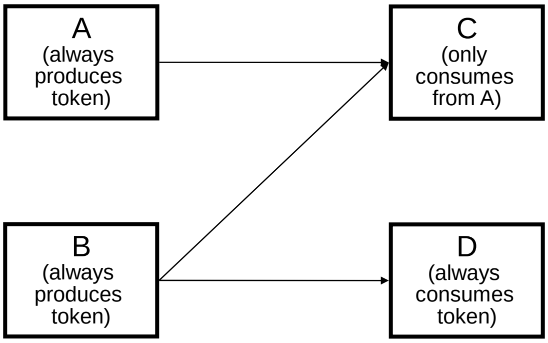


- Each node corresponds to one program/task;
- Communication is only via channels;
- Channels include FIFOs as large as needed;
- **Send operations are non-blocking, reads are blocking.**

Kahn process networks are deterministic

- There is only one sender per channel.
- A process cannot check whether data is available before attempting a read.
- A process cannot wait for data for more than one port at a time.
- Therefore, the order of reads depends only on data, not on the arrival time.
- Therefore, Kahn process networks are deterministic (!); for a given input, the result will always the same, regardless of the speed of the nodes.

Scheduling Kahn Networks



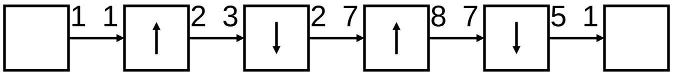
Problem: run processes with finite buffer

Parks’ Scheduling Algorithm (1995)

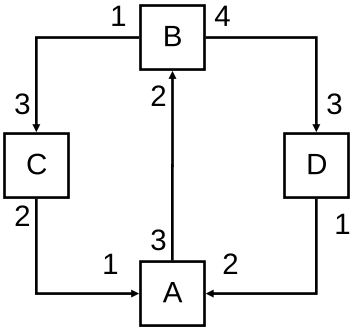
- Set a capacity on each channel
- Block a write if the channel is full
- Repeat
  - Run until deadlock occurs
  - If there are no blocking writes → terminate
  - Among the channels that block writes, select the channel with least capacity and increase capacity until producer can fire.

Synchronous data flow (SDF)

- Asynchronous message passing=
  tasks do not have to wait until output is accepted.
- Synchronous data flow =
  all tokens are consumed at the same time.



PASS example: 1) firing rates



d(AB)=6

3a – 2b = 0

4b – 3d = 0

b – 3c = 0

2c – a = 0

d – 2a = 0

Solution:

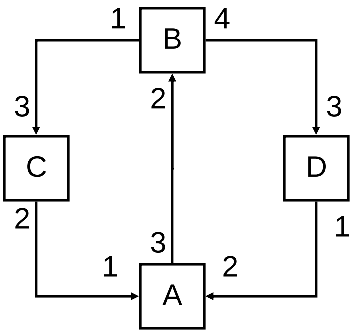
a = 2c

b = 3c

d = 4c

Smallest solution: a=2; b=3; d=4; c=1

PASS example: 2) Simulation



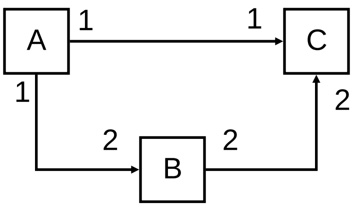
d(AB)=6

Smallest solution:
a=2; b=3; d=4; c=1

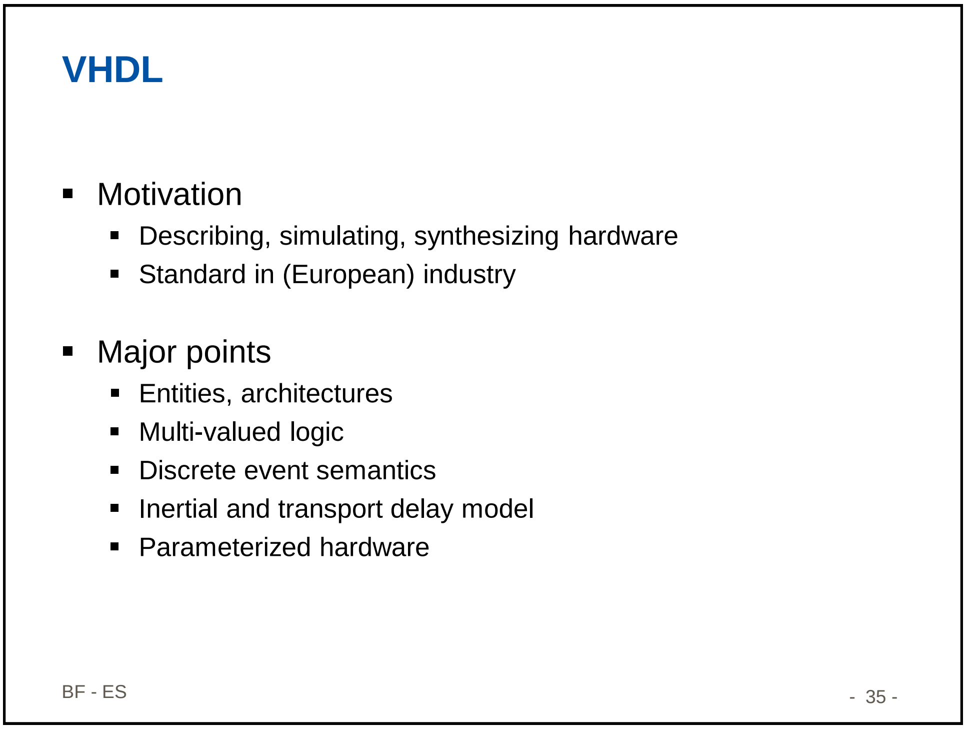
Possible schedules:
BBBCDDDDAA
BDBDBCADDA
BBDDBDDCAA
(and many more)

BC... not valid

Q: Schedule?



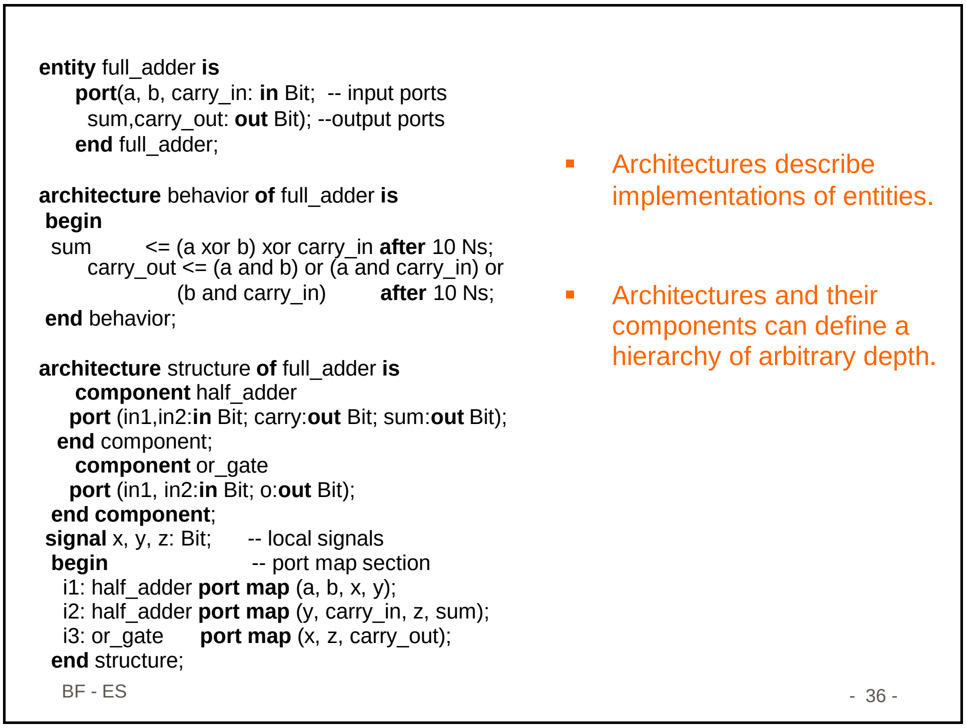
d(AB)=2



VHDL

- Motivation
  - Describing, simulating, synthesizing hardware
  - Standard in (European) industry

- Major points
  - Entities, architectures
  - Multi-valued logic
  - Discrete event semantics
  - Inertial and transport delay model
  - Parameterized hardware

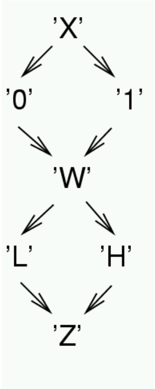


```
entity full_adder is
    port(a, b, carry_in: in Bit;  -- input ports
      sum,carry_out: out Bit); --output ports
    end full_adder;

architecture behavior of full_adder is
 begin
  sum       <= (a xor b) xor carry_in after 10 Ns;
      carry_out <= (a and b) or (a and carry_in) or
                (b and carry_in)        after 10 Ns;
 end behavior;

architecture structure of full_adder is
    component half_adder
    port (in1,in2:in Bit; carry:out Bit; sum:out Bit);
   end component;
    component or_gate
    port (in1, in2:in Bit; o:out Bit);
  end component;
 signal x, y, z: Bit;      -- local signals
  begin                    -- port map section
    i1: half_adder port map (a, b, x, y);
    i2: half_adder port map (y, carry_in, z, sum);
    i3: or_gate      port map (x, z, carry_out);
  end structure;
```

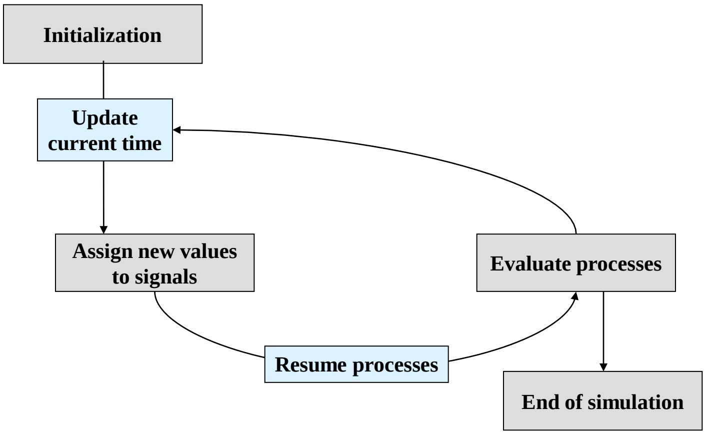
- Architectures describe implementations of entities.
- Architectures and their components can define a hierarchy of arbitrary depth.

Resolution function for IEEE 1164

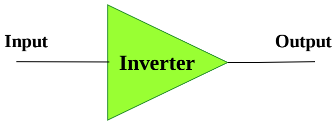
```
constant resolution_table : stdlogic_table := (
--U    X    0    1    Z   W     L    H   –
('U',  'U',  'U',  'U', 'U', 'U',    'U', 'U', 'U'),   --| U |
('U',  'X',  'X',  'X', 'X', 'X',    'X', 'X', 'X'),   --| X |
('U',  'X',  '0',  'X', '0', '0',    '0', '0', 'X'),   --| 0 |
('U',  'X',  'X',  '1', '1', '1',    '1', '1', 'X'),   --| 1 |
('U',  'X',  '0',  '1', 'Z', 'W',    'L', 'H', 'X'),   --| Z |
('U',  'X',  '0',  '1', 'W', 'W',    'W', 'H', 'X'),   --| W |
('U',  'X',  '0',  '1', 'L', 'W',    'L', 'W', 'X'),   --| L |
('U',  'X',  '0',  '1', 'H', 'W',    'W', 'H', 'X'),   --| H |
('U',  'X',  'X',  'X', 'X', 'X',    'X', 'X', 'X')    --| - |
);
```



Semantics

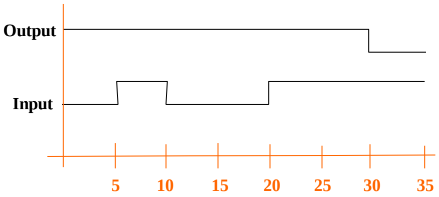


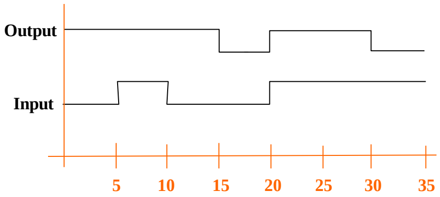
Inertial vs. transport delay model



-- INERTIAL is the default
Output <= NOT input AFTER 10 ns;

-- TRANSPORT must be specified
Output <= TRANSPORT NOT input AFTER 10 ns;





Q: What is the resulting wave form?

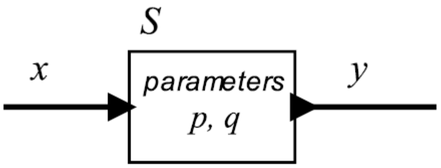
```
o <= transport '0', '1' after 5 ns,
           '0' after 10 ns, ‘0' after 20 ns;
wait for 5 ns;
o <= '1' after 7 ns;
wait;
```

Actor model

- Motivation
  - Combining components into larger systems
  - Combining different computational models
- Major points
  - Continuous actors, discrete actors
  - Ptolemy
  - Virtual Prototyping (→ Matlab/Simulink/Stateflow)

Actor Model of Continuous-Time Systems

A *system* is a function that accepts an input *signal* and yields an output signal.



The domain and range of the system function are sets of signals, which themselves are functions.

$$x\colon \mathbb{R} \to \mathbb{R}, \quad y\colon \mathbb{R} \to \mathbb{R}$$

$$S\colon X \to Y$$

$$X = Y = (\mathbb{R} \to \mathbb{R})$$

Parameters may affect the definition of the function *S*.

Discrete Signals

Let *e* be a signal $\mathbb{R} \to \{absent\} \cup X$
where *X* is any set of values.

Let $T = \{t \in \mathbb{R} \ : \ e(t) \neq absent\}$

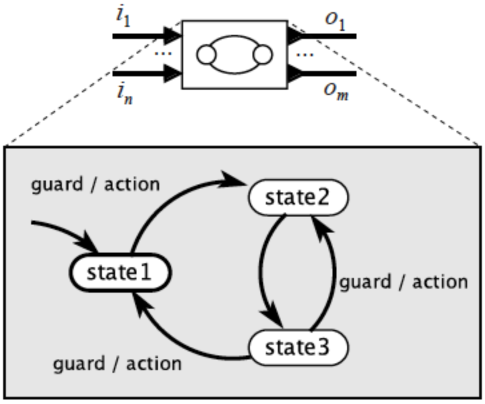
Then *e* is **discrete** iff there exists a one-to-one function

$$f : T \to \mathbb{N}$$

that is order-preserving, i.e., for all $t_1 \leq t_2$, $f(t_1) \leq f(t_2)$.

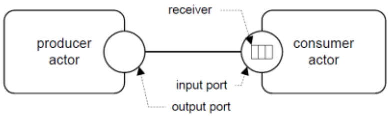
Actor Model for State Machines

Expose inputs and outputs, enabling composition:

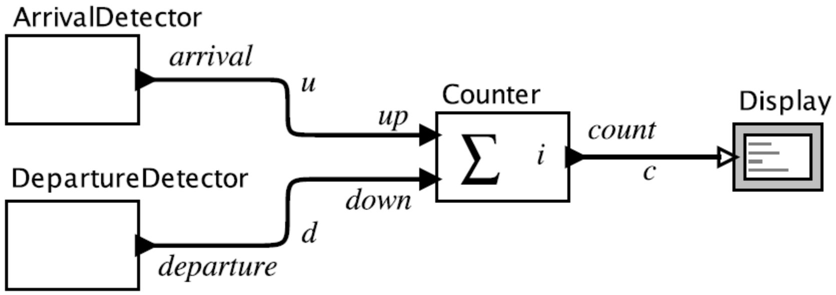


Ptolemy

- A model is a set of interconnected *actors* and one *director*
- Actor
  - Input & output *ports*, states, & parameters
- Models of computation
  - Define the interaction semantics
  - Implemented in Ptolemy II by a *domain*
    - Director + Receiver
- Director
  - Manages the data flow and the scheduling of the actors
  - The director fires the actors
- Receiver
  - Defines the semantics of the port buffers



Q: Is the output discrete?

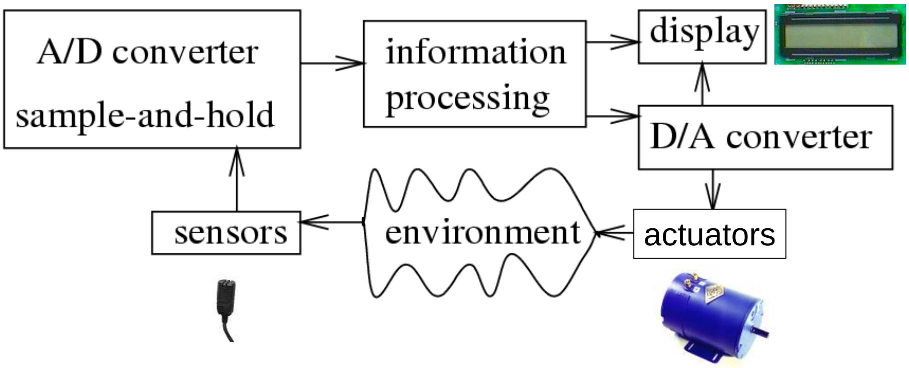


Assume that *arrival* and *departure* are discrete signals. Is *count* a discrete signal?

Sensors & Actuators

Embedded System Hardware

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



Sensors and Actuators

Sensors:

- Cameras
- Accelerometers
- Rate gyros
- Strain gauges
- Microphones
- Magnetometers
- Radar/Lidar
- Chemical sensors
- Pressure sensors

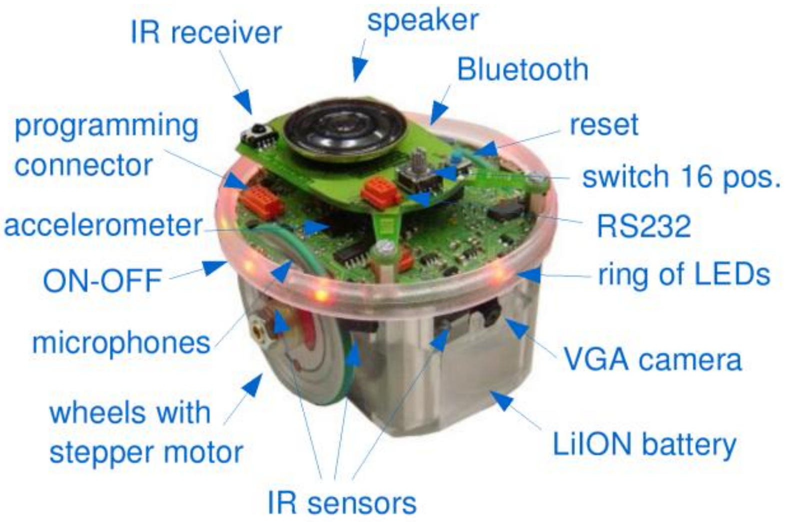
Actuators:

- Motor controllers
- Solenoids
- LEDs, lasers
- LCD and plasma displays
- Loudspeakers

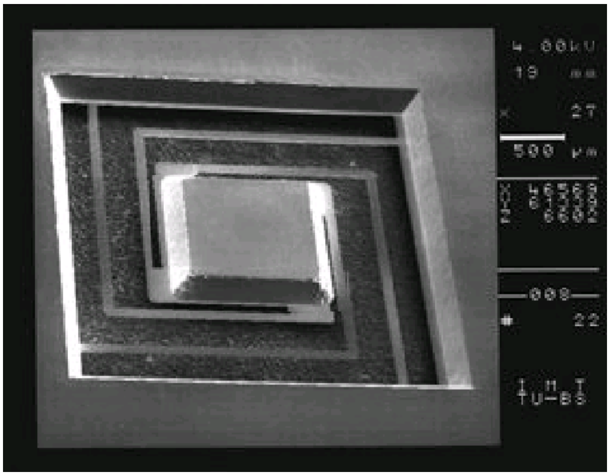
Modeling Issues:

- Physical dynamics
- Noise
- Bias
- Sampling
- Interactions

E-puck



Acceleration Sensor



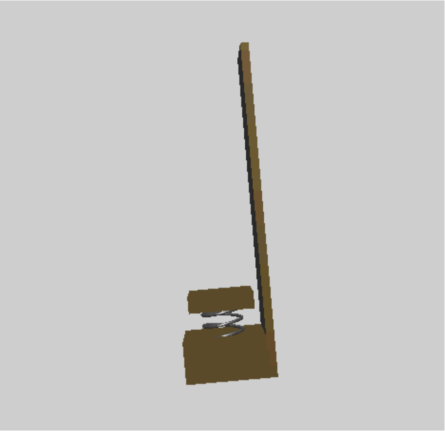
Courtesy & ©: S. Bütgenbach, TU Braunschweig

Spring-Mass-Damper Accelerometer

By Newton’s second law, F=ma.

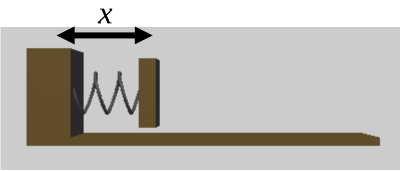
For example, F could be the earth’s gravitational force.

The force is balanced by the restoring force of the spring.



Spring-Mass-Damper System

- mass: M
- spring constant: k
- spring rest position: p
- position of mass: x
- viscous damping constant: c



Force due to spring extension:

$$F_1(t) = k(p - x(t))$$

Force due to viscous damping:

$$F_2(t) = -c\dot{x}(t)$$

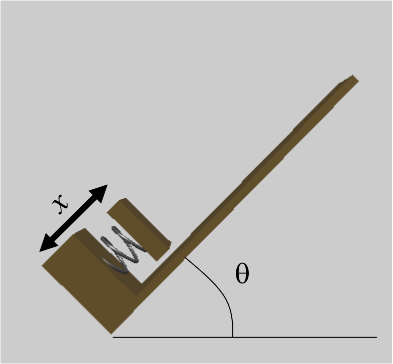
Newton's second law:

$$F_1(t) + F_2(t) = M\ddot{x}(t)$$

or

$$M\ddot{x}(t) + c\dot{x}(t) + kx(t) = kp.$$

Measuring tilt



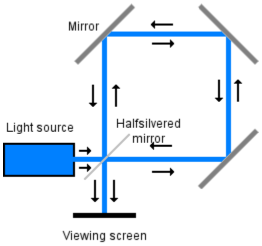
Component of gravitational force in the direction of the accelerometer axis must equal the spring force:

$$Mg\sin(\theta) = k(p - x(t))$$

Difficulties Using Accelerometers

- Separating tilt from acceleration
- Integrating twice to get position: Drift
- Vibration
- Nonlinearities in the spring or damper

Measuring Changes in Orientation: Gyroscopes



Optical gyros: Leverage the Sagnac effect, where a laser light is sent around a loop in opposite directions and the interference is measured. When the loop is rotating, the distance the light travels in one direction is smaller than the distance in the other. This shows up as a change in the interference.

Images from the Wikipedia Commons

Inertial Navigation Systems

Combinations of:

- GPS (for initialization and periodic correction)
- Three axis gyroscope measures orientation
- Three axis accelerometer, double integrated for position after correction for orientation

Magnetometers

- A very common type is the Hall Effect magnetometer.
- Charge particles (electrons) flow through a conductor (2) serving as a Hall sensor. Magnets (3) induce a magnetic field (4) that causes the charged particles to accumulate on one side of the Hall sensor, inducing a measurable voltage difference from top to bottom.
- The four drawings at the right illustrate electron paths under different current and magnetic field polarities.

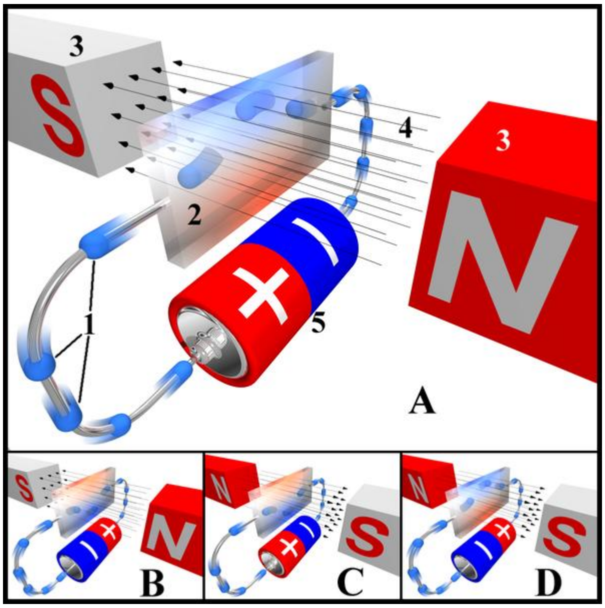
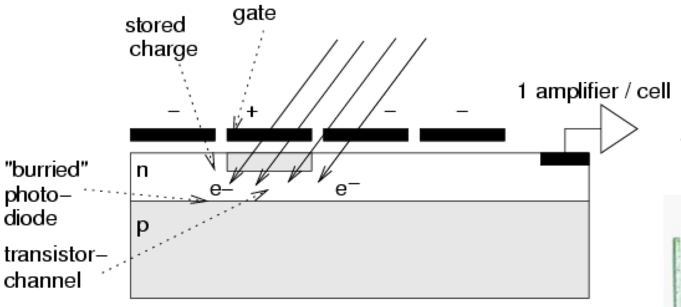


Image source: Wikipedia Commons

Edwin Hall discovered this effect in 1879.

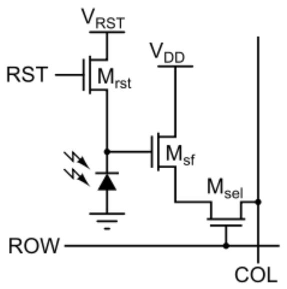
Charge-coupled devices (CCD) image sensors

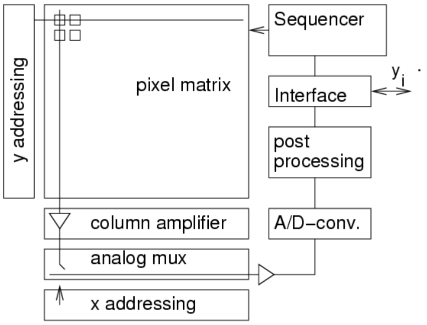
Based on charge transfer to next pixel cell



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

CMOS image sensors





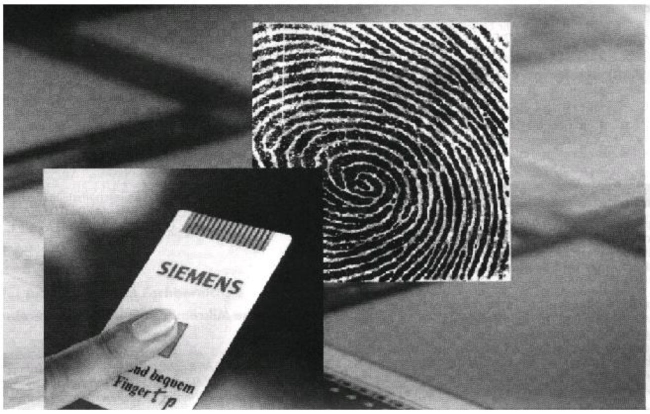
- Lower power consumption
- Lower cost

- Based on standard production process for CMOS chips, allows integration with other components

- low cost devices
- Automotive
- medical

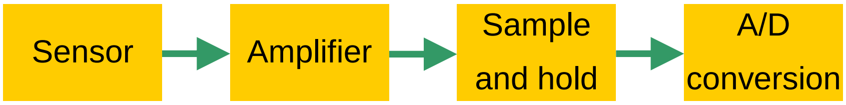
Example: Biometrical Sensors

Example: Fingerprint sensor (© Siemens, VDE):



Matrix of 256 x 256 elem.
Voltage ~ distance.
Resistance also computed

Standard layout of sensor systems



- 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