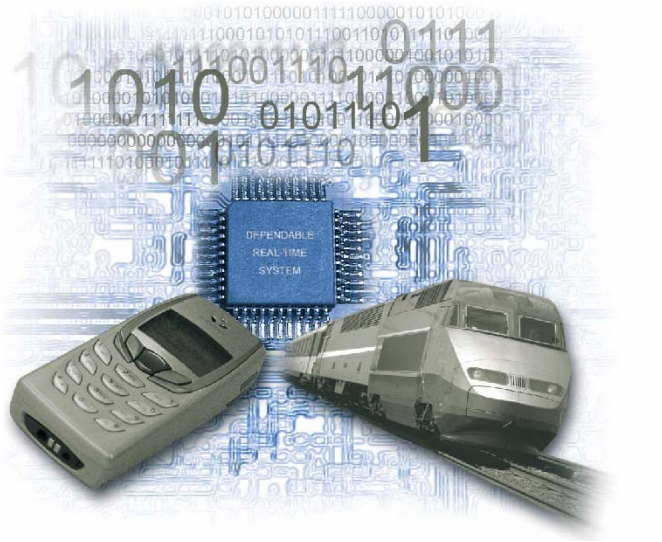
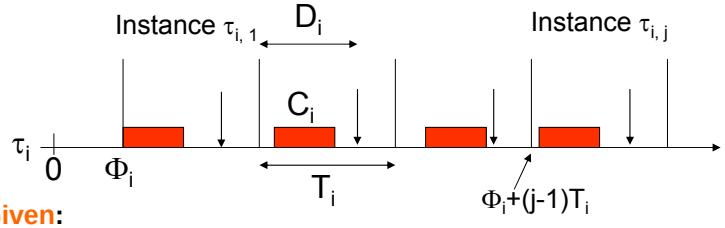
Embedded Systems 17



Periodic scheduling REVIEW



- **Given:**
  - A set of periodic tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with
    - phases Φ_i (arrival times of first instances of tasks),
    - periods T_i (time difference between two consecutive activations)
    - relative deadlines D_i (deadline relative to arrival times of instances)
    - computation times C_i

  $\Rightarrow$ *j* th instance $\tau_{i,j}$ of task τ_i with
    - arrival time $a_{i,j} = \Phi_i + (j-1)\, T_i$,
    - deadline $d_{i,j} = \Phi_i + (j-1)\, T_i + D_i$,
    - start time $s_{i,j}$ and
    - finishing time $f_{i,j}$
- **Find a feasible schedule**

Rate monotonic scheduling (RM) REVIEW

- Rate monotonic scheduling (RM) (Liu, Layland '73):
  - Assign fixed priorities to tasks τ_i:
    - priority(τ_i) = $1/T_i$
    - I.e., priority reflects release rate
  - Always execute ready task with highest priority
  - Preemptive: currently executing task is preempted by newly arrived task with shorter period.
- **Theorem (Liu, Layland, 1973):**
  RM is optimal among all fixed-priority scheduling algorithms.

Schedulability check REVIEW

- $U_{lub} = n(2^{1/n} - 1)$
- A set of tasks can be scheduled by RM if
  $U < U_{bnd}(\text{RM}) = \ln 2 \approx 0.69$
- But what can we tell about schedulability when processor utilization factor is larger than $n(2^{1/n} - 1)$?
- Answer:
  We can compute a more precise result, if we make use of the knowledge of periods T_i and computation times C_i.

Schedulability check

REVIEW

- Remember:
  The response time $R_{i,j}$ of an instance j of task i is the time (measured from the arrival time) at which the instance is finished: $R_{i,j} = f_{i,j} - a_{i,j}$.
- Compute an upper bound R_i on the response time:
  - Suppose that $\tau_1, ..., \tau_n$ are ordered with increasing periods (i.e. decreasing priorities).
  - Consider an arbitrary periodic task τ_i.
  - At a critical instant t, when an instance of τ_i arrives together with all higher priority tasks, we have:
    - $R_i = C_i + \sum_{k=1}^{i-1}$ (# activations of τ_k during $[t, t + R_i]) \cdot C_k$
      $= C_i + \sum_{k=1}^{i-1} \lceil R_i/T_k \rceil \cdot C_k$

Schedulability check

REVIEW

- Solution of fixed point equation?
- Compute the following sequence:
  - $R_i^{(0)} = C_i$.
  - $R_i^{(j+1)} = C_i + \sum_{k=1}^{i-1} \lceil R_i^{(j)} / T_k \rceil \cdot C_k$.
- It is easy to see that this sequence is monotonically increasing, i.e., $f(x) = C_i + \sum_{k=1}^{i-1} \lceil x / T_k \rceil \cdot C_k$ is monotonically increasing.
- $\Rightarrow$ If a least fixed point of f(x) exists, then the sequence converges to this fixed point.

Schedulability check

REVIEW

$\Rightarrow$ Algorithm:

$\forall$ i: $R_i^{(0)} = C_i$
repeat
  $\forall$ i: $R_i^{(j+1)} = C_i + \sum_{k=1}^{i-1} \lceil R_i^{(j)} / T_k \rceil \cdot C_k$
until ($\exists$ i with $R_i^{(j+1)} > D_i$) **or** ($\forall$ i $R_i^{(j+1)} = R_i^{(j)}$);
if ($\forall$ i $R_i^{(j+1)} = R_i^{(j)}$) **then**
  report("RM schedulable");

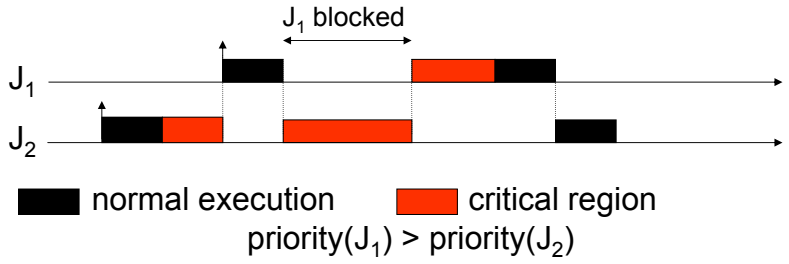
REVIEW

Rate Monotonic Scheduling in Presence of Task Dependencies

The priority inversion problem

REVIEW

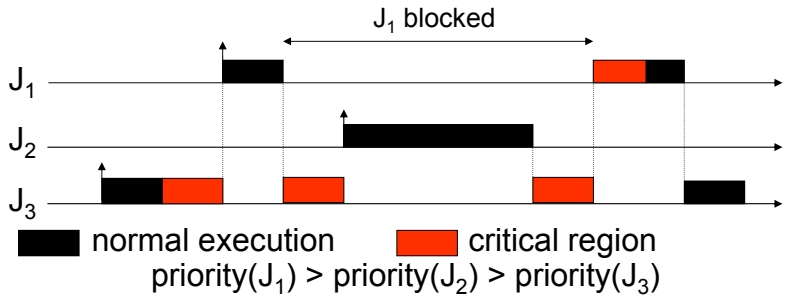
- **Priority inversion** can occur due to resource conflicts (exclusive use of shared resources) in fixed priority schedulers like RM:



- Here: Blocking time equal to length of critical section.

The priority inversion problem

REVIEW



- Blocking time equal to length of critical section + computation time of J_2.
- Unbounded time of priority inversion, if J_3 is interrupted by tasks with priority between J_1 and J_3 during its critical region.

Coping with priority inversion: The priority inheritance protocol

REVIEW

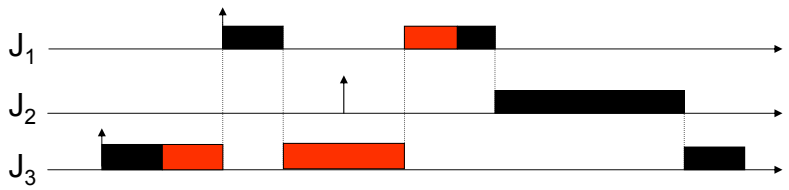
Idea of **priority inheritance protocol**:

- If a task J_h blocks, since another task J_l with lower priority owns the requested resource, then J_l inherits the priority of J_h.
- When J_l releases the resource, the priority inheritance from J_h is undone.
- Rule: Tasks always inherit the highest priority of tasks blocked by it.

Direct vs. push-through blocking

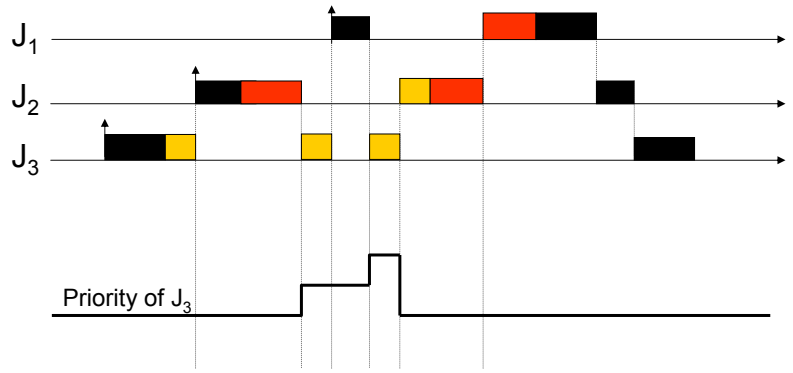
REVIEW

- **Direct blocking:** High-priority job tries to acquire resource already held by lower-priority job
- **Push-through blocking:** Medium-priority job is blocked by lower-priority job that has inherited a higher priority.



Transitive priority inheritance

REVIEW



Schedulability check

REVIEW

Let B_i be the maximum blocking time due to lower-priority jobs that a job J_i may experience.

$\forall$ i: $R_i^{(0)} = C_i$

repeat

$\forall$ i: $R_i^{(j+1)} = C_i + B_i + \sum_{k=1}^{i-1} \lceil R_i^{(j)} / T_k \rceil \cdot C_k$

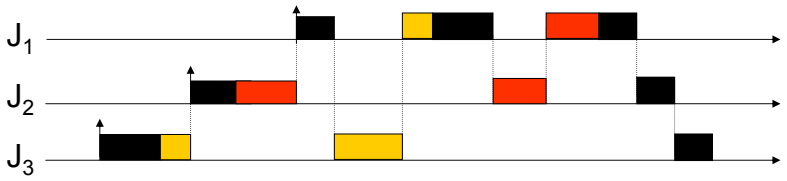
until ($\exists$ i with $R_i^{(j+1)} > D_i$) **or** ($\forall$ i $R_i^{(j+1)} = R_i^{(j)}$);

if ($\forall$ i $R_i^{(j+1)} = R_i^{(j)}$) **then**

report("RM schedulable");

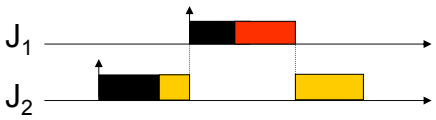
Problem: Chained Blocking

REVIEW



Problem: Deadlock

REVIEW



J_1:	J_2:
wait(S_a)	wait(S_b)
wait(S_b)	wait(S_a)
signal(S_b)	signal(S_a)
signal(S_a)	signal(S_b)

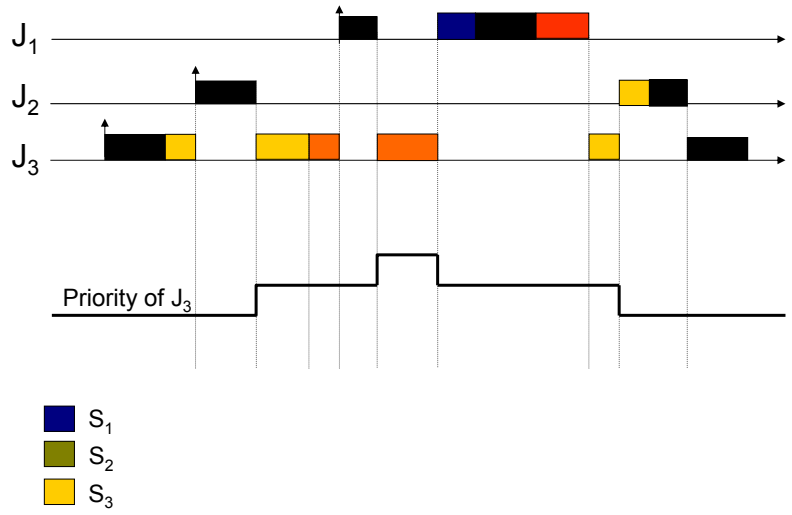
Priority Ceiling Protocol

REVIEW

- Each semaphore S is assigned a **priority ceiling**:
  C(S)= priority of the highest-priority job that can lock S
- The processor is assigned to a ready job J with highest priority.
- To enter a critical section, J needs priority > C(S*),
  where S* is the currently locked semaphore with max C.
  → otherwise J „blocks on semaphore“ and
  priority of J is inherited by job J‘ holding S*.
- When J‘ exits critical section, its priority is updated to the highest priority of some job that is blocked by J‘ (or to the nominal priority if no such job exists).

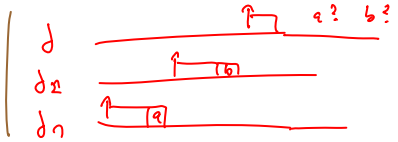
Example

REVIEW



Priority Ceiling Protocol

Theorem (Sha/Rajkumar/Lehoczky): Under the Priority Ceiling Protocol, a job can be blocked for at most the duration of one critical section.



- Suppose that J is blocked by lower-priority jobs J_1, J_2
- Let J_1 enter the critical section first with semaphore S_A; let C_1 be the max ceiling among the semaphores it blocks.
- Then J_2 enters its crit. section $\Rightarrow$ priority $(J_2) > C_1$
- Since J uses $S_A \Rightarrow$ priority $(J) \leq C_1$.

$\Rightarrow$ priority (J) < priority (J_2) ↯

Priority Ceiling Protocol

The Priority Ceiling Protocol prevents deadlocks.

Deadlock $\Rightarrow$ $\exists$ jobs $J_0, \ldots, J_{n-1}$ s.t.
J_0 is blocked by J_1, J_1 is blocked by $J_2, \ldots$,
J_{n-1} is blocked by J_0.

Let J_i be the last job to enter the critical section $\Rightarrow$ priority (J_i) > max ceiling of semaphores locked by $J_{i+1 \bmod n}$

Hence, since J_i blocked by $J_{i+1 \bmod n}$,
priority $(J_i) \leq$ ceiling of blocked semaphore. ↯

Incorporating a-periodic tasks

- In real systems, not all tasks are periodic
  - Environmental events to be processed
  - Exceptions raised
  - Background tasks running whenever CPU time budget permits
- Thus, real systems tend to be a combination of
  - periodic and
  - a-periodic tasks

  and of
  - hard real-time and
  - soft real-time tasks.

A-periodic and periodic tasks together (1)

- A-periodic and periodic tasks together
  - can be handled by dynamic-priority schedulers like EDF
- Problem:
  - Off-line guarantees can not be given without assumptions on a-periodic tasks.
  - If deadlines for a-periodic tasks are hard, a-periodic tasks need to be characterized by a minimum interarrival time between consecutive instances
    $\Rightarrow$ bounds on the a-periodic load
  - A-periodic tasks with maximum arrival rate may be modeled as periodic tasks with this rate

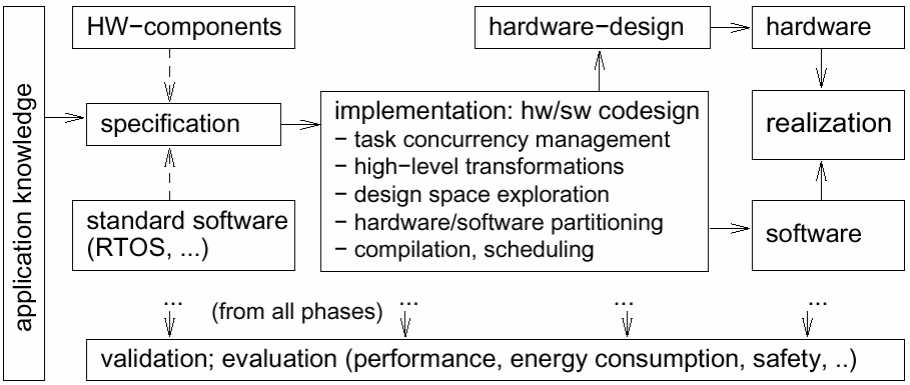
    $\Rightarrow$ periodic scheduling
  - A-periodic tasks with maximum arrival rate are called sporadic tasks.

A-periodic and periodic tasks together (2)

- Other solutions for the case that periodic tasks have hard deadlines, a-periodic tasks have soft deadlines.
  - Simplest solution: Background scheduling
    - A-periodic tasks are only executed when no periodic task is ready
    - Guarantees for periodic tasks do not change
    - Only applicable when load is not too high
  - Other solutions:
    - Define new periodic tasks, a so-called **server**
    - A-periodic tasks are executed during “execution time” of server process
    - Independent scheduling strategies possible for periodic tasks and a-periodic tasks “inside the server”

REVIEW

Overview



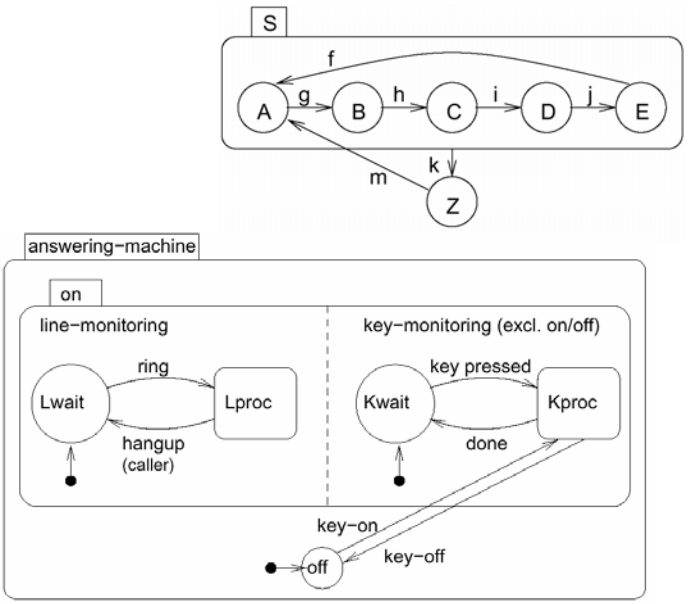
Specification

Communication/ local computations	**Shared memory**	**Message passing**	
		Synchronous	**Asynchronous**
Communicating finite state machines	StateCharts, StateFlow		SDL, MSCs
Data flow model ⊂ **Computational graphs**			Kahn process networks, SDF Petri nets
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA	
Discrete event (DE) model	VHDL, Simulink	Only experimental systems, e.g. distributed DE in Ptolemy	

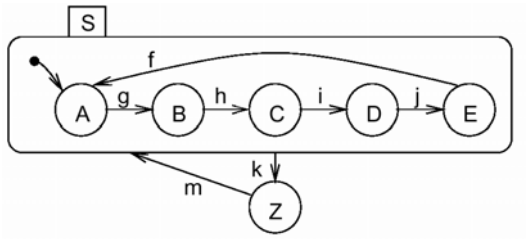
StateCharts

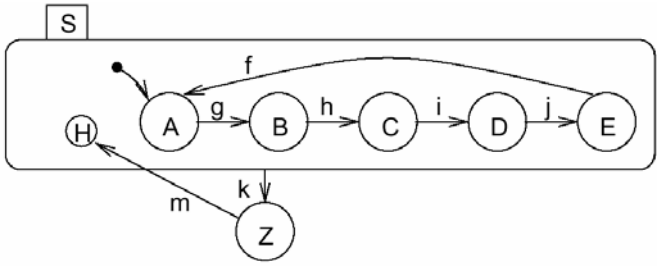
- Why?
  - Concise models of complex systems:
    StateCharts = FSMs + Hierarchy + Orthogonality + Broadcast communication
  - Commercial tools (StateMate, StateFlow, …)
- What?
  - Semantics
  - Virtual Prototyping (→ Matlab/Simulink/Stateflow)

StateCharts: Hierarchy + Orthogonality



StateCharts: Default-state mechanism

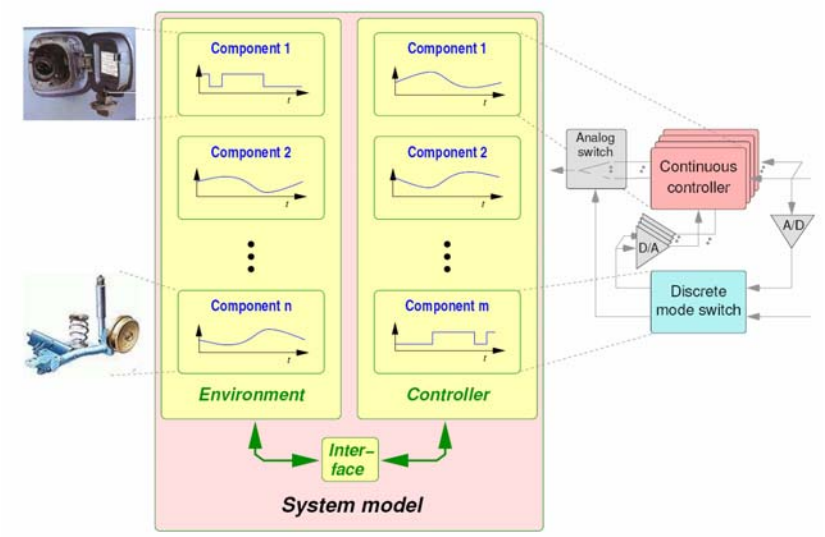




StateCharts: Semantics (Statemate)

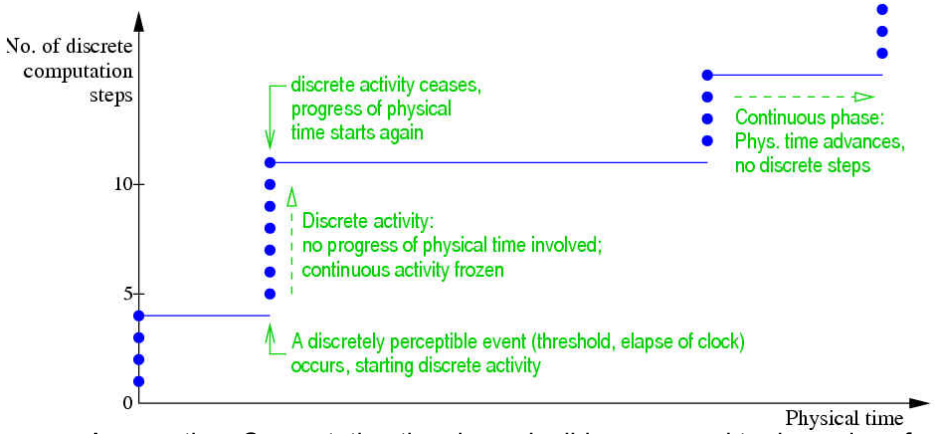
- Two stages
  - Preparation (for timeout events and scheduled actions)
  - Execution
- Preparation stage:
  - Fix scheduled actions that will be executed
  - Fix timeout events that will be generated
- Execution stage:
  - Determine the set of transitions to be taken based on internal and external events and on values of internal and external variables
  - Compute the next states and the reactions (evaluate right hand sides of assignments)
  - Transitions become effective, variables obtain new values.

Hybrid Systems



Hybrid Systems: The super-step time model

- Two-dimensional time:



- Assumption: Computation time is neglegible compared to dynamics of the environment.

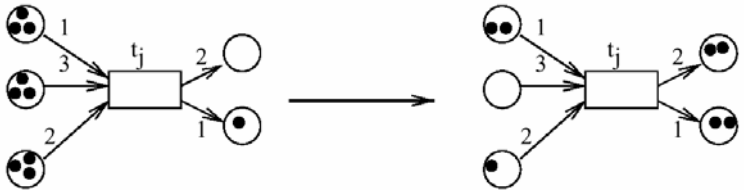
Petri Nets

- Why?
  - Modeling causal dependencies
  - Distributed systems
- What?
  - Reachability graph
  - Invariant generation
  - Deadlocks
- Advanced material:
  - Fairness
  - Nets with priorities
  - Predicate/transition nets

Activated transitions

Transition *t* is „activated“ iff

$$(\forall p \in {}^\bullet t : M(p) \geq W(p,t)) \wedge (\forall p \in t^\bullet : M(p) + W(t,p) \leq K(p))$$



Activated transitions can „take place“ or „fire“,
but don‘t have to.
The order in which activated transitions fire is not fixed
(it is non-deterministic).

Shorthand for changes of markings

Firing transition:

$$M'(p) = \begin{cases} M(p) - W(p,t), & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ M(p) + W(t,p), & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ M(p) - W(p,t) + W(t,p), & \text{if } p \in {}^\bullet t \cap t^\bullet \\ M(p) & \text{otherwise} \end{cases}$$

Let

$$\underline{t}(p) = \begin{cases} -W(p,t) \text{ if } p \in {}^\bullet t \setminus t^\bullet \\ +W(t,p) \text{ if } p \in t^\bullet \setminus {}^\bullet t \\ -W(p,t) + W(t,p) \text{ if } p \in t^\bullet \cap {}^\bullet t \\ 0 \end{cases}$$

$\Rightarrow$ $\forall p \in P$: $M'(p) = M(p) + \underline{t}(p)$

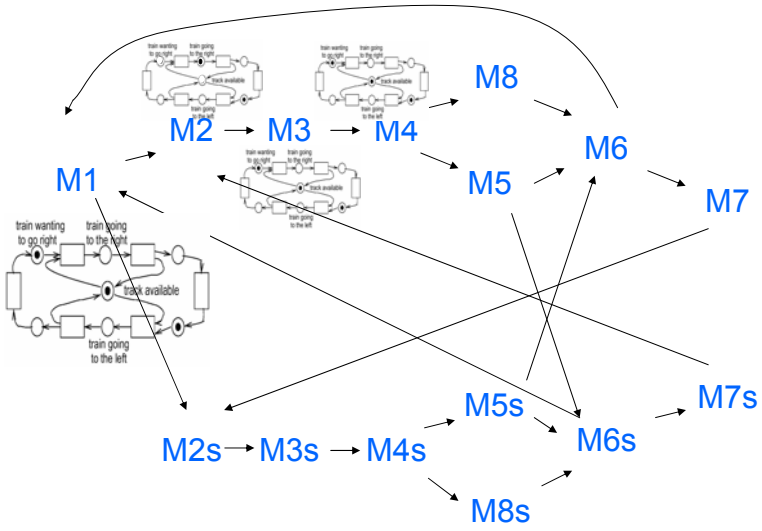
$\Rightarrow$ $M' = M + \underline{t}$ +: vector add

Reachability graph

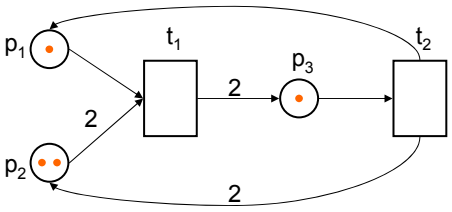
- M [t> M` iff $M' = M + \underline{t}$
- M [ε> M` iff M=M`
- M [qt> M` iff ∃ M`` . M [q> M`` and M`` [t> M`
- M [*> M‘ iff ∃ q . M [q> M`

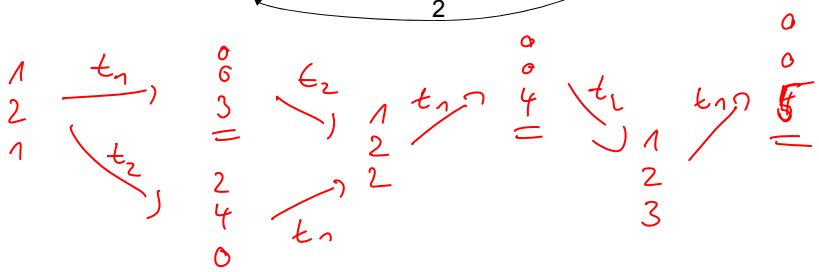
- **Reachability set** R(M) = {M‘ | M [*> M‘ }
- **Reachability graph** RG(M):
  nodes R(M), edges { (M,t,M`) | M [t> M´ }

Reachable markings



Unbounded Petri net





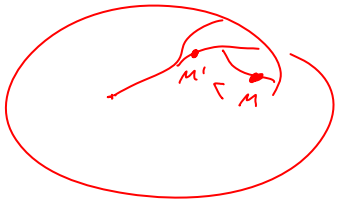
Boundedness

- A P/T net is unbounded iff there exist two reachable markings M, M', such that
  M[*>M' and M' > M.

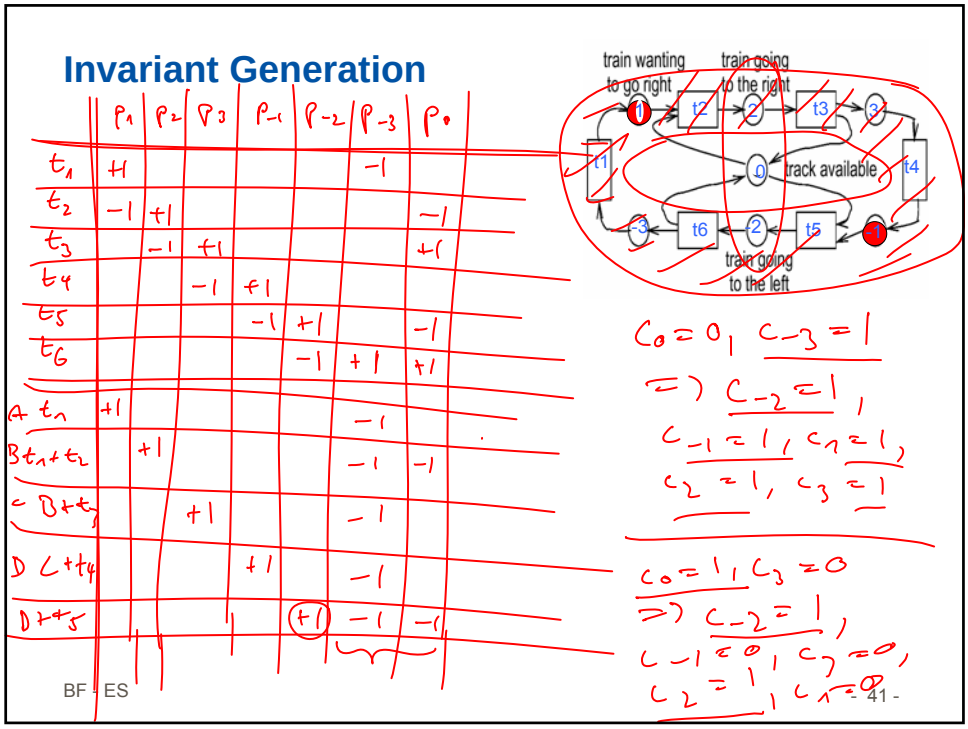
- Every infinite sequence of markings (M_i) contains a weakly monotonically growing infinite subsequence ($M`_i$), i.e., for j<k, $M`_j \leq M`_k$.

Algorithm for deciding boundedness

- Explore RG(M_0) depth-first:
  - If there exists a marking M‘ on the stack such that M‘<M,
    stop with result UNBOUNDED;
- If entire graph explored, return BOUNDED.



Invariant Generation

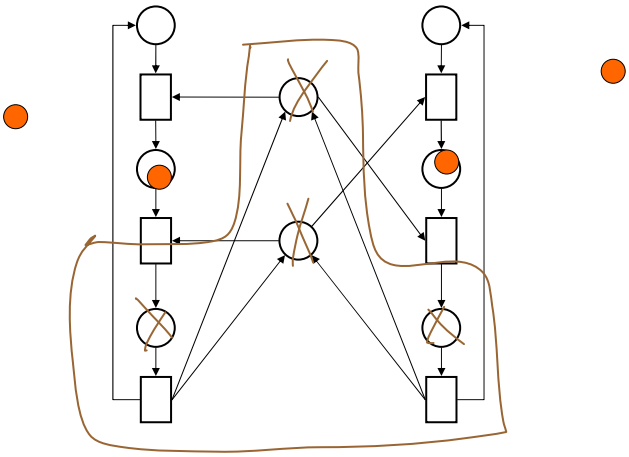


Invariants & boundedness

- **Theorem:**
  a) If R is an invariant and p ∈ R, then p is bounded.
  b) If a net is covered by invariants then it is bounded.

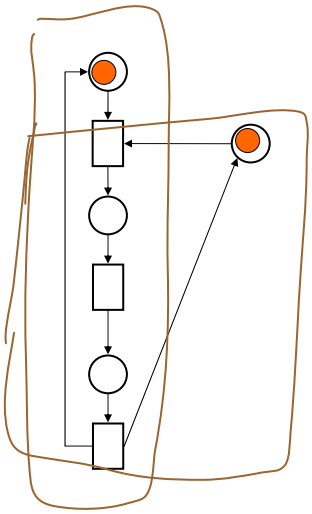
Deadlock

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



Structural properties: deadlock-traps

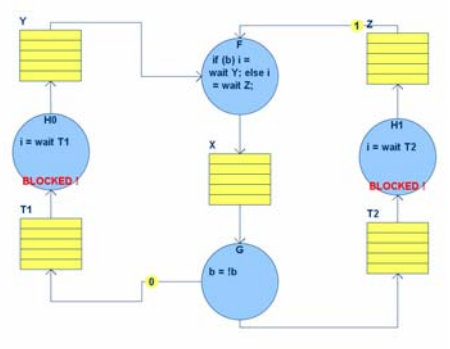
- A place set *S* is a **(static) deadlock** if every transition that adds token to S also removes token from *S*.
- A place set *S* is a **trap** if every transition that removes token from S also adds token to *S*.
- A P/T net has the **deadlock-trap property**, if every (static) deadlock contains a trap that is sufficiently marked in M_0.
- Every homogeneous P/T net with non-blocking weights that has the deadlock-trap property is deadlock-free.



Data flow models

- Why?
  - 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
- What?
  - Kahn process networks
    - Park‘s runtime scheduling algorithm
  - Synchronous data flow (SDF)
    - Lee/Messerschmitt‘s static scheduling algorithm

Kahn process networks

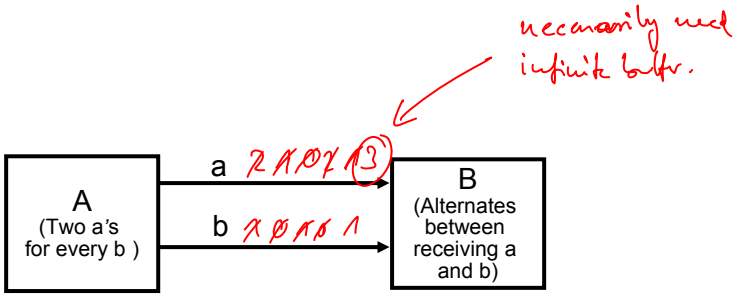


- Each node corresponds to one program/task;
- Communication is only via channels;
- Channels include FIFOs as large as needed;
- **Send operations are non-blocking, reads are blocking.**

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 may be impossible

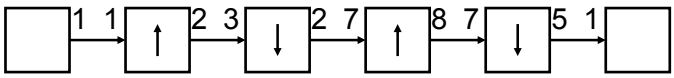


Parks' Scheduling Algorithm (1995)

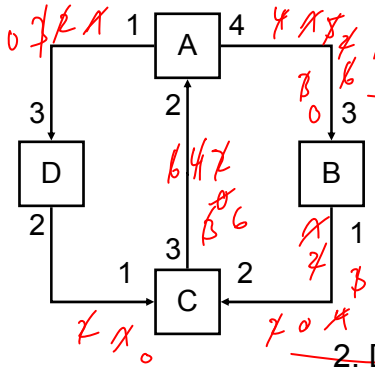
- Set a capacity on each channel ← FIFOs always bounded !
- Block a write if the channel is full
- Repeat
  - Run until deadlock occurs
  - If there are no blocking writes → terminate
  - Among the channels that block writes, select the channel with least capacity and increase capacity until producer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



SDF Scheduling Algorithm Lee/Messerschmitt 1987



d(CA)=6

1. Establish relative execution rates

2. Determine periodic schedule

BF - ES

- 51 -

Message Sequence Charts

- Why?
  - Modeling scenarios (instead of state-based behavior)
  - Parital-order semantics
  - ITU-T Standard Z.120
  - Integrated as *sequence diagrams* in UML
- What?
  - Synchronous vs. asynchronous concatenation
  - Regularity
- Advanced material:
  - Live sequence charts

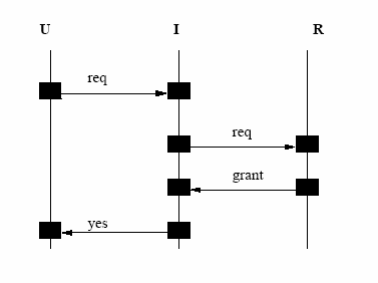
BF - ES

- 52 -

Message Sequence Charts

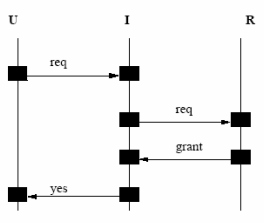
$Ch = (E, \leq, \lambda)$

- the elements of E_p are arranged along a life-line with the earlier elements appearing above the later elements.



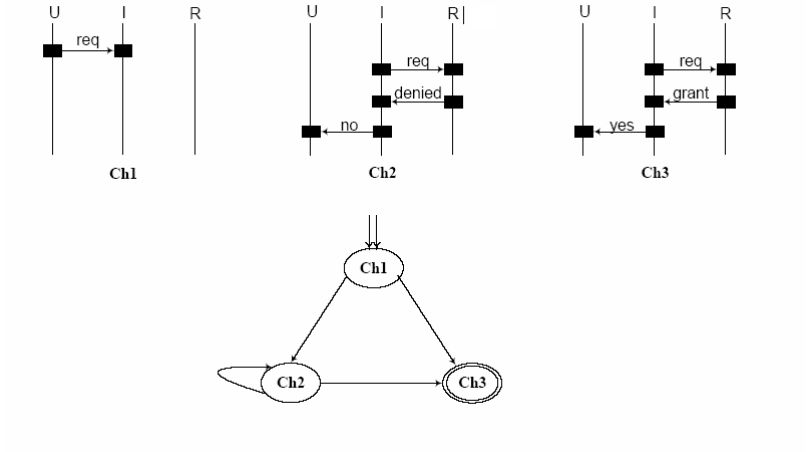
Language = Set of linearizations

- A **linearization** of a basic MSC is a sequence of actions $\lambda(e_0), \lambda(e_1), \ldots, \lambda(e_n)$ such that $E=\{e_0,e_1, \ldots,e_n\}$ and $e_0 \leq e_1 \leq \ldots \leq e_n$.
- Each basic MSC $Ch = (E, \leq, \lambda)$ defines a set of linearizations: $lin(Ch) \subseteq \Sigma^*$



lin (Ch) =
{ U!I(req) I?U(req)
I!R(req) R?I(req)
R!I(grant) I?R(grant)
I!U(yes) U?I(yes) }

Message Sequence Graphs



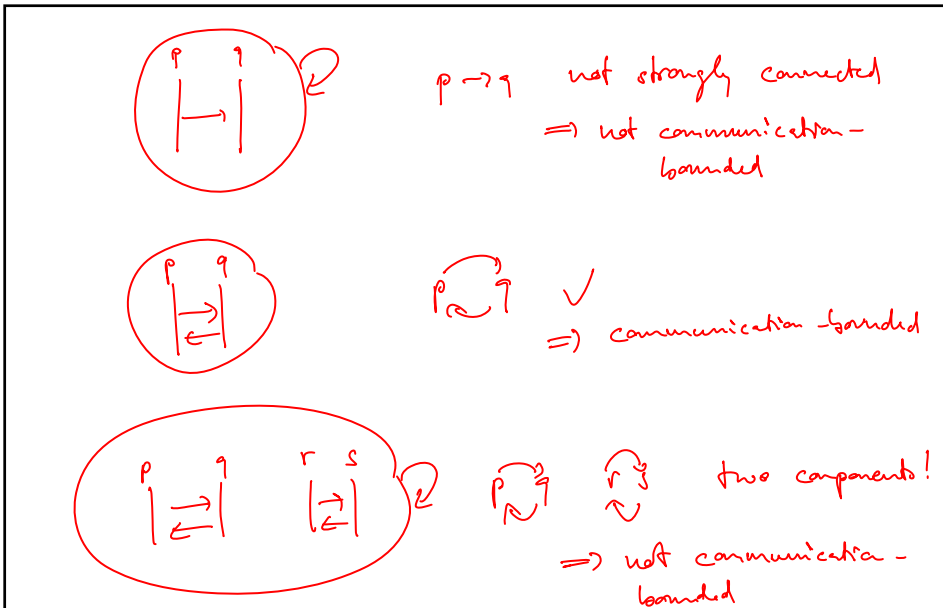
Synchronous vs. asynchronous concatenation

- **edges** in an MSG represent chart **concatenation:**
- **Synchronous concatenation** *Ch:Ch´*
  means that *all* the events in *Ch* must finish before *any* event in *Ch´* can occur.
- **Asynchronous concatenation** *Ch*1 ◦ *Ch*2
  is carried out at the level of life-lines.
- asynchronous concatenation of two charts is also a chart.
- synchronous concatenation of two charts
  may not result in a chart.
- asynchronous concatenation may lead to **non-regular languages**

Communication-boundedness

Communication-boundedness is a sufficient condition for regularity.

- The **communication graph** of a basic MSC is a directed graph, where the nodes are the processes, edge $p \rightarrow q$ if $p!q(m)$ for some m in chart.
- MSC is **communication-bounded** iff communication graph consists of a single strongly-connected component (+ isolated nodes)
- MSG is **communication-bounded** iff communication graph of all loops is communication-bounded.



VHDL

- Why?
  - Describing, simulating, synthesizing hardware
  - Standard in (European) industry
- What?
  - Entities, architectures
  - Multi-valued logic
  - Semantics
    - Transport delay model
- Advanced material:
  - IEEE 1164
  - Parameterized hardware

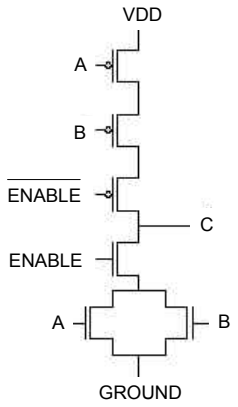

```
entity full_adder is
     port(a, b, carry_in: in Bit;  -- input ports
       sum,carry_out: out Bit); --output ports
     end full_adder;

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.

Multi-valued logic

- Many subcircuits can be effectively disconnected from the rest of the circuit (they provide „high impedance“ values to the rest of the circuit).
- Example: subcircuits with tri-state outputs.

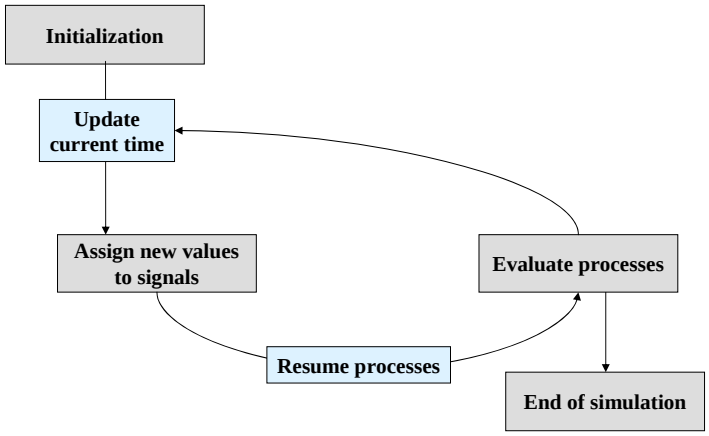
Example: Tristate NOR



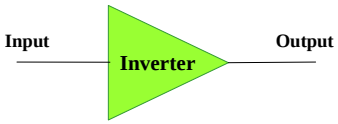
ENABLE = `0`
⇒ C is disconnected from the rest of the circuit

☞ We introduce signal value 'Z', meaning „high impedance “

Semantics

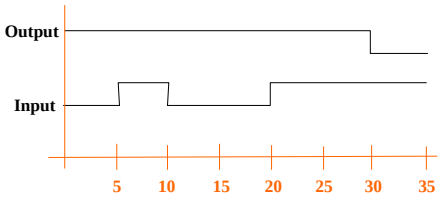


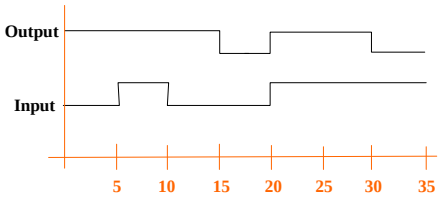
Inertial vs. transport delay model



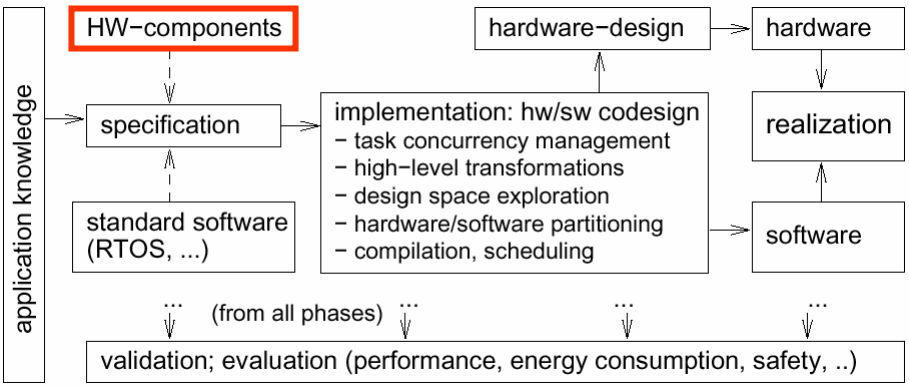

```
-- INERTIAL is the default
Output <= NOT input AFTER 10 ns;
```

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



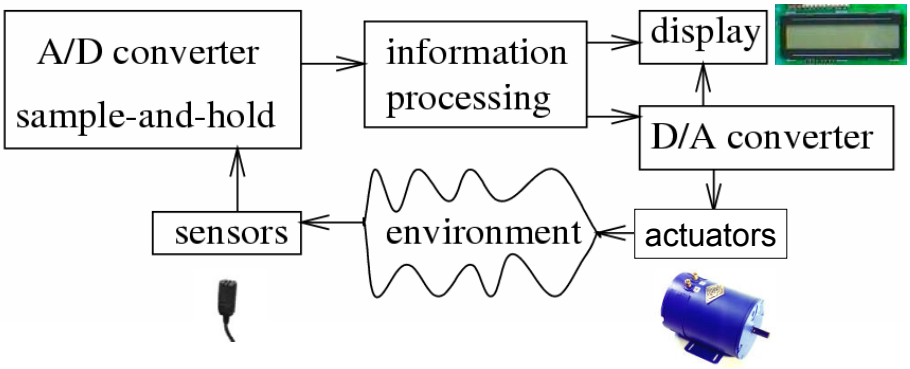


Overview of embedded systems design



Embedded System Hardware

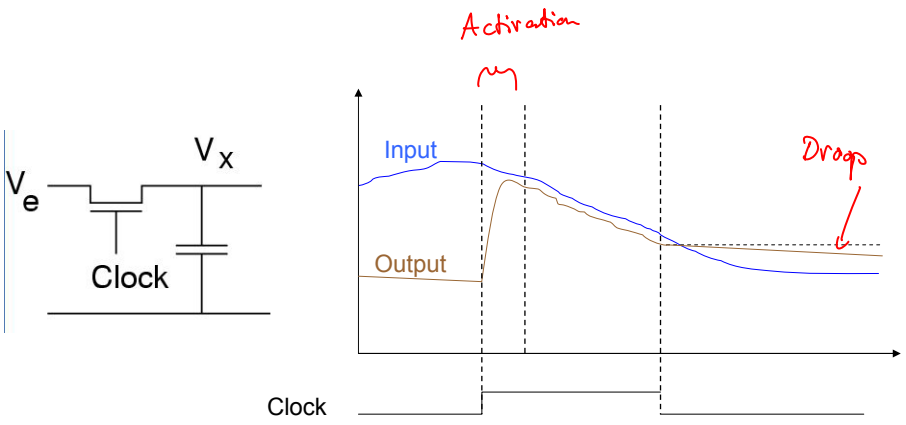
- Embedded system hardware is frequently used in a loop (***„hardware in a loop“***):



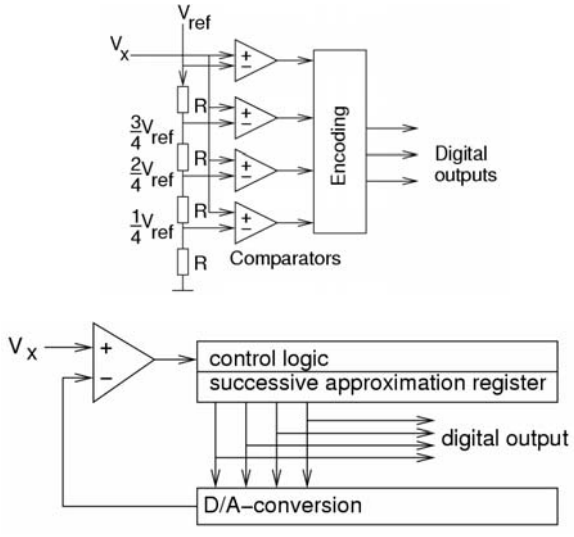
Sensors, A/D + D/A converters

- Why?
  - Embedded systems interact with physical environment.
- What?
  - Sample & hold circuits
  - A/D converters
  - D/A converters
- Advanced material:
  - Image sensors, …

Sample and Hold



Flash conversion vs. successive approximation



Information Processing

- Why?
  - Embedded systems must be efficient
  - Embedded processors need not be instruction set compatible with standard PCs
- What?
  - Power/energy efficiency
  - Code-size efficiency
  - Runtime efficiency
- Advanced material:
  - Reconfigurable logic, Multimedia processors, scratch pad memory, …

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:

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

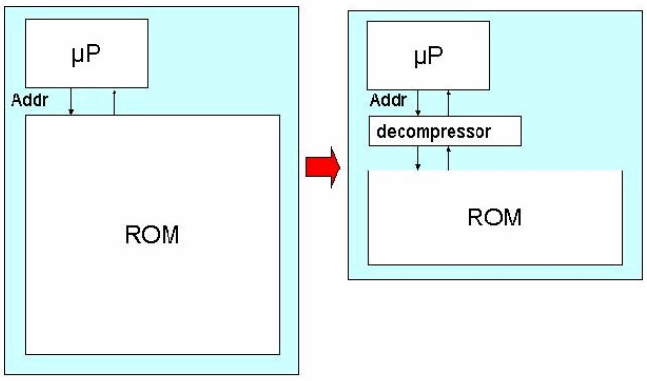
V_t : threshhold voltage

(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)

Code-size efficiency

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



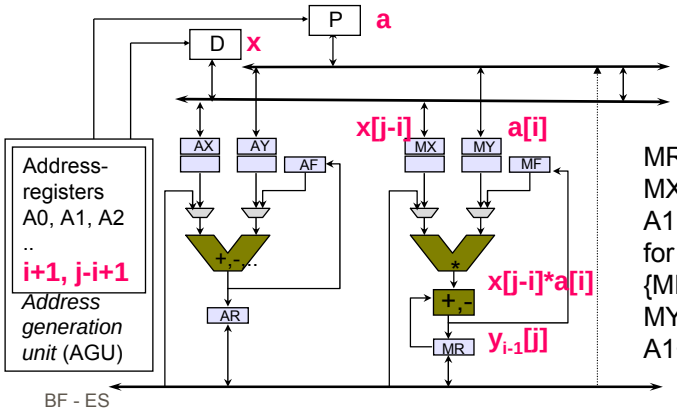
Run-time efficiency

- 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--}
```

Real-time communication

- Why?
  - Modular system development, support and evolution
  - Single network vs. wiring harness
- What?
  - Bus-master approach
  - TDMA, CSMA
  - Collision handling
- Advanced material:
  - FlexRay

TDMA - Time Division Multipl. Access

Operational principle:

- Progress of time is divided into TDMA rounds, within which the individual nodes have private time slots with different phase delay to start of the round; the slots are non-overlapping

Problems:

- Private slots waste bandwith
- Need for global clock synchronisation
- Number of nodes and their worst-case message lengths need to be fixed a priori
- This leads to either designs using huge safety margins or to complex interference between node performance and TDMA setup (lacking separation of concerns between computation and communication)

Carrier Sense Multiple Access (CSMA)

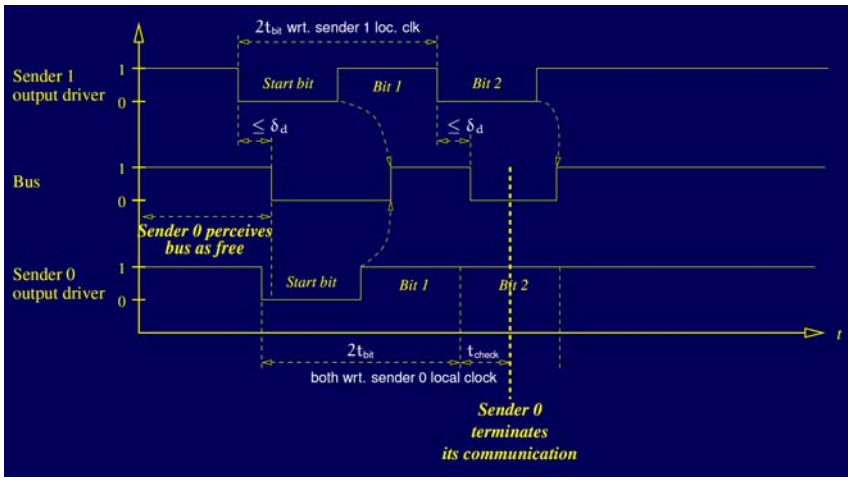
Operational principle:

- If communication medium is idle then send a message (node decides on its own — no global authority)

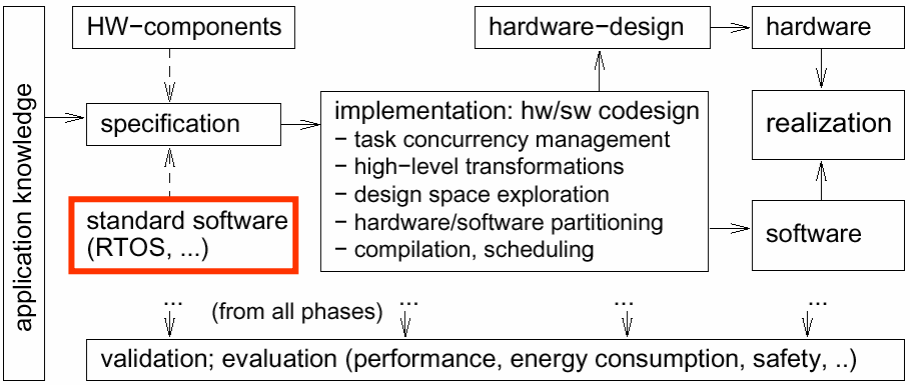
Problems:

- Multiple nodes may start almost synchronously, leading to collision on the medium
- Message may be crippled
- Message may be overwritten and thus not delivered
- If message delivery is vital (std. in ES) then collision has to be resolved
  - Collision detection or collision avoidance, arbiting the bus such that at least one of thecolliding messages is delivered uncrippled

Collision handling



Overview of embedded systems design



Scheduling

- Why?
  - Scheduling key issue in implementing RT-systems
  - Different algorithms with different assumptions and cost
- What?
  - Aperiodic scheduling
  - Periodic scheduling
  - Scheduling with resource constraints
- Advanced material:
  - Priority ceiling protocol

Aperiodic scheduling: EDF – Earliest Deadline First

- EDF: At every instant execute the task with the earliest absolute deadline among all the ready tasks.
- **Theorem (Horn '74):**
  Given a set of n independent task with arbitrary arrival times, any algorithm that at every instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

Aperiodic scheduling: Non-preemptive version

- **Theorem** (Jeffay et al. '91): EDF is an optimal *non-idle* scheduling algorithm also in a non-preemptive task model.
- When idle schedules are allowed: problem is NP-hard.
- Possible approaches:
  - Heuristics
  - Bratley's algorithm: branch-and-bound

Aperiodic scheduling: : Scheduling with precedence constraints

- Non-preemptive scheduling with non-synchronous arrival times, deadlines and precedence constraints is NP-hard.
- Restrictions:
  - Consider synchronous arrival times (all tasks arrive at 0)
  - Allow preemption.
- **Theorem (Lawler 73):**
  LDF (Latest Deadline First) is optimal wrt. maximum lateness.

Periodic scheduling: EDF

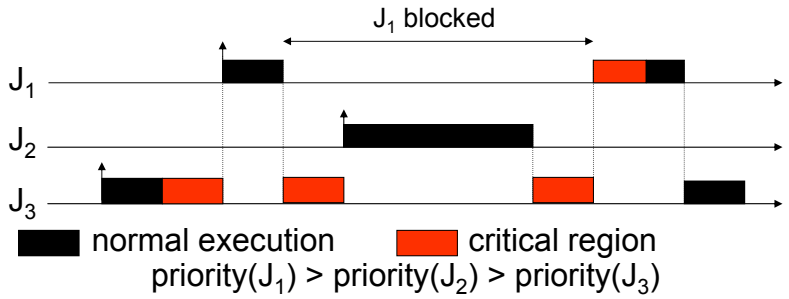
- **Theorem: A set of periodic tasks τ_1, ..., τ_n with $D_i = T_i$ is schedulable with EDF iff $U \leq 1$.**
- EDF is applicable to both periodic and a-periodic tasks.
- If there are only periodic tasks, priority-based schemes like “rate monotonic scheduling (RM)” (see later) are often preferred, since
  - They are simpler due to fixed priorities
    $\Rightarrow$ use in “standard OS” possible
  - sorting wrt. to deadlines at run time is not needed

Rate monotonic scheduling (RM)

- Rate monotonic scheduling (RM) (Liu, Layland '73):
  - Assign fixed priorities to tasks τ_i:
    - priority(τ_i) = $1/T_i$
    - I.e., priority reflects release rate
  - Always execute ready task with highest priority
  - Preemptive: currently executing task is preempted by newly arrived task with shorter period.

- **Theorem (Liu, Layland, 1973):**
  RM is optimal among all fixed-priority scheduling algorithms.

The priority inversion problem



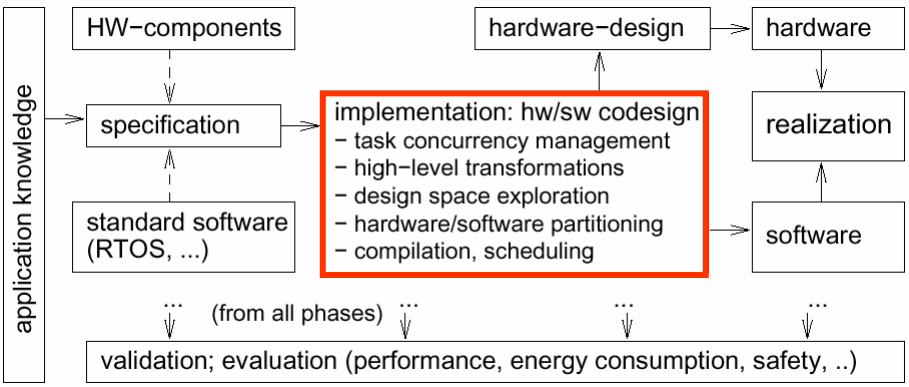
- Blocking time equal to length of critical section + computation time of J_2.
- Unbounded time of priority inversion, if J_3 is interrupted by tasks with priority between J_1 and J_3 during its critical region.

Coping with priority inversion: The priority inheritance protocol

Idea of **priority inheritance protocol**:

- If a task J_h blocks, since another task J_l with lower priority owns the requested resource, then J_l inherits the priority of J_h.
- When J_l releases the resource, the priority inheritance from J_h is undone.
- Rule: Tasks always inherit the highest priority of tasks blocked by it.

Coming up…



Thursday

- Midterm exam: December 18, 2008, HS I, Math building, 16:15 - 17:45
- Open book: bring any handwritten or printed notes, or any books you like.
- Please bring your ID.