Embedded Systems
Overview of embedded systems design  REVIEW

- HW-components
  - specification
    - standard software (RTOS, ...)

- hardware-design
  - implementation: hw/sw codesign
    - task concurrency management
    - high-level transformations
    - design space exploration
    - hardware/software partitioning
    - compilation, scheduling

- hardware
  - realization
    - software

- application knowledge

- validation; evaluation (performance, energy consumption, safety, ..)
Scheduling

- Support for multi-tasking/multi-threading – several tasks to run on shared resources
- Task ~ process – sequential program
- Resources: processor(s) + memory, disks, buses, communication channels, etc.
- Scheduler assigns shared resources to tasks for durations of time
- Most important resource(s) – processor(s)
- Scheduling – mostly concerned with processor(s)
  - Online – scheduling decisions taken when input becomes available
  - Offline – schedule computed with complete input known
- Other shared resources with exclusive access complicate scheduling task
Point of departure: Scheduling general IT systems

- In general IT systems, not much is known about the set of tasks a priori
  - The set of tasks to be scheduled is dynamic:
    - new tasks may be inserted into the running system,
    - executed tasks may disappear.
    - Tasks are activated with unknown activation patterns.
  - The power of schedulers thus is inherently limited by lack of knowledge – only online scheduling is possible
Scheduling processes in ES:
The difference in process characterization

- Most ES are “closed shops”
  - Task set of the system is known
  - at least part of their activation patterns is known
    - periodic activation in, e.g., signal processing
    - maximum activation frequencies of asynchronous events determinable from environment dynamics, minimal inter-arrival times
  - Possible to determine bounds on their execution time (WCET)
    - if they are well-built
    - if we invest enough analysis effort
- Much better prospects for guaranteeing response times and for delivering high-quality schedules!
Scheduling processes in ES: Differences in goals

- In classical OS, quality of scheduling is normally measured in terms of performance:
  - Throughput, reaction times, … in average case

- In ES, the schedules do often have to meet stringent quality criteria under all possible execution scenarios:
  - A task of an RTOS is usually connected with a deadline. Standard operating systems do not deal with deadlines.
    - There are **hard** deadlines which have to be fulfilled under all circumstances and
    - “**soft**” deadlines” which should be fulfilled if possible
  - Scheduling of an RTOS has to be **predictable**.
  - Real-time systems have to be designed for **peak load**. Scheduling for meeting deadlines should work for all anticipated situations.
Three types of constraints for real-time tasks:

- Timing constraints
- Precedence constraints (priority c.)
- Mutual exclusion constraints on shared resources
Model-based Code Generation: Esterel Scade

Daniel Kästner
kaestner@absint.com
AbsInt Angewandte Informatik GmbH
Model-based Software Development

- Model is software specification.
- Hardware/Software codesign.
- Prototyping.
- Formal verification.

- **Automated** & integrated development tools:
  - Simulation.
  - Documentation.
  - Automatic code generation.

- **Automated** & integrated verification and test methods
  - Model checking
  - Static system analysis
  - Synthesis of test suites
Model-based Software Development

SCADE programs (~Esterel/Lustre)

C Code

Binary Code

Esterel Scade Suite
- SCADE language
- SyncCharts/SSM

Generator

Compiler

Compiler

SymTA/S
- System-level Schedulability Analysis

- aiT WCET Analyzer
- StackAnalyzer
SyncCharts

- Visual formalism for describing states and transitions of a system in a modular fashion.

- Extension of state-transition diagrams (Mealy/Moore automata)
  - Hierarchy
  - Modularity
  - Parallelism

- Is fully deterministic.

- Tailored to control-oriented applications (drivers, protocols).

- Implements synchronous principle.
Synchronous Programming

- Important requirement: guaranteeing **deterministic** behavior.
- Time is divided into discrete ticks (also called cycles, steps, instants).
- Simple implementation: **sampling / cyclic executive**:
  
  ```
  Initialize Memory
  Foreach period do
    Read Inputs
    Compute Outputs
    Update Memory
  ```
  
- Verification of timing behavior: prove that the **worst-case execution time** (WCET) of any reaction fits between two iteration of the cyclic executive.
- Implicit assumption: presence of a **global clock**. This makes application in **distributed** environments difficult.
Overview

- **StateCharts:**
  - First, and probably most popular formal language for the design of reactive systems.
  - Focus on specification and design, not designed as a programming language.
  - Determinism is not ensured.
  - No standardized semantics.

- Programming languages for designing reactive systems:
  - **ESTEREL** [Berry]: textual imperative language.
  - **LUSTRE** [Caspi, Halbwachs]: textual declarative language. Tailored to data-flow oriented systems (e.g. regulation systems).
  - **SCADE** [Esterel Inc.]. Enhanced LUSTRE, graphical and textual formalism.
  - **SyncCharts / SSM**: Graphical formalism corresponding to ESTEREL.
Concurrency vs. Parallelism

- Concurrency is central to embedded systems. A computer program is said to be concurrent if different parts of the program conceptually execute simultaneously.

- A program is said to be parallel if different parts of the program physically execute simultaneously on distinct hardware (multi-core, multi-processor or distributed systems).
Petri Nets
Petri’s nets - complex foundations for simple models

For his nets, Carl Adam Petri has made an attempt to combine automata from theoretical CS, insights from physics, and pragmatic expertise from engineers:

- **state is distributed, transitions are localised** (space is relevant)
- **local causality replaces global time** (time as a derived concept)
- **subsystems interact by explicit communication**
  (information transport is as relevant as information processing)

**Engineers can often ignore the background - Petri nets just work!**

*but the background explains why things work, why concepts from other disciplines, such as logic, have been integrated into Petri nets so easily, and why foundational research has to continue*
Application areas

- modelling, analysis, verification of distributed systems
- automation engineering
- business processes
- modeling of resources
- modeling of synchronization
Key Elements

- **Conditions**
  Either met or not met. Conditions represent “local states”. Set of conditions describes the potential state space.

- **Events**
  May take place if certain conditions are met. Event represents a state transition.

- **Flow relation**
  Relates conditions and events, describes how an event changes the local and global state.

- **Tokens**
  Assignments of tokens to conditions specifies a global state.

Conditions, events and the flow relation form a bipartite graph (graph with two kinds of nodes).
Example 2: Synchronization at single track rail segment

- mutual exclusion: there is at most one train using the track rail

"Preconditions" of x fulfilled

"Postcondition" of x fulfilled
Playing the „token game“: dynamic behavior

Diagram:

- Train wanting to go right
- Train going to the right
- Track available
- Train going to the left
Playing the „token game“: dynamic behavior
Playing the „token game“: dynamic behavior

REVIEW
Conflict for resource „track“: two trains competing
A Petri nets is nondeterministic

When multiple transitions are enabled at the same time, any one of them may fire.

If a transition is enabled, it may fire (but it doesn't have to).
Condition/event Petri nets

Def.: \( N=(C,E,F) \) is called a net, iff the following holds
1. \( C \) and \( E \) are disjoint sets
2. \( F \subseteq (C \times E) \cup (E \times C) \); is binary relation, ("flow relation")

Def.: Let \( N \) be a net and let \( x \in (C \cup E) \).
   \( \bullet x := \{ y \mid y F x \} \) is called the set of preconditions.
   \( x^* := \{ y \mid x F y \} \) is called the set of postconditions.

Example:
Def.: Let \((c,e) \in C \times E\). \((c,e)\) is called a **loop** iff \(cFe \land eFc\).

Def.: Net \(N=(C,E,F)\) is called **pure**, if \(F\) does not contain any loops.
Simple nets

- **Def.:** A net is called **simple** if no two nodes $n_1$ and $n_2$ have the same pre-set and post-set.

- Example (not simple):
Thalys trains between Cologne, Amsterdam, Brussels and Paris.
Example Thalys trains: more complex

- Thalys trains between Cologne, Amsterdam, Brussels and Paris.
- Synchronization at Brussels and Paris

- Places 3 and 10: trains waiting in A and C
- Transitions 9 and 2: trains driving from A and C to Brussels
- T1: connecting the two trains
- Break for driver P13
- T5 synchronization with trains at Gare du Nord
Realistic scenarios need more general definitions

- More than one token per condition, capacities of places
- weights of edges
- state space of Petri nets may become infinite!

Producer

Consumers
Place/transition nets

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 \( p \in P \) and transitions \( t \in T \)
2. \( K: P \rightarrow (\mathbb{N}_0 \cup \{\omega\}) \setminus \{0\} \) denotes the **capacity** of places
   \((\omega)\) 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

**defaults:**
- \( K = \omega \)
- \( W = 1 \)
Example

- $P = \{p_1, p_2, p_3\}$
- $T = \{t_1, t_2\}$
- $F = \{(p_1, t_1), (p_2, t_2), (p_3, t_1), (t_1, p_2), (t_2, p_1), (t_2, p_3)\}$
- $W = \{(p_1, t_1) \rightarrow 2, (p_2, t_2) \rightarrow 1, (p_3, t_1) \rightarrow 1, (t_1, p_2) \rightarrow 1, (t_2, p_1) \rightarrow 2, (t_2, p_3) \rightarrow 1\}$
- $m_0 = (2, 0, 1)$
Reachability

Reachability graph

$m_0 = (2, 0, 0)$
Is there a sequence of transition firings such that $M \rightarrow M'$?
From conditions to resources (1)

- c/e-systems model the flow of information, at a fundamental level (true/false)

- there are natural application areas for which the flow/transport of resources and the number of available resources is important (data flow, document-/workflow, production lines, communication networks, ..)

- place/transition-nets are a suitable generalisation of c/e-systems:
  - state elements represent places where resources (tokens) can be stored
  - transition elements represent local transitions or transport of resources
From conditions to resources (2)

- a transition is enabled if and only if
  - sufficient resources are available on all its input places
  - sufficient capacities are available on all its output places

- a transition occurrence
  - consumes one token from each input place and
  - produces one token on each output place
Computing changes of markings

- “Firing” transitions \( t \) generate new markings on each of the places \( p \) according to the following rules:

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

When a transition \( t \) fires from a marking \( M \), \( w(p, t) \) tokens are deleted from the incoming places of \( t \) (i.e. from places \( p \in \bullet t \)), and \( w(t, p) \) tokens are added to the outgoing places of \( t \) (i.e. to places \( p \in t^* \)), and a new marking \( M' \) is produced.
Activated transitions

- Transition $t$ is „activated“ iff

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

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).
Boundedness

- A place is called **k-safe** or **k-bounded** if it contains in the initial marking $m_0$ and in all other reachable from there markings at most $k$ tokens.

- A net is **bounded** if each place is bounded.

- **Boundedness**: the number of tokens in any place cannot grow indefinitely
- **Application**: places represent buffers and registers (check there is no overflow)

- A Petri net is inherently bounded if and only if all its reachability graphs (i.e. reachability graphs with all possible starting states) all have a **finite number of states**.
Liveness

- A transition $T$ is live if in any marking there exists a firing sequence such that $T$ becomes enabled.
- An entire net is live if all its transitions are live.
- Important for checking deadlock.

Live? NO

Live? YES
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.
Shorthand for changes of markings

Firing transition:

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

Let

\[
t(p) = \begin{cases} 
-W(p, t) & \text{if } p \in \cdot t \setminus t \\
+W(t, p) & \text{if } p \in t \setminus \cdot t \\
-W(p, t) + W(t, p) & \text{if } p \in \cdot t \cap t \\
0 & \end{cases}
\]

\[\forall p \in P: M'(p) = M(p) + t(p)\]

\[M' = M + t\]

+: vector add
Matrix $N$ describing all changes of markings

$$
\begin{cases}
-W(p,t) & \text{if } p \in t \setminus t^* \\
+W(t, p) & \text{if } p \in t^* \setminus t \\
-W(p,t) + W(t, p) & \text{if } p \in t^* \cap t^* \\
0 & \text{otherwise}
\end{cases}
$$

Def.: Matrix $N$ (incidence matrix) of net $N$ is a mapping

$$
N : P \times T \rightarrow Z \text{ (integers)}
$$

such that $\forall t \in T$: $N(p,t) = t(p)$

Component in column $t$ and row $p$ indicates the change of the marking of place $p$ if transition $t$ takes place.
Incidence matrix

incidence matrix $N$ of a pure (no elementary loops) place/transition-net:

$$N_{p,t} := \begin{cases} -W(t, p), \text{ arc from } p \text{ to } t \\ +W(t, p), \text{ arc from } t \text{ to } p \\ 0, \text{ otherwise} \end{cases}$$

Contribution of $t$ on $p$
Example: $N =$

![Diagram of a network with nodes and edges]

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
<th>$t_9$</th>
<th>$t_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_2$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_3$</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_4$</td>
<td></td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_5$</td>
<td></td>
<td></td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_6$</td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_7$</td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_9$</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$p_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{12}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>$p_{13}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>
State equation

\[ N_{t+1} = \begin{bmatrix} p_1 & \cdots & p_n \end{bmatrix} \begin{bmatrix} r_1 & \cdots & r_n \end{bmatrix} + \begin{bmatrix} \sum_{i=1}^{n} p_i \cdot \tau_i \end{bmatrix} \]

\[ m' = m_0 + N \cdot \tau_i \]

\[ h' = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \\ V_2 \end{bmatrix} \]

\[ \begin{bmatrix} f \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \begin{bmatrix} -2 \end{bmatrix} \]
State equation

reachability graph
Computation of Invariants

We are interested in subsets $R$ of places whose number of labels remain invariant under firing of transitions:

- e.g. the number of trains commuting between Amsterdam and Paris (Cologne and Paris) remains constant

Important for correctness proofs
Computation of Invariants

$P_1 + P_2 = 2$
Place - invariants

Standardized technique for proving properties of system models

For any transition $t_j \in T$ we are looking for sets $R \subseteq P$ of places for which the accumulated marking is constant:

$$\sum_{p \in R} t_{-j}(p) = 0$$

Example:
Characteristic Vector

\[ \sum_{p \in R} t_j(p) = 0 \]

Let:
\[ c_R(p) = \begin{cases} 1 & \text{if } p \in R \\ 0 & \text{if } p \not\in R \end{cases} \]

\[ \Rightarrow \sum_{p \in R} t_j(p) = t_j \cdot c_R = \sum_{p \in P} t_j(p) c_R(p) = 0 \]

Scalar product
Condition for place invariants

\[
\sum_{p \in R} t_j(p) = t_j \cdot c_R = \sum_{p \in P} t_j(p) c_R(p) = 0
\]

Accumulated marking constant for all transitions if

\[
t_1 \cdot c_R = 0
\]

\[
\ldots \ldots \ldots
\]

\[
t_n \cdot c_R = 0
\]

Equivalent to \( N^T c_R = 0 \) where \( N^T \) is the transposed of \( N \)
More detailed view of computations

System of linear equations.

Solution vectors must consist of zeros and ones.

Different techniques for solving equation system (Gauss elimination, tools e.g. Matlab, …)
Application to Thalys example

$$N^T c_R = 0,$$ with $$N^T = \begin{bmatrix}
1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

$$c_{R,1} = (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0\ 0\ 0)$$
Interpretation of the 1\textsuperscript{st} invariant

\[ C_{R,1} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \]

Characteristic vector describes places for Cologne train. We proved that: the number of trains along the path remains constant.
Application to Thalys example

\[ N^T c_R = 0, \text{ with } N^T = \]

\[
\begin{bmatrix}
  t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \\
  1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

\[ c_{R,2} = (1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0) \]
Interpretation of the 2\textsuperscript{nd} invariant

\[ c_{R,2} = (1,0,0,0,1,1,0,0,1,1,1,0,0) \]

We proved that:
None of the Amsterdam trains gets lost.
Application to Thalys example

\[ N^T c_R = 0, \text{ with } N^T = \]

\[
\begin{array}{cccccccccccc}
\hline
 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} & p_{12} & p_{13} \\
 t_1 & 1 & -1 &  &  &  &  &  &  & -1 &  &  & 1 \\
t_2 & 1 & -1 &  &  &  &  &  &  &  & 1 &  &  \\
t_3 & 1 & -1 &  &  &  &  &  &  &  &  & 1 &  \\
t_4 & -1 &  &  &  &  &  &  &  &  &  &  &  \\
t_5 & 1 & -1 &  &  &  &  &  &  &  &  &  &  \\
t_6 & -1 &  &  &  &  &  &  &  &  &  &  &  \\
t_7 & 1 & -1 &  &  &  &  &  &  &  &  &  &  \\
t_8 & 1 & -1 &  &  &  &  &  &  &  &  &  &  \\
t_9 & 1 & -1 &  &  &  &  &  &  &  &  &  &  \\
t_{10} & 1 & -1 &  &  &  &  &  &  &  &  &  &  \\
\hline
\end{array}
\]

\[ c_{R,2} = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0) \]
Solution vectors for Thalys example

We proved that:

- the number of trains serving Amsterdam, Cologne and Paris remains constant.
- the number of train drivers remains constant.

\[ c_{R,1} = (1 1 1 1 1 1 0 0 0 0 0 0 0) \]
\[ c_{R,2} = (0 0 0 0 0 0 1 1 0 0 0 1 0) \]
\[ c_{R,3} = (0 0 0 0 0 0 0 0 1 1 0 0 1) \]
\[ c_{R,4} = (1 0 0 0 1 1 0 0 1 1 1 0 0) \]
Solution vectors for Thalys example

It follows:
• each place invariant must have at least one label at the beginning, otherwise “dead”
• at least three labels are necessary in the example
Place Invariants – Animation

http://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/aalst/trafficlight2_PI.swf
$N^T c_R = 0$, with $N^T = \begin{array}{cccccc}
T1 & P1 & P2 & P3 & P4 & P5 & P6 \\
T2 & & & & & & \\
T3 & & & & & & \\
T4 & & & & & & \\
T5 & & & & & & \\
\end{array}$