Embedded Systems Development

Lecture 3
SyncCharts & Esterel

Daniel Kästner
AbsInt Angewandte Informatik GmbH
kaestner@absint.com
SyncCharts: Example from last week
SyncCharts: Advanced Constructs

- Immediate transition
  - Syntax: \#<trigger>/<effect>
  - The trigger may be satisfied as soon as the state is entered: An active state waits for the satisfaction of the trigger of one of its outgoing transitions, at an instant strictly posterior to its entering, or immediately in case of an immediate transition.

- Count delays for transitions
  - Syntax: <factor><trigger>/<effect>
  - <factor> is the natural number of instants a transition must be active before it is executed. These active instants need not be consecutive, but the source state (S1) must be active all the time.
Suspension

- A suspension is associated with a trigger. If the trigger is satisfied the reaction is suspended in the target state: the execution of the preempted state is frozen.
- Notation: \( T \), or: \( S^T \)
- Note: aborts take priority over suspensions.
Run Modules

- Macrostates are states that have their own behavior (also called processes). They can be abstracted as modules, similarly to procedures in programming languages.
- Modules are instantiated by using run modules (corresponding to procedure calls).
- A signal interface renaming has to be defined for run modules.

Module M with interface:
- input I;
- output O;

... used as a run module with the following signal binding:
- signal S1 / I;
- signal S2 / O;

Main module:
- signal S1,S2;
Conditional Connector

- Introduction of conditional pseudostate
- Concise representation of scenarios where a common trigger is shared by several outgoing transitions.
- All departing transitions are immediate.
- One departing “default” transition without condition must be present.
History Connector

- Directly attached to macrostates
- Only incoming transitions can connect
- The previous state of the macrostate is restored when it is entered through a history connector.
Example: Watchdog
Example: Watchdog

Diagram:
- IsON
- Counter
- Alarm
  - Signal C1, C2

States:
- B0
- B0'
- B1
- B1'
- B2
- B2'

Signals:
- reset
- set
- inhib
Example: Watchdog

Watchdog

IsON

Counter

signal C1, C2

Alarm

B0

B1

B2

C0/C1

C1

C2

C0/C1

C1/C2

inhib

C0+

set

reset
Example: Watchdog
Example: Watchdog

Watchdog

IsON

Counter

\[ \text{C0}^+ \]

set

reset

B2'

C2

C1/C2

C1

C0/C1

C0

inhib

signal C1,C2

Alarm
Example: Watchdog

Watchdog

IsON

Counter

set
reset

B2
C2
C1/C2
C1
C0/C1
C0

B1

signal C1,C2

B0

Alarm

inhib

C0+
Example: Watchdog
Example: Watchdog
Example: Watchdog

Watchdog

IsON

Counter

C0/C1

B0'

signal C1,C2

B1

B2'

C1/C2

C0

C2

B1'

B0

set

reset

C0+

Alarm

inhibit

Example: Watchdog
Example: Watchdog

Watchdog

IsON

Counter

inhib

signal C1, C2

Alarm

inhib+

C0+

set

reset

B2'

C2

C1/C2

B1'

C1

C0/C1

B0'

B0

B1

B2
Example: Watchdog
Example: Watchdog

Watchdog

IsON

Counter

B2'

C2

C1/C2

B1'

C1

C0/C1

B0'

C0

B0

B2

set

reset

Alarm

inhib

inhib+

C0+
Example: Watchdog
Example: Watchdog

**Watchdog**

**IsON**

**Counter**

- B2' → C2 → C1/C2 → C1 → C0/C1 → C0
- B1' → signal C1,C2
- B0' → inhibit

**Alarm**

**C0+**

**set**

**reset**
Example: Watchdog
Example: Watchdog

Watchdog

IsON

Counter

B2' C2 C1/C2 C1 C0/C1 C0

set+ reset

C1/C2

signal C1,C2

inhib

Alarm

set+

B0'

B1'

B2
Example: Watchdog

Watchdog

IsON

Counter

B2'  C2  C1/C2  C1  C0/C1  C0
B1'  B0'

signal C1,C2

set
reset

inhibit

C0+

Alarm
Example: Watchdog

- reset
- inhib
- C0
Computing a Reaction – Definitions

- Each SSM can be represented by unique macrostate, called Top, which designates the root of the state containment hierarchy.

- A state-transition graph (STG) $G$ is a tuple $G = (S, ini)$ where $S$ is a non-empty set of reactive cells, and $ini$ is the initial reactive cell.

- A reactive cell is a tuple $C=(B,R,S)$ such that its body $B$ is a state (simple state or macrostate) and $R$ the set of all its outgoing transitions. The status $S$ of a reactive cell is either IDLE or ACTIVE.

- A macrostate $M$ is a quadruple $M=(G,I,O,L)$ composed from a non-empty set $G$ of STGs, and three possibly empty sets of signals: input signals ($I$), output signals ($O$), local signals ($L$).

- A transition has a destination and a label. The destination is a reactive cell, the label is composed of three optional fields: a trigger, a guard, an effect. A transition is denoted as a quadruple $R=<type, trigger, effect, targetID>$. Feasible types are $sA$ (strong abort), $wA$ (weak abort), $nT$ (normal termination).
Illustration

A macrostate made of 1 STG.

STG made of 2 reactive cells.

Initial reactive cell. Body is macrostate

Reactive cell whose body is simple state.

Macrostate made of 2 STGs.

STG made of 2 reactive cells.

Initial reactive cell. Body is simple state
Detailed Example

- Macrostate $\text{ABO}=$TOP:
  - $\text{ABO.G} = \{\text{ABO.G}_1\}$
  - $\text{ABO.I} = \{A, B\}$
  - $\text{ABO.O} = \{O\}$
  - $\text{ABO.L} = \emptyset$
  - $\text{RC}_{\text{ABO}.\text{out}} = \emptyset$
- State-Transition Graph $\text{ABO.G}_1$:
  - $\text{ABO.G}_1.S = \{\text{RC}_{\text{WaitAandB}}, \text{RC}_{\text{done}}\}$
  - $\text{ABO.G}_1.i_{ni} = \text{RC}_{\text{WaitAandB}}$
- Macrostate $\text{WaitAandB}$:
  - $\text{WaitAandB.G} = \{\text{WaitAandB.G}_1,
    \text{WaitAandB.G}_2\}$
  - $\text{WaitAandB.I} = \{A, B\}$
  - $\text{WaitAandB.O} = \emptyset$
  - $\text{WaitAandB.L} = \emptyset$
  - $\text{RC}_{\text{WaitAandB}.\text{out}} = \{<nT,,O,\text{done}>\}$
- Simple state done:
  - $\text{RC}_{\text{done}.\text{out}} = \emptyset$
Detailed Example

- State-Transition Graph WaitAandB.G₁
  - WaitAandB.G₁.S={RC_{wA}, RC_{dA}}
  - WaitAandB.G₁.ini=RC_{wA}
- State-Transition Graph WaitAandB.G₂
  - WaitAandB.G₂.S={RC_{wB}, RC_{dB}}
  - WaitAandB.G₂.ini=RC_{wB}

- Simple State wA
  - RC_{wA}.out = \{<sA,A,,dA>\}
- Simple State dA
  - RC_{dA}.out = Ø
- Simple State wB
  - RC_{wB}.out = \{<sA,B,,dB>\}
- Simple State dB
  - RC_{dB}.out = Ø

Naming convention: RCₓ ≅ x
Configurations

- A **configuration** is a maximal set of states (macrostates or simple states) the system could be in simultaneously. (Note that formally status is associated with reactive cells.)

- Let $T$ be the top macrostate associated with an SSM. A **legal configuration** $C$ for $T$ must satisfy the following rules:
  1. $T$ in $C$
  2. If a macrostate $M$ is in $C$, then $C$ must also contain for each STG $G$ directly contained in $M$ exactly one state directly contained in $G$.
  3. $C$ is maximal and contains only states satisfying rules 1 and 2.

- A **stable configuration** is a legal configuration that the SSM can reach after a sequence of reactions. Only the stable configurations are of interest for the user.
Example

- **Legal configurations:**
  - \{ABRO,ABO,done\}
  - \{ABRO,ABO,WaitAandB,wA,wB\}
  - \{ABRO,ABO,WaitAandB,wA,dB\}
  - \{ABRO,ABO,WaitAandB,dA,wB\}
  - \{ABRO,ABO,WaitAandB,dA,dB\}

- **Stable configurations:**
  - all legal configurations except
    - \{ABRO,ABO,WaitAandB,dA,dB\}
Computing a reaction

- Computing a reaction is done by concurrent threads which suspend their execution when a trigger cannot be evaluated and can resume when new signal statuses are broadcast.
- Reactions are computed as a sequence of microsteps, all executed during the same instant but in the order that respects causality.
- A transition is taken (ie a microstep is executed) only when its trigger is surely satisfied (no possibility of backtracking).
- Termination codes of components (reactive cell, STG, macrostate, simple state):
  - **DONE**: execution has been terminated
  - **DEAD**: nothing left to do at the current instant and in the future (final state); component is candidate to join a normal termination.
  - **PAUSE**: nothing left to do until next instant
  - Partial order: DEAD < PAUSE
Concurrency and Weak Abort

Microstep 1
Concurrency and Weak Abort

Microstep 2
Concurrency and Weak Abort

Microstep 3
Computing the reaction of an SSM

- Reaction of an SSM. Given a stable configuration, a reaction is computed by:
  1. Read input signals (presence status of all input signals is known)
  2. Set all output signals to the unknown presence status ($\bot$)
  3. Compute reaction of the top macrostate:
     
     ```
     react(T)
     /* yields emitted signals and the next stable configuration */
     ```
State Reaction

- Reaction of macrostate M:
  1. Set all local signals to ⊥
  2. For each STG G directly contained in M do in parallel
     - Compute reaction of STG G and store the termination code in c(g):
       \[ c(G) = \text{react}(G) \]
  3. When all parallel executions are done
     - Compute \( C = \text{maximum of } c(G) \) for all STGs G in M
  4. Return C

- Reaction of a simple state S
  1. If S is a final state return DEAD.
  2. If an effect is associated with S, then emit all signals of the effect.
  3. return PAUSE
State Transition Graph Reaction

- Reaction of a STG G
  1. If there is no current state in G then set current state to the initial state: G.current = G.ini;
  2. Compute the reaction of the reactive cell C=(M,t) whose body M is the current state (M=G.current):
     \( r = \text{react}(C) \)
  3. If \( r = \text{DONE} \) then G.current = c.nextState; goto 2.
  4. return \( r \) /* here \( r \) cannot be DONE */

- Comments:
  - When entering a macrostate the current state of each STG is undefined. If the STG is already active, the current state is the (unique) currently active state.
  - Reactions of all STGs from a macrostate are computed in parallel (fork).
Reactive Cell Reaction

- Reaction of a reactive cell $C$:
  1. if (!firstInstant) Strong abort test:
     - If a strong abort transition is enabled then take this transition
  2. Execute the body of the reactive cell
     - If it is a macrostate $M$, then recursive call: $B = \text{react}(M)$
     - If it is a simple state $S$, then terminal call: $B = \text{react}(S)$
  3. if (!firstInstant) Weak abort test:
     - /* Note: body has completed execution. */
     - If a weak abort transition is enabled then take this transition
  4. Normal termination test:
     - If ($B == \text{DEAD}$) then take normal termination transition
  5. End of reaction:
     - Set $C\text{.status} = \text{ACTIVE}$
     - return PAUSE
Reactive Cell Reaction

- The triggers are not tested at the first instant when the reactive cell is activated (strong/weak abort test).
- If the presence status of a triggering signal is unknown, the execution is suspended till another concurrent execution thread will fix the status of the tested signal.
- Taking a transition t (strong/weak abort, or normal termination) means:
  - Recursively “kill” the body of the reactive cell C:
    - set k.status=IDLE for all transitively contained reactive cells k
    - reset G.current for all transitively contained STGs G
  - Execute the effect associated with t and set C.current = t.target;
  - Set C.status = IDLE;
  - return DONE;
Immediate Weak/Strong Aborts

- What happens for $a^+b^+$?
Model-based Software Development

Lustre programs

Esterel programs

C Code

Generator

Compiler

Compiler

Esterel SCADE
- SCADE language
- SyncCharts

aiT WCET Analyzer
- Timing Validation

SymTA/S
- System-level Schedulability Analysis

Binary Code
module ABRO:
input A,B,R;
output O;
loop
    [await A || await B];
    emit O each R
end module
ESTEREL - Principles

- Imperative language.

- Tailored for programming hardware or software synchronous controllers dominated by control-handling aspects.

- Most ESTEREL statements are conceptually instantaneous, i.e. are executed in the same reaction than other statements that sequentially precede or follow them in the program.
Esterel: General Structure

module $M$:
  input names;
  output names;
  statement
end module

- **Interface declaration** specifies which objects a module exports or imports:
  - **Data objects**, which are declared abstractly in Esterel. Their actual value is given in the host language.
  - **Signals and sensors**. Host objects implementing them depend on the host language.

- **Body** is an executable statement.
module WATCH:

input UL,UR,LL,LR;
type Time;
custom Noon: Time;
custom WordLength = 16: integer;
function CompareTime (Time, Time): boolean;
procedure IncrementTime (Time) (integer);
type Beep;
output Beeper: Beep;
output CurrentTime := Noon : Time;

Signal declaration
Data declarations

Possibly modified (Call by reference)
Not modified (Call by value)
Signals and Sensors

- Interface signals or local signals, declared by the `signal` statement (see later).
- Instantaneously broadcast throughout the program.

  - **Pure signals**: status is present or absent.
  - **Valued signals**: have status and carry a value of arbitrary type.

- One predefined signal `tick`:
  - pure signal
  - represents activation clock of the reactive program
  - Status is present in each instant.

- **Sensors** have a value but no status;
  Example: `sensor temperature : integer;`
Statements

- A statement starts in some instant $t$, remains active for a while, and may terminate in some instant $t' >= t$.
  - $t=t'$: statement is instantaneous
  - $t'>t$: statement takes time
Kernel Statements

- Selection of basic statements, most other statements can be programmed with:
  - nothing
  - pause
  - emit $S$
  - $p ; q$
  - $p \parallel q$
  - present $S$ then $p$ else $q$ end
  - suspend $p$ when $S$
  - loop $p$ end
  - trap $T$ in $p$ end
  - exit $T$
  - signal $S$ in $p$ end
Informal Semantics

- **Basic pure control statements:**
  - **nothing**: does nothing, ie terminates instantaneously when started.
  - **pause**: pauses when started and terminates in the next instant.

- **Signal emission:**
  - **emit** \(S\): emits signal \(S\) (ie sets its status to present) and terminates instantaneously.
  - **emit** \(S(e)\): evaluates the data expression \(e\), emits \(S\) with that value and terminates instantaneously.
  - Valid for the current instant only; happens only once.
Informal Semantics

- Sequencing ( \( p ; q \)):
  - \( p \) is instantaneously started when \( p ; q \) is started and is executed up to completion or trap exit.
  - If \( p \) terminates, \( q \) is immediately started and the sequence behaves as \( q \) from then on.
  - If \( p \) exits via traps, the exits are immediately propagated and \( q \) is never started.

  - Example: emit S1; emit S2
Informal Semantics

- Parallel Statement ($p || q$):
  - Denotes explicit concurrency
  - Any signals emitted are instantaneously broadcast to all branches in each instant.
  - The sequencing operator $;$ binds tighter than $||$.
  - Upon start both branches $p$ and $q$ are instantaneously started.
  - It terminates in the precise instant where both branches are terminated (branch synchronization).

- Example: $p;q||r$ vs $[p;q]||r$ vs $p;[q||r]$
Informal Semantics

- **present** $S$ then $p$ else $q$ end
  - immediately starts $p$ if the signal $S$ is present, otherwise $q$ is immediately started.

- **suspend** $p$ when $s$
  - $s$ is a signal expression (see later)
  - When the suspend statement starts, $p$ is immediately started.
  - $s$ has no effect in the initial instant in which the statement becomes active.
  - If the signal expression $s$ is true, $p$ remains in its current state and the suspend statement pauses for the instant.
  - If $s$ is false, $p$ is executed for the instant. If $p$ pauses, terminates or exits a trap, so does suspend.
Informal Semantics

- **loop** $p$ **end loop**:
  - $p$ is instantaneously restarted anew upon termination.
  - $p$ must never be able to terminate instantaneously when started. Note: the condition check is static!
  - If $p$ exists some enclosing trap, the loop is exited.

  ```
  trap $T$
  
  loop $p$ end loop || present $S$ then exit $T$

  end trap
  ```

- Example: `loop emit $S$ end loop (not allowed)`
Static Termination Check

loop
  present I then
  present J else
    p
  end present
else
  q
end present;
end loop

Program is rejected!
The ABRO Example

module ABRO:
  input A,B,R;
  output O;
  loop
    [await A || await B];
    emit O each R
  end module
Informal Semantics

- trap $T$ in $p$ end
  - A trap defines a lexically scoped exit for $p$.
  - $p$ is immediately started when the trap statement starts.
  - If $p$ terminates so does the trap statement.
  - exit $T$ (occurring inside of $p$) causes $T$ to terminate immediately.
  - When traps are nested, the outer one takes priority.

```
trap $U$ in
  trap $T$ in
    $p$
  end trap;
$q$
end trap;
$r$
```

- $p$ exits $T$: $q$ is immediately started
- $p$ exits $U$: $r$ is immediately started
- $p$ exits $T$ and $U$ simultaneously: $U$ takes priority.
Informal Semantics

- Local signal declaration: `signal S in p end signal`
  - Signal $S$ is local to $p$.
  - Scoping is lexical, i.e., any redeclaration of a signal hides the outer declaration.
  - A local signal placed within a loop can be executed several times in the same instant. Then each execution declares a new copy / incarnation of the signal.

- Example:

  ```
  signal Alarm,
  Distance : integer,  
in
  p
  end signal
  ```
Expressions

- **Data expressions:**
  - references to constants or variables
  - \(?S\) yields the current value of signal \(S\)
  - \(\text{pre}(?S)\) yields the value of signal \(S\) at the previous instant

- **Signal expressions:**
  - \(S\): current status of signal \(S\)
  - \(\text{pre}(S)\): status of signal \(S\) at previous instant
  - Boolean expressions over signal statuses (using the logical `and`, `or`, `not` operators, the `pre` operator and the predefined `tick` signal). `present` is considered true, `absent` false.
  - First instant of a signal \(S\):
    - interface signal: first instant of program execution
    - local signal: any instant where the corresponding local signal declaration is entered.
Expressions

- Delay expressions:
  - Used in temporal statements like `await` or `abort`.
  - Standard delays:
    - Defined by a signal expression.
    - Never elapse instantaneously.
    - Example: `meter and not second`

- Immediate delays
  - Defined as `immediate s`, where s is a signal expression
  - Can elapse instantaneously.
  - Example: `immediate [meter and not second]`
Expressions

- Count delays
  - Defined by an integer count expression \( e \) followed by a signal expression \( s \).
  - The expression is evaluated only once when the delay is initiated. If the value is 0 or less, it is set to 1. Thus a count delay never elapses instantaneously.
  - There is no immediate count delay, and counts cannot be combined with Boolean signal operators.
  - Example: \( 3 \) [second and not meter]
Further Statements

- Third basic control statement:
  - `halt`: pauses forever and never terminates.
- `sustain S / sustain S(e)`: continuous emission of signal
- Assignment (instantaneous):
  - `X := e` where `X` is a variable and `e` is a data expression
- Procedure call (instantaneous): `call P(X,Y) (e1,e2)`
- `repeat e times`
  - `p`
  - `end repeat`

- Local variable declaration:

```plaintext
var X : double,
    Count := ?Distance : integer
in
    p
end var
```
Further Statements

- if Data Test: if \( e \) then \( p \) else \( q \) end if
  - \( e \) is a data expression: The conditions are evaluated in sequence.
- await \( d \)
  - \( d \) is a delay expression
  - The delay is started when await is started. The statement pauses until the delay elapses and terminates in that instant.
- abort \( p \) when \( d \) / weak abort \( p \) when \( d \)
  - \( p \) is run until termination or until the delay \( d \) elapses.
  - If \( p \) terminates before the delay elapses, so does the abort statement. Otherwise, \( p \) is preempted when the delay elapses.
- strong abort vs. weak abort:
  - strong abort: If the delay elapses before termination of \( p \), \( p \) is preempted and not executed.
  - weak abort: If the delay elapses before termination, \( p \) is preempted and executed for a last time.