

## *Co-Synthesis*

- 1 Vulcan (Stanford - DeMicheli et al)
-  1 Polis (UC Berkeley - Vicentelli et al)

---

---

Margarida Jacome - UT Austin - 1997

## *Polis*


- 1 Leverages research in logic synthesis
  - ◆ modeling: FSM ( $\leq$  CFSM)
  - ◆ automatic path to logic synthesis and formal verification (VIS)
- 1 Assisted partitioning (non-automatic)

---

---

Margarida Jacome - UT Austin - 1997

## *Polis Outline*

-  1 System Modeling
- 1 Target Architecture & Partitioning
- 1 Software Implementation
- 1 Scheduling
- 1 Validation

---

---

Margarida Jacome - UT Austin - 1997

## *FSM Model*

popular model for describing control systems: the behavior of a system is represented in terms of states and transitions between states

**FSM model consists of:**

- a set of *states*
- a set of *transitions* between states
- a set of *actions* associated with these states or transitions

---

---

Margarida Jacome - UT Austin - 1997

## FSM Model

FSM is a quintuple

$\langle S, I, O, f: S \times I \rightarrow S, h: S \times I \rightarrow O \rangle$

where:

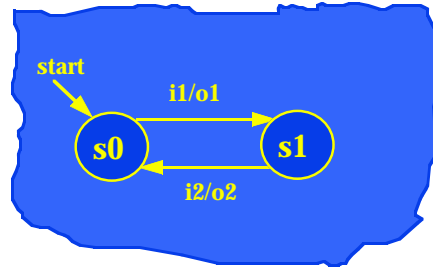
$S = \{s_1, s_2, \dots, s_l\}$  is a set of *states*;

$I = \{i_1, i_2, \dots, i_m\}$  is a set of *inputs*;

$O = \{o_1, o_2, \dots, o_n\}$  is a set of *outputs*;

$f$  is a *next state function*, which determines the next state from the current state and inputs;

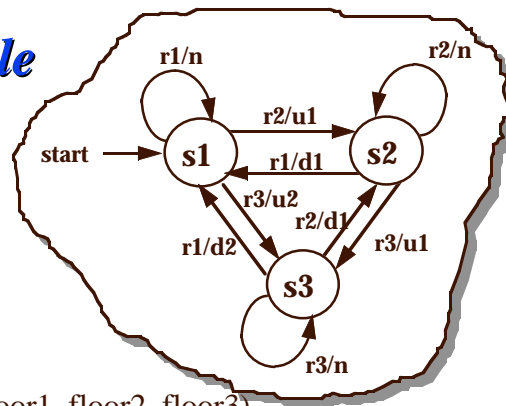
$h$  is an *output function*, which determines the outputs from the current state and inputs.



Margarida Jacome - UT Austin - 1997

## A FSM Example

FSM State Diagram  
for an Elevator  
Controller  
(3 floors)



$S = \{s_1, s_2, s_3\}$  ← current floor (floor1, floor2, floor3)

$I = \{r_1, r_2, r_3\}$  ← floor requested (floor1, floor2, floor3)

$O = \{d_2, d_1, n, u_1, u_2\}$  ← direction and number of floors the elevator should go (“u” denotes up, “d” denotes down, “n” denotes idle)

Margarida Jacome - UT Austin - 1997

## FSMs

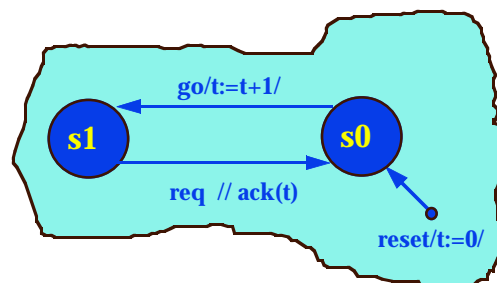
- 1 "Classical" FSMs have an implied synchronous hypothesis :
  - ◆ all the FSMs used to model a system must change state and produce their outputs *simultaneously*

Margarida Jacome - UT Austin - 1997

## The Synchronous Hypothesis

### Operational Cycle of a FSM

- ➔ 1. Idle
- ➔ 2. Detect input events
- ➔ 3. Transition, according to which events are present and a transition relation
- ➔ 4. Emit output events



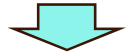
**FSM**  
 phase 1: duration between zero and infinity  
 phases 2/3/4: duration of zero

Margarida Jacome - UT Austin

## ■■■■ The Synchronous Hypothesis

System *instantaneously* reacts to events...

- 1 The chronometric notion of time is replaced by a notion of *order among events*



only relevant notions are *simultaneity* and *precedence* between events

---

---

Margarida Jacome - UT Austin - 1997

## ■■■■ FSMs and CFSMs

- 1 Mixed hardware-software systems may contain components that proceed at very different speeds
  - ◆ synchronous hardware modules
    - » execute concurrently
    - » compute next state and outputs at each clock cycle
  - ◆ software modules
    - » execute sequentially
    - » reaction to conditions may take hundreds of clock cycles to compute and propagate

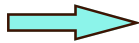
---

---

Margarida Jacome - UT Austin - 1997

## FSMs and CFSMs

FSMs can be used to model such systems but their use would be excessively cumbersome...

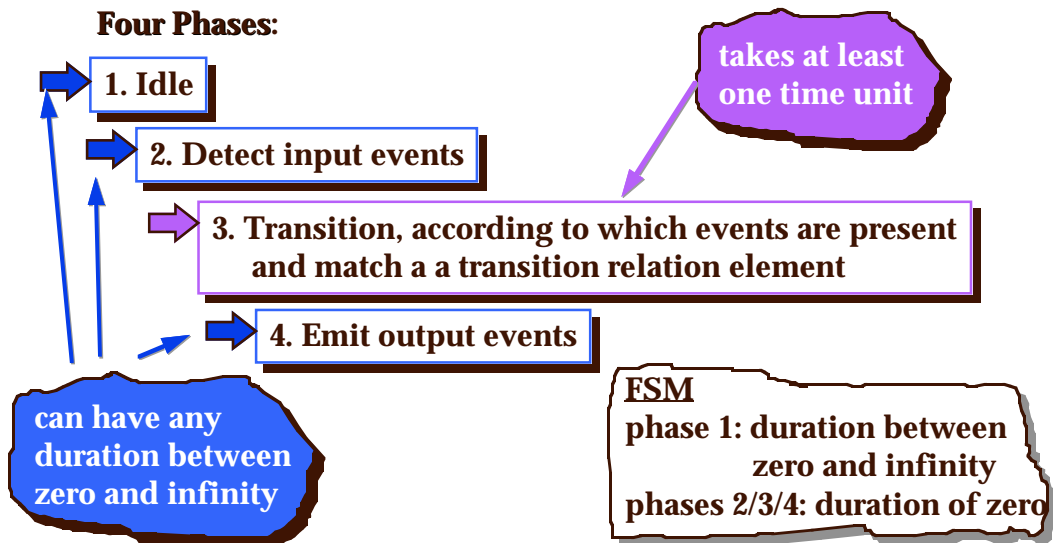


**CFSMs: specialized model that incorporates the unbounded delay assumption**

Margarida Jacome - UT Austin - 1997

## Operational Cycle of a CFSM

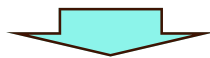
Four Phases:



Margarida Jacome - UT Austin - 1997

## Basics

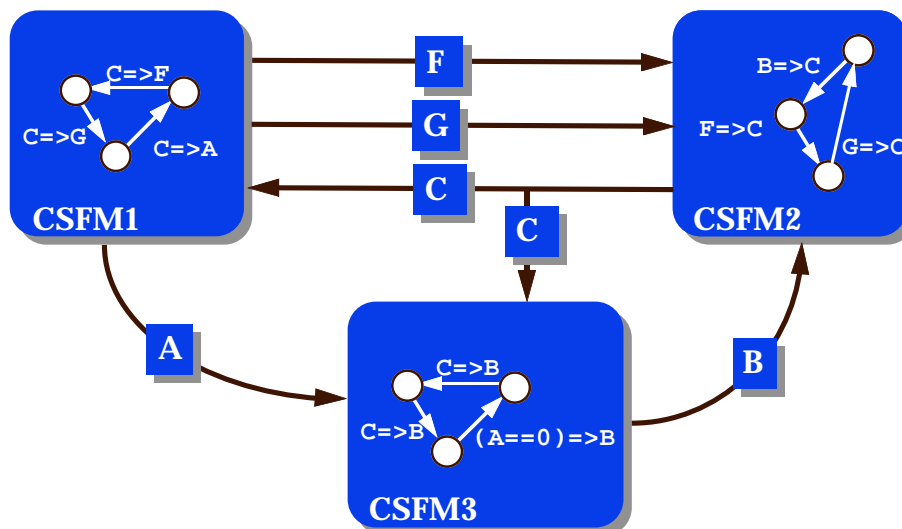
- 1 System modeled as a network of interacting CFSMs communicating through *events*
  - ◆ Each CFSM takes a non-zero unbounded time to perform its task
    - » at least before an implementation is chosen
- 1 Protocol between communicating CFSMs
  - ◆ receiver waits for the sender to emit the event
  - ◆ sender can proceed after emitting the event without the need to wait



implicit *one place buffer* between the sender and each receiver saves the event until it is detected (or overwritten)

Margarida Jacome - UT Austin - 1997

## Network of CFSMs: Depth-1 Buffers



Margarida Jacome - UT Austin - 1997

## Events

An event is a triple  $e = (e_n, e_v, e_t)$ :

- 1  $e_n$  is the name of the event
  - ◆ i.e., the “communication port” where it occurs
- 1  $e_v \in e_v$  is the value of the event;  
( $e_v$  is the set of values the event can take)
- 1  $e_t$ , a non-negative integer, is the time of occurrence of a particular instance of an event

Ex.: event with a name "temperature" could occur every time a certain sensor reports a new value, in the range between 0 and 100°C.

→ Some events may not have “interesting” values (e.g., reset)-- in this case  $e_v$  is the special symbol  $\epsilon$ .

---

---

Margarida Jacome - UT Austin - 1997

## Example: Seat Belt

Five seconds after the key is turned on, if the belt has not being fastened, an alarm will beep for ten seconds or until the key is turned off.

→ *input* events of the system:

event name	event values
------------	--------------

*BELT	ON/OFF
-------	--------

*KEY	ON/OFF
------	--------

→ *output* events of the system:

event name	event values
------------	--------------

*ALARM	ON/OFF
--------	--------

---

---

Margarida Jacome - UT Austin - 1997



## Example (cont.)

Five seconds after the key is turned on, if the belt has not been fastened, an alarm will beep for ten seconds or until the key is turned off.

→ *internal events of the system (i.e., events exchanged by the system components and not visible outside):*

event name	event values
*START	$\epsilon$ starting of the timer
*END	5/10 elapsed time

Margarida Jacome - UT Austin - 1997

## CFSMs

- 1 A CFSM is basically constituted by
  - ◆ a set of *input events*
    - » each with its associated set of values
  - ◆ a set of *output events*
    - » each with its associated set of values and possibly with an initial value
  - ◆ a *transition relation*

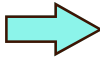
The transition relation describes how input events can cause output events.

Margarida Jacome - UT Austin - 1997

## Transition Relation

Describes how input events can cause output events.

- 1 It is a set of *pairs of sets*
  - ◆ First member of each pair: set of *input names and values*
  - ◆ Second member of each pair: set of *output names and values*



Transition:

- *triggered* by the input events with the appropriate values
- *emits* the output events with the appropriate values

The *reaction time* is unbounded and non-zero

Margarida Jacome - UT Austin - 1997

## Input Events

- 1 *Trigger* events
  - ◆ can be used *only once* to cause a transition of a given CFM
    - » each occurrence is *consumed* by the triggered transition
  - ◆ can cause many transitions in *different* CFM
- 1 *Pure value* events
  - ◆ cannot directly cause a transition
  - ◆ can be used to choose among different possibilities involving the same set of trigger events (and their values).

Margarida Jacome - UT Austin - 1997

## Input Events

Ex.: a given system must *sample the temperature every minute*, and react appropriately.

System can be modeled as a CFSM with two input events:  
time (trigger) and temperature (pure value).

- the reaction (CFSM transition) can occur only due to a time change
- the reaction must take into account the value of the temperature event when the time change event occurs.

Modeling both as events allows some other system component to react to temperature changes rather than time changes

---

Margarida Jacome - UT Austin - 1997

## States

- 1 the *state* of a CFSM consists of a *set of event types* that are at the same time input and output for it.
  - ◆ the non-zero reaction time of this feedback loop provides the "storage" capability that is required to implement the concept of *state*.



CFSMs: *reaction time* is unbounded and *non-zero*

---

Margarida Jacome - UT Austin - 1997

## ////// CFSMs

A CFSM is a quintuple  $C = (I, E, O, R, F)$ :

- 1  $I = \{(i'_n, i'_v), (i''_n, i''_v), \dots\}$  is a finite set of *input event names* and of the corresponding *finite set of allowed values*
- 1  $E, I \supseteq E$ , is the set of "*trigger*" *input event names*  
Events with names in  $(I - E)$  are "*pure data*" events
- 1  $O = \{(o'_n, o'_v), (o''_n, o''_v), \dots\}$  is a finite set of *output event names* and of the corresponding *finite set of allowed values*, such that  $E \cap O = \emptyset$  (i.e., the same event cannot be a trigger input and an output)
- 1  $R, \{(e_n, e_v) \mid (e_n, e_v) \in O, e_v \in e'_v\} \supseteq R$ , is a set of possible *initial values* of (some) *output events*
- 1  $F, \{(f_i, f_o) \mid f_i = \{(e'_n, e'_v) \mid (e'_n, e'_v) \in I, e'_v \in e'_v\}, f_o = \{(e''_n, e''_v) \mid (e''_n, e''_v) \in O, e''_v \in e''_v\} \supseteq F$ , is the *transition relation*
  - ◆ for all  $(f_i, f_o) \in F$  there must exist at least one  $(i_n, i_v) \in E, i_v \in i_v$  such that  $(i_n, i_v) \in f_i$  (i.e., at least one trigger event)

---

Margarida Jacome - UT Austin - 1997

## ////// Seat Belt Example Revisited

Five seconds after the key is turned on, if the belt has not being fastened, an alarm will beep for ten seconds or until the key is turned off.

→ *input events:*

event name	event values
------------	--------------

\*BELT ----- ON/OFF

\*KEY ----- ON/OFF

→ *output events:*

event name	event values
------------	--------------

\*ALARM ----- ON/OFF

→ *internal events:*

event name	event values
------------	--------------

\*START -----  $\epsilon$

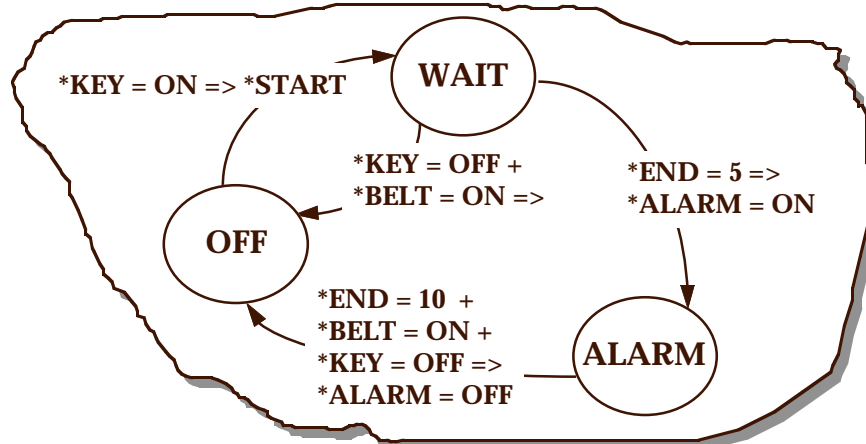
\*END ----- 5/10

---

Margarida Jacome - UT Austin - 1997

## Seat Belt Example

A CFSM describing the desired event/reaction pattern:



+ denotes the logic or condition

=> separates input and output events of a given transition

Margarida Jacome - UT Austin - 1997

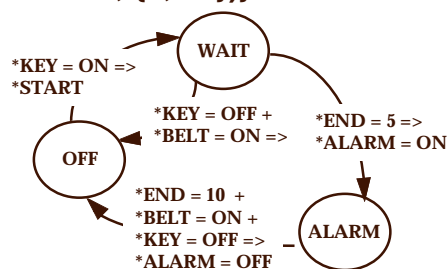
## Example: Formal Description

The formal description of the same CFSM

$C_1 = (I_1, E_1, O_1, R_1, F_1)$  is

1  $I_1 = \{(*KEY, \{ON, OFF\}), (*BELT, \{ON, OFF\}), (*END, \{5, 10\}), (s_1, \{OFF, WAIT, ALARM\})\}$

1  $E_1 = \{(*KEY, \{ON, OFF\}), (*BELT, \{ON, OFF\}), (*END, \{5, 10\})\}$



$s_1 =>$  pure data event  
(convention: name  
not preceded by “\*”)

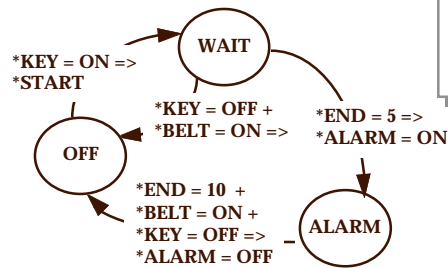
Margarida Jacome - UT Austin - 1997

## Example: Formal Description (cont.)

$C_1 = (I_1, E_1, O_1, R_1, F_1)$ :

1  $O_1 = \{(*START, \{\varepsilon\}), (*ALARM, \{ON, OFF\}), (s_1, \{OFF, WAIT, ALARM\})\}$

initial values  $\rightarrow$  1  $R_1 = \{(s_1, OFF)\}$



**s1: appears as input and output event => state event**

Margarida Jacome - UT Austin - 1997

## Example: Formal Description (cont.)

$C_1 = (I_1, E_1, O_1, R_1, F_1)$ :

1  $F_1 = \{$

$\{(*KEY, ON), (s_1, OFF)\} \Rightarrow \{(s_1, WAIT), (*START, \varepsilon)\},$

$\{(*KEY, ON), (*BELT, ON), (s_1, OFF)\} \Rightarrow \{(s_1, OFF)\},$

$\{(*KEY, OFF), (s_1, WAIT)\} \Rightarrow \{(s_1, OFF)\},$

$\{(*BELT, ON), (s_1, WAIT)\} \Rightarrow \{(s_1, OFF)\},$

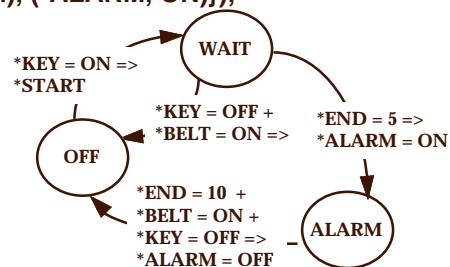
$\{(*END, 5), (s_1, WAIT)\} \Rightarrow \{(s_1, ALARM), (*ALARM, ON)\},$

$\{(*END, 10), (s_1, ALARM)\} \Rightarrow \{(s_1, OFF), (*ALARM, OFF)\},$

$\{(*BELT, ON), (s_1, ALARM)\} \Rightarrow \{(s_1, OFF), (*ALARM, OFF)\},$

$\{(*KEY, OFF), (s_1, ALARM)\} \Rightarrow \{(s_1, OFF), (*ALARM, OFF)\}$

$\}$



Margarida Jacome - UT Austin - 1997

## Network of CFSMs

A Network of CFSMs is a set of CFSMs

$$N = \{C_1 = (I_1, E_1, O_1, R_1, F_1), C_2 = (I_2, E_2, O_2, R_2, F_2), \dots\}$$

such that no two different CFSMs have an output event name in common (i.e.,  $i \neq j$  implies that  $O_i \cap O_j = \emptyset$ )

- ➡ Output sets are disjoint in order to avoid the difficulties inherent in the implementation of the *update of a single object by two concurrent agents* (would require some mutual exclusion mechanism or some resolution function)
- ➡ Input sets need not be disjoint, thus implying a *broadcast communication mechanism* (as opposed to point-to-point)

---

Margarida Jacome - UT Austin - 1997

## Seat Belt Example

The network of CFSMs would be composed by  $C_1$  plus a CFSM implementing the timer,  $C_2$ , defined as follows:

$$C_2 = (I_2, E_2, O_2, R_2, F_2):$$

$$1 \quad I_2 = \{(*START, \{\epsilon\}), (*TICK, \{\epsilon\}), (s_2, \{0,1, 2, 3, 4, 5, 6, 7, 8, 9\})\}$$

represents an input event from the environment occurring once a second

$$1 \quad E_2 = \{(*START, \{\epsilon\}), (*TICK, \{\epsilon\})\}$$

$$1 \quad O_2 = \{(*END, \{5, 10\}), (s_2, \{0,1, 2, 3, 4, 5, 6, 7, 8, 9\})\}$$

$$1 \quad R_2 = \{(s_2, 0)\}$$

---

Margarida Jacome - UT Austin - 1997

## Seat Belt Example


$C_2 = (I_2, E_2, O_2, R_2, F_2)$ :

```
1 F2 = {  
    ((*TICK, ε), (s2,0)) => {(s2, 1)},  
    ((*START, ε), (*TICK, ε), (s2,0)) => {(s2, 0)},  
    ((*TICK, ε), (s2,1)) => {(s2, 2)},  
    ((*START, ε), (*TICK, ε), (s2,1)) => {(s2, 0)},  
    ((*TICK, ε), (s2,2)) => {(s2, 3)},  
    ((*START, ε), (*TICK, ε), (s2,2)) => {(s2, 0)},  
    ...  
    ((*TICK, ε), (s2,4)) => {(s2, 5), (*END, 5)},  
    ((*START, ε), (*TICK, ε), (s2,4)) => {(s2, 0), (*END, 5)},  
    ...  
    ((*TICK, ε), (s2,9)) => {(s2, 0), (*END, 10)},  
}
```

---

Margarida Jacome - UT Austin - 1997

## Polis Outline

- 1 System Modeling
-  1 Target Architecture & Partitioning
- 1 Software Implementation
- 1 Scheduling
- 1 Validation

---

Margarida Jacome - UT Austin - 1997



## Partitioning, Target Architecture

- 1 Non-automated partitioning  $\Rightarrow$  (manually) specified by the designer
- 1 Each partition may comprise one or more CSFM's
  - ◆ partitions are synthesized separately

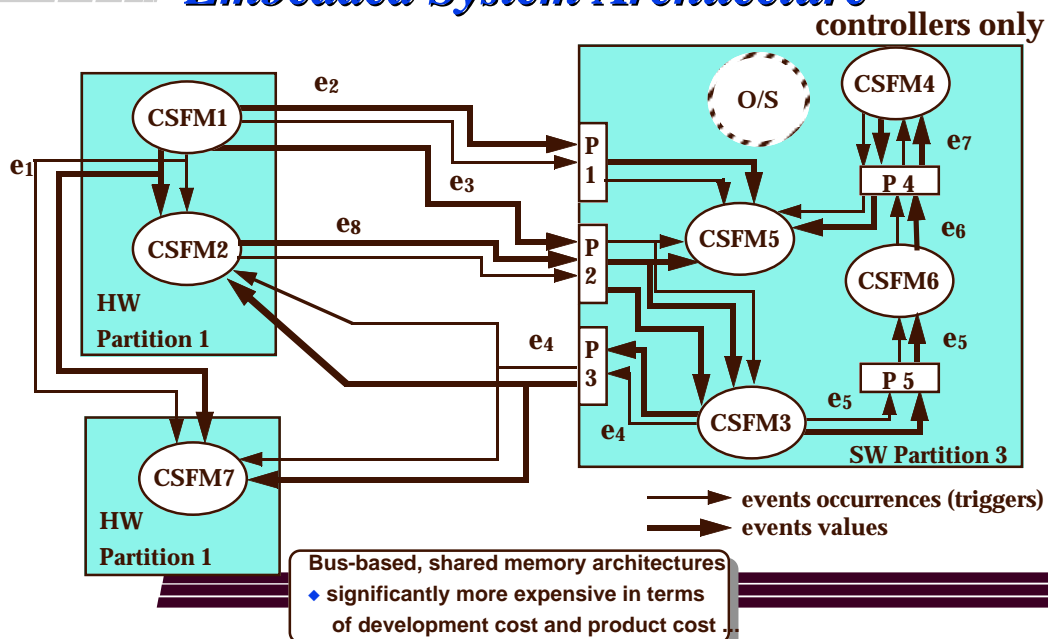
Target Architecture



General Embedded System Architecture

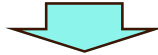
Margarida Jacome - UT Austin - 1997

## Embedded System Architecture



## ■■■■ Interface Among Partitions

- 1 Communication between partitions is based on discrete event exchange
- 1 In heterogeneous systems, “events” may be implemented differently



Two *implementation domains* can currently be handled:

- ⊗ Synchronous hardware
- ⊗ Software embedded in a micro-controller

---

---

Margarida Jacome - UT Austin - 1997

## ■■■■ Hardware Domain

- 1 An event type is represented as a wire
  - ◆ event detection: input wire is found high
  - ◆ event emission : setting an output wire for a single clock tick

---

---

Margarida Jacome - UT Austin - 1997

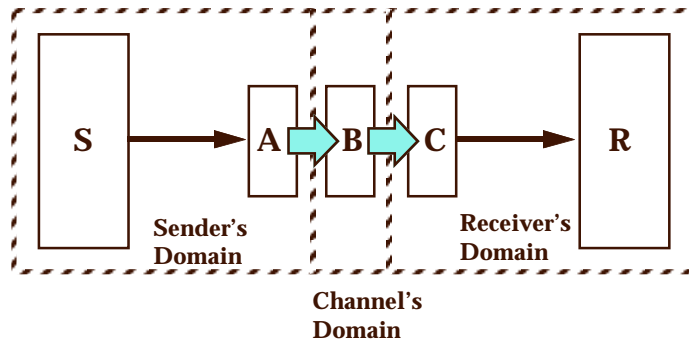
## Software Domain

- 1 detecting the occurrence of an input event:  
whenever task polls its input buffer, finds it non-zero
  - ◆ `occurred (event, InpBuff) != 0`
- 1 event emission : writing a value to a virtual port
  - ◆ `emit(event)`

---

Margarida Jacome - UT Austin - 1997

## Interface Between Partitions



- ➡ Translation of representations: sender to channel (A) and channel to receiver (C)  
Block B: gets the event across (mixed hardware software implementation)

---

Margarida Jacome - UT Austin - 1997

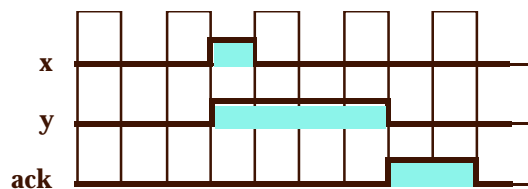
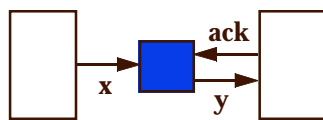
## Interface Types

- 1 Hardware to hardware
- 1 Software to hardware
- 1 Hardware to non-interrupt software
- 1 Software to non-interrupt software on a separate processor
- 1 Software to non-interrupt software on the same processor
- 1 Software to interrupt software on a separate processor
- 1 Hardware to interrupt software

Margarida Jacome - UT Austin - 1997

## An Example of Interface

Hardware to non-interrupt software




- ★ Transform 1-clock pulse (event) into a value of a bit of an input port of a processor
- ★ Presence of the event must be saved until it is copied into the receiver's input buffer

### Mixed hardware software implementation.

1. The pulse from the hardware sender is stored by an interface sequential circuit that keeps it until the receiver's scheduler, after reading it, resets it
2. The scheduler sets the input buffers of the tasks which are sensitive to this event

Margarida Jacome - UT Austin - 1997

## *Polis Outline*

- 1 System Specification
- 1 System Modeling
- 1 Target Architecture & Partitioning
-  1 Software Implementation
- 1 Scheduling
- 1 Validation

---

---

Margarida Jacome - UT Austin - 1997

## *Software Implementation*

- 1 Input
  - ◆ set of tasks (specified by CFSMs)
  - ◆ set of timing constraints
    - » e.g., input event rates and response constraints
- 1 Output
  - ◆ set of C procedures that implement the tasks
  - ◆ Scheduler that satisfies the timing constraints

 Intermediate representation  $\Rightarrow$  S-Graphs(control/data flow graph)

---

---

Margarida Jacome - UT Austin - 1997

## ////// S-Graphs

An s-graph is DAG containing the following types of nodes:

- 1 BEGIN (source)
- 1 END (sink)
- 1 TEST (disjoint paths -- expression might be detect event)
- 1 ASSIGN (includes emit event)

---

Margarida Jacome - UT Austin - 1997

## ////// Software Time/Size Estimation

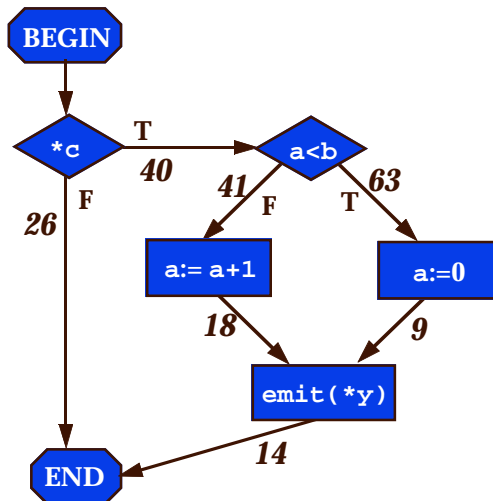
➡ Cost assigned to S-Graph edges

- timing estimation for a specific micro-controller

➡ Estimated time

- min: 26 cycles
- max: 126 cycles


- ◆ Cost parameters evaluated via simple benchmarks
  - ◆ need timing and size measurements for each target system
  - ◆ implemented for several micro-controllers




---

T Austin - 1997

## *Polis Outline*

- 1 System Specification
- 1 System Modeling
- 1 Target Architecture & Partitioning
- 1 Software Implementation
-  1 Scheduling
- 1 Validation

---

---

Margarida Jacome - UT Austin - 1997

## *The Scheduling Problem*

- 1 Given
  - ◆ estimates on the minimum and maximum execution times for each CFM transition (from the S-graph)
  - ◆ a set of timing constraints
    - » e.g., input event rates and input-to-output deadlines
- 1 Find
  - ◆ an execution ordering for the CFM transitions that satisfies the constraints
    - » static, pre-computed
    - » dynamic, decided at run-time

---

---

Margarida Jacome - UT Austin - 1997

## *Supported Scheduling Algorithms*


- 1 round-robin
- 1 static, I/O rate-based
- 1 static, pre-emptive, I/O rate based
- 1 dynamic, pre-emptive, earliest deadline first

---

---

Margarida Jacome - UT Austin - 1997

## *Scheduling: Open Issues*

- 1 Propagation of constraints from external I/O behavior to each CFM
- 1 Satisfaction of constraints within a single transition
-  1 automatic choice of scheduling algorithm, based on performance estimates and constraints


---

---

Margarida Jacome - UT Austin - 1997



## *Polis Outline*

- 1 System Specification
- 1 System Modeling
- 1 Target Architecture & Partitioning
- 1 Software Implementation
- 1 Scheduling
-  1 Validation

---

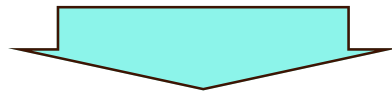
---

Margarida Jacome - UT Austin - 1997

## *System Validation*

Ensuring correctness of an implementation:

- 1 with respect to specification
- 1 with respect to some user defined property



**Formal Verification**

**Cosimulation**

---

---

Margarida Jacome - UT Austin - 1997

## *Formal Verification*

- 1 Model described as a network of CFSMs ⇒ mapped into FSMs
- 1 Time-independent and time-dependent properties

☀ Ex.:

“alarm will not be on forever”

“alarm will not be on for more than 6s”

---

---

Margarida Jacome - UT Austin - 1997

## *Cosimulation*

- 1 Used in a closed loop with system partitioning - trade-off analysis
- 1 Multiple simulations can be compared to evaluate
  - ◆ timing constraints
  - ◆ processor occupation
  - ◆ run time
  - ◆ etc.



Uses Ptolemy as cosimulation engine

---

---

Margarida Jacome - UT Austin - 1997

## **Cosimulation Environment**

Ptolemy Discrete Event (DE) Domain:

- 1 Each CFSM is mapped into a Star
- 1 Each Star has input and output portholes -- carrying *events*

⇒ A scheduling policy was implemented on “top” of Ptolemy’s DE scheduler -- decides what “software” stars are to be interrupted/executed

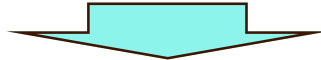
---

---

Margarida Jacome - UT Austin - 1997

## **Polis: Trade-off Analysis**

- 1 Main emphasis: speed
  - ◆ during simulation
  - ◆ what-if analysis -- architectural changes (⇒ target processor and H/S partition)



- 1 embedded software executes on a host workstation
  - ◆ instructions that *accumulate clock cycles* (estimated) are appended to each statement in the C code generated from the S-graph.

---

---

Margarida Jacome - UT Austin - 1997

## **Trade-off Analysis**

- 1 Perform simulation and gather profiling data
  - ◆ event response times
  - ◆ analysis of bottlenecks
- 1 Use estimated clock cycles for fast H/S cosimulation
  - ◆ processor selection at early stages of design



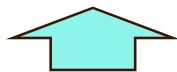
log files used to record information that may be relevant for timing analysis (e.g., when an event is overwritten, missing deadlines)

---

Margarida Jacome - UT Austin - 1997

## **Trade-off Analysis**

- 1 simulation NOT comparable with that provided by a cycle-accurate processor model (considering specific compilation options, etc.), and a hardware simulator...



(reported 20% accuracy for a characterized microcontroller...)

goal: support exploration of architectural trade-offs



Precise validation of final implementation must use a much more accurate model.

→ POLIS does not provide support for detailed simulation

---

Margarida Jacome - UT Austin - 1997