

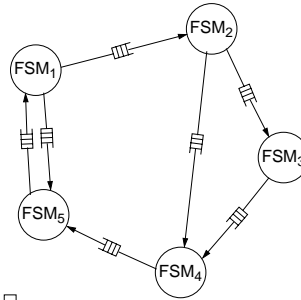
### Globally Asynchronous Locally Synchronous Systems

1. Globally Asynchronous Locally Synchronous Systems
2. Globally Asynchronous Locally Synchronous System Models

### Globally Asynchronous Locally Synchronous Systems

Globally asynchronous and locally synchronous (GALS) models:

- Each FSM individually behaves like a synchronous systems  $\Rightarrow$  reacts instantaneously on a set of available inputs and generates output.
- The global system is asynchronous  $\Rightarrow$  communication time is finite and non-zero; reaction time of each FSM, as viewed by other FSMs is finite and non-zero.
- With global asynchrony, buffering of signals could be needed.



### Globally Asynchronous Locally Synchronous Systems (cont'd)

$\Leftrightarrow$  With a GALS model, the set of FSMs is not any more equivalent with a single FSM (as was the case for the synchronous model).



Several nice features are gone:

- With synchronous FSMs we had the nice theoretical background and the possibility of formal verification of the whole system. *Not the case with GALS.*
- Every implementation of a synchronous FSM model is guaranteed to be functionally equivalent to the initial model and behave exactly and deterministically like the model (in the case we are able to produce an implementation!). *Not the case with GALS.*

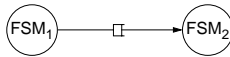
### Globally Asynchronous Locally Synchronous Systems (cont'd)

$\Leftrightarrow$  The GALS model is not deterministic, in the sense that its behavior depends on the amount of time taken for a certain communication or transition.



Two different implementations of the same GALS model can behave differently.

### Globally Asynchronous Locally Synchronous Systems (cont'd)



- A GALS model in which FSM<sub>1</sub> and FSM<sub>2</sub> communicate through a single-slot buffer.
- FSM<sub>1</sub> outputs a signal (writes into the buffer) every 2 ms (we neglect communication time).
  1. If the reaction time of FSM<sub>2</sub> is 6ms, every third signal from FSM<sub>1</sub> will be reacted on.
  2. If we have a faster implementation of FSM<sub>2</sub>, with reaction time 2ms, every signal from FSM<sub>1</sub> will be captured.

### Globally Asynchronous Locally Synchronous Systems (cont'd)

- Each individual FSM can be still verified and even formal methods can be used.
- However, individual correctness of each FSM does not guarantee the correctness of the whole system. The system behaves correctly only if, in addition, certain assumptions regarding the timing of components and of communications are satisfied.

#### Example on previous slide:

- Each FSM can be functionally verified individually.
- The global system will be correct (no signal is lost) if FSM<sub>2</sub> has a reaction time which is smaller than the production rate of FSM<sub>1</sub>.
- Estimation and simulation can be used in order to verify that a certain implementation (like FSM<sub>1</sub> as software on a certain  $\mu$ processor, and FSM<sub>2</sub> as an ASIC) satisfies this assumption.

### GALS System Models

⇒ A GALS system is modelled as a *network of FSMs*:

- Each FSM has a *locally synchronous* behavior: it executes a transition by producing a single output reaction based on a single, snap-shot input assignment in zero time.
- A System has a globally asynchronous behavior: each FSM reads inputs, executes a transition, and produces outputs in a finite amount of time as seen by the rest of the system.

### GALS System Models (cont'd)

⇒ FSMs communicate through signals.

- A signal, in general, carries an event and associated data.
- A signal is communicated between two FSMs via a connection that has an associated input buffer.
- A sender can communicate a signal to several receivers; each receiver buffers the signal in its own input buffer (of a certain size) associated to the connection.
- Communication is asynchronous and has undefined (finite) delays. Each input buffer stores the most recently received events and values.
- In general, the transmitter sends without waiting for the receiver; nothing prevents the transmitter from sending a new event before the last one was consumed and, thus, potentially, overwriting it.

### GALS System Models (cont'd)

- A FSM reacts when at least on event is available on any of its inputs; in this case the FSM
  - reads and consumes the available input signal(s);
  - identifies the matching transition and performs the corresponding state transition with the associated action set;
  - writes the outputs associated to the transition.

### GALS System Models (cont'd)

- ☞ The reaction takes a certain, finite, amount of time.  
After executing a certain transition, the FSM will be ready to react to new inputs (according to the rules above).

**Question:** When? Immediately, just after it finished the current transition?

**Answer:** Not necessarily!

- ☞ When a certain FSM is ready to check inputs and react, depends on the particular scheduling policy used at implementation.
  - Imagine you have several FSMs synthesised as software on a single  $\mu$ processor. A certain scheduling policy will decide when a particular FSM is ready to check inputs and execute a transition.
    - The scheduling policy has to be considered when checking if the timing of a certain implementation is correct (see F0 10).

### Summary

- For many systems and, in particular, larger distributed hardware/software systems, only GALS is a realistic approach for implementation.
- Some of the nice features of synchronous FSMs are gone in this case. Formal reasoning about the global system is not possible any more.
- GALS systems can be represented as a network of FSMs. A FSM has a locally synchronous behavior; however, each FSM reads inputs, executes a transition, and produces outputs in a finite amount of time as seen by the rest of the system. FSMs communicate through signals in an asynchronous manner.

### Timed Automata

- ☞ Restrictions with synchronous FSMs:
  - With the synchronous FSM model, reasoning about time is possible only for globally synchronous systems.
  - The clock is unique for the whole system and can be explicitly modelled as a FSM delivering ticks; all transitions in the system are synchronised on the clock tick. Time is captured counting these clock ticks  $\Rightarrow$  *discrete time model*.
  - All time values are non-negative integers; Events only occur at integer time values.

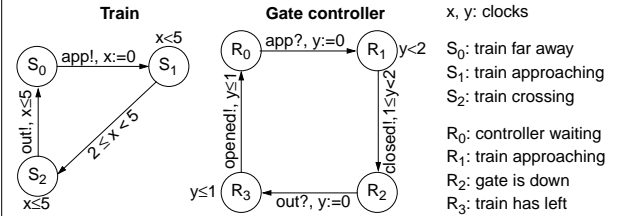
### Timed Automata (cont'd)

- ☞ For modeling real-time asynchronous systems, continuous time models are the natural representation.
- ☞ In timed automata time is considered a continuous quantity. No global synchrony, in the sense of a unique clock, is assumed.
- ☞ Timed automata are an extension of the FSM model which allows modelling of certain real-time systems and formal reasoning.
- ☞ A system is modelled as a set of concurrent timed automata.

A timed automaton is a finite automaton (similar to a FSM) augmented with a finite set of real-valued clocks.



### Timed Automata (cont'd)



- When the train approaches, it sends signal *app* at least 2 minutes before it enters the crossing; after leaving the crossing it sends the signal *out*; it leaves the crossing maximum 5 minutes after signalling *app*.
- When the controller gets signal *app* it closes the gate, which takes at least 1 minute, but less than 2; then it waits for signal *out*; when *out* arrives it opens the gate within maximum 1 minute.



### Timed Automata (cont'd)

- Transitions are instantaneous; time elapses when the automaton is in a certain state.
- When a transition occurs, some of the clocks can be reset; at any moment, the value of a clock is equal to the time elapsed since the last time it has been reset.
- Time passes at the same rate for all clocks.
- When a transition occurs, signals (events) can be generated.
- Transitions can have associated guards expressed as conditions on clock values; the transition can be taken only if the current values of the clocks satisfy the guard.
- Transitions can have input signals (events) associated; when the signal arrives and the associated guard is satisfied, the transition will be taken.
- States can have associated invariants, expressed as conditions on the clocks; the automaton can stay in that state only as long as the invariant is true.



### Timed Automata (cont'd)

- ☞ Like FSMs, timed automata can be extended with variables.
  - Actions on variables can be associated to transitions.
  - Guards expressed as conditions on the variables can be associated to transitions
- ☞ Timed automata are, by definition, *infinite state models* (continuous time!). However, they admit a *finite state representation* (by exploiting equivalence relations on certain portions of the state space)!
  - Model checking techniques can be used to prove properties of timed automata.
  - The state explosion problem is more severe than for synchronous concurrent FSMs!



### What Modeling Approach to Choose?

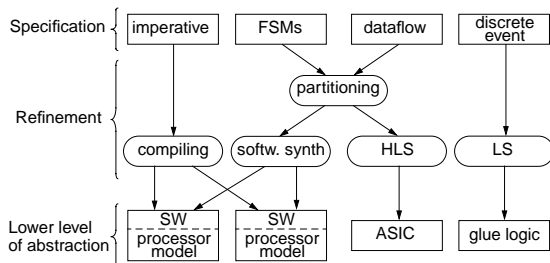
- It depends on the characteristics of the system:
  - control or data flow dominated (e.g. DSP application or reactive system);
  - synchronous or asynchronous; centralised or distributed;
  - how large?
  - what aspects related to timing are we interested in?
  -
- It depends on what you intend to do with the model:
  - simulation
  - formal verification
  - automatic synthesis
  -
- It depends on what tools you have available and which approach you (or your company or your boss!) prefer.

### What Modeling Approach to Choose?

- ☞ Don't use the "strongest"! Go for exactly that expressive power you need; not more.  
Remember F0 3, slide 8:
- Large expressive power: imperative model (e.g. unrestricted use of C or Java):
    - Can specify "anything".
    - No formal reasoning possible (or extremely complex).
  - Limited expressive power, based on well chosen computation model (e.g. Esterel):
    - Only particular systems can be specified.
    - Formal reasoning is possible.

### What Modeling Approach to Choose?

Large embedded systems are heterogeneous ⇒ mixture of models:



### Specification Languages

☞ The choice of a specification language is, to a large extent, connected to the choice of the modelling approach.

This, because certain specification languages are strongly connected to a particular model of computation:

- Communicating asynchronous state machines: SDL, Lotos
- Synchronous systems: Esterel, StateCharts;
- Dataflow and continuous computation: Matlab, Lustre, Silage

### Specification Languages (cont'd)

⇨ Some languages do not support particular models of computation

- General purpose programming languages:
  - imperative: C, C++, Java, Ada
  - functional: Lisp, Scheme, Haskell
  - logic: Prolog
- Hardware description languages:
  - VHDL, Verilog, SystemC: imperative, Discrete Event

⇨ When used according to certain *restrictions* and programming guide-lines, specifications based on particular models of computation can be realised in these languages too.

### Specification Languages (cont'd)

⇨ Different parts of the specification can be realised in different languages.

- Will we ever get "*THE*" System Specification Language?
- Will multi-language specification become the standard?