

# Synchronous Modelling of Complex Systems

Nicolas Halbwachs  
*Verimag, Grenoble*

joint work with

L. Mandel — *LRI*

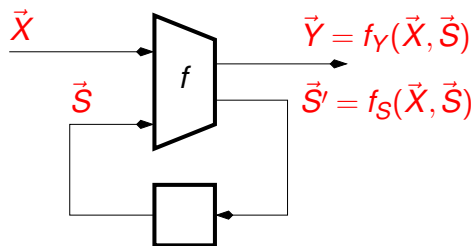
E. Jahier, P. Raymond, X. Nicollin — *Verimag*  
and D. Lesens — *Astrium Space Transportation*

- 1 **The Synchronous Paradigm**
- 2 Synchrony and Asynchrony
- 3 Synchronous Modelling of Asynchrony
- 4 A case study
- 5 Quasi-synchrony
- 6 Translating AADL concepts
- 7 Other work and conclusion

# The Synchronous Paradigm (1/8)

## Synchronous machines

Basic components: generalized Mealy machines



Behaviour:  $(\vec{S}_0, \vec{X}_0, \vec{Y}_0), (\vec{S}_1, \vec{X}_1, \vec{Y}_1), \dots, (\vec{S}_n, \vec{X}_n, \vec{Y}_n), \dots,$

with  $Y_n = f_Y(\vec{X}_n, \vec{S}_n)$  and  $\vec{S}_{n+1} = f_S(\vec{X}_n, \vec{S}_n)$

Deterministic!

# The Synchronous Paradigm (2/8)

## Basic components, simple examples

- **unit delay  $\delta$** : outputs the value received at the previous reaction

$$f_O(\vec{X}, \vec{S}) = \vec{S} \quad , \quad f_S(\vec{X}, \vec{S}) = \vec{X}$$

- **sampler  $\beta(b)$** : outputs the current input if  $b$  is true, the previous output otherwise

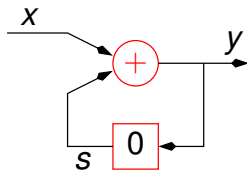
$$f_O(b, \vec{X}, \vec{S}) = f_S(b, \vec{X}, \vec{S}) = \text{if } b \text{ then } \vec{X} \text{ else } \vec{S}$$

# The Synchronous Paradigm (3/8)

## Example: an integrator

Outputs the sum of inputs received so far

$$f_Y(x, s) = f_S(x, s) = s + x$$

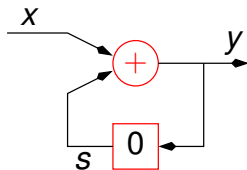


# The Synchronous Paradigm (3/8)

## Example: an integrator

Outputs the sum of inputs received so far

$$f_Y(x, s) = f_S(x, s) = s + x$$



In Lustre:

$$y = x + s ;$$

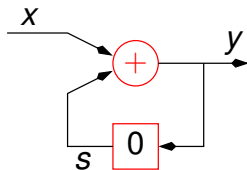
$$s = 0 \rightarrow \text{pre}(y) ;$$

# The Synchronous Paradigm (3/8)

## Example: an integrator

Outputs the sum of inputs received so far

$$f_Y(x, s) = f_S(x, s) = s + x$$



In Lustre:

$$y = x + s ;$$

$$s = 0 \rightarrow \text{pre}(y) ;$$

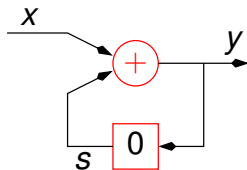
or  $y = x + (0 \rightarrow \text{pre}(y)) ;$

# The Synchronous Paradigm (3/8)

## Example: an integrator

Outputs the sum of inputs received so far

$$f_Y(x, s) = f_S(x, s) = s + x$$



In Lustre:

$$y = x + s ;$$

$$s = 0 \rightarrow \text{pre}(y) ;$$

or  $y = x + (0 \rightarrow \text{pre}(y)) ;$

step	0	1	2	3	...
x	$x_0$	$x_1$	$x_2$	$x_3$	...

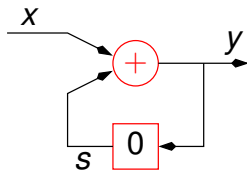


# The Synchronous Paradigm (3/8)

## Example: an integrator

Outputs the sum of inputs received so far

$$f_Y(x, s) = f_S(x, s) = s + x$$



In Lustre:

$$y = x + s ;$$

$$s = 0 \rightarrow \text{pre}(y) ;$$

or  $y = x + (0 \rightarrow \text{pre}(y)) ;$

step	0	1	2	3	...
x	$x_0$	$x_1$	$x_2$	$x_3$	...
y	$x_0$	$x_0 + x_1$	$x_0 + x_1 + x_2$	$x_0 + x_1 + x_2 + x_3$	...

# The Synchronous Paradigm (4/8)

Example: a counter of events

Detect a “minute” every 60 “second”

In Lustre:

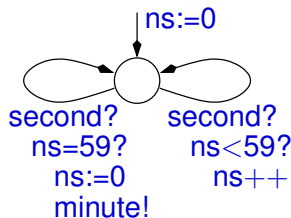
```

ns = 0 -> if second then
            if pre(ns) < 59
            then pre(ns)+1 else 0
            else pre(ns) ;
minute = (second and ns=0) ;
  
```

In Esterel

```

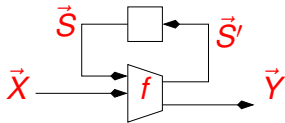
every 60 second do emit minute ;
  
```



# The Synchronous Paradigm (5/8)

## Synchronous machines

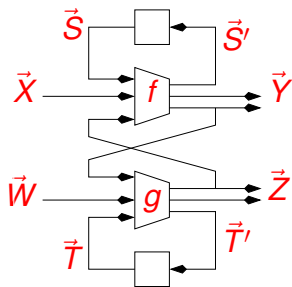
Parallel composition:



# The Synchronous Paradigm (5/8)

## Synchronous machines

Parallel composition:



$$(\vec{S}', \vec{Y}) = f(\vec{X}, \vec{S}, \vec{Z})$$

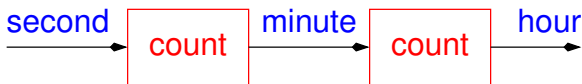
$$(\vec{T}', \vec{Z}) = g(\vec{W}, \vec{T}, \vec{Y})$$

(**deterministic**, provided there is no combinational loop)

# The Synchronous Paradigm (6/8)

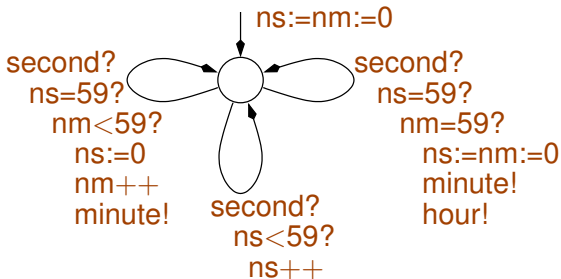
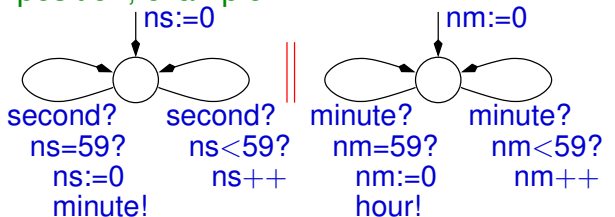
## Parallel composition, example

```
every 60 second do emit minute ;  
||  
every 60 second do emit minute ;
```



# The Synchronous Paradigm (7/8)

## Parallel composition, example



# The Synchronous Paradigm (8/8)

**Synchronous languages** are well accepted as description formalisms

- precise, formally defined
- match the way of thinking of users
- various, complementary, compatible paradigms (imperative/data-flow, textual/graphical)

Powerful **associated tools**

- efficient code generation (for centralized, statically scheduled applications)
- precise and inexpensive modelling, no overhead in analysis and verification
- can be used for expressing properties (observers)

**Industrial environments:**

Esterel Studio, Lustre-Scade, Signal-Sildex

- 1 The Synchronous Paradigm
- 2 Synchrony and Asynchrony
- 3 Synchronous Modelling of Asynchrony
- 4 A case study
- 5 Quasi-synchrony
- 6 Translating AADL concepts
- 7 Other work and conclusion



# Synchrony and Asynchrony (1/3)

Synchronous languages and associated tools are well-established for centralized, statically scheduled applications

## What about more complex situations?

- Need for dynamic scheduling:  
urgent sporadic events, multiple periods
- Need for distribution:  
redundancy, performances, physical constraints

## Synchrony and Asynchrony (2/3)

**First remark:** In real-time systems, purely asynchronous situations are rare

Partial synchrony, or strongly constrained asynchrony: e.g.,

- known periods
- known clock drift
- quite precise WCET

# Synchrony and Asynchrony (3/3)

Related works:

# Synchrony and Asynchrony (3/3)

## Related works:

- extend the synchronous model
  - CRP [Berry-Shyamasundar-Ramesh],
  - Multiclock-Esterel [Berry-Sentovitch],
  - $n$ -synchrony [Cohen-Duranton-Eisenbeis-Pagetti-Plateau-Pouzet],
  - GALS [Metropolis], [Polychrony],
  - Tag machines [Benveniste-Caillaud-Carlioni-Sangiovanni]

# Synchrony and Asynchrony (3/3)

## Related works:

- **extend the synchronous model**  
CRP [Berry-Shyamasundar-Ramesh],  
Multiclock-Esterel [Berry-Sentovitch],  
*n*-synchrony [Cohen-Duranton-Eisenbeis-Pagetti-Plateau-Pouzet],  
GALS [Metropolis], [Polychrony],  
Tag machines [Benveniste-Caillaud-Carlioni-Sangiovanni]
- **less synchronous implementations**  
Multi-task implementations [SYNDEX], [Caspi-Scaife],  
Distributed code [Caspi-Girault],[Caspi-Salem], [Potop-Caillaud]

# Synchrony and Asynchrony (3/3)

## Related works:

- **extend the synchronous model**  
CRP [Berry-Shyamasundar-Ramesh],  
Multiclock-Esterel [Berry-Sentovitch],  
 $n$ -synchrony [Cohen-Duranton-Eisenbeis-Pagetti-Plateau-Pouzet],  
GALS [Metropolis], [Polychrony],  
Tag machines [Benveniste-Caillaud-Carlioni-Sangiovanni]
- **less synchronous implementations**  
Multi-task implementations [SYNDEX], [Caspi-Scaife],  
Distributed code [Caspi-Girault],[Caspi-Salem], [Potop-Caillaud]
- **model asynchrony within the synchronous framework**  
SafeAir, SafeAir-II projects [Baufreton et-al],  
Polychrony [Le Guernic-Talpin-Le Lann], [Gamatié-Gautier],  
**this talk** (same approach, in the ctxt of the Assert project)

# The ASSERT Project (1/2)

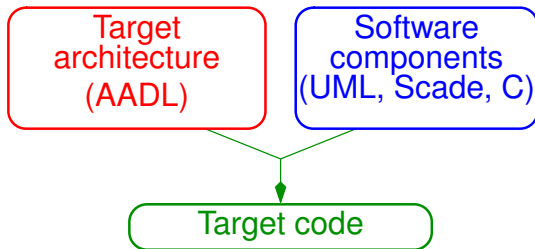
European “Integrated Project” (2005-08) on model-driven design of embedded systems

Main application domain: aerospace applications

# The ASSERT Project (1/2)

European “Integrated Project” (2005-08) on model-driven design of embedded systems

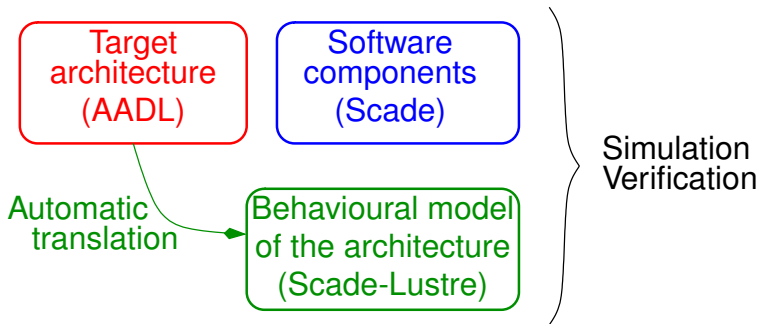
Main application domain: aerospace applications





# The ASSERT Project (2/2)

What this talk is about:



- 1 The Synchronous Paradigm
- 2 Synchrony and Asynchrony
- 3 Synchronous Modelling of Asynchrony
- 4 A case study
- 5 Quasi-synchrony
- 6 Translating AADL concepts
- 7 Other work and conclusion

# Synchronous Modelling of Asynchrony (1/5)

Need to

- prevent a component from reacting (sporadic reactions)
- non-determinism
- model execution time

# Synchronous Modelling of Asynchrony (2/5)

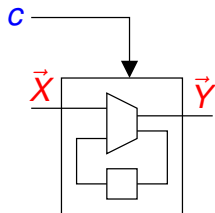
## Prevent a component from reacting

- available in all synchronous languages:
  - **clocks** in Lustre and Signal
  - **activation conditions** in Scade
  - **suspend** statement in Esterel

# Synchronous Modelling of Asynchrony (3/5)

## Activation condition in Scade

A distinguished Boolean input, say  $c$ , decides if the component must react.

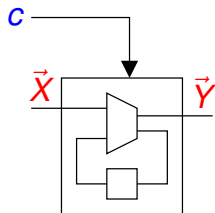


# Synchronous Modelling of Asynchrony (3/5)

## Activation condition in Scade

A distinguished Boolean input, say  $c$ , decides if the component must react.

- when  $c = 1$  the normal reaction occurs

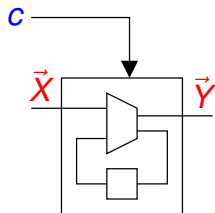


# Synchronous Modelling of Asynchrony (3/5)

## Activation condition in Scade

A distinguished Boolean input, say  $c$ , decides if the component must react.

- when  $c = 1$  the normal reaction occurs
- when  $c = 0$ 
  - the state does not change

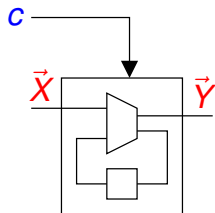


# Synchronous Modelling of Asynchrony (3/5)

## Activation condition in Scade

A distinguished Boolean input, say  $c$ , decides if the component must react.

- when  $c = 1$  the normal reaction occurs
- when  $c = 0$ 
  - the state does not change
  - the output keeps its previous value



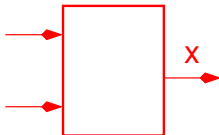


# Synchronous Modelling of Asynchrony(4/5)

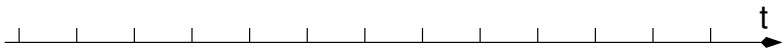
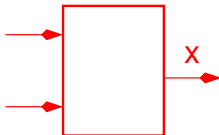
## Non determinism

- Just by adding auxiliary inputs (**oracles**)
- Restriction of non-determinism:
  - constraints/assumptions on oracles ensured by “assertions” or transducer (scheduler)

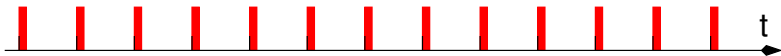
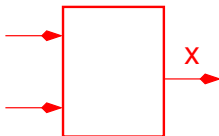
# A task in the synchronous world



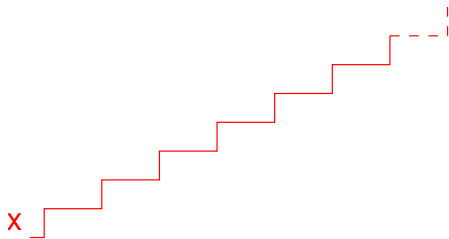
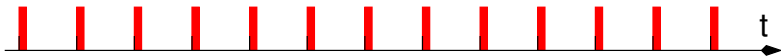
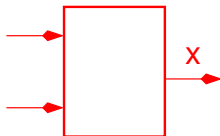
# A task in the synchronous world



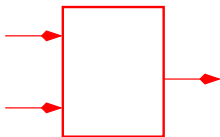
# A task in the synchronous world



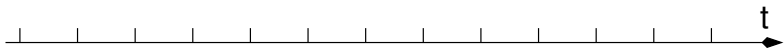
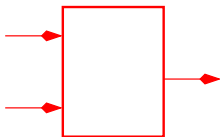
# A task in the synchronous world



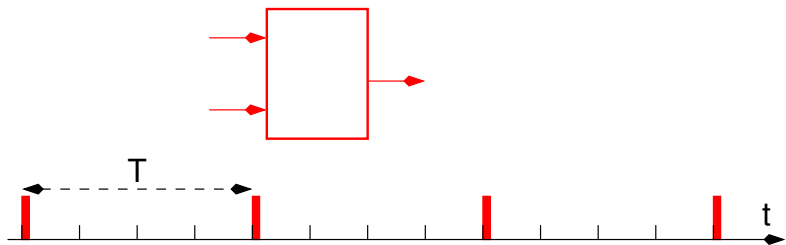
# A sporadic or periodic task



# A sporadic or periodic task

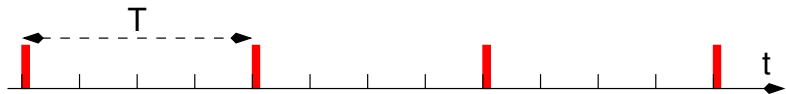
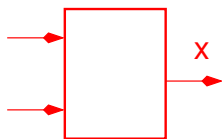


# A sporadic or periodic task

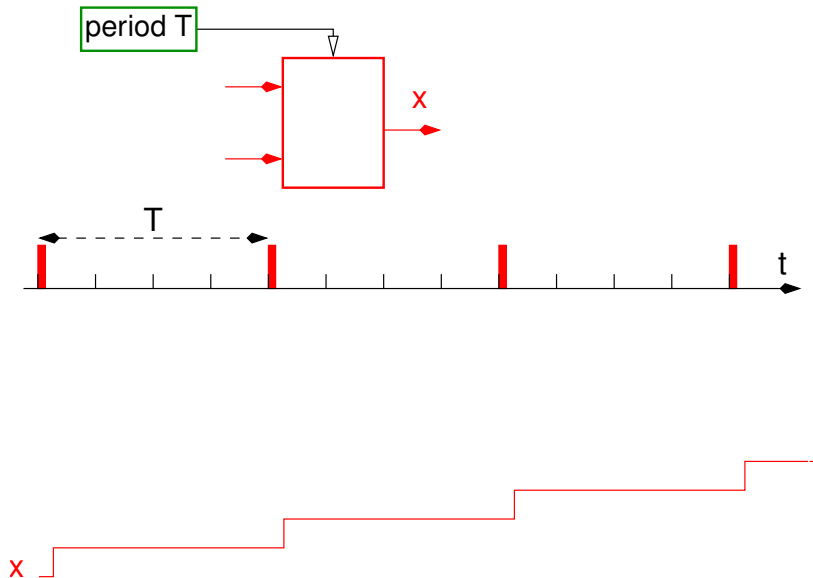




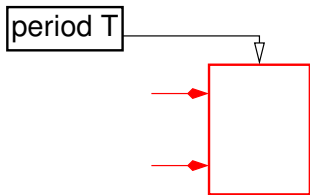
# A sporadic or periodic task



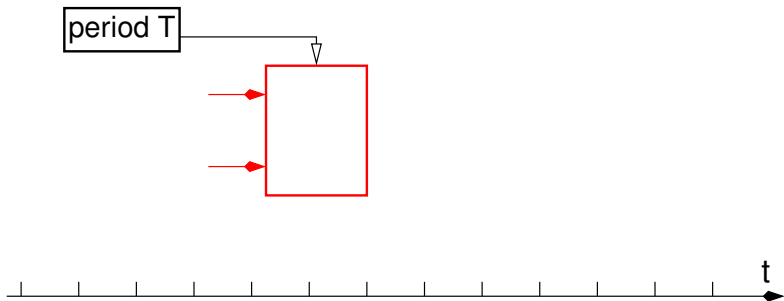
# A sporadic or periodic task



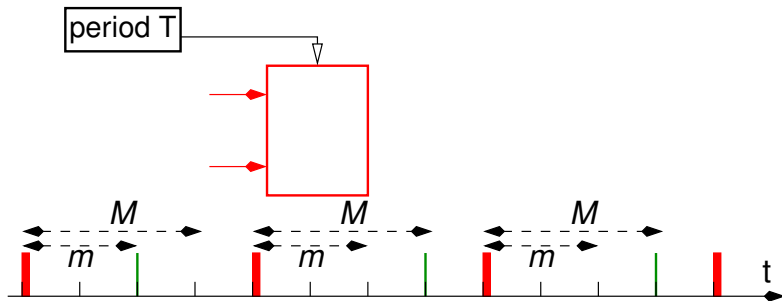
# Execution time



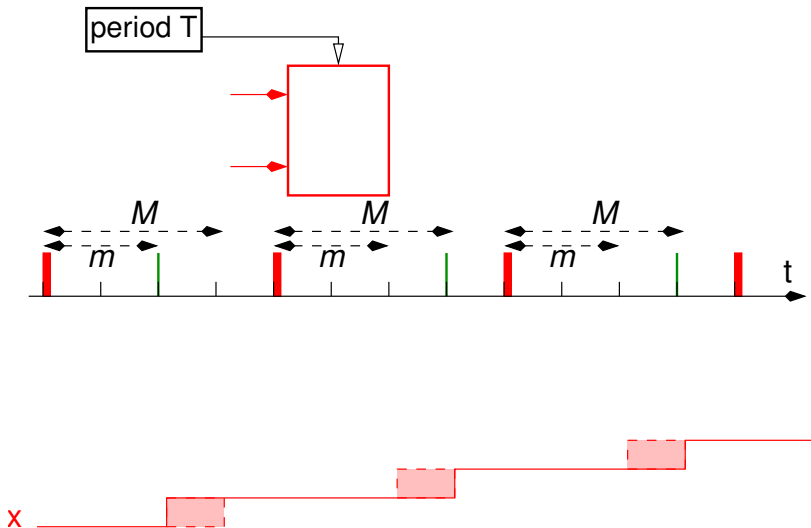
# Execution time



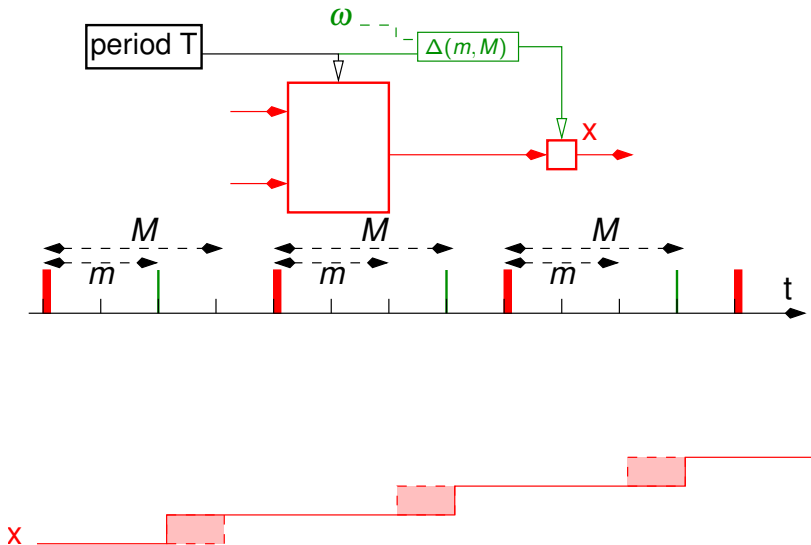
# Execution time



# Execution time

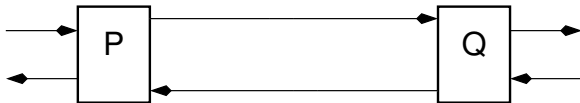


# Execution time



# Synchronous Modelling of Asynchrony (5/5)

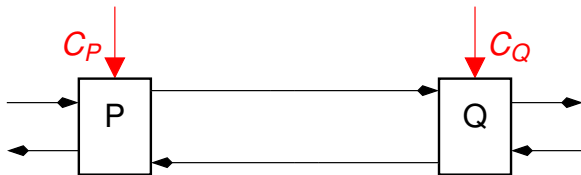
A quite general structure:





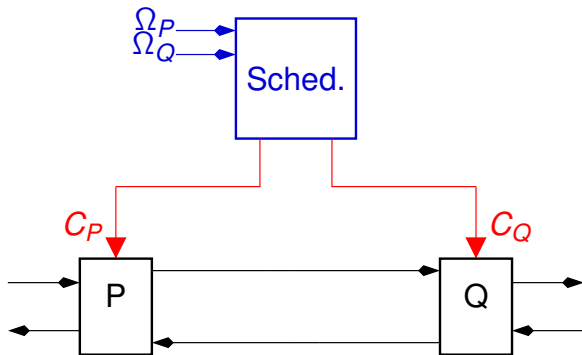
# Synchronous Modelling of Asynchrony (5/5)

A quite general structure:



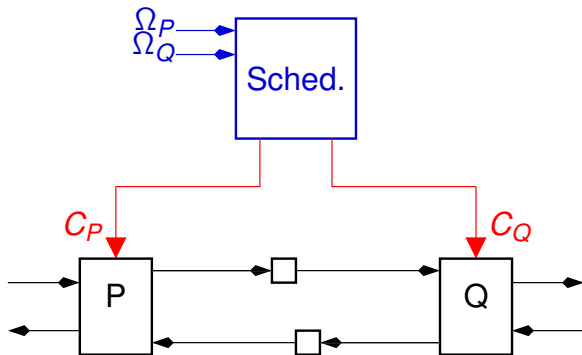
# Synchronous Modelling of Asynchrony (5/5)

A quite general structure:



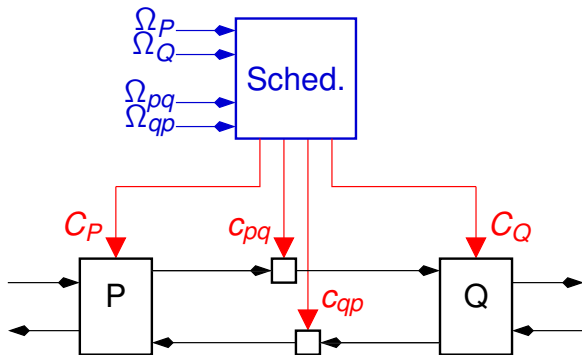
# Synchronous Modelling of Asynchrony (5/5)

A quite general structure:



# Synchronous Modelling of Asynchrony (5/5)

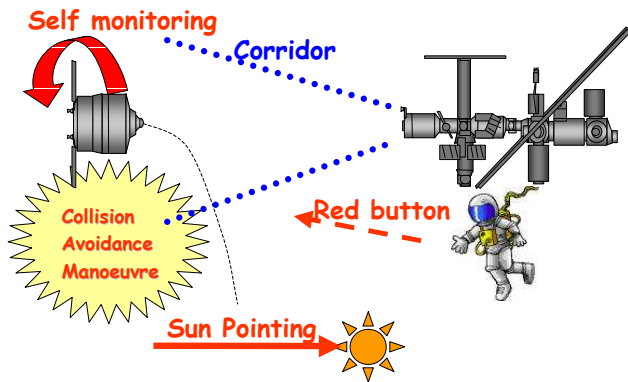
A quite general structure:



- 1 The Synchronous Paradigm
- 2 Synchrony and Asynchrony
- 3 Synchronous Modelling of Asynchrony
- 4 A case study**
- 5 Quasi-synchrony
- 6 Translating AADL concepts
- 7 Other work and conclusion

# The PFS case study (1/5)

- Proximity Flight Safety (PFS), part of the Automatic Transfer Vehicle (ATV), spacecraft in charge of supplying the International Space Station (ISS) ESA, Astrium-ST
- Ensures the safety of the approach of the ATV to the ISS (most safety critical part of the mission)



When anything goes wrong, the PFS is in charge of safely moving the ATV apart from the ISS, and to orient it towards the sun (“Collision Avoidance Manoeuvre”, CAM)

## The PFS case study (3/5)

The system is made of two redundant “Monitoring and Safety Units” (MSU): one master, one backup

Each MSU:

- **detects anomalies:** failures of the main computer, abnormal state of the bus, erroneous position or speed of the ATV, “red button” pressed from inside the ISS
- **detects its own failures** (master change)
- **is able to perform a CAM**



## The PFS case study (4/5)

At each instant, one of the MSU is the **master**. If the master detects its own failure, it transmits its mastership to the other MSU.

However,

- such a master change can only occur once in a mission
- master change is forbidden during a CAM

# The PFS case study (5/5)

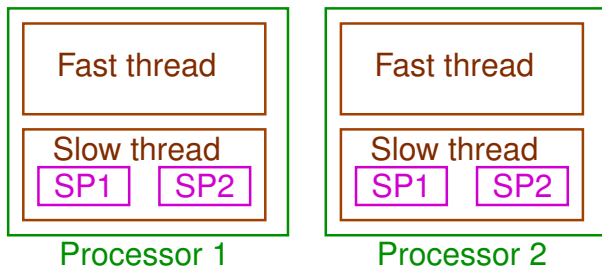
**Distribution:** Two computers (one for each MSU) running in quasi-synchrony

**Multitasking:** Each MSU consists of two periodic tasks (one fast, one slow). Each task specified in **Scade**

# The PFS case study (5/5)

**Distribution:** Two computers (one for each MSU) running in quasi-synchrony

**Multitasking:** Each MSU consists of two periodic tasks (one fast, one slow). Each task specified in **Scade**



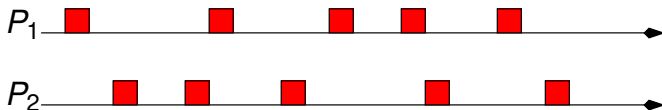
- 1 The Synchronous Paradigm
- 2 Synchrony and Asynchrony
- 3 Synchronous Modelling of Asynchrony
- 4 A case study
- 5 **Quasi-synchrony**
- 6 Translating AADL concepts
- 7 Other work and conclusion

# Quasi-synchrony (1/2)

[Caspi et al, FTRTFT'00, Safecomp'01]

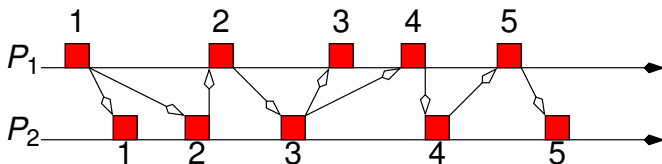
Several periodic processes on different computers Supposed to run with the same clock Small clock drift, under which the following assumption can be made:

*Between two successive activations of one periodic process, each other process is activated either 0, or 1, or at most 2 times*



## Quasi-synchrony (2/2)

In case of simple communication (e.g., by shared memory), each process can only **miss** or **duplicate** at most **one output in a row** from any other process:



## Modeling quasi-synchrony (1/2)

2 processes  $P$  and  $Q$ , activated by conditions  $C_P, C_Q$

**Assumption:** between 2 occurrences of  $C_i$  there are at most 2 occurrences of  $C_j$

Ambiguous, because of simultaneity...

**Precise assumption:** Each condition cannot be true alone more than twice in a row. If a condition occurs alone twice in a row, the other condition must follow alone

# Modeling quasi-synchrony (2/2)

Non-deterministic quasi-synchronous scheduler



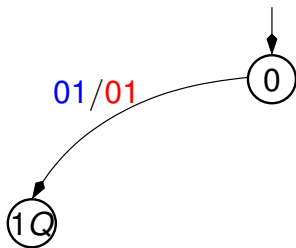


# Quasi-synchronous scheduler

 $\Omega_P, \Omega_Q / C_P, C_Q$ 

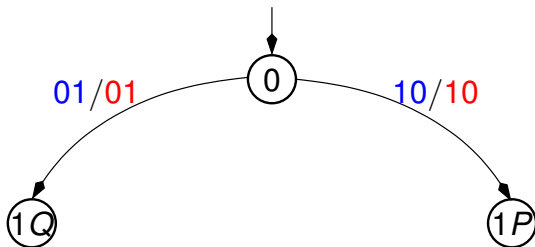
# Quasi-synchronous scheduler

$$\Omega_P, \Omega_Q / C_P, C_Q$$



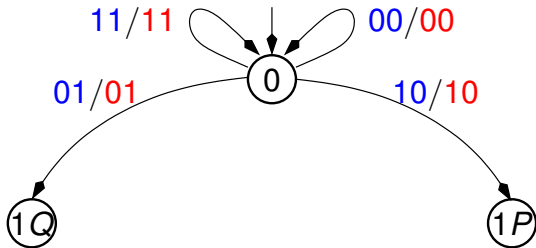
## Quasi-synchronous scheduler

$$\Omega_P, \Omega_Q / C_P, C_Q$$



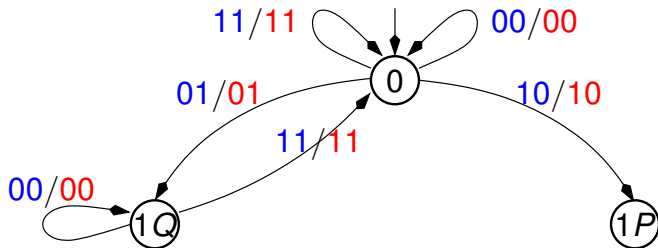
# Quasi-synchronous scheduler

$$\Omega_P, \Omega_Q / C_P, C_Q$$



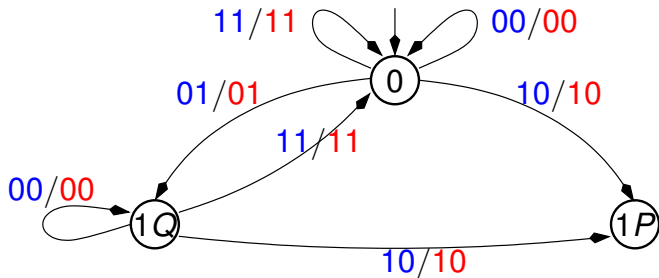
# Quasi-synchronous scheduler

$$\Omega_P, \Omega_Q / C_P, C_Q$$



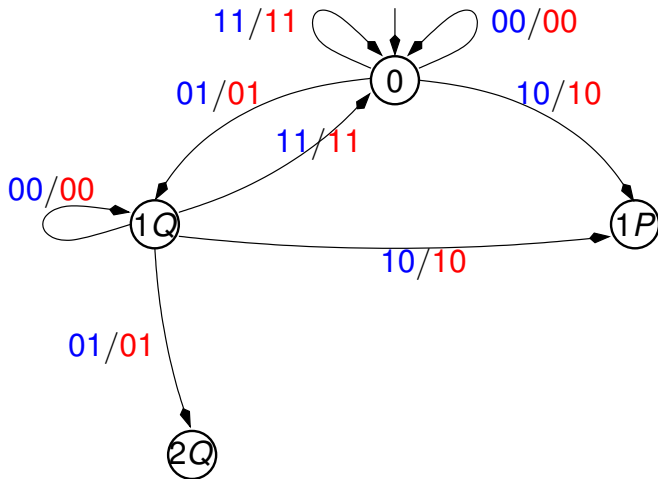
# Quasi-synchronous scheduler

$$\Omega_P, \Omega_Q / C_P, C_Q$$



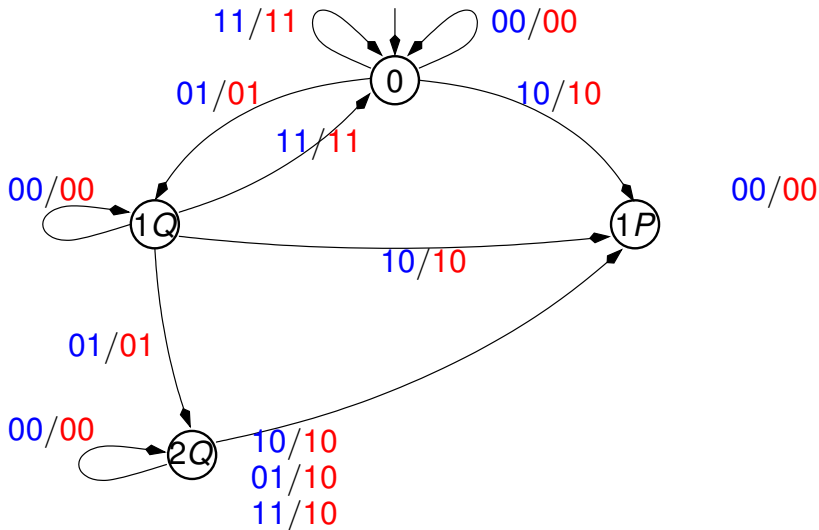
# Quasi-synchronous scheduler

$$\Omega_P, \Omega_Q / C_P, C_Q$$



## Quasi-synchronous scheduler

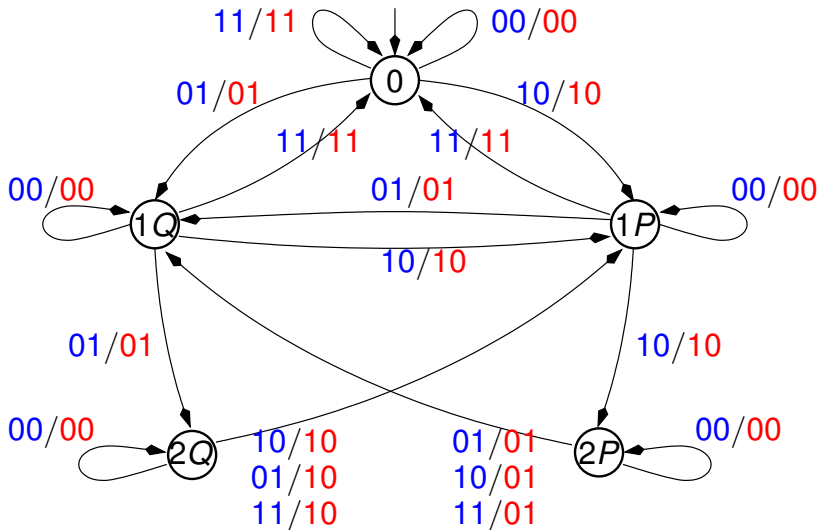
$$\Omega_P, \Omega_Q / C_P, C_Q$$





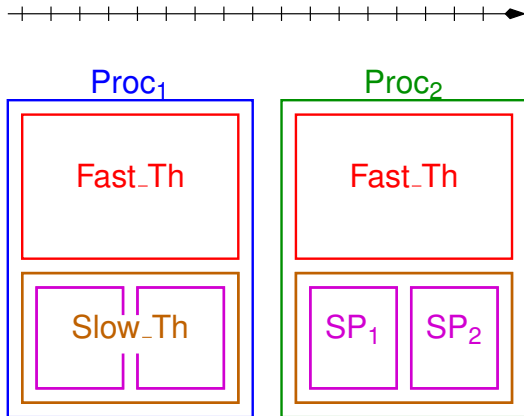
## Quasi-synchronous scheduler

$\Omega_P, \Omega_Q / C_P, C_Q$

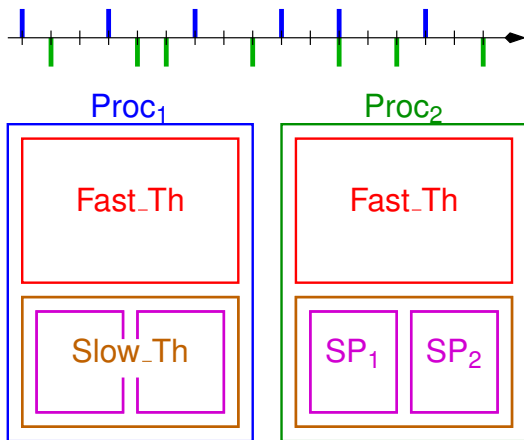


- 1 The Synchronous Paradigm
- 2 Synchrony and Asynchrony
- 3 Synchronous Modelling of Asynchrony
- 4 A case study
- 5 Quasi-synchrony
- 6 Translating AADL concepts**
- 7 Other work and conclusion

# Processes: actual clocks



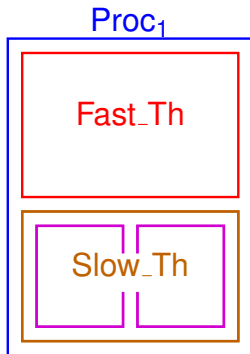
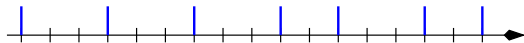
# Processes: actual clocks



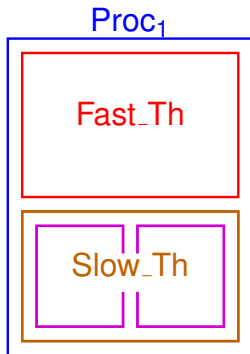
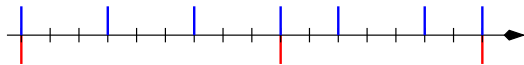
“Quasi-synchronous” clocks

used to count periods and deadlines

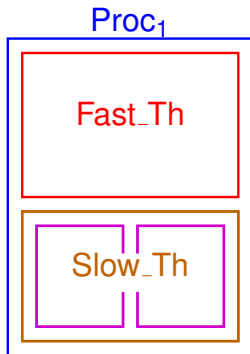
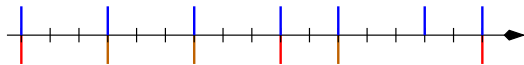
# Threads: sharing the processor



# Threads: sharing the processor



# Threads: sharing the processor



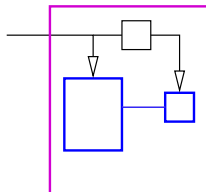
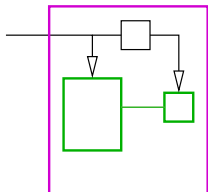
Activity clocks, used to count execution times

# Subprograms: sequencing

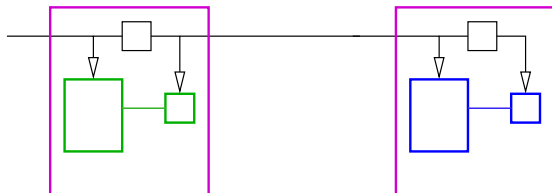




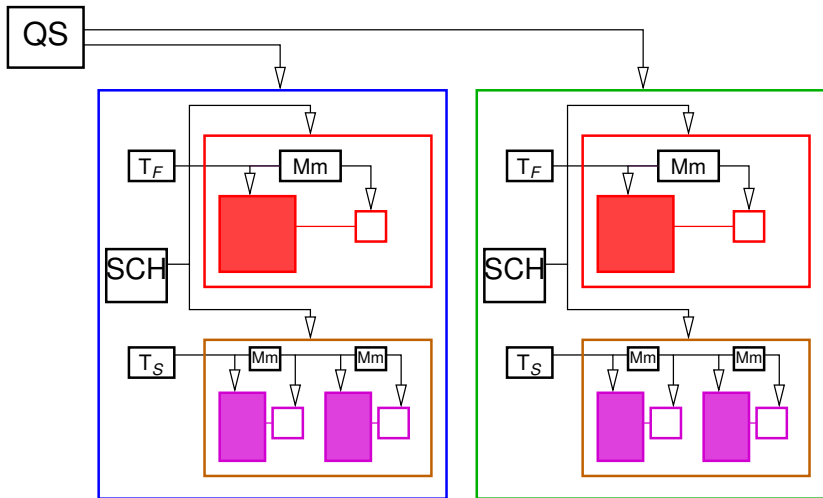
# Subprograms: sequencing



# Subprograms: sequencing



## Final model



# Applications

- extensive simulation  
(using the tool LURETTE to generate oracles automatically)
- automatic verification
  - Example of property of the PFS:  
“at each instant, one and only one MSU is the master”

# Applications

- extensive simulation  
(using the tool LURETTE to generate oracles automatically)
- automatic verification
  - Example of property of the PFS:  
“at each instant, one and only one MSU is the master”  
Wrong, because of asynchrony.  
Right property:  
“at each instant, there is at most one master”  
“there are at most two clock cycles without master”

- 1 The Synchronous Paradigm
- 2 Synchrony and Asynchrony
- 3 Synchronous Modelling of Asynchrony
- 4 A case study
- 5 Quasi-synchrony
- 6 Translating AADL concepts
- 7 Other work and conclusion

## Other works

- deterministic communication [[ACSD2006](#)]
- scheduling policies and resource management [[FASE2009](#)]

## Other works

- deterministic communication [[ACSD2006](#)]
- scheduling policies and resource management [[FASE2009](#)]

## Conclusion

- Gives precise semantics to AADL
- Makes it executable (early simulation/validation)
- One more non-synchronous application of synchrony