

Some issues in model-based development for embedded control systems

Paul Caspi

Verimag-Cnrs

www-verimag.imag.fr

EMSOC Villard de Lans

june 2006

Introduction

- Model-based development widely recognised as a method of choice for efficient and safely design
- More effectively used in embedded control
 - e.g., automatic code generation in Airbus fly-by-wire (1984)
 - Simulink-CAN, Simulink-TTA,...
- But rather empirical, lack of foundations

Some issues:

1. Model-based development in control and in computer science
2. Sampling theory for discrete event and hybrid systems
3. Preserving a synchronous model semantic in asynchronous implementations
4. UML vs Simulink: what is object orientation in block-diagram approaches?

Model-based design in computer science and control _____

Model-based design in computer science

- Starts from a non deterministic **specification**
- Based on successive property-preserving **refinements**
- Until an **implementation** is reached

Remarks:

- This is an idealised scheme, seldom fulfilled
- Yet has a paradigmatic value
- Some real-world impressive achievements in control!!
 - **B method (Abrial)**: Paris, Barcelona, New York subways

Model-based design in computer science

MACHINE

initial

SETS

persons, buildings

ABSTRACT_CONSTANTS

state, authorisation

PROPERTIES

persons $\neq \emptyset \wedge$ buildings $\neq \emptyset$

\wedge state \in persons \rightarrow buildings

\wedge authorisation \in persons \leftrightarrow buildings

INVARIANT

state \subseteq authorisation

OPERATION

move $\hat{=}$ ANY (p, b)

WHERE (p, b) \in authorisation

\wedge state(p) \neq b

THEN state(p) := b

END

END

Model-based design in computer science

Further steps:

- Add implementation details:
 - paths, doors, badge controls,...
- Separate controllers from environment !!!
- Generate control programs

Model-based design in computer science and control _____

Model-based design in control science

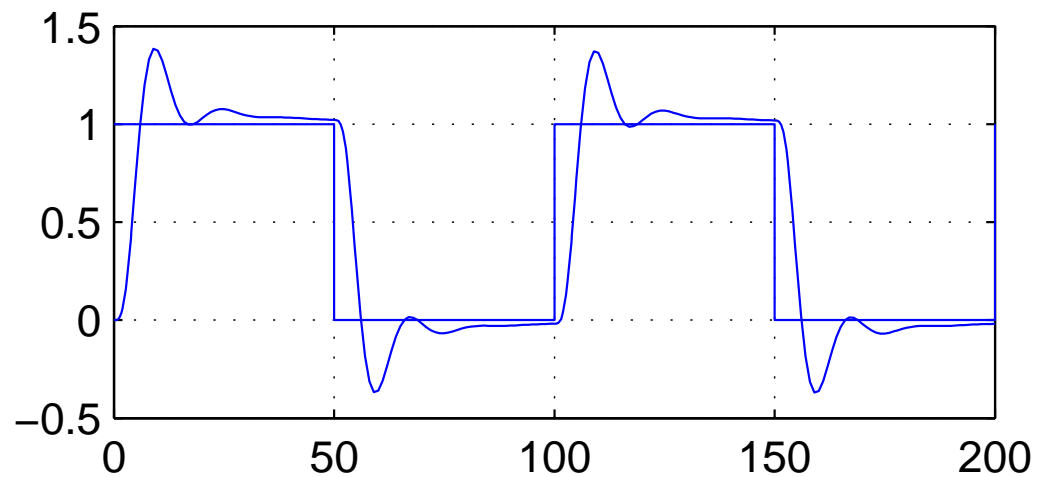
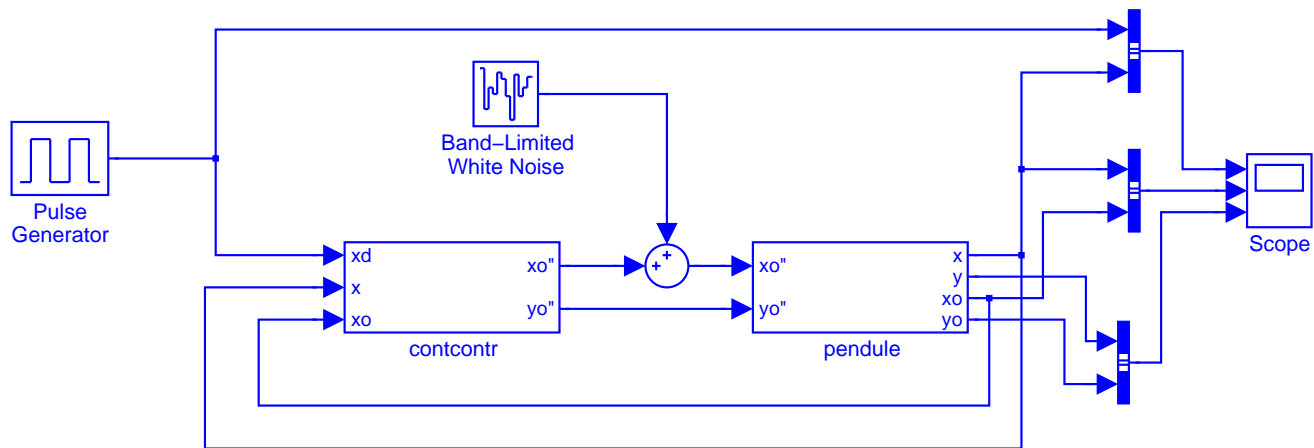
- Start from a perfect model
- Design a robust controller
- Add perturbations and implementation details and checks for robustness

Remarks:

- This is also an idealised scheme

Model-based design in control

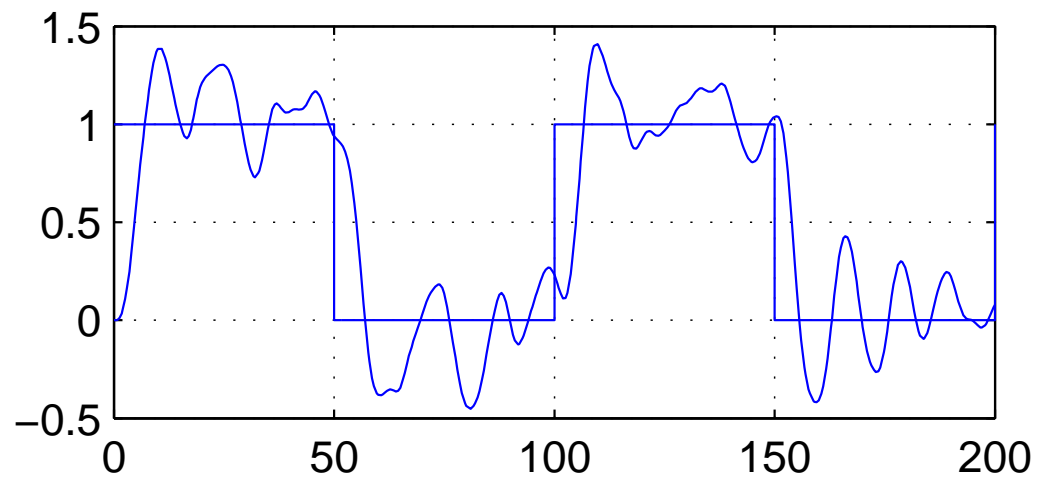
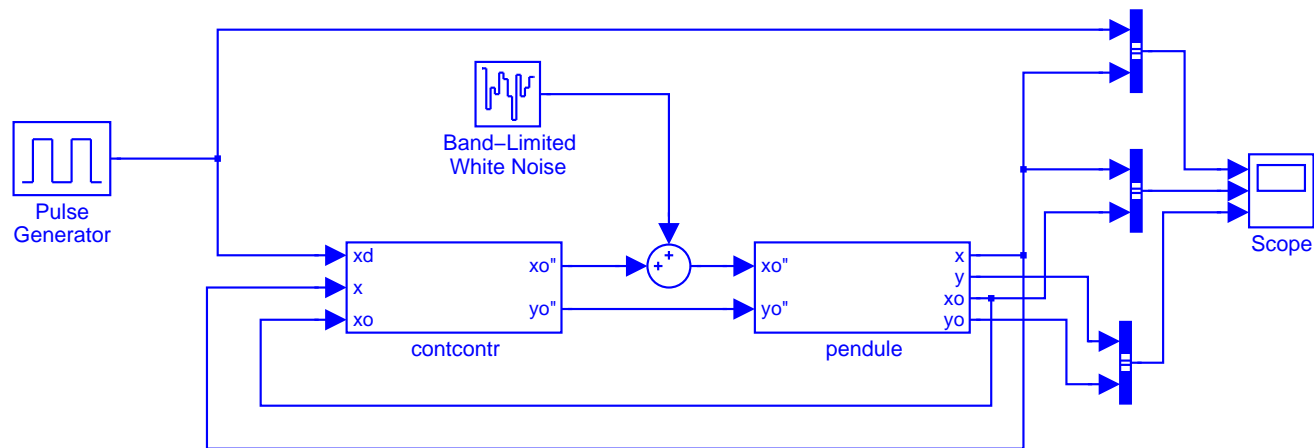
Perfect model



Time offset: 0

Model-based design in control

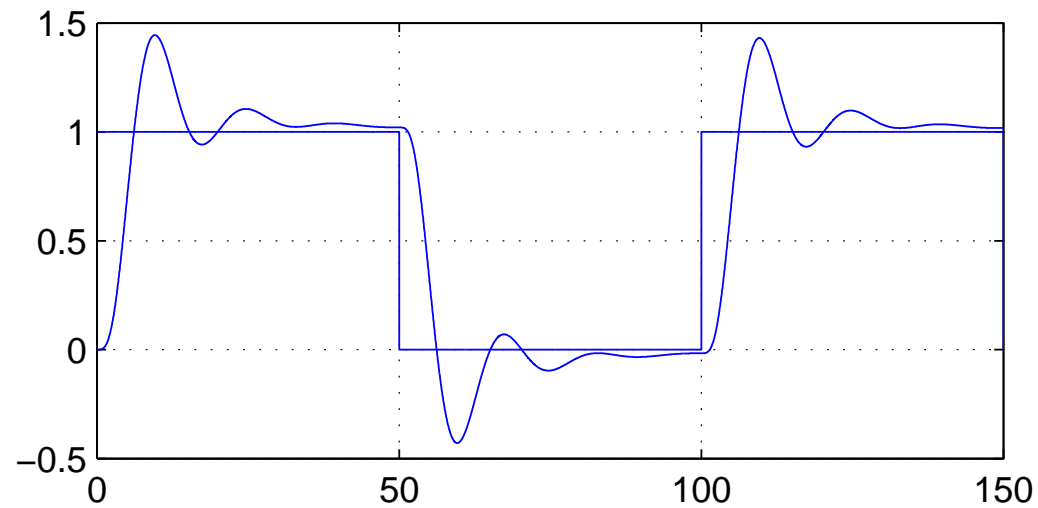
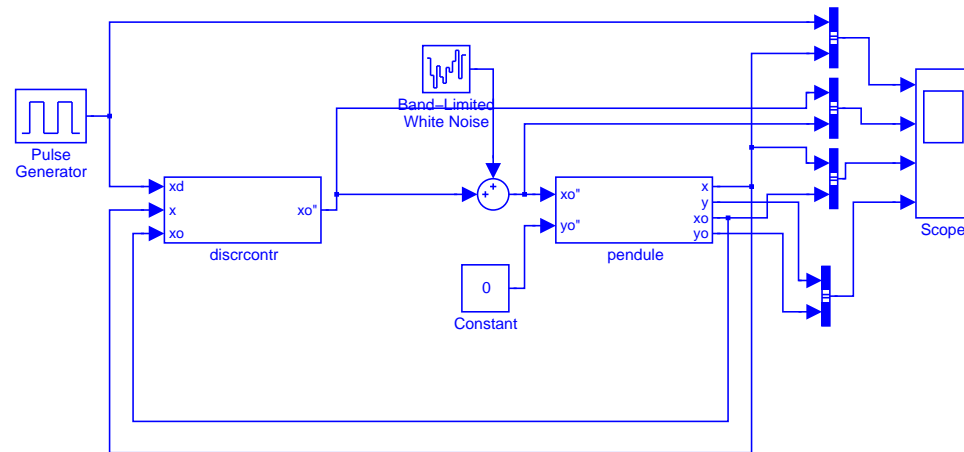
Perfect model with noise



Time offset: 0

Model-based design in control

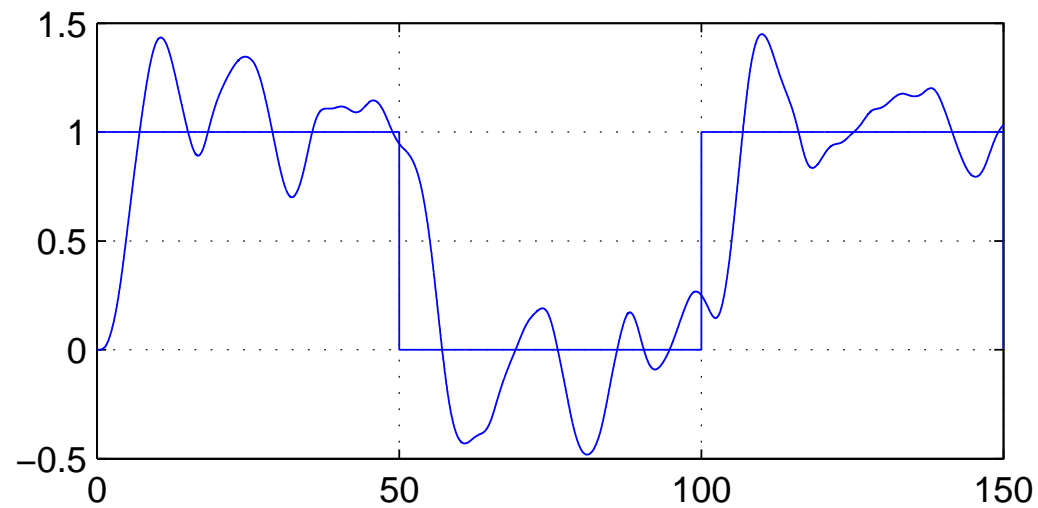
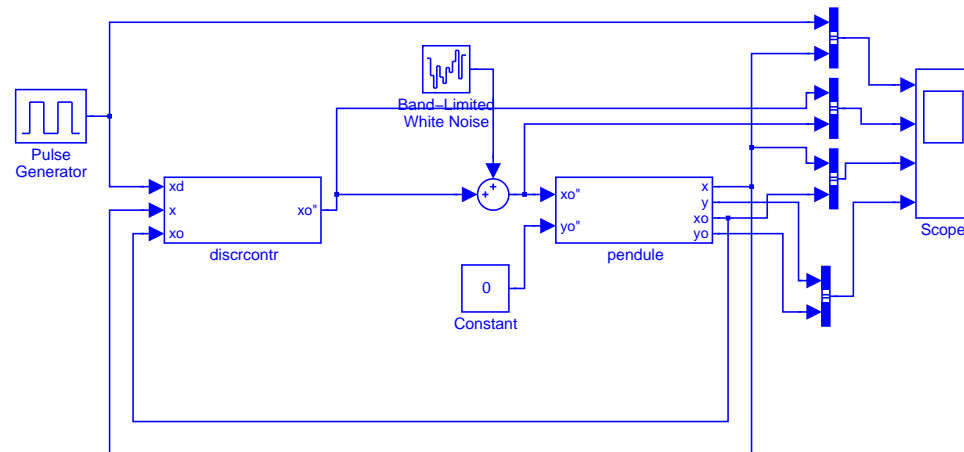
Discrete-time controller



Time offset: 0

Model-based design in control

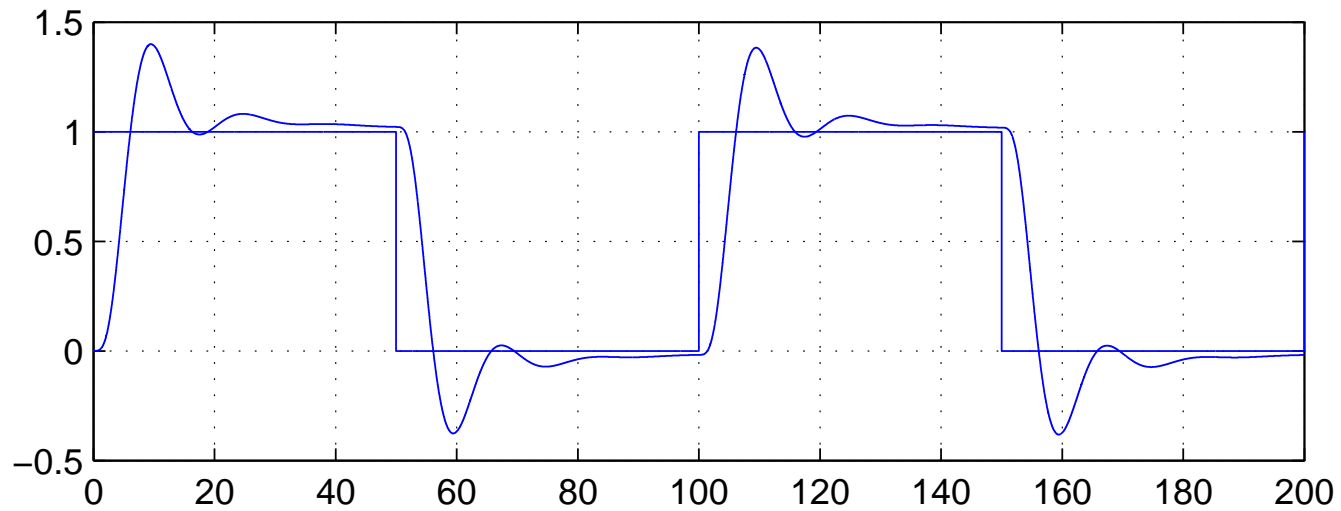
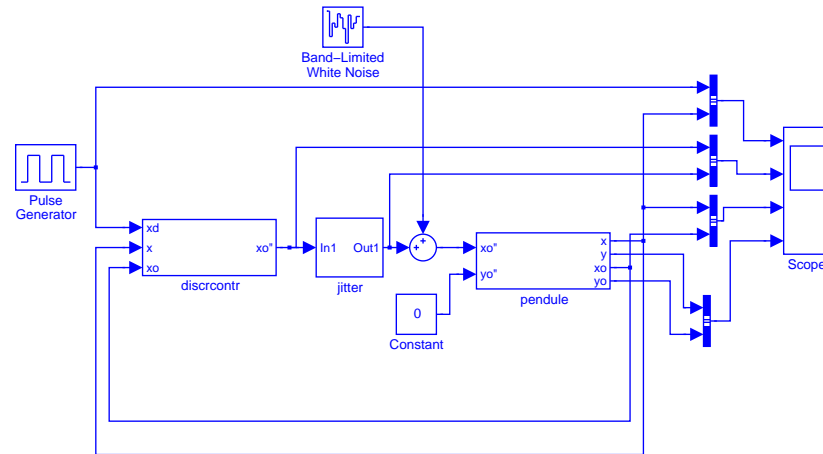
Discrete-time controller with noise



Time offset: 0

Model-based design in control

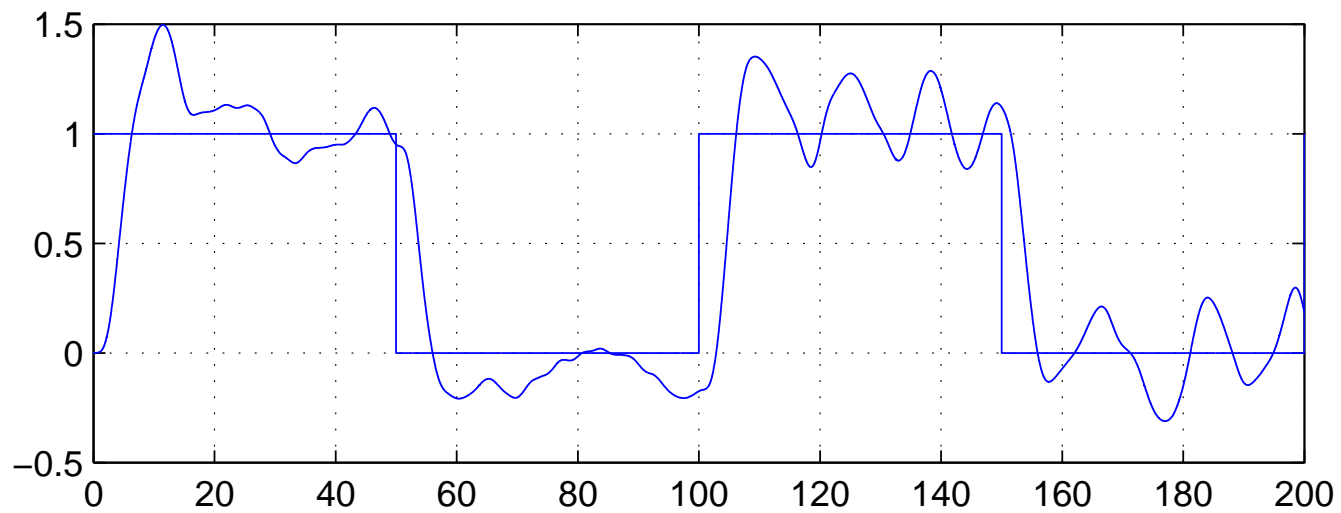
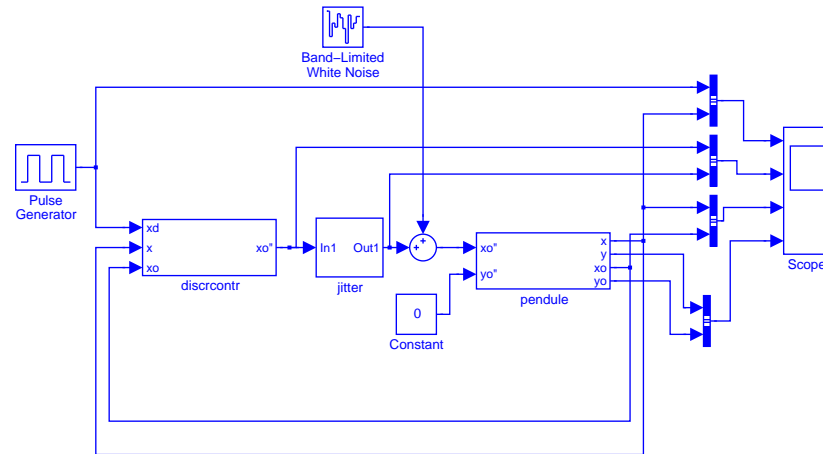
Discrete-time controller with jitter



Time offset: 0

Model-based design in control

Discrete-time controller with jitter and noise



Time offset: 0

Model-based design in computer science and control _____

How can we make them converge ??

A suggestion :

Consider the perfect control model as specifying a set of behaviours, those behaviours which are within some “distance” of the perfect model behaviour.

This requires some notion of “distance”, able to account for

- perturbations
- modelling errors
- discretisation
- jitter and communication delays
- ...

Sampling discrete event and hybrid systems

Continuous control is implemented by periodic sampling (time-triggered)

- sampled-data control theory
- numerical analysis

Discrete event control is implemented by event triggered systems

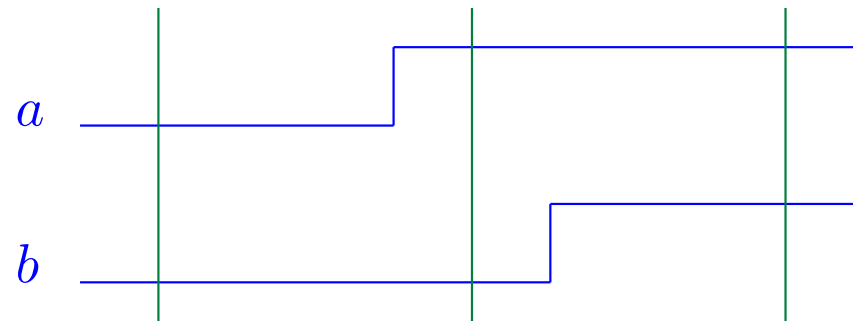
What about mixed (hybrid) systems ??

Experience shows that periodic sampling is a popular solution

but an empirical one

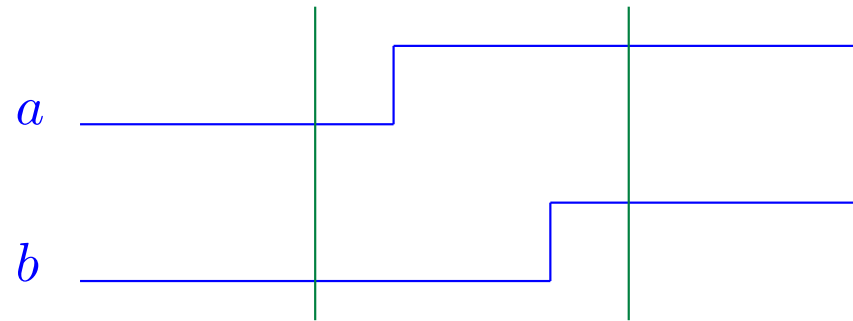
Sampling discrete event systems

A possible sampling



Sampling discrete event systems

Another one

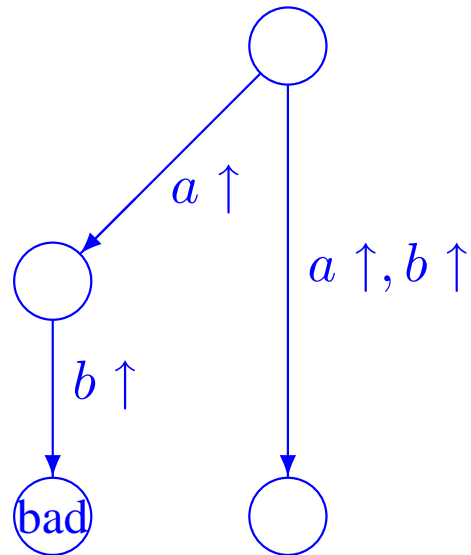


Races

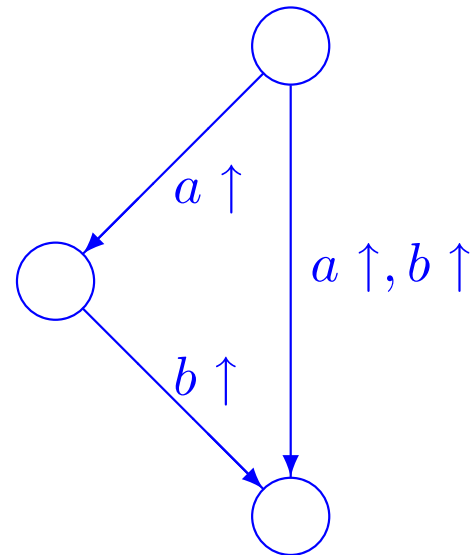
A **race** takes place when two variables can change in distinct orders

A race is **critical** if different states can be reached according to which variable changes first

A critical race



A non-critical race

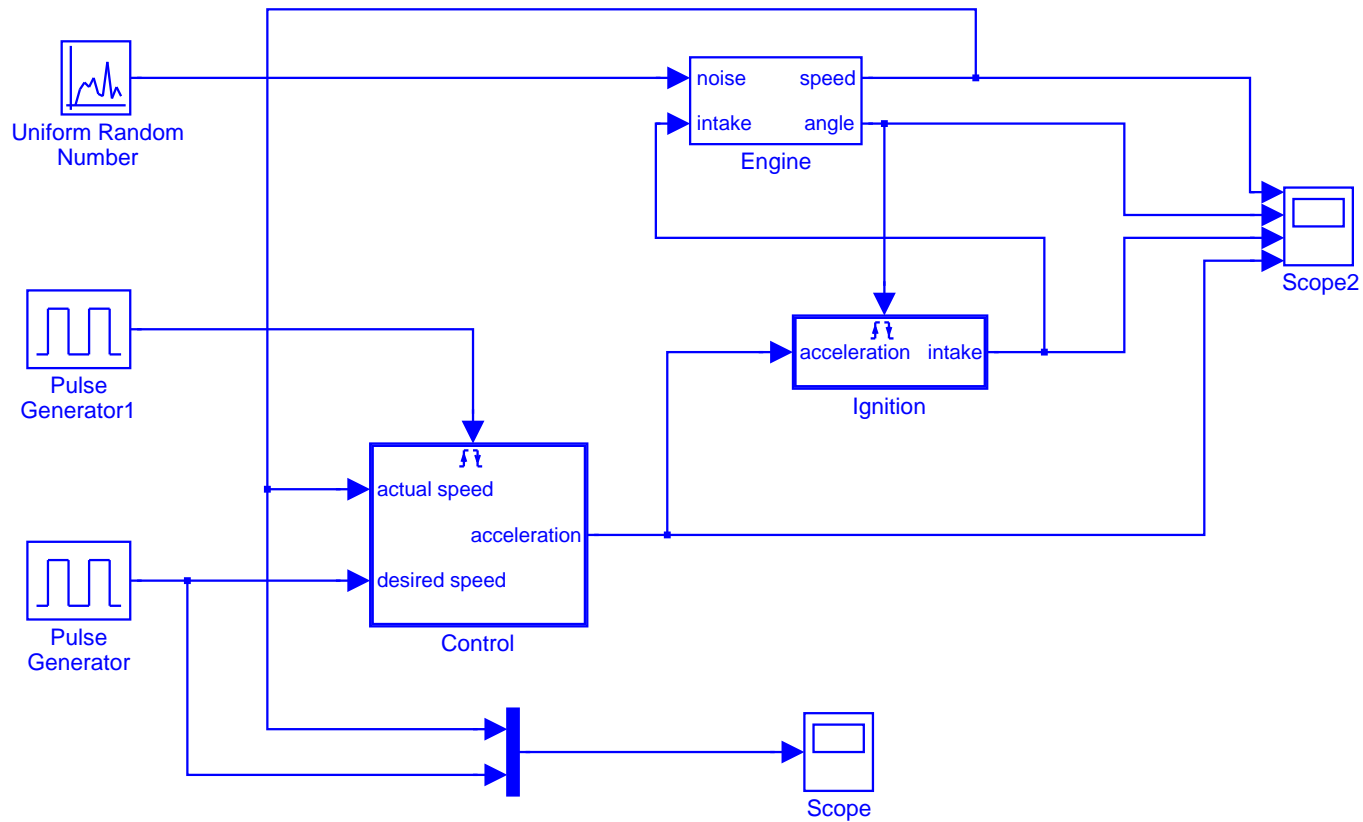


Sampling discrete event and hybrid systems

Which kind of “distance” can account for

- perturbations
- modelling errors
- discretisation
- jitter and communication delays
- sampling discrete events
- races
- ...

Event and time-triggered systems



Characteristics of the model

Based on several idealisations:

- The engine model is more or less accurate
- Computations are exact
- Computations take no time (synchronous abstraction)

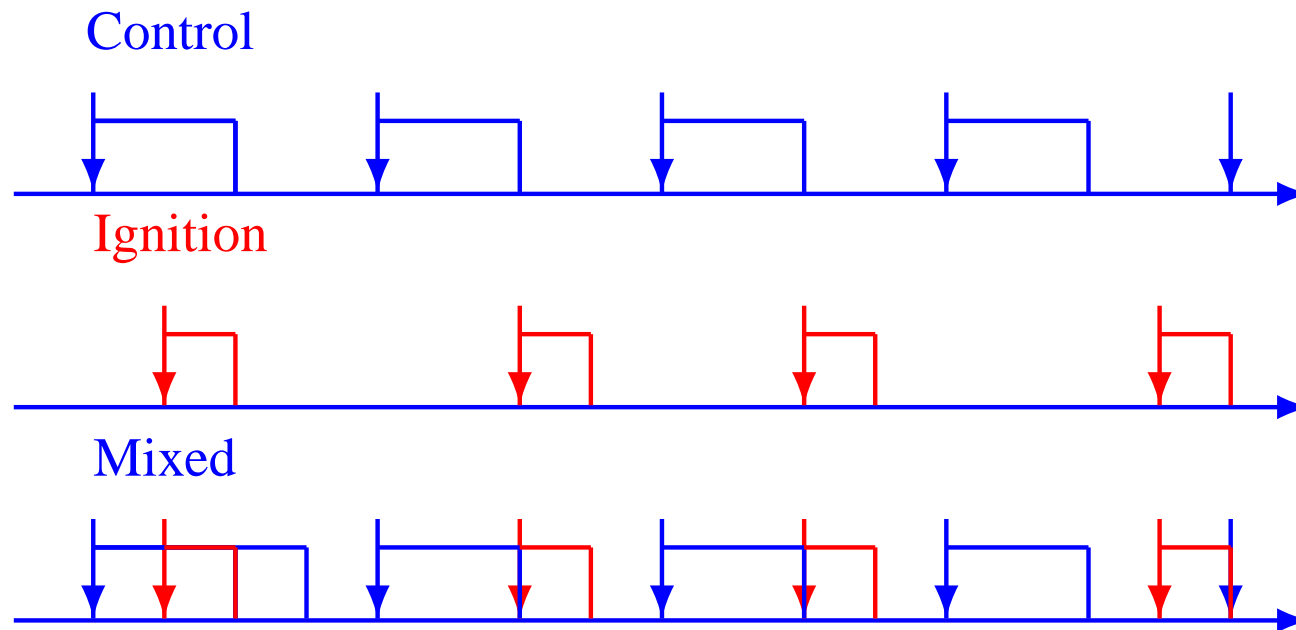
Implementation approximations

- Bounds on computation errors.
- Deadlines on executions

Domain dependent

Preemptive scheduling

If the deadline associated with event-triggered computations is smaller than the execution time of time-triggered tasks, preemptive scheduling is mandatory:



A solution : deadline monotonic scheduling

Fixed priorities in the reverse order of deadlines D_i (*Burns & al. 94*)

Schedulability test: iterative computation of response times

$$R_j = \sum_{i=1, j-1} \left\lceil \frac{R_j}{T_i} \right\rceil C_i + C_j$$

- task priorities in decreasing order
- T_i minimum inter-arrival time ($D_i < T_i$)
- C_i worst execution times

It suffices then to verify for every i : $R_i < D_i$

Inter-task communication

Communication integrity, several approaches:

- Blocking approaches based on semaphores
 - Priority inversion (pathfinder !!)
 - priority inheritance, priority ceiling protocols
- Lock-free methods
- Loop-free, wait-free methods
 - Burns et Chen* (triple buffer)
 - provide easier schedulability analysis ?

What about semantics?

...and model-based development?

Preemption alters the ordering of computations

- In many cases it does not matter (robustness, continuity, ...)
- In some cases it can (discontinuities, critical races, ...)

Can we propose executions that be functionally equivalent to the model?

Proposed solution

Ensures communication integrity and provides executions that are functionally equivalent to the model:

Based on:

1. **Syntactic checks:** communications from low to high priority tasks should go through a unit delay on the low task trigger
2. **Double buffer protocols** where distinction is made between the occurrence of triggering events and the task executions

Simulink and UML

Simulink

Functional block-diagrams
+ Automata (Stateflow)

Continuous time, discrete and event triggered systems

Simulation, code generation (Scade...), verification

Many libraries, (TTA, CAN, preemptive scheduling,...)

UML

Class, flow and activity diagrams,
Message sequence charts

Simulation, code generation, verification

OS support

Ways of mixing them

A “popular way”:

- Devote Simulink (Scade) to time-triggered systems
- Devote UML to event-triggered systems

Some dubious statements:

- “Simulink can’t handle event-triggered systems”
- “Control people don’t care of objects and classes”

Are these the right ways ?

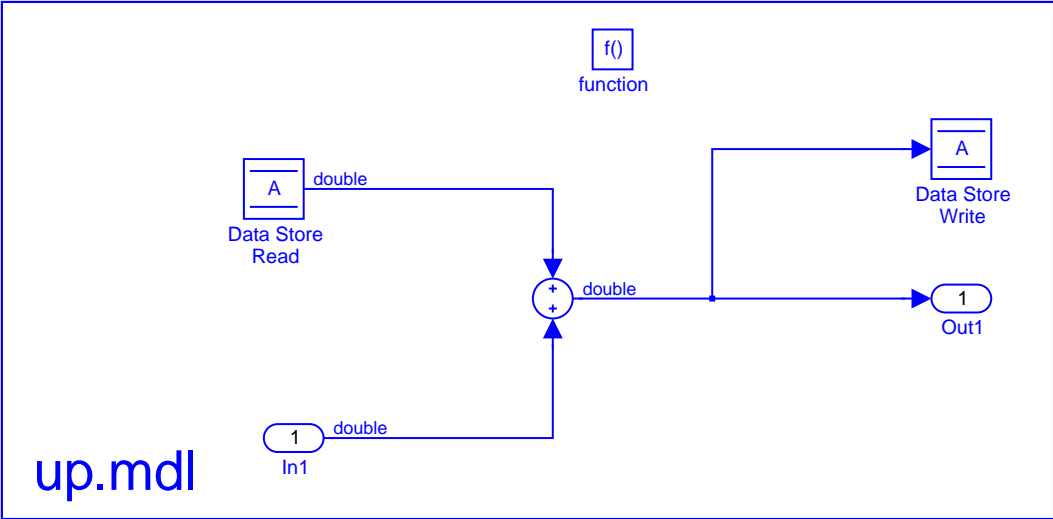
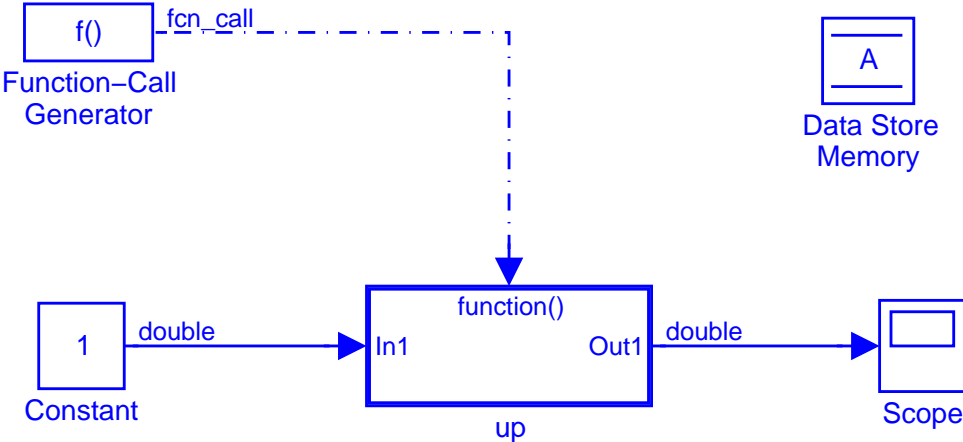
Do these allow bridging the enlarging cultural gap between Control and Computer sciences ?

Mode-Automata in Simulink

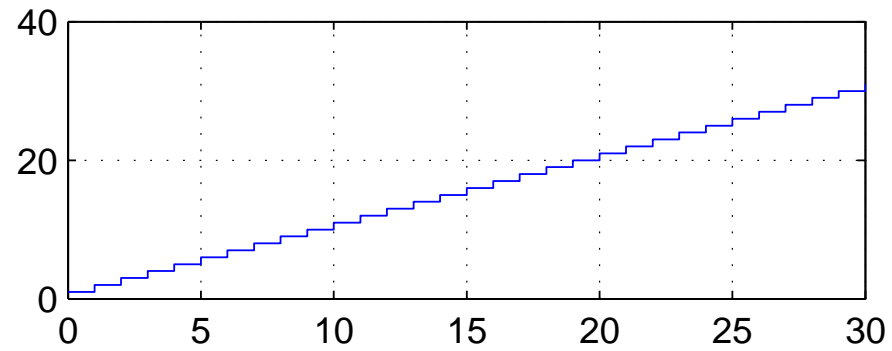
A cooperative way of designing complex systems:

- teams begin to agree on shared variable **names**,
- then each team can independently design and validate its own mode.

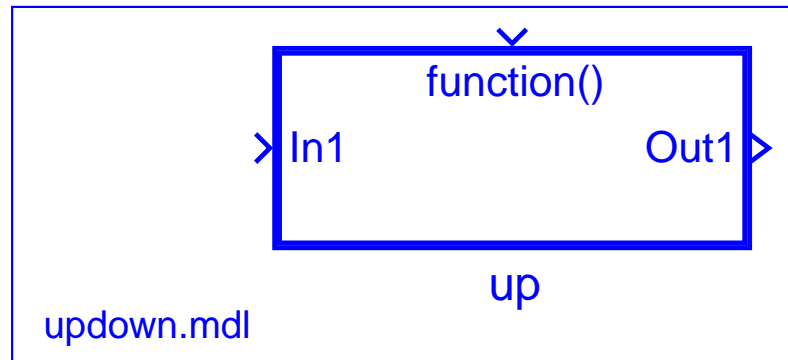
The Up team builds the “up” mode ...



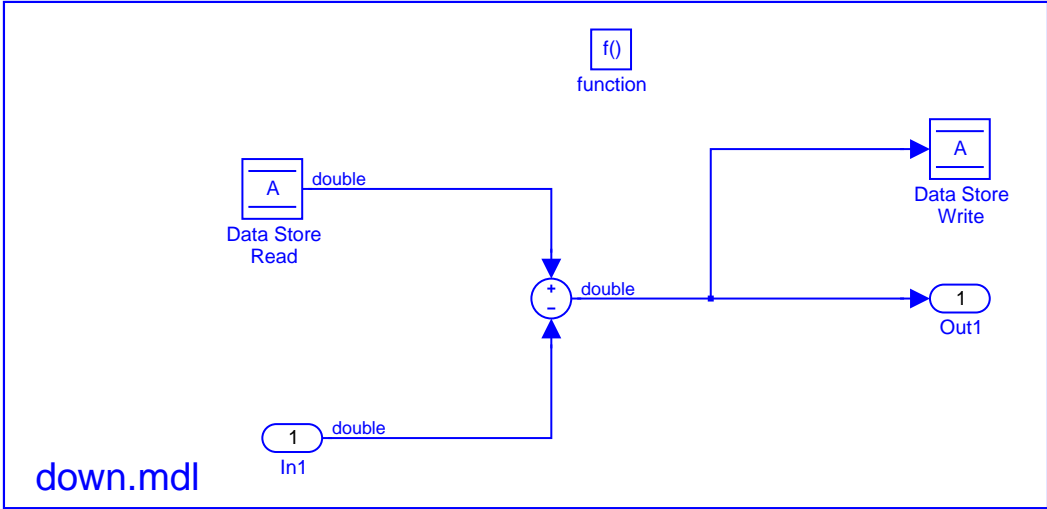
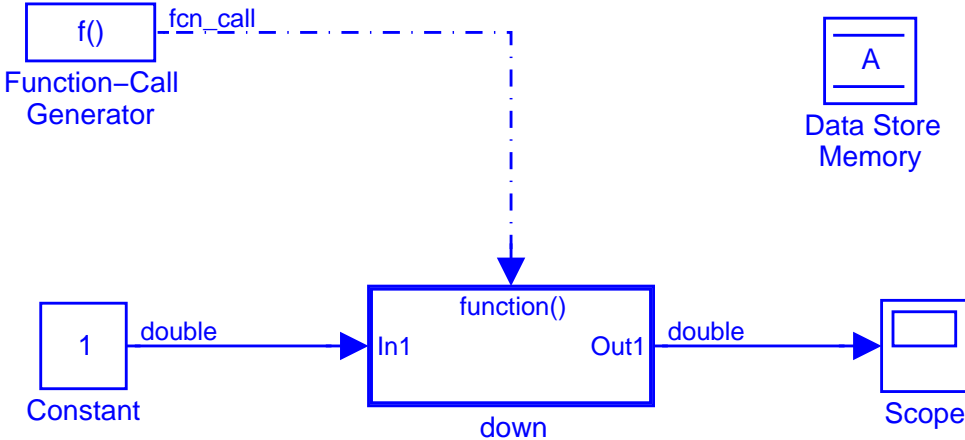
..., tries it and saves it



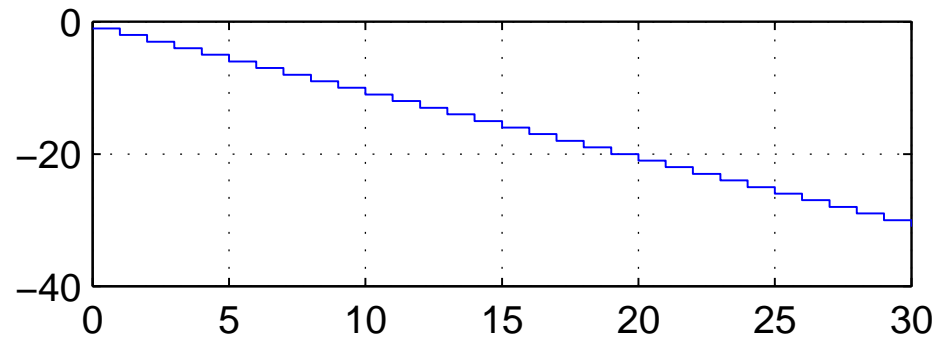
Time offset: 0



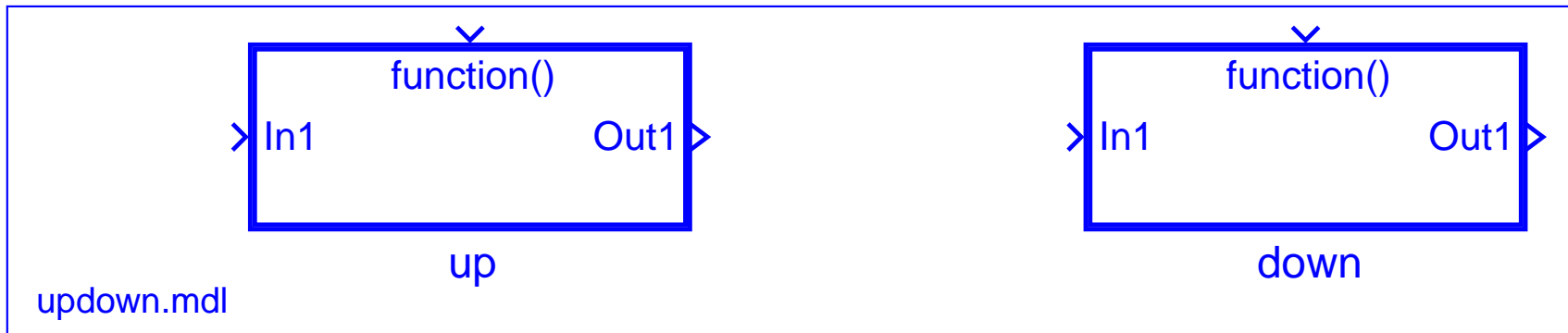
The Down team builds the “down” mode ...



..., tries it and saves it

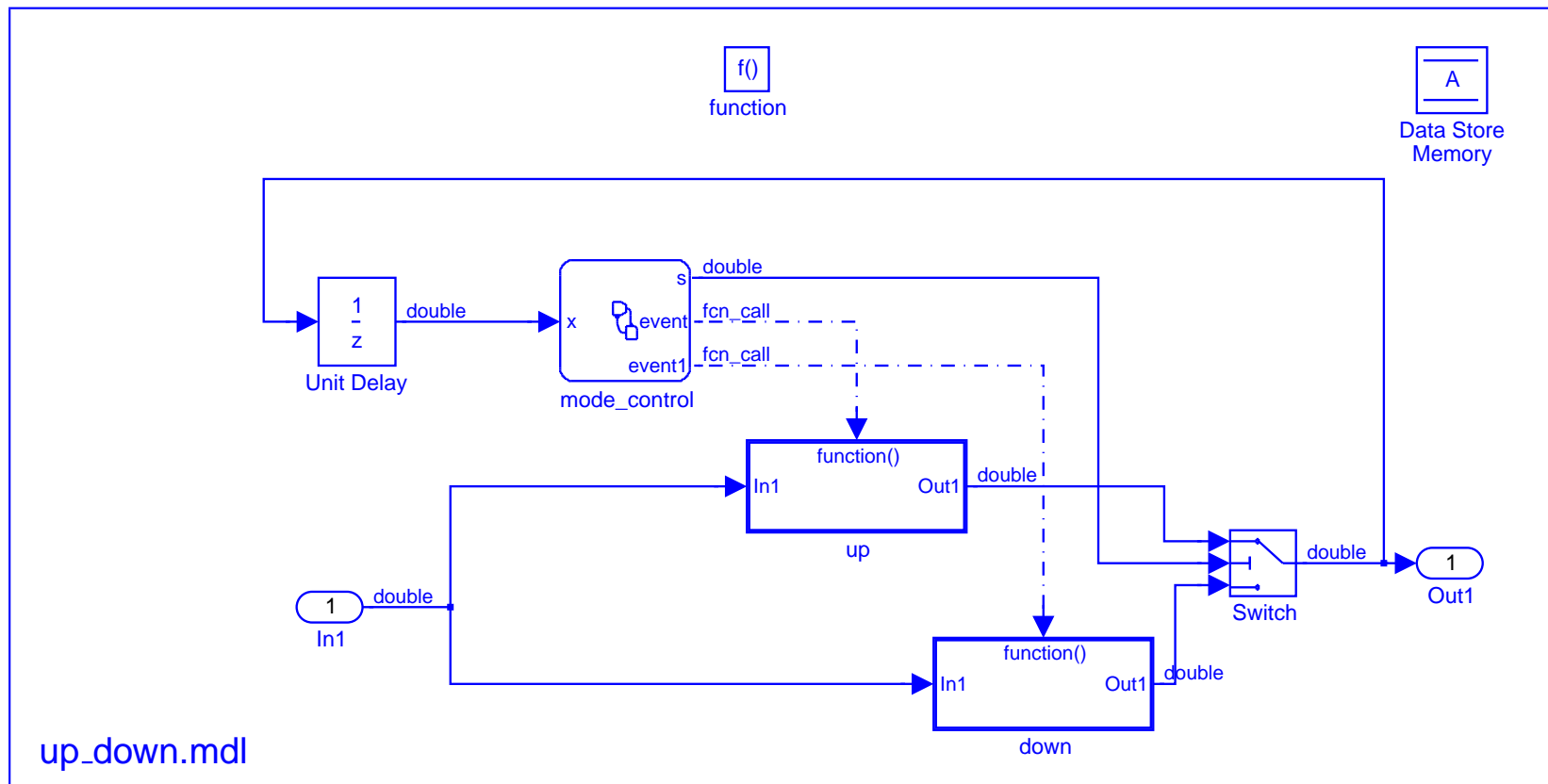
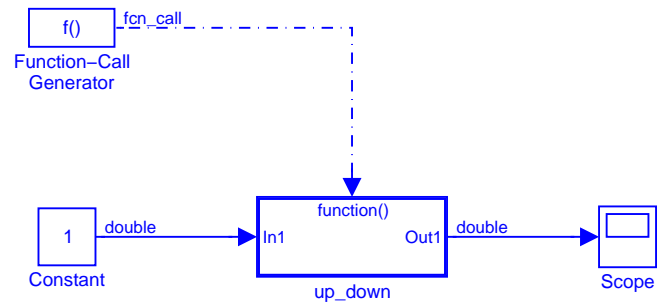


Time offset: 0



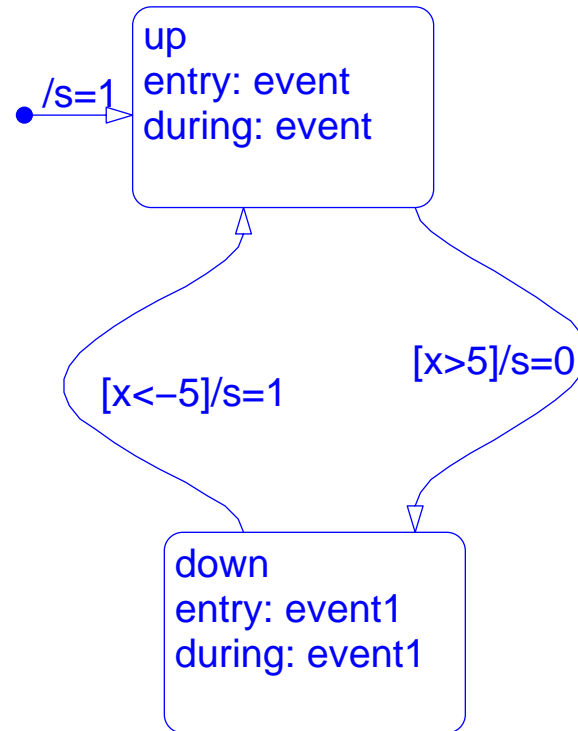
The Simulink library looks like a Java “static class”

The UpDown team combines the two models...

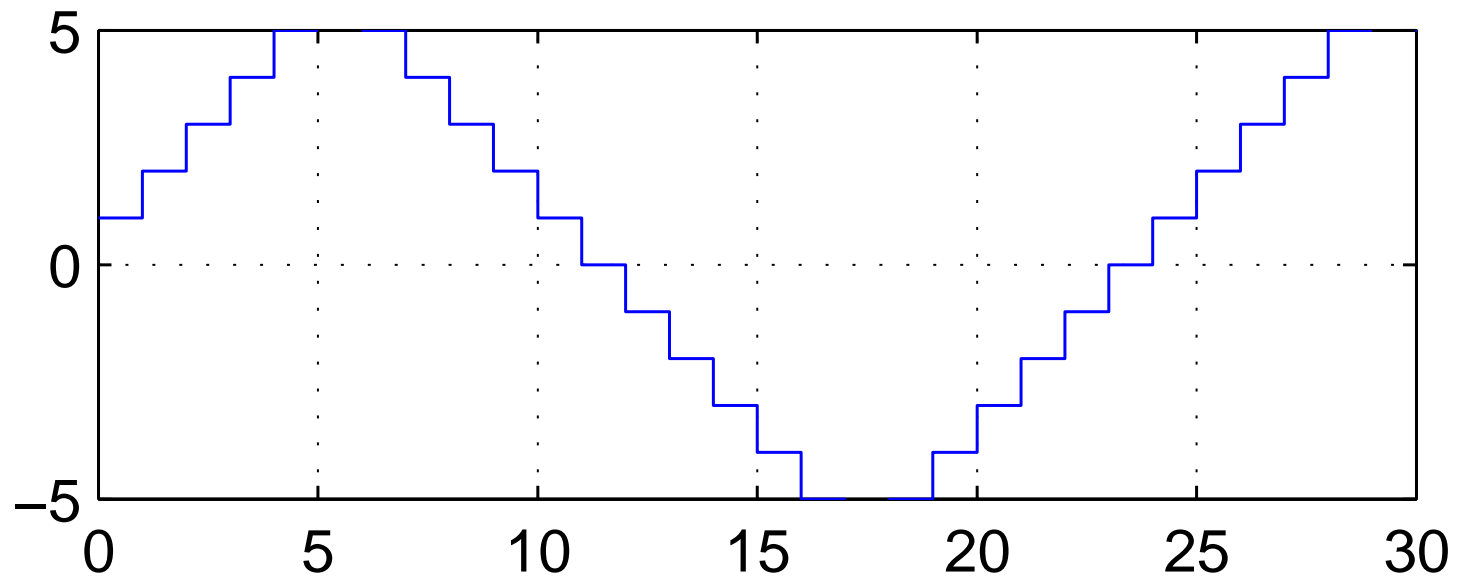


...through the activation automaton...

other_up_down/up_down/mode_control



...and tries it



Time offset: 0

It works...

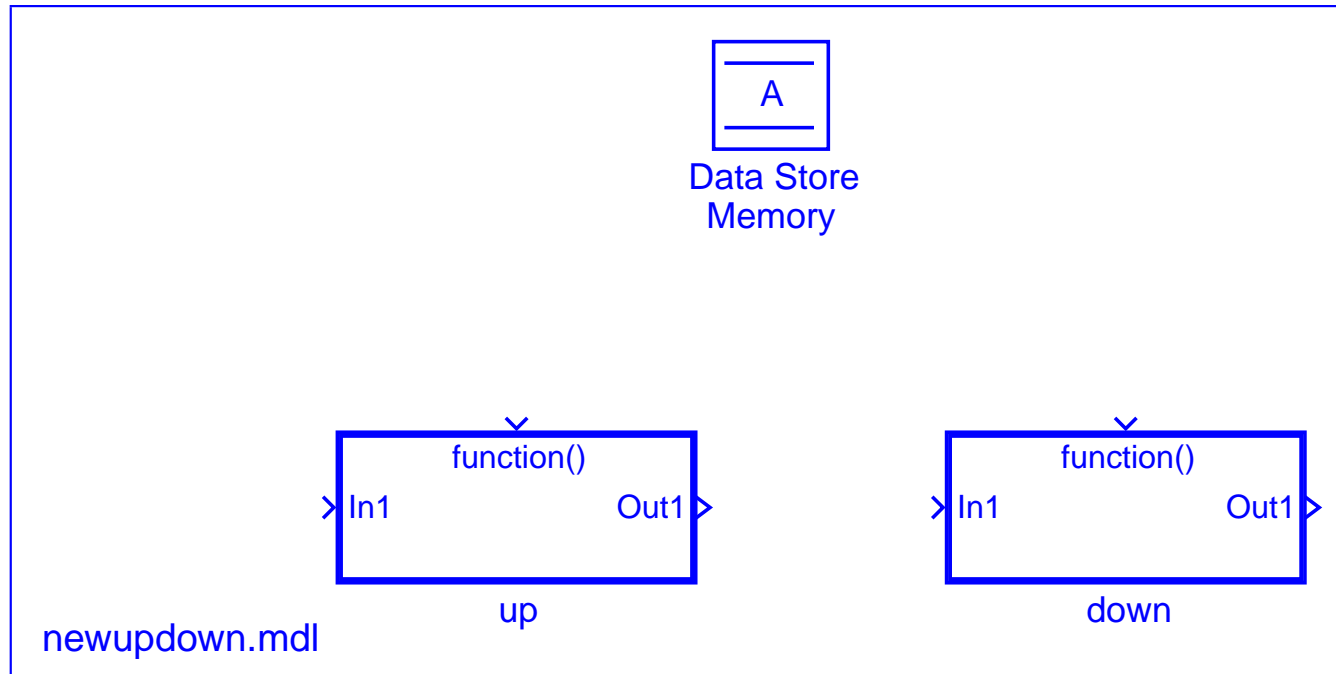
Interest and Drawbacks

- Interest :
 - Modular approach
 - No redesign
 - No complex wiring
- Drawbacks
 - Unsafe features
 - when activations are not exclusive
 - lexicographic execution order!!!
 - Dynamic binding
 - the shared variable binding is performed at instantiation

How to improve on it?

- Retrieve the advantages
 - last construct
 - no redesign
 - no complex wiring
- Propose a safe construct
 - statically check the clock mutual exclusion
 - (use the `merge` construct for instance)
- Move to a much safer static binding
 - binding takes place at the `definition`
 - libraries become true `classes !!!`

Libraries become true classes !!!



Simulink could now be endowed with a class diagram : the distance with UML would get narrower

Conclusion

This seems to show some need for

making control and computing cultures converge and cross fertilise

Thank you for your attention