The *true* inventors of Synchronous Programming! [a]

Marc Pouzet

LIENS

Institut Universitaire de France

Marc.Pouzet@ens.fr

Workshop SYNCHRON, Fréjus, Dec. 3rd, 2010

[a] With help from Timothy Bourke, Gwenael Delaval, Louis Mandel, Simon Pouzet, Pascal Raymond.

How to program/control systems in which time is involved?

Two families of systems...

Transformational systems

The machine expects an input, computes a result then stops.

Examples

- transform a colored image into a B&W image
- translate text
- ask for a route on a map (ViaMichelin, GoogleMap)
- a Google request: an input $\longrightarrow$ a result after a certain time

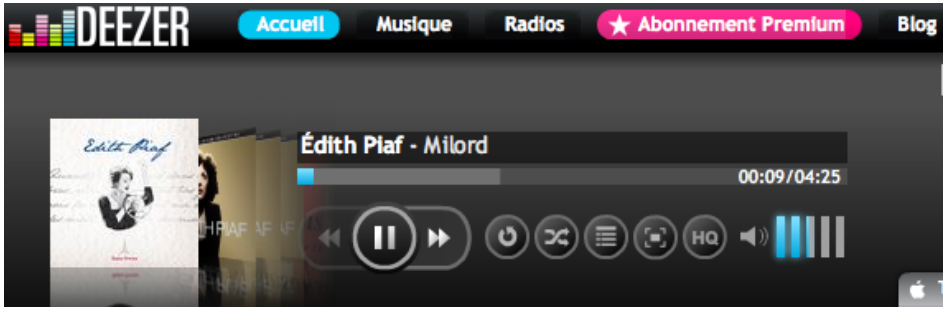
Here, time does not matter

Reactive systems

Interactions all the time; requests/observations/computations continuously

Examples

- a window manager; a keyboard
- a stream of music or video on the internet (Deezer, DailyMotion)



If the external environment is too fast,

it can wait or be slowed down

Here, physical real time does not really matter

Real-time reactive systems

A new constraint: the environment cannot wait.

A car: electronic ignition, trajectory control (ESP), braking system (ABS)

A plane: fly-by-wire control

They are now mostly programs

Problem: command **at the same time** the left and right wing and back spoilers; control rolling and pitching, etc.

The plane receives **at the same time** informations from the gyroscopes, anemometers, etc.

There is no longer a mechanical connection between the pilot and the actuators.

How to reproduce it with software?

Time and parallelism

- How is it possible to open several windows at the same time on a computer when it only has a single processor?
- It can only do one thing at a time.
- This is because it goes faster than our perception.
- This is quite easy when applications are independent from each other.

Does this parallelism help when programming planes and real-time systems?

Draw a platform

(Demo)

```
loop
  move platform
end
```

Two platforms?

Put two in parallel.

```
loop                        loop
  move platform1  ||          move platform2
end                         end
```

What's wrong?

Decompose time as a sequence of logical instants

- It executes a little bit of the first program, then a bit of the second one, etc.
- what about adding a new platform?
- The programmer has no simple way to control the scheduler's choices so that the platforms move at the same rhythm.
- No precise synchronization: the processor suits itself

Parallelism by *interleaving*

- `move platform1; move platform1; move platform1; move platform1; move platform2;;...`
- `move platform1; move platform1; move platform2; move platform1; move platform2;...`

Two painters (video)

Taken from “Je ne sais rien mais je dirai tout”, Pierre Richard, 1973.

Explanation

Victor Lanoux: **“tu dois compter jusqu’à trois, sinon tu perds un temps !”**

“you’ve got to count to 3, otherwise, you lose an instant!”

A notion of absolute logical time, shared by everyone.

Do not try to go as fast as possible but rather agree on a common global time scale.

Just respect the rhythm of the process that you are controlling

```
loop                          loop
  move platform1;      ||       move platform2;
  pause                         pause
end                           end
```

It has been called “The synchronous model of time” (Berry et al., 80’s)

Two synchronized platforms

(Demo with ReactiveML, Louis Mandel)

Is it really all that original?

The conductor of an orchestra: all musicians share a common time scale defined by the conductor.

Dancers: they synchronize on the music. This is the only way for several dancers to do the same thing.

Synchronous circuits: they synchronize on the ticks of a clock.

Reason ideally, neglecting the speed of light (orchestra), or of sound (dancers), or of electricity (circuits)

It is possible to determine later that this is a reasonnable approximation.

This is far simpler.

Global time is not necessarily fast

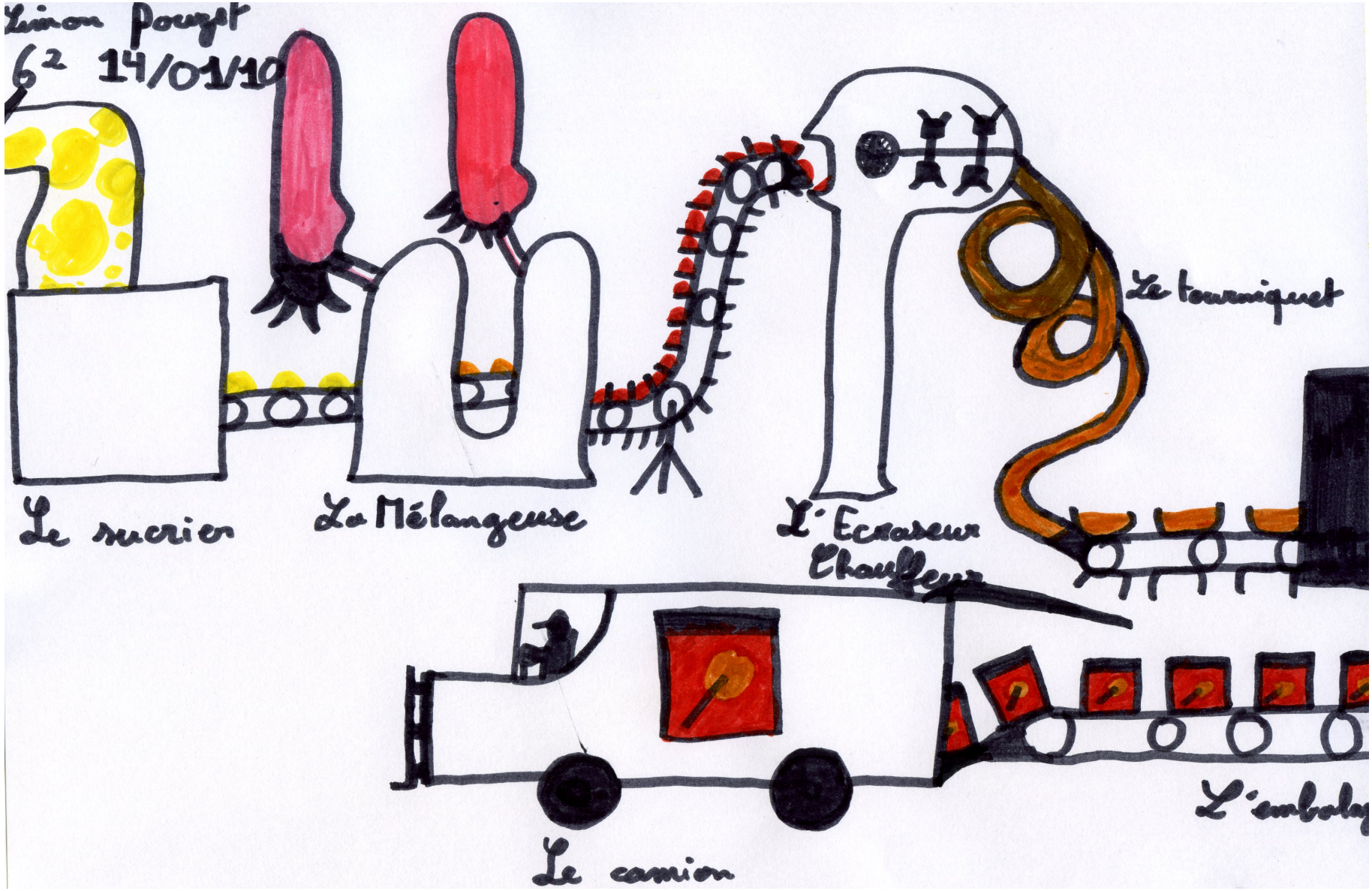
One time password

Compared to the time taken to enter a password, the clock drift is negligible.

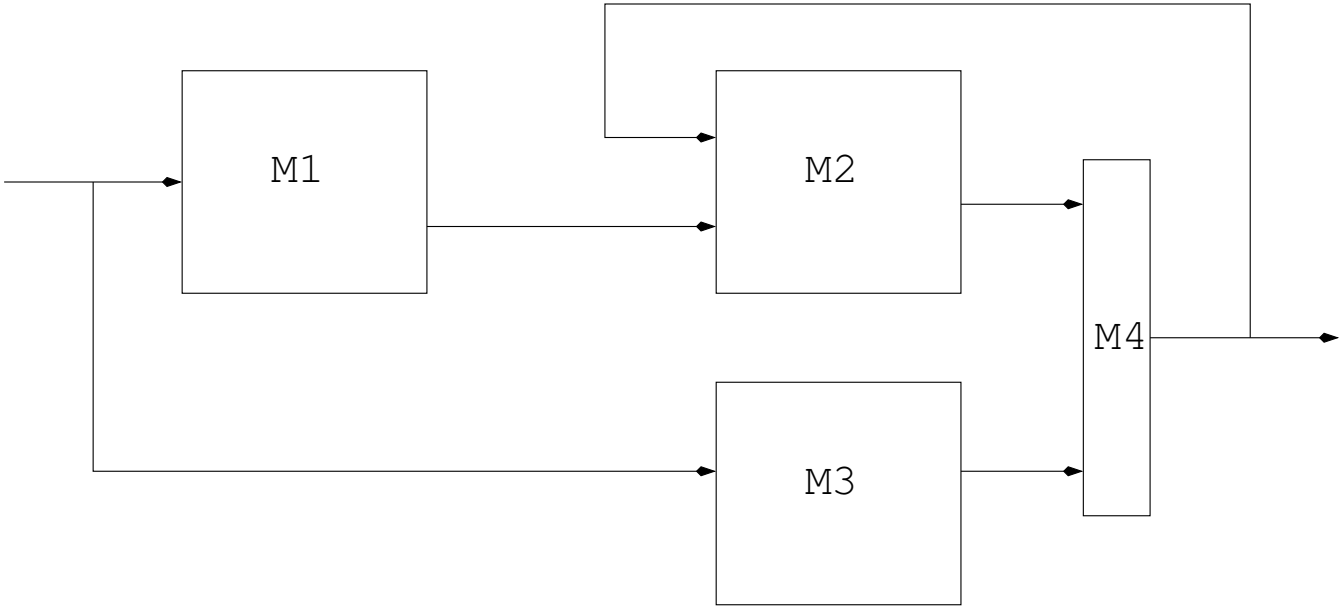
Thus, it is possible to act “as if” the calculation were instantaneous and to focus on the more important things.

Here, there is no synchronization. We rely on time.

Synchronous machines that communicate



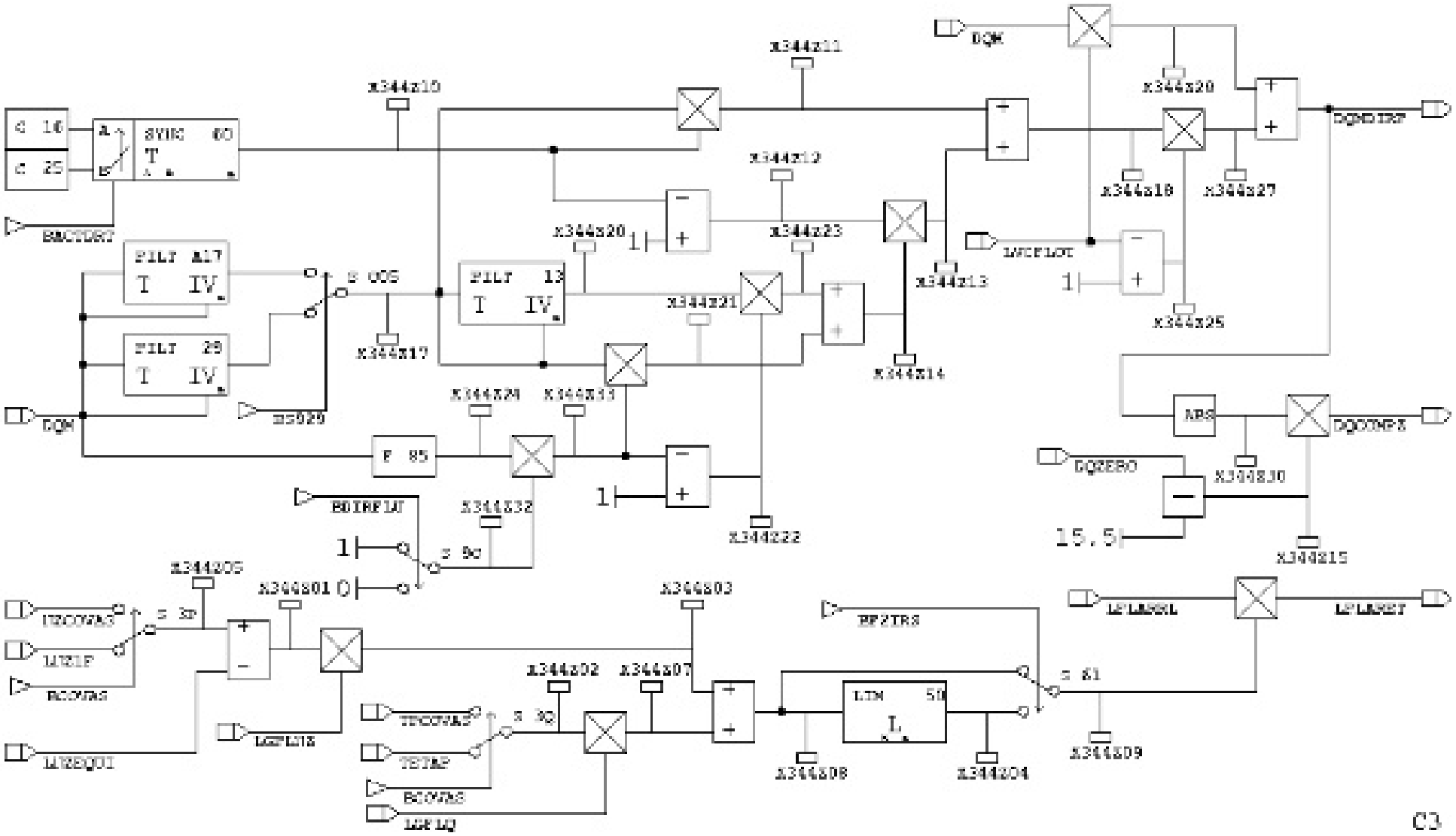
Synchronous Kahn Networks



- parallel processes communicating through data-flows
- communication in zero time: data is available as soon as it is produced.
- a shared notion of global time, even though individual rhythms may differ
- these drawings are not so different from actual computer programs

SAO (Spécification Assistée par Ordinateur) — Airbus 80's

Describe the system as block diagrams (synchronous communicating machines)



Programming with data-flow equations

The language Lustre (Caspi & Halbwachs, 1984).

X	1	2	1	4	5	6	...
Y	2	4	2	1	1	2	...
1	1	1	1	1	1	1	...
$X+Y$	3	6	3	5	6	8	...
$X+1$	2	3	2	5	6	7	...

The equation $Z = X + Y$ means that at every instant n, $Z_n = X_n + Y_n$.

Time is logical: the two inputs X and Y arrive "at the same time"; the output Z is produced at the very same instant.

Practically speaking, it suffices to check that the current output is produced before the input for the next instant arrives.

Memorizing values

We add operators to memorize the value produced at the previous instant.

`X`	1	2	1	4	5	6	...
`pre X`	nil	1	2	1	4	5	...
`Y`	2	4	2	1	1	2	...
`Y -> pre X`	2	1	2	1	4	5	...
`S`	1	3	4	8	13	19	...

The sequence (S_n) such that $S_0 = X_0$ and $S_n = S_{n-1} + X_n$ for all $n > 0$ is written:

```
S = X -> pre S + X
```

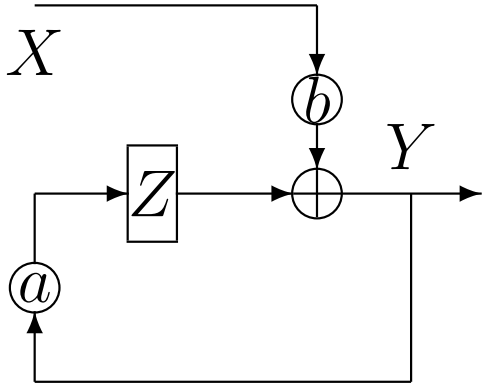
As in mathematics, intermediate equations can be introduced:

```
S = X -> I; I = pre S + X
```

A classical model of control theory and electronics

Example: a linear filter

$$Y_0 = bX_0 \ , \ \forall n \ Y_{n+1} = aY_n + bX_{n+1}$$



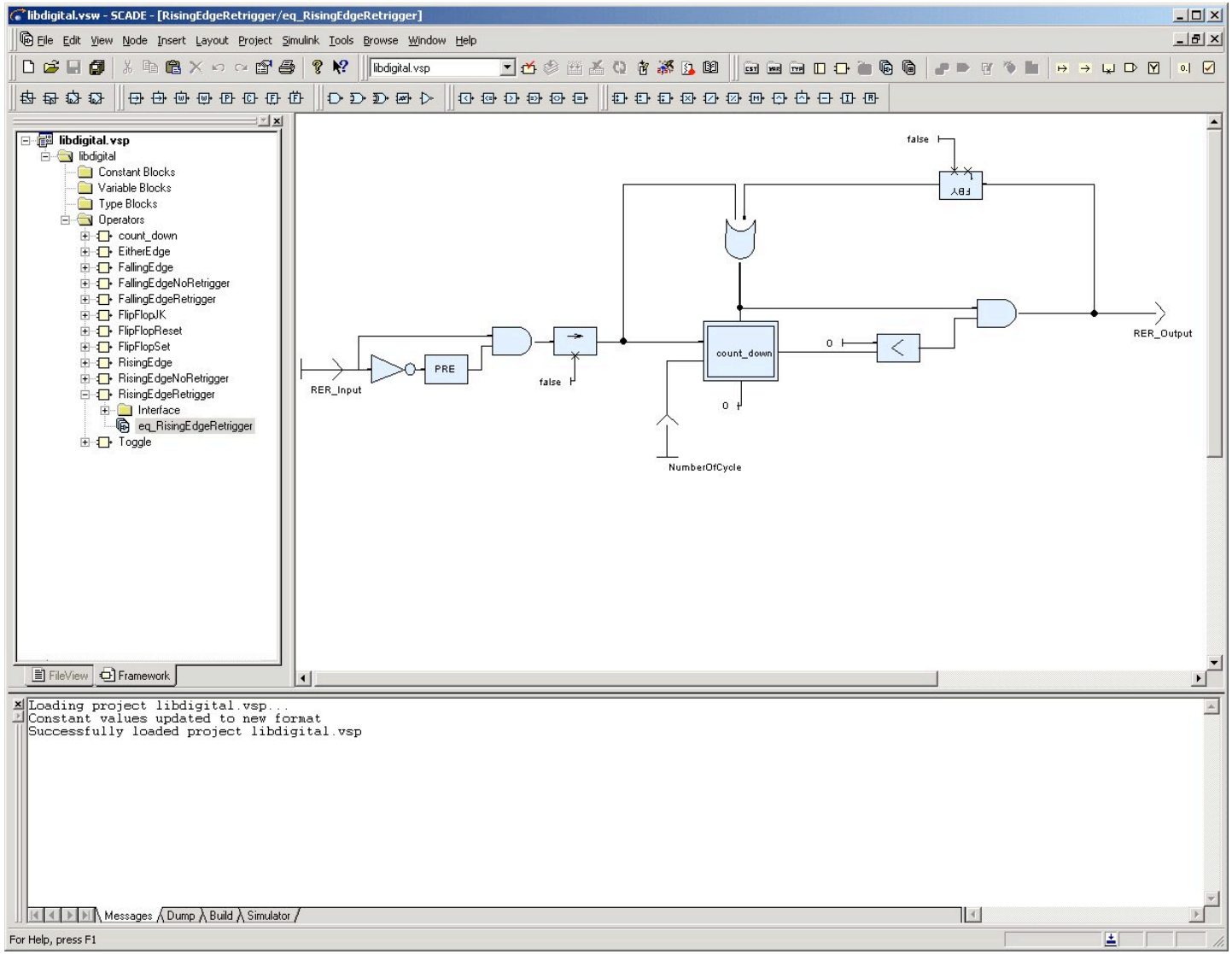
(demo — linear filter in Lustre)

The beautiful idea of Lustre:

- directly write mathematical equations
- analyze, transform and simulate them
- automatically translate them into executable programs

SCADE (Safety Critical Application Development Environment – Esterel-Tech.)

From computer assisted drawings to executable programs



Merge different rhythms

(Video – bricks)

A single brick at a time; bricks must not arrive too fast...

- $\mathtt{G_i} = 1$ when the brick is on my left
- $\mathtt{D_i} = 1$ when the brick is on my right

$\mathtt{G_1}$	1	0	0	0	0	0	...
$\mathtt{D_1}$	0	1	0	0	0	0	...
$\mathtt{G_2}$	0	1	0	0	0	0	...
$\mathtt{D_2}$	0	0	1	0	0	0	...
$\mathtt{G_3}$	0	0	1	0	0	0	...
$\mathtt{D_3}$	0	0	0	1	0	0	...

Merge different rhythms

(Video – bricks)

A single blick at a time; bricks must not arrive too fast...

- $\mathtt{G_i} = 1$ when the brick is on my left
- $\mathtt{D_i} = 1$ when the brick is on my right

$\mathtt{G_1}$	1	0	1	0	0	0	...
$\mathtt{D_1}$	0	1	0	1	0	0	...
$\mathtt{G_2}$	0	1	0	1	0	0	...
$\mathtt{D_2}$	0	0	1	0	1	0	...
$\mathtt{G_3}$	0	0	1	0	1	0	...
$\mathtt{D_3}$	0	0	0	1	0	1	...

Synchronize them

(Vidéo – briques)

Taken from “Je ne sais rien mais je dirai tout”, Pierre Richard, 1973.

What happened?

Two different non synchronizable rhythms!

First case:

- Pierre Richard can only do one thing at a time ($\mathtt{C}$ (pull up the cord) or G_6 (take brick on the left)).
- The cord must always be pulled, i.e., $\mathtt{C} = 1$.

Second case:

- He does two things at a time (that is, it goes twice as fast).
- This time, it works... almost!

Is it that important to be perfectly synchronous?

Check rhythms

- Find methods to check that all rhythms are correct.
- Once the rhythms are known, establish once and for all who is going to work and when.

Even if the rhythms are correct and well synchronized, the brick must still arrive at the right time.

Solution: a table to place the brick (buffer). This introduces a delay (latency).

This is exactly what happens in a digital TV (TVB-T) *vs* an analog TV: the first image arrives with a delay of one second.

Buffers are used to synchronise image and sound.

Conclusion

Synchronous languages were invented to design/program real-time reactive systems. They are used in the large (planes, subways, trains, etc.)

Synchronous model of time

- A notion of **absolute logical time**, shared by everyone.
- **reason ideally**, then check **a posteriori** that the implementation is fast enough

Programming by writing mathematical equations

- a system = a set of data-flow equations
- just like the classical model from control, electronics and signal processing
- these mathematical equations are automatically translated into executable code, reducing the risk of bugs