Synchronous languages

Lecture 3:
LUSTRE (full version)

ENSEEIHT 3A – parcours E&L
2021/2022

Frédéric Boniol (ONERA)
frederic.boniol@onera.fr

Simplified LUSTRE: first conclusion

So far…

LUSTRE is a data flow synchronous language:

- Determinsitic semantics
- Functional semantics (node are similar to pure functions, without side-effect)
- Modular programming (reuse of nodes by nodes)
- Bounded execution time (no dynamic processes, no infinite loop…)
- Bounded memory (no data creation)

However

Only one clock (the sampling clock)

⇒ All the flows are sampled on the same clock

⇒ However, real embedded systems are sampled on multiple clocks

=> Full LUSTRE = multi-clock programming language

Full LUSTRE: main concepts

Clock

- A clock is a Boolean flow

Basic clock

- The most frequent sampling clock in the system
- Is equal to the constant flow true

Semantics of flows

- Each flow is typed by a clock (the clock of the flow)
- Let X be a flow, and let C be the clock of X (clock(X)=C), then
  - X is present (with a defined value) whenever C is true
  - X is absent (without any value) whenever C is false

Full LUSTRE: back to the main concepts

Equation and clock

Equations must be homogenous in term of clock

Example :

Equation

```
Z = X + Y
```

makes sense iff X and Y have the same clock ; in that case, Z has the same clock as X and Y.

Otherwise, this equation is not correct.

Node and clock

=> A node can receive input flows on different clocks

=> The clock of the node (called the basic clock of the node from the internal point of view of the node) is the union of the clocks of its input flows.

Full LUSTRE: sampling operators

Two new temporal operators

- Under-sampling

  when

- Over-sampling

  current

=> **when** and **current** allow to change the clock of a flow

Full LUSTRE: Under-sampling

Under-sampling operation: when

Let X be a flow and B a Boolean flow (assimilated to a clock) of same clock;

The equation

```
Y = X when B
```

defines a flow Y, of same type than X, and of clock B, defined by

- When B is true, Y is present and the value of Y is the value of X
- When B is false, Y is absent
- When B is absent, Y is absent

⇒ **when** returns a less frequent flow than its input

Example:

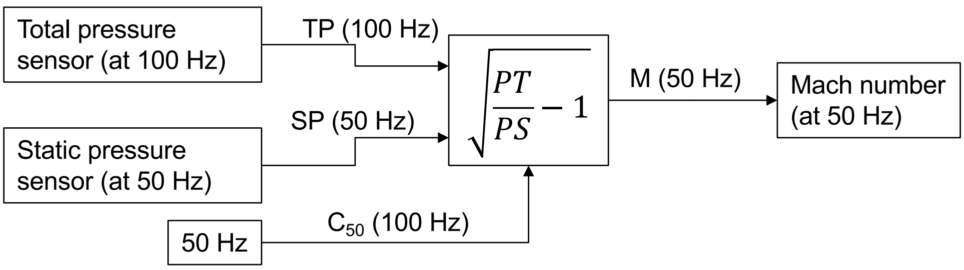
Top of the clock of X and B

	n=0	n=1		n=2	n=3	n=4	n=5	…
X_n	4	1		0	2	7	8	
B_n	true	false		true	true	false	true	…
$(X \text{ when } B)_p$	4			0	2		8	…
	p=0			p=1	p=2		p=3	…

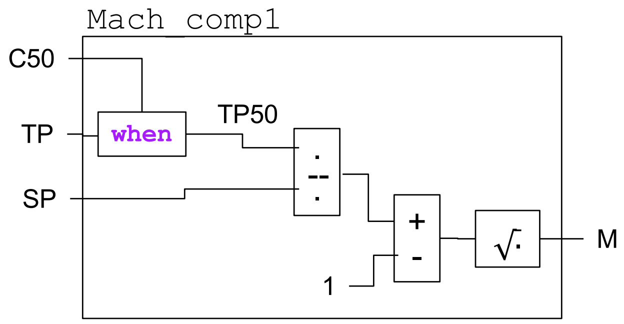
Top of the clock of X **when** B

Full LUSTRE: Under-sampling

Example: Mach number computation from pressure sensors at different frequency




```
node Mach_comp1 (TP:real;
                 C50:bool;
                 SP:real when C50)
returns (M:real when C50)
var TP50:real when C50;
let
   M = sqrt( (TP50 / SP) - 1);
   TP50 = TP when C50;
tel.
```



Full LUSTRE: Over-sampling

Over-sampling operation: `current`

Let X be a flow, and let B be the clock of X (assimilated to a Boolean flow) ;

The equation

Y = `current` X

defines a flow Y, of same type than X, defined by

- When X is present, Y is present and the value of Y is the value of X
- When X is absent and B (the clock of X) is present (with value false), then Y is present with the last value of X
- When X is absent and B (the clock of X) is absent, then Y is absent

⇒ `current` returns a more frequent flow than its input

Example:

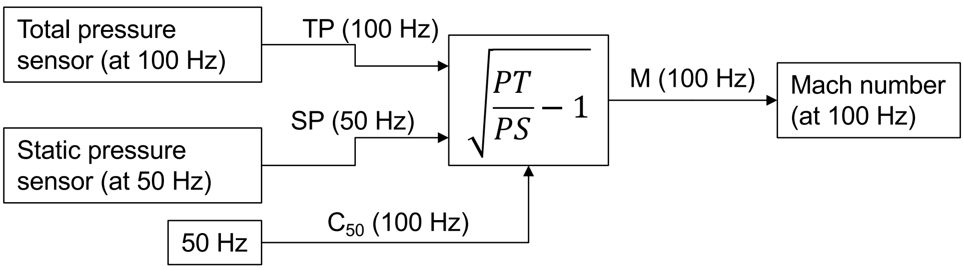
Top of the clock of X

	p=0			p=1	p=2		p=3	…
X_p	4			0	2		8	…
B_n	true	false		true	true	false	true	…
(`current` X)$_n$	4	4		0	2	2	8	…
	n=0	n=1		n=2	n=3	n=4	n=5	…

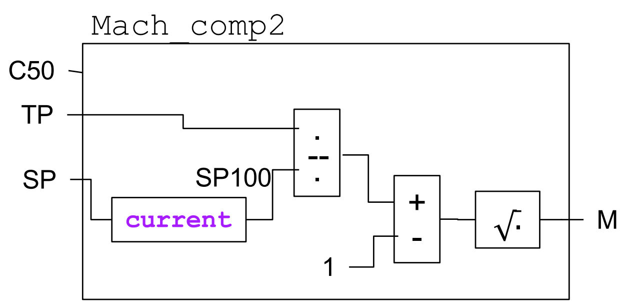
Top of the clock of B and `current` X

Full LUSTRE: Under-sampling

Example: Mach number computation from pressure sensors at different frequency




```
node Mach_comp2 (TP:real;
                 C50:bool;
                 SP:real when C50)
returns (M:real)
var SP100:real;
let
   M = sqrt( (TP / SP100) - 1);
   SP100 = current SP;
tel.
```



Full LUSTRE: full examples

Example : timer2

- Input set : bool — activation of the timer (Boolean flow)
- Output on : bool — state of the timer (Boolean flow)
- Constant d : int — duration of the timer (w.r.t the global clock)

```
const d : int;
node timer2 (set : bool) returns (on : bool)
var count : int; C : bool;
let
   on = count > 0;
   count = if set then d
           else if (C)then pre(count) - 1
                else (0 -> pre(count));
C = false -> pre(on);
tel.
```

Full LUSTRE: full examples

Example : timer2

```
const d : int;
node timer2 (set : bool) returns (on : bool)
var count : int;
let
   on = count > 0;
   count = if set then d
           else if (false -> pre(on)) then pre(count) - 1
                else (0 -> pre(count));
tel.
```

	n=0	n=1	n=2	n=3	n=4	n=5	n=5	n=7	n=8	n=9	n=10
D_n	2	2	2	2	2	2	2	2	2	2	2
set_n	false	true	false	false	true	true	false	false	false	false	false
on_n	false	true	true	false	true	true	true	false	false	false	false

Full LUSTRE: full examples

Example : param_timer2

- Input `set : bool` activation of the timer (Boolean flow)
- Input `ck : bool`
- Output `on : bool` state of the timer (Boolean flow)
- Constant `d : int` duration of the timer (w.r.t ck)

Idée : reuse of timer2

```
const d : int;
node param_timer2(set, ck : bool) returns (on : bool)
var tick : bool;
let
   level = current(timer2(set when tick));
   tick = true -> (set or ck);
tel.
```

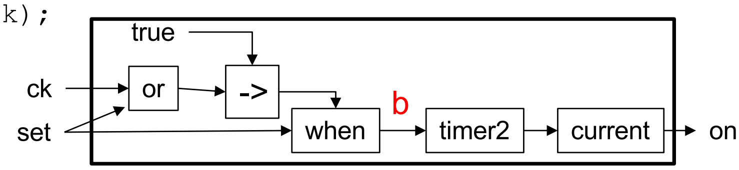
=> `timer2` is activated each time `tick` is true, that is, at each true occurrence of `ck` and at each occurrence true of `set`.

=> The internal clock of timer2 is `tick`

Full LUSTRE: full examples

Example : param_timer2

```
const d : int;
node param_timer2(set, ck : bool) returns (on : bool)
var tick : bool;
let
   level = current(timer2(set when tick));
   tick = true -> (set or ck);
tel.
```



	n=0	n=1	n=2	n=3	n=4	n=5	n=5	n=7	n=8	n=9	n=10
D_n	2	2	2	2	2	2	2	2	2	2	2
ck_n	false	**true**	false	false	**true**	false	false	**true**	false	false	**true**
set_n	false	false	**true**	false	false	false	false	false	false	false	false
b_p	false	false	true		false			false			false
on_n	false	false	false	false	false	false	false	**true**	false	false	false

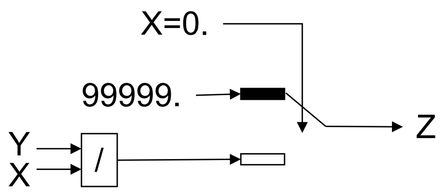
Full LUSTRE: full examples

Exercise:

Let X and Y be two real flows

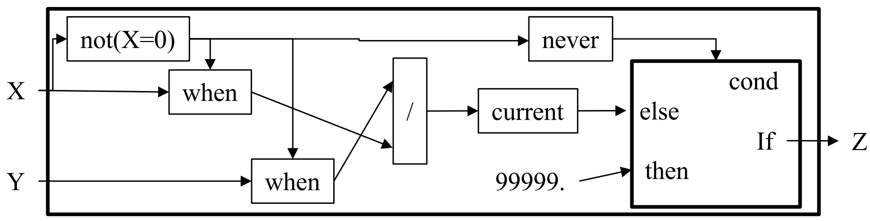
Let Z be defined by

Z = if X=0. then 99999. Else Y/X



This equation fails at run time when X is equal to zero. Why ? How to fix it ?

```
node guarded_div(X, Y : real)
returns (Z : real)
var ck : bool;
let
        Z = if never(ck) then current( (X when ck) / (Y when ck)))
                else 10000. ;
        ck = not (Y=0);
tel.
```



Full LUSTRE: full examples

Exercise : node scheduling

Let us consider the following program

```
node P (X:int) returns (Z:int)
var Y:int;
let
  Y = F1(X);
  Z = F2(Y);
tel.
```

X → [F1] → Y → [F2] → Z

(where F1 and F2 are two Lustre nodes)

We want to execute F1 and F2 alternatively:

n=0	n=1	n=2	n=3	n=4	n=5	n=5	n=7	n=8	n=9	n=10
F1		F1		F1		F1		F1		F1
	F2		F2		F2		F2		F2	

⇒ F1 is scheduled on clock B1

⇒ F2 is scheduled on clock B2

	n=0	n=1	n=2	n=3	n=4	n=5	n=5	n=7	n=8	n=9	n=10
$B1_n$	true	false	true	false	true	false	true	false	true	false	true
$B2_n$	false	true	false	true	false	true	false	true	false	true	false

⇒ Modify the node P to implenent this scheduling

Full LUSTRE: full examples

Exercise : node scheduling

```
node P (X:int) returns (Z’:int)
var Y:int when B1; Y’:int; Z when B2;
    B1:bool; B2:bool;
let
  B1 = true -> not pre(B);
  B2 = not B1;
  Y = F1(X when B1);
  Y’ = current Y;
  Z = F2(Y’ when B2);
  Z’ = 0 -> current Z;
tel.
```

	n=0	n=1	n=2	n=3	n=4	n=5	n=5	n=7	n=8
B1	true	false	true	false	true	false	true	false	true
B2	false	true	false	true	false	true	false	true	false
X	X0	X1	X2	X3	X4	X5	X6	X7	X8
F1(X when B1)	Y0		Y2		Y4		Y6		Y8
current Y	Y0	Y0	Y2	Y2	Y4	Y4	Y6	Y6	Y8
F2(Y’ when B2)		Z0		Z2		Z4		Z6	
current Z	nil	Z0	Z0	Z2	Z2	Z4	Z4	Z6	Z6
Z’	0	Z0	Z0	Z2	Z2	Z4	Z4	Z6	Z6

Full LUSTRE: summary

B	false	true	false	true	false	false	true	true
X	x0	x1	x2	x3	x4	x5	x6	x7
Y	y0	y1	y2	y3	y4	y5	y6	y7
pre(X)	nil	x0	x1	x2	x3	x4	x5	x6
Y->pre(X)	y0	x0	x1	x2	x3	x4	x5	x6
Z=X when B		x1		x3			x6	x7
T=current Z	nil	x1	x1	x3	x3	x3	x6	x7
pre(Z)		nil		x1			x3	x6
0->pre(Z)		0		x1			x3	x6

Full LUSTRE: conclusion

LUSTRE is

- A language dedicated to control systems
- Based on a time simplification
- Based on a synchronous data flow semantics

Advantage

- Simple language with a formal semantics
- Verification done at compile time
  - Causality
  - Type consistency
  - Clock consistency

  ⇒ Any program which succeeds at compile time does not crash at run time!

Drawback

- Not suitable for dynamic behaviour
- Do not allow expression of unbounded loop (for i=1 to n …)

=> **The price to pay for simplicity!**

End of lecture 3

⇒Next lecture: Formal aspects of the LUSTRE language