

# CS:5810 Formal Methods in Software Engineering

## Reactive Systems and the Lustre Language<sup>1</sup> Part 2

Adrien Champion    Cesare Tinelli

---

<sup>1</sup>Copyright 2015-21, Adrien Champion and Cesare Tinelli, the University of Iowa. These notes are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holder.

# Lustre: a synchronous dataflow language

Design of **reactive** systems:

- run in an infinite loop, and
- produce an output every  $n$  milliseconds

clock

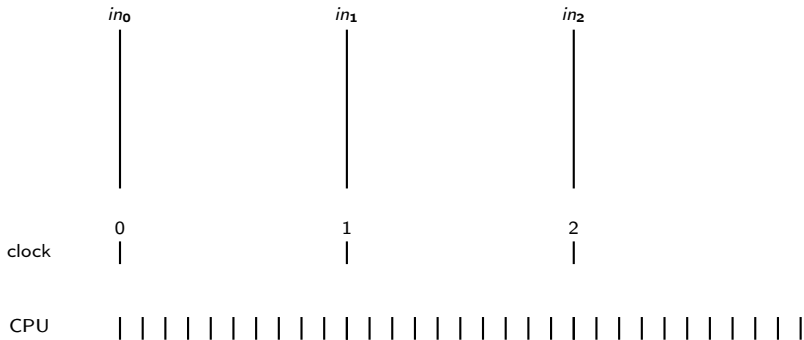
CPU



# Lustre: a synchronous dataflow language

Design of **reactive** systems:

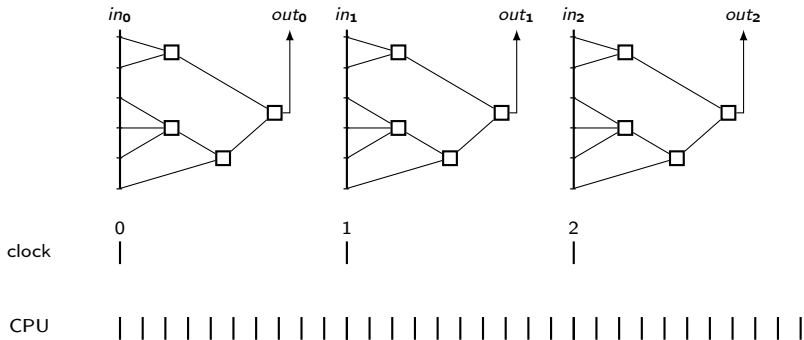
- run in an infinite loop, and
- produce an output every  $n$  milliseconds



# Lustre: a synchronous dataflow language

Design of **reactive** systems:

- run in an infinite loop, and
- produce an output every  $n$  milliseconds



# Exercises

Model a switch with two buttons, Set and Reset.

```
node Switch( Set, Reset, Init : bool )  
returns ( Out : bool );
```

such that:

- pressing Set turns the switch on;
- pressing Reset turns the switch off;
- the initial position of the switch is determined by a third signal `Init` if Set and Reset are initially both unpressed.

# Exercises

Model a switch with two buttons, Set and Reset.

```
node Switch( Set, Reset, Init : bool )
returns ( Out : bool );
```

such that:

- pressing Set turns the switch on;
- pressing Reset turns the switch off;
- the initial position of the switch is determined by a third signal Init if Set and Reset are initially both unpressed.

```
node Switch( Set, Reset, Init : bool )
returns ( X : bool );
let
    X =      if Set then true
            else if Reset then false
            else (Init -> pre X);
tel
```

# Exercises

Model a switch with two buttons, Set and Reset.

```
node Switch( Set, Reset, Init : bool )
returns ( Out : bool );
```

such that:

- pressing Set turns the switch on;
- pressing Reset turns the switch off;
- the initial position of the switch is determined by a third signal Init if Set and Reset are initially both unpressed.

Equivalently, and more concisely:

```
node Switch( Set, Reset, Init : bool )
returns ( X : bool );
let
  X = Set or (not Reset and (Init -> pre X)) ;
tel
```

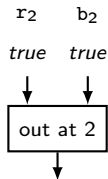
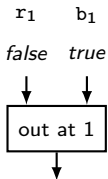
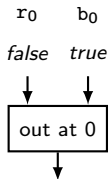
# Exercises

```
node ??? (r,b: bool) returns (out: int);  
let  
  
    out =      if r then 0  
               else if b then (0 -> pre out) + 1  
               else          (0 -> pre out);  
tel
```



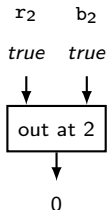
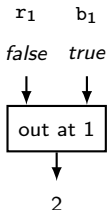
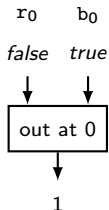
# Exercises

```
node ??? (r,b: bool) returns (out: int);  
let  
  
    out =      if r then 0  
              else if b then (0 -> pre out) + 1  
              else          (0 -> pre out);  
tel
```



# Exercises

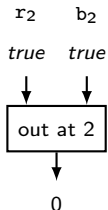
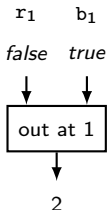
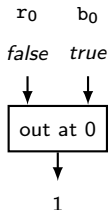
```
node ??? (r,b: bool) returns (out: int);  
let  
  
    out =      if r then 0  
              else if b then (0 -> pre out) + 1  
              else          (0 -> pre out);  
tel
```



# Exercises

Counter with reset:

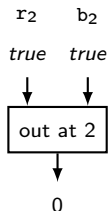
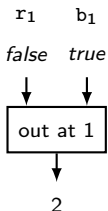
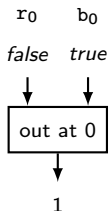
```
node ??? (r,b: bool) returns (out: int);  
let  
  
    out =      if r then 0  
              else if b then (0 -> pre out) + 1  
              else          (0 -> pre out);  
  
tel
```



# Exercises

Counter with reset:

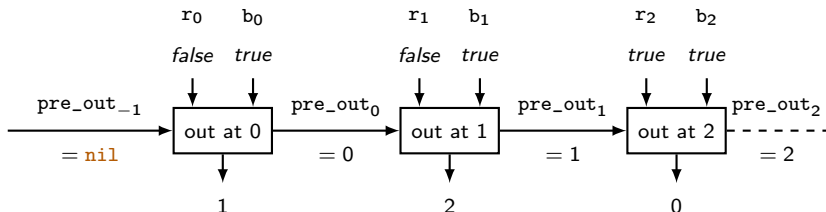
```
node cnt (r,b: bool) returns (out: int);  
var pre_out: int;  
let pre_out = 0 -> pre out;  
  out =      if r then 0  
            else if b then pre_out + 1  
            else      pre_out;  
tel
```



# Exercises

Counter with reset:

```
node cnt (r,b: bool) returns (out: int);  
var pre_out: int;  
let pre_out = 0 -> pre out;  
    out =      if r then 0  
              else if b then pre_out + 1  
              else      pre_out;  
tel
```



# Modularity

---

Once defined, a node can be used as a basic operator

# Modularity

---

Once defined, a node can be used as a basic operator

What does A look like?

```
X = true -> (pre A = 3)
A = cnt(X, true);
```

# Modularity

---

Once defined, a node can be used as a basic operator

What does A look like?

```
X = true -> (pre A = 3)
```

```
A = cnt(X, true);
```

```
A = 0,
```



# Modularity

---

Once defined, a node can be used as a basic operator

What does A look like?

```
X = true -> (pre A = 3)
```

```
A = cnt(X, true);
```

```
A = 0, 1,
```

# Modularity

---

Once defined, a node can be used as a basic operator

What does A look like?

```
X = true -> (pre A = 3)
```

```
A = cnt(X, true);
```

```
A = 0, 1, 2,
```

# Modularity

---

Once defined, a node can be used as a basic operator

What does A look like?

```
X = true -> (pre A = 3)
```

```
A = cnt(X, true);
```

```
A = 0, 1, 2, 3,
```

# Modularity

---

Once defined, a node can be used as a basic operator

What does A look like?

```
X = true -> (pre A = 3)
```

```
A = cnt(X, true);
```

```
A = 0, 1, 2, 3, 0,
```

# Modularity

---

Once defined, a node can be used as a basic operator

What does A look like?

```
X = true -> (pre A = 3)
```

```
A = cnt(X, true);
```

```
A = 0, 1, 2, 3, 0, 1, 2, 3, 0, 1...
```

# Modularity

A node can have several outputs:

```
node MinMax( X : real ) returns ( Min, Max : real );  
let  
  Min = X -> if (X < pre Min) then X else pre Min ;  
  Max = X -> if (X > pre Max) then X else pre Max ;  
tel
```

```
node minMaxAverage ( X: real ) returns ( Y: real ) ;  
var Min, Max: real ;  
let  
  Min, Max = MinMax(X) ;  
  Y = (Min + Max)/2.0 ;  
tel
```

# Complete example: specification

## Stopwatch:

- one integer output: `time` “to display”;
- three input buttons:
  - `on_off` starts and stops the stopwatch,
  - `reset` resets the stopwatch **if not running**,
  - `freeze` freezes the displayed time **if running**, cancelled if stopped

# Complete example: available nodes

```
-- Bistable switch
node switch (on, off: bool) returns (out: bool);
let
  out =
    if (false -> pre out) then not off else on;
tel

-- Counts steps if inc is true, can be reset
node counter (reset, inc: bool) returns (out: int);
let
  out =      if reset then 0
             else if inc then (0 -> pre_out) + 1
             else           (0 -> pre_out);
tel

-- Detects raising edges of a signal
node edge (in: bool) returns (out: bool);
let
  out = false -> in and (not pre in);
tel
```



# Complete example: solution(s)

Unsatisfactory solution not using edge:

```
node stopwatch (on_off, reset, freeze: bool)
returns (time: int);
var actual_time: int;
    running, frozen: bool;

let

    running = switch(on_off, on_off);
    frozen = switch(
        freeze and running, freeze or on_off
    );
    actual_time = counter(reset and not running, running);
    time = if frozen then (0 -> pre time) else actual_time;
tel
```

# Complete example: solution(s)

Satisfactory solution:

```
node stopwatch (on_off, reset, freeze: bool)
returns (time: int);
var actual_time: int;
    running, frozen,
    on_off_pressed, r_pressed, f_pressed: bool;
let
    on_off_pressed = edge(on_off);
    r_pressed = edge(reset);
    f_pressed = edge(freeze);
    running = switch(on_off_pressed, on_off_pressed);
    frozen = switch(
        f_pressed and running, f_pressed or on_off_pressed
    );
    actual_time = counter(r_pressed and not running, running);
    time = if frozen then (0 -> pre time) else actual_time;
tel
```

Part of these notes are based on the following lectures notes:

The Lustre Language — Synchronous Programming  
by Pascal Raymond and Nicolas Halbwachs  
Verimag-CNRS