Reactive System Programming
Lustre (Part 2)

SF7 (EPU-SI) / E2 (Master STIC)

October 22, 2007

Several LUSTRE files and documents are available at `//www.i3s.unice.fr/~andre/CAdoc` (direct access to this directory). Copy td2St.zip.

1 Associativity of $->$

1.1 Boolean flows

Write an observer of the following property:

(X –> Y) –> Z = X –> (Y –> Z)

1.2 Integer flows

Question a: Why LESAR is unconclusive in this case?

Question b: Observe the behavior of X = 1 -> 2 -> 3;

Remark: remember that X -> Y -> Z $\equiv$ X ->Z.

2 -> and pre

Observe the behavior of the programs studied in the lecture.

1. X = A –> **pre**(X);
2. X = A –> **pre**(B –> **pre**(X));
3. **node** T(X:**bool**) **returns** (Y:**bool**);
4. **node** F(X:**bool**) **returns** (Y:**bool**);

Sources codes are in src/ExWithPre.lus and src/TF.lus

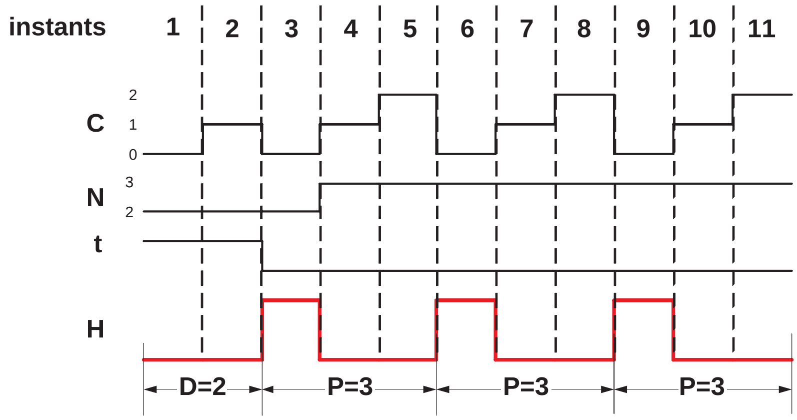

Figure 1: Timing diagrams of a Timer.

2.1 Application: Fibonacci Numbers

Write a LUSTRE program that generates Fibonacci[1] numbers, defined by the following recurrence:

$$F_n = F_{n-1} + F_{n-2}$$
$$F_1 = 1$$
$$F_2 = 1$$

3 Clock generation

This is an improved version of the clock generated in the previous Lab.

3.1 Specification

The clock emits a periodic pulse: the output is set to true for one instant every P instants (period = P). There is an initial delay (delay = D): the clock remains to false for D instants before entering the periodic behavior. This kind of clock is often called a *Timer* in RTOS. Fig. 1 shows the behavior when $D = 2$ and $P = 3$.

3.2 Use of counters

Propose a solution making use of the modulo counter studied in the previous Lab. D and P are known at run-time.

3.3 Use of Boolean functions

Now D and P are constants known at compile time. The solution shall make use of Boolean shift registers.

4 Mutual exclusion algorithm

Adapted from "A Tutorial Of LUSTRE", N. Halbwachs.

[1] Fibonacci is a nickname. His real name is Leonardo Pisano.

4.1 Specification

n processes $p_0, p_1, \cdots, p_{n-1}$ compete for an exclusive resource. The arbiter receives a Boolean array REQ, where REQ[i] is true whenever the process p_i requests the resource — and returns an array GRANT, such that GRANT[i] is true whenever the resource is granted to p_i.

The arbiter proceeds by letting a token travel around the processes. When the process, which has the token, is requesting the resource, it takes the resource and keeps the token until it releases the resource. If it does not use the resource, it passes the token at the next instant.

4.2 Behavior of a process

The behavior of a process is specified as follows:

```
-- tokenRing.lus
-- adapted from "A Tutorial of Lustre", N. Halbwachs
-- Charles André
-- November 16, 2004

node SR (S, R, I:bool) returns (Q:bool);
let
   Q = I ->
       if S and not pre(Q) then true
       else  if R then false
          else pre(Q);
tel

node FALLING_EDGE (X:bool) returns (Y:bool);
let
   Y = false -> not X and pre(X);
tel

node process (req, tok:bool) returns (grant, new_tok:bool);
let
   grant = SR(tok and req, not req, tok and req);
   new_tok = false -> pre(tok and not req) or
      FALLING_EDGE(grant);
tel

node arbiter (const n:int; REQ:bool^n) returns (GRANT:bool^n);
var TOK, NTOK:bool^n;
let
   (GRANT, NTOK) = process(REQ, TOK);
   -- intially: process 0 has the token
   TOK[0] = true -> pre (NTOK[n-1]);
   TOK[1..n-1] = false^(n-1) -> pre(NTOK[0..n-2]);
tel
```

Question a: Analyze node SR, draw the associated automaton.

Question b: Justify node FALLING_EDGE.

Question c: Justify node process. new_tok is set to true when the process passes the token to its successor on the ring.

4.3 Behavior of the arbiter

Question a: Justify node arbiter.

4.4 Proof of the exclusion

Build an observer of the exclusion property.

Hints: use the EX and OR arrays seen in the lecture.