RAPPORT DE TRAVAUX PRATIQUES LUSTRE

Hugo Blain
Bizot Louis
Dirigé par - Mme Hermine Chatoux

Contents

1 Introduction

Ce rapport présente le résultat de nos travaux pratiques en langage Lustre. Il est à noter que nous n'avons pas réussi certaines exercices. De plus, ayant des problèmes d'installation avec Luciole, nous n'avions pas accès à certains aspects graphiques du logiciel comme les chronogrammes.

2 Exercice 1: Initiation

2.1 La valeur absolue

Le but de ce premier exercice est de créer un noeud Lustre permettant de renvoyer la valeur absolue d'un flot d'entrée. Pour cela, on vérifie si la valeur de l'entrée est positive ou négative. Si elle est positive, on conserve la valeur actuelle. Sinon, on prend "-" la valeur d'entrée.

```
node ABS (r: real) returns (ab: real);
let
ab = if r < 0.0 then -r else r;
tel
```

Figure 1: Code du noeud de valeur absolue

2.2 Alterner Vrai et Faux

Ici, on va chercher à générer un signal True puis False et ainsi de suite. On produit cette alternance grâce au code suivant:

```
node OSC (b:bool) returns (s:bool);
let
    s = true -> not pre(s);
tel
```

Figure 2: Code du noeud de l'alternance True et False

2.3 Alternance modifiée

Même principe que l'exercice précédent mais cette fois ci nous prenons : True puis True puis False et False etc...
Nous avons réalisé cela grâce à un xor entre la valeur précedente et le précédent du précédent:

```
node OSC2 (b:bool) returns (s:bool);
let
    s = true -> if (pre(s) xor pre(pre(s))) then s else not pre(s)
tel
```

Figure 3: Code du noeud de l'alternance modifiée

2.4 Compteur

Le noeud de cet exercice va imiter le fonctionnement d'un compteur. Le compteur progresse lorsqu'on appuie sur le bouton *step* et se réinitialise lorsqu'on appuie sur le bouton *reset* :

```
node COMPTEUR (reset:bool) returns (n:int);
let
    n = 1 -> if (reset) then 0 else pre(n)+1;
tel
```

Figure 4: Code du noeud du compteur

2.5 Compteur plus lent

On reprend le principe de l'exercice précédent mais le compteur doit être 2 fois plus lent. Afin de diminuer la vitesse, on utilise le noeud d'un exercice précédent alternant les True et False. On augmentera donc le compteur une fois sur deux:

```
node COMPTEUR2 (reset :bool) returns (n:int);
var A: bool;
let
    A = OSC(true);
    n = 1 -> if (reset or pre(reset)) then 0 else current((pre(n) + 1) when not A);
tel
```

Figure 5: Code du noeud du compteur plus lent

2.6 Différence entre le *when* et le *if*

Nous allons ici tester le programme fourni:

```
node ChangeCompteur(reset,c :bool) returns(n :int) ;
let
    n = if c then COMPTEUR(reset) else COMPTEUR2(reset);
tel
```

Figure 6: Code du programme à tester

On observe que ce programme permet d'alterner entre les deux compteurs réalisés précédemment, l'un étant plus rapide que l'autre. Néanmoins, lors d'un appuie sur le bouton *step*, les deux compteurs s'incrémente en parallèle même si un seul est observable.

3 Exercice 2: Additionneur

Dans ce nouvel exercice, nous allons tenter de réaliser un additionneur 4 bits.

3.1 Additionneur 1 bit

Pour cela, nous allons tout d'abord réaliser un additionneur 1 bit. Nous aurons besoin d'utiliser la fonction xor et de prendre en compte des retenus. Voici le code correspondant à ce premier:

```
node ADD1(a, b, ci: bool) returns (s, co: bool);
let

s = (a xor b) xor c;
co = (a and b) or (b and c) or (a and c);

tel;
```

Figure 7: Code du noeud ADD1 , additionneur 1 bit

3.2 Additionneur 4 bits

Nous réalisons maintenant l'additionneur 4 bits souhaités. Nous nous servirons évidemment de l'additionneur 1 bit créé précédemment. Ce noeud prendra en entrée 8 bits et donnera en sortie 4 bits ainsi que la valeur "carry" correspondant à la retenue de fin. Voici le code associé:

```
65  node FIRST_ADD4(a0 ,a1, a2, a3, b0, b1, b2, b3: bool) returns (s0, s1, s2, s3,carry: bool);
66  var c0,c1, c2: bool;
67
68  let
69  s3, c2 = ADD1(a3, b3, false);
70  s2, c1 = ADD1(a2, b2, c2);
71  s1, c0 = ADD1(a1, b1, c1);
72  s0, carry = ADD1(a0, b0, c0);
73
74  tel;
```

Figure 8: Code du noeud FIRST_ADD4

‘

3.3 Additionneur 4 bits simplifié

Il est possible de simplifié le code précédent par l'ajout de tableau. Comme précisé dans le sujet de TP, Lustre va tout de même instancier un nombre de variable égal à la taille du tableau, ne permettant pas de créer un code plus optimisé. Néanmoins, une simplification d'écriture est tout de même la bienvenue:

```
node ADD4(a, b: bool^4) returns (carry, s: bool^4);
let

s[0], carry[0] = ADD1(a[0], b[0], false);
s[1..3], carry[1..3] = ADD1(a[1..3],b[1..3],carry[0..2]);

tel;
```

Figure 9: Code du noeud ADD4, simplification de FIRST_ADD4

On voit une réelle amélioration dans la qualité d'écriture du code. L'utilisation de tableau et de noeud récursif entraîne une grosse réduction du nombre de ligne du noeud, le rendant par la même occasion bien plus lisible.

4 Exercice 3: Le contrôleur de souris

Dans cet exercice nous programmons un noeud Lustre "détectant" les doubles clics. Ce noeud renvoie "2" si il reçoit deux clics en moins de 4 top, sinon il renvoie "1".

```
node COMPTEUR (reset:bool) returns (n:int);
let
    n = 1 -> if (reset) then 0 else pre(n)+1;
tel;


node SOURIS(click: bool) returns (s: int);
var premierClick: bool; nbrPas : int;
let
    nbrPas = COMPTEUR(false) -> if click then COMPTEUR(true) else pre(nbrPas)+1;
    premierClick = false -> if (click or pre(premierClick)) and nbrPas < 4 then true else false;
    s = 1 -> if click and pre(premierClick) then 2 else 1;
tel;
```

Figure 10: Code pour le noeud SOURIS

Pour compter le nombre de "top", nous utilisons le noeud "COMPTEUR" et la variable "nbrPas". Tandis que la variable "premierClick" permet de savoir si un premier click a déjà eu lieu.

5 Exercice 4: Chien de garde

Dans cette section nous allons travailler sur un noeud chien de garde. C'est un dispositif qui permet de gérer le temps de réponse d'un système et d'éventuellement déclencher une alarme si une échéance n'est pas respectée.

5.1 Chien de garde Version 1

Dans cette première version nous avons:

- Deux commandes *set* et *reset*
- Une variable booléenne *deadline* signalant l'occurrence de l'échéance
- Une sortie *alarm* signalant le dépassement de l'échéance

L'alarme se produit quand le *watchdog* est actif (a reçu la commande set) et que l'événement *deadline* se produit : alarm = is_set and deadline;

is_set est initialement dans un état vrai ou faux suivant que set est vrai ou faux. Elle reste identique à elle même tant que set ou reset ne sont pas vrais :
is_set = set -> if reset then False else if set then True else pre(is_set);

```
node WD1(set, reset, deadline: bool) returns (alarm: bool);
var is_set: bool;
let
    alarm = is_set and deadline;
    is_set = set -> if reset then false else if set then true else pre(is_set);
tel;
```

Figure 11: Code complet pour WD1

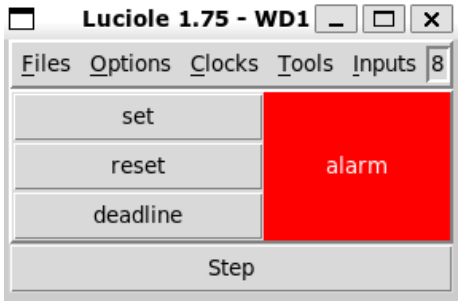

Figure 12: Simulation du noeud WD1 avec luciole

En testant, on peut bien voir que quand *is_set* et *deadline* sont True alors *alarm* passe aussi à True. Et si on click sur *reset* pour passer *is_set* à False, malgré *deadline* à True, l'alarme ne se déclenche pas.

5.2 Chien de garde Version 2

La 2ème version du *watchdog* va recevoir les mêmes commandes set et *reset* mais l'alarme sera déclenchée si l'événement *reset* ne s'est pas produit avant un certain temps *delay* (mesuré en nombre de cycles de l'horloge de base) après l'occurrence de set.

- Le temps écoulé depuis l'occurrence de *set* est mesuré par un registre *remain* dont le contenu est décrémenté à chaque cycle de l'horloge. On va réutiliser le noeud *WD1*
- *deadline* sera produit quand le registre atteint 0. On réutilisera aussi le noeud *EDGE* du détecteur de front (vu en cours) pour détecter le fait que *remain* atteint 0

```
node EDGE (b: bool) returns (edge: bool);
let
    edge = false -> (b and not pre(b));
tel;


node WD2(set, reset: bool; delay: int) returns (alarm: bool);
var remain: int; deadline: bool;
let
    alarm = WD1(set, reset, deadline);
    deadline = false -> EDGE(remain = 0);
    remain = if set then delay
             else if pre(remain) > 0 then pre(remain)-1
                  else pre(remain);
tel;
```

Figure 13: Code WD2

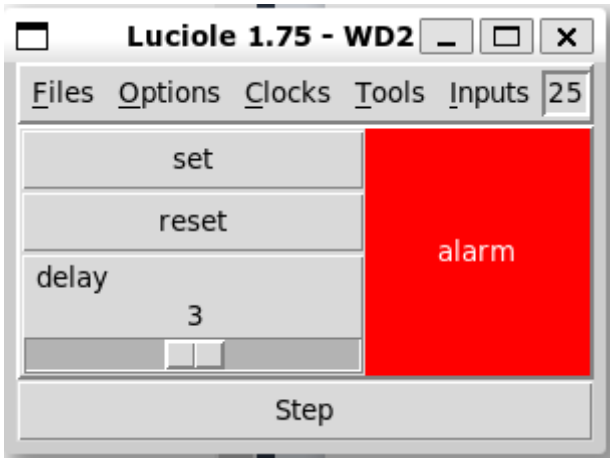

Figure 14: Simulation de WD2 avec luciole

Avec la simulation on peut choisir la valeur de *delay*. Si on clique sur *set*, on "déclenche" le compte à rebours, on décrémente de un à chaque clic sur *Step*. Arrivé à 0 l'alarme se déclenche sauf si entre temps nous cliquons sur *reset*.

5.3 Chien de garde Version 3

Dans la dernière version, on va exprimer le délai suivant une échelle de temps correspondant à l'occurrence d'un signal *time_unit*. Pour cela, on reprend *WD2* en utilisant une horloge appropriée générée par la variable booléenne *clock*.

```
node WD3 (set, reset, time_unit: bool; delay: int) returns (alarm: bool);
var clock: bool;
let
    alarm = current(WD2((set, reset, delay) when clock));
    clock = true -> (set or reset or time_unit);
tel
```

Figure 15: Code WD3

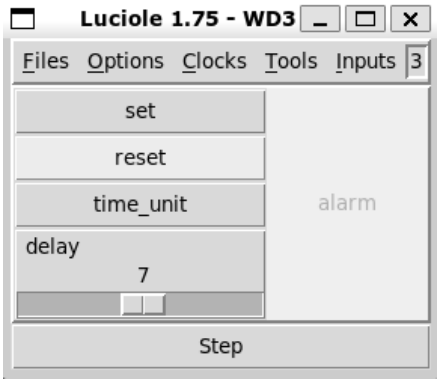

Figure 16: Simulation de WD3 avec luciole

Pour déclencher l'alarme :

- *set* et *time_unit* à True
- *set* relaché
- *alarm* se déclenche quand le délai est écoulé et si *time_unit* est toujours à True

6 Exercice 5: Délai

L'opérateur *DELAY* va prendre en entrée un entier d (une constante) et un flot booléen X et va retourner une version « retardée » du flot, c'est-à-dire un flot booléen Y tel que $yn = xn\text{-}d$ pour tout $n > d$.
On suppose que $yn = False$ pour tout n inférieur ou égale à d.

6.1 Noeud Lustre *DELAY*

Pour cela, dans un premier temps, on utilise une variable auxiliaire qu'on appelle A. C'est un tableau booléen de dimension d tel que $A[i]_n = X_{n-i}$.

```
node DELAY (const d: int; X: bool) returns (Y: bool);
var A: bool^(d+1);
let
    A[0] = X;
    A[1..d] = (false^(d)) -> pre(A[0..d-1]);
    Y = A[d];
tel ;
```

Figure 17: Code du noeud *DELAY*

Avec l'expression *A[1..d] = (False $\hat{}$ (d))*, on initialise à False toute la portion du tableau *A[1..d]*.

6.2 Noeud Lustre *MAIN_DELAY*

Pour appeler le noeud *DELAY* créer précédemment nous avons besoin d'un noeud principal que nous appellerons *MAIN_DELAY* :

```
node MAIN_DELAY (A: bool) returns (A_delayed: bool);
let
    A_delayed = DELAY(3, A);
tel;
```

Figure 18: Code du noeud *MAIN_DELAY*

Ici on donne l'exemple avec $d = 3$. La sortie sera notre entrée, décalée de 3 tics d'horloge.

7 TP2 Exercice2: Programmation synchrone

La comparaison de deux variables est un problème très courant quand on cherche à valider un programme. En général on ne cherche pas une égalité stricte à chaque instant mais plutôt à montrer que les deux "ne diffèrent pas trop".

Dans cet exercice, nous travaillerons sur le cas particulier de variables booléennes représentées au cours du temps par des signaux de type niveau :

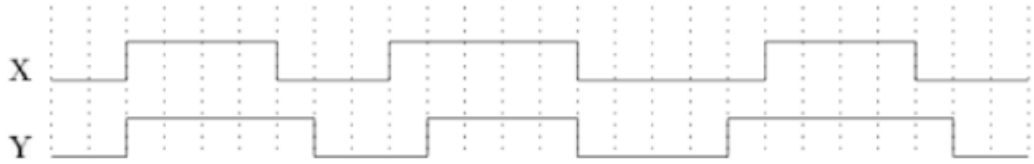

Figure 19: Signaux de type niveau pour les variables booléennes X et Y

Pour comparer deux signaux de type niveau, un bon critère consiste à dire que leurs présences doivent toujours être simultanées, sauf, éventuellement, au moment des changements de niveau, où on accepte un certain décalage. Autrement dit, les deux signaux ne doivent pas être différents (de point de vue de la présence) plus de k instants consécutifs.

Pour simplifier, on prendra ici $k = 1$. Avec $k = 1$, il suffit de vérifier si $X = Y$ ou $X = Y(t\text{-}1)$ ou $X(t\text{-}1) = Y$:

```
node CRITERE(X, Y: bool) returns(OK: bool);
let
    OK = true -> if( (X = Y) or (pre(X) = Y) or (pre(Y) = X) ) then true
                 else false;
tel;
```

Figure 20: Code