Travaux Dirigés
Langages Synchrone (LUSTRE)

Question 1

Soit le programme Lustre TEST1 suivant :

```
node TEST1 (I:int)
returns (B:bool ; (X1, X2:int) when B) ;
let
     B = true -> not pre(B) ;
     X1 = compteur(I) when B ;
     X2 = compteur(I when B) ;
tel.
```

Avec

```
node compteur (I:int) returns (N:int) ;
let
     N = I -> (pre(N) + 1) ;
tel.
```

Quelle est la réponse de TETS1 au flot constant I=0 ?

---------- Eléments de réponse…..

Il faut d'abord bien identifier les horloges des flots.

- I est déclaré dans l'interface d'entrée sans précision d'horloge. Il est donc sur l'horloge de base du nœud principal TEST1 :
  => clk(I) = true
- B est déclaré dans l'interface de sortie sans précision d'horloge. Donc B est sur l'horloge de base du nœud principal TEST1 :
  => clk(B) = true
- En revanche X1 et X2 sont déclarés comme étant sur l'horloge B, donc
  =>clk(X1) = clk(X2) = B

B est ensuite défini comme une flot qui vaut alternativement true et false (équation de B).

Dans l'équation de X1, il faut bien comprendre où se situe de « when » :

- Le sous-nœud compteur est appelé sur le flot I. Dit autrement, I est le flot d'entrée de compteur. Or, Lustre étant un langage à flot de données, cela implique que compteur(I) sera exécuté à chaque occurrence de I, et donc sur l'horloge de base. Donc
  => clk(compteur(I)) = true
- compteur(I) et B sont donc sur la même horloge (l'horloge de base), donc on peut construire l'expression compteur(I) when B, ce qui produit un flot sur l'horloge B
  => clk(compteur(I) when B) = B
- ce qui confirme que X1 est sur l'horloge B.
  - donc X est présent chaque fois que B est vrai et vaut compteur(I)
  - et absent chaque fois que B est faut.

Dans l'équation de X2, le when est appelé en entrée de compteur. Dit autrement, on applique compteur sur I when B. Comme Lustre est un langage flots de données, cela implique que

- compteur ne sera exécuté que lorsque B sera vrai,
- et il prendra en entrée le flot I sous-échantillonné sur B
  => clk(I when B) = B
  => clk(compteur(I when B)) = B

Si on représente l'exécution du programme par un chronogramme on obtient :

I	1	1	1	1	1	1
B	true	false	true	false	true	false
compteur(I)	1	2	3	4	5	6
X1 = compteur(I) when B	1		3		5	
I when B	1		1		1	
X2 = compteur(I when B)	1		2		3	

Question 2

Soit le programme Lustre défini partiellement par

```
node prog1 (B1, B2:bool ; H1:bool when B1 ;
            H2:bool when B2 ;
            X1:int when H1 ;
            X2:int when H2 ;
            Y:int when (B1 and B2))
returns (S : int when …) ;
var  Z1:int when …;
     Z2:int when …;
     Z3:int when …;
     Z4:int when … ;
let
     Z1 = current(X1) ;
     Z2 = current(X2) ;
     Z3 = current(Z1) + current(Z2) ;
     Z4 =(current(Z3) when (B1 and B2)) + Y ;
     S = current(Z4) ;
tel.
```

Il manque dans ce programme les informations d'horloge. Lorsqu'elles existent, précisez les horloges de Z1… Z4 et S. Dans le cas contraire, dite simplement que tel flot est « mal typé du point de vue des horloge ».

---------- Eléments de réponse…..

En appliquant le même raisonnement que dans l'exercice précédent, on a

- clk(B1) = clk(B2) = true
- clk(H1) = B1
- clk(H2) = B2
- clk(X1) = H1
- clk(X2) = H2
- clk(Y) = B1 and B2

Donc, sachant que current ramène un flot sur l'horloge de son horloge, les deux premières équations donnent

- clk(Z1) = clk(clk(X1)) = clk(H1) = B1
- clk(Z2) = clk(clk(X2)) = clk(H2) = B2

Ensuite

- clk(current(Z1)) = clk(clk(Z1)) = clk(B1) = true
- clk(current(Z2)) = clk(clk(Z2)) = clk(B2) = true

Donc l'équation de Z3 est bien typée du point de vue des horloges (parce qu'on y additionne deux flots qui ont la même horloge)

- clk(Z3) = clk(current(Z1)) = clk(current(Z2)) = true

En revanche, il n'est pas possible de faire current(Z3) puisque Z3 est sur la base (on ne peut pas sur-échantillonner un flot qui est déjà au rythme le plus rapide). Donc l'équation de Z4 est mal typée du point de vue des horloges.

Et du coup, on ne peut pas typer l'équation de S.

Question 3

Ecrire les équations Lustre (et uniquement les équations) qui calculent la suite de Fibonacci : $u_0=1$, $u_1=1$, $u_n=u_{n-1} + u_{n-2}$ pour tout $n>1$

---------- Eléments de réponse…..
```
node Fibonacci() returns (U : int) ;
let
        U = 1 -> pre(1 -> (U + pre(U))) ;
tel.
```

Qui est aussi équivalent à

```
node Fibonacci() returns (U : int) ;
var V : int ;
let
        U = 1 -> pre(V) ;
        V = 1 -> (U + pre(U)) ;
tel.
```

Question 4

On considère un passage à niveau composé de deux voies. La circulation sur chaque voie se fait toujours dans le même sens comme indiqué sur le figure ci-après. Sur chaque voie est disposé un capteur ENTREE (ENTREE1 pour la voie 1 et ENTREE2 pour la voie 2) en amont du passage à niveau, et un capteur SORTIE (SORTIE1 pour la voie 1 et SORTIE2 pour la voie 2) en aval du passage à niveau. Ces capteurs émettent un signal (i.e., une occurrence vraie sur le flot booléen correspondant) à chaque passage d'un train (chaque signal est émis qu'une seule fois lors du passage du train). Le système de contrôle reçoit ces signaux et émet à destination des barrières des ordres de fermetures et d'ouvertures, sous la forme de signaux DOWN et UP (DOWN pour l'ordre de fermeture, et UP pour l'ouverture). On appelle zone S1 (resp. S2) la portion de voie comprise entre ENTREE1 (resp. ENTREE2) et SORTIE1 (resp. SORTIE2).

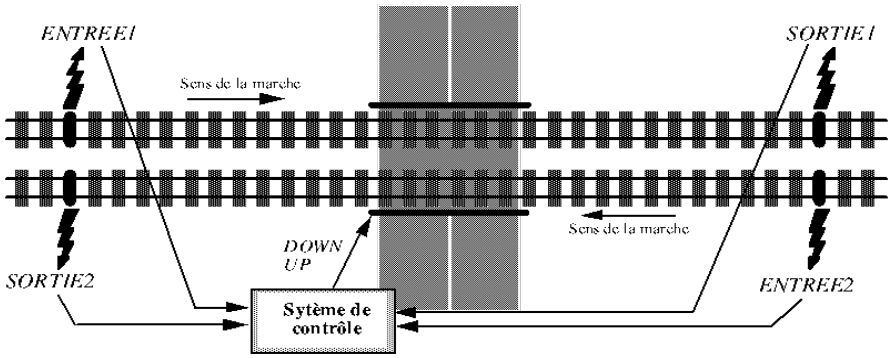

figure 1.

La spécification du système à réaliser devient alors: *les barrières doivent être fermées lorsqu'il y a au moins un train en zone S1 ou S2, et réciproquement les barrières doivent être ouvertes lorsque les zones S1 et S2 sont vides*. Ecrire un programme Lustre implémentant cette spécification.

***Attention** : il peut y avoir plusieurs trains en même temps dans chaque zone S1 et S2. Un train peut par exemple entrer dans S1 alors que le train qui le précède n'est pas encore sorti.*

---------- Eléments de réponse…..
Il suffit de compter les trains sur chaque voie.

```
node passage (ENTREE1, ENTRE2, SORTIE1, SORTIE2 : bool) returns (UP, DOWN : bool) ;
```

```
var N1, N2, var1, var2 : int ;
let
        UP = not DOWN ;
        DOWN = (N1 + N2) > 0 ;
        N1 = (0 -> pre(N1)) + var1 ;
        N2 = (0 -> pre(N2)) + var2 ;
        var1 = (if ENTREE1 then 1 else 0) + (if (SORTIE1) then -1 else 0) ;
        var2 = (if ENTREE2 then 1 else 0) + (if (SORTIE2) then -1 else 0) ;
tel.
```

Quelques commentaires :

- on rappelle que les équations sont écrites dans un ordre quelconque. Les équations sont des définitions et non des instructions informatiques séquentielles.
- UP est défini simplement comme le contraire de DOWN
- DOWN est vrai lorsque N1+N2 est strictement positif, et faux sinon
- var1 est la variation du nombre de trains sur la voie 1. Si seul ENTREE1 se produit, alors var1 vaut +1, si seul SORTIE1 se produit alors var1 vaut -1, et si les deux se produisent en même temps (un train entre pendant qu'un autre sort), alors var1 vaut 1-1 soit 0.
- Idem pour var2
- Et N1 est simplement égal à sa valeur précédente + la variation du nombre de trains sur la voie 1
- Idem pour N2.
- Noter que initialement, si ni ENTREE1 ni SORTIE1 se produisent (au premier instant), alors var1 vaut 0 au premier instant, et donc N1 vaudra aussi 0 au premier instant. En revanche, si ENTREE1 seul se produit au premier instant, alors var1 vaut 1 au premier instant, et N1 vaut donc 1 au premier instant. De même, si c'est SORTIE1 seul qui se produit au premier instant, alors N vaudra -1 au premier instant (ce qui est étrange, mais permi par le programme).

Un exemple d'exécution est donné dans le chronogramme ci-dessous :

instant	0	1	2	3	4	5	6	7	8
ENTREE1	**false**	**true**	**false**	**true**	**false**	**false**	**true**	**false**	**false**
SORTIE1	**false**	**False**	**false**	**false**	**true**	**false**	**true**	**true**	**false**
ENTREE2	**false**	**False**	**true**	**false**	**false**	**false**	**false**	**false**	**false**
SORTIE2	**false**	**False**	**false**	**true**	**false**	**false**	**false**	**false**	**false**
var1	0	1	0	1	-1	0	0	-1	0
N1	0	1	1	2	1	1	1	0	0
var2	0	0	1	-1	0	0	0	0	0
N2	0	0	1	0	0	0	0	0	0
N1+N2	0	1	2	2	1	1	1	0	0
DOWN	**false**	**true**	**true**	**true**	**true**	**true**	**true**	**false**	**false**
UP	**true**	**false**	**false**	**false**	**false**	**false**	**false**	**true**	**true**

Dans cet exemple (ce n'est qu'un exemple) : A l'instant 1, un train arrive sur la voie 1, puis un autre à l'instant 2 sur la voie 2. A l'instant 3 un nouveau train arrive sur la voie 1 tandis que celui sur la voie 2 sort. A l'instant 4, un des deux trains de la voie 1 sort. Puis à l'instant 6 un train arrive pendant qu'un autre sort (en même temps) sur la voie 1. Et enfin à l'instant 7 le dernier train sort de la voie 1.
Résultat, dans cet exemple, les barrières sont fermées de l'instant 1 à l'instant 6 inclus.