Examen
Langages Temps Réel - LUSTRE (duration 1h30)
(All documents are allowed)

Question 1

Let us consider the following LUSTRE programs :

```
node accumulator (X: int) returns (S:int);
     let
          S = X + (0 -> pre(S));
     tel

node TEST1 (X: int) returns (S1:int);
     var B1:bool;
     let
          B1 = true -> not pre(B1);
          S1 = current (accumulator (X when B1));
     tel

node TEST2 (X: int) returns (S2:int);
     var B2:bool;
     let
          B2 = false -> not pre(B2);
          S2 = current (accumulator (X) when B2);
     tel
```

Let `X = (1,1,1,1,1,1…)`.

Precise the behaviour of :

- `accumulateur(X)`
- `TEST1(X)`
- `TEST2(X)`

For that purpose you can use (and fullfil) the chronogram given in appendix A.

Question 2

Let us consider the following LUSTRE programs:

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

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

Precise the behaviour of the following program. Explain the value of the output flow n. You can use an example and a chronogram to support your explanation.

```
node double_counter (reset, c : bool) returns (n : int) ;
var n1, n2 : int;
let
     n1 = current ( counter(reset when (reset or c)) );
     n2 = current ( decounter(reset when (reset or not c)) );
     n = if c then n1 else n2;
tel;
```

Question 3 Clock calculus

Let us consider the following LUSTRE program:

```
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 …;
let
     Z1 = current(current(X1)) ;
     Z2 = current(current(X2)) ;
     Z3 = (Z1 + Z2) when (B1 and B2) ;
     S = current(Z3 + Y) ;
tel.
```

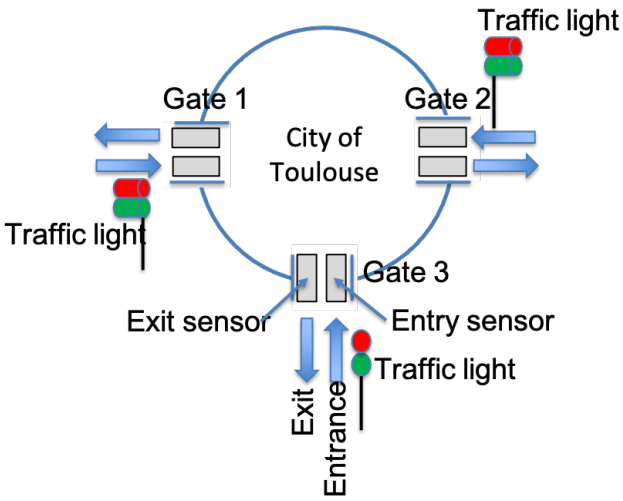
The clocks are missing in the output and local declarations.
Indicate the clock of S, Z1, Z2, and Z3.

Recall: `X : int when true` means that the clock of X is the basic clock.

Question 4 Entrance controller

It has been suggested that the city of Toulouse could put a limit on the number of motorists that can enter into the city at any one time. One proposal is to establish traffic lights at each gate of the city and, when the city is full, to turn the traffic lights to red for incoming traffic. To indicate to the motorists that the city is full, the red light is set to flashing. Otherwise, if the city is not full, the traffic lights are set to green. In order to achieve this goal, pressure sensors are placed in the road at the gates' entry and exit points. Every time a car enters the city, a signal "entry" is sent to a Gate_Controller task; similarly every time a car exits the city, a signal "exit" is sent to the controller.

To simplify the problem, we suppose that the city of Toulouse has only three gates, as shown below:



We note `max_cars` the maximun number of cars in the city. As soon this threshold is reached, the traffic lights have to be turned to red (in flashing mode).
We note then `min_cars` the number of cars in the city to reached before allowing again new motorists to enter the city. For instance, if `max_cars - min_cars` is equal to 100, that means that when the traffic lights have been turned to red (that is, when there are `max_cars` cars in the city), then we have to wait that 100 cars exit the city before turning the traffic lights to green again.

Specify the Gate_Controller tasks as a LUSTRE node:

```
const max_cars : int;
const min_cars : int = max_cars - 100;
node Gate_controller  (entry1, exit1: bool ;
                       entry2, exit2 : bool ;
                       entre3, exit3 : bool)
returns (red1, green1 : bool;
         red2, green2 : bool;
         red3, green3 : bool)
var …
let
…
tel.
```

Remark: setting the red lights to flashing means to set each `red` flow to true and false alternatively.

Correction :

Exercice 1

X	1	1	1	1	1	1
accumulator(X)	1	2	3	4	5	6
B1	True	False	True	False	True	False
accumulator(X when B1)	1		2		3	
current(accumulator(X when B1))	1	1	2	2	3	3
TEST1(X)	1	1	2	2	3	3
B2	False	True	False	True	False	True
accumulator(X) when B2		2		4		6
current(accumulator(X) when B2)	nil	2	2	4	4	6
TEST2(X)	nil	2	2	4	4	6

Exercice 2

Counter un nœud qui incrémente sa sortie de 1 à chaque fois qu'on l'appelle, sauf quand reset est vrai. Dans ce cas il remet sa sortie à 0.
Decounter fait l'inverse, il décrémente sa sortie de 1 à chaque fois qu'on l'appelle, sauf quand reset est vrai, dans ce cas il remet sa sortie à 0.

Dans l'équation de n1, on a l'expression `counter(reset when (reset or c))`.
Dans cette expression, le nœud counter est appelé avec la valeur de reset sous-échantillonnée sur le flot booléen reset ot c. Ce qui implique que dans cette expression, l'instance du nœud counter est appelée uniquement lorsque reset est vrai ou que c est vrai.
En conséquence, le chronogramme de n1 sera du type

reset	False	True	False	False	False	Fasle
c	False	False	True	False	True	False
counter(reset when (reset or c))		0	1		2	

La valeur retournée par « counter(reset when (reset or c)) » est incrémentée à chaque fois que c est vrai et reset est faux. Elle est remise à zéro à chaque fois que reset est vrai. Et elle est absente (le nœud counter n'est pas appelé) dans les autres cas.
Et donc, le flot n1 (qui vaut « `current (counter(reset when (reset or c)))` ») ramène la sortie de counter sur l'horloge de base, ce qui donne

reset	False	True	False	False	False	Fasle
c	False	False	True	False	True	False
counter(reset when (reset or c))		0	1		2	
n1	nil	0	1	1	2	2

En faisant le même raisonnement, on obtient le chronogramme suivant pour n2

reset	False	True	False	False	False	Fasle
c	False	False	True	False	True	False
decounter(reset when (reset or not c))	0	0		-1		-2
n2	0	0	0	-1	-1	-2

Et au final, comme n = if c then n1 else n2, on obtient

c	False	False	True	False	True	False
n1	nil	0	1	1	2	2

n2	0	0	0	-1	-1	-2
n = if c then n1 else n2	0	0	1	-1	2	-2

Si on poursuivait le chronogramme, on verrait l'entrelacement des deux compteurs, l'un montant, l'autre descendant, sans jamais se mélanger. Lorsque c repasse à vrai, la valeur de n repart de la dernière valeur de n1 et l'incrémente, et lorsque c est faux, la valeur de n repart de la dernière valeur de n2 et la décrémente. On a réalisé un double compteur qui compte à la fois le nombre d'occurrence vraie sur c et le nombre d'occurrence faux sur c.

- A l'instant t, si n est positif, sa valeur est égale au nombre d'occurrences vraies sur c depuis le dernier reset
- Et si n est négatif, alors sa valeur est égale au nombre d'occurrences faux sur c depuis le dernier reset.

Exercice 3

Il faut voir que

- B1, et B2 sont sur l'horloge de base. Ils ont une valeur à chaque instant
- H1 a comme horloge B1. Il n'a une valeur que lorsque B1 est vrai
- H2 a comme horloge B2. Il n'a une valeur que lorsque B2 est vrai
- X1 a comme horloge H1. Il n'a une valeur que lorsque H1 est vrai
- X2 a comme horloge H2. Il n'a une valeur que lorsque H2 est vrai
- Y a comme horloge (B1 and B2). Il n'a une valeur que lorsque B1 et B2 sont vrais simultanément.

Une fois qu'on a compris les horloges des entrées, on peut commencer à inférer les horloges des flots internes et de sortie.

Note : on ne peut pas commencer à S puisqu'il faut d'abord connaitre Z3. Et on ne peut pas commencer par Z3 puisqu'il faut d'abord connaitre Z1 et Z2.

Concernant Z1 : l'expression current(X1) sur-échantillonne le flot X et le ramenant sur l'horloge de l'horloge de X. L'horloge de X est H1. L'horloge de H1 est B1. Donc current(X1) a comme horloge B1. Donc current(current(X1)) a comme horloge l'horloge de B1, c'est-à-dire l'horloge de base.

Idem pour Z2. Z2 a comme horloge l'horloge de base.

Concernant Z3, on vérifie d'abord qu'on a le droit de faire Z1+Z2, et qu'on a le droit de faire B1 and B2. Ces opérations sont effectivement correctes puisque Z1 et Z2 sont sur la même horloge, ainsi que B1 et B2.
Ensuite, on vérifie que Z1+Z2 et B1 and B2 sont sur la même horloge. Ce qui est le cas. C'est l'horloge de base.
Donc Z3 a comme horloge B1 and B2.

Enfin, concernant S, on vérifie que Y+Z3 est une opération correcte du point de vue des horloge (et des types). C'est le cas puisque Y et Z3 ont la même horloge, c'est-à-dire B1 and B2 (et sont des int). En conséquence, current(Y+Z3) a comme horloge l'horloge de B1 and B2, c'est à l'horloge de base.

A final, les réponses sont

```
     S : int when true) ;
     Z1:int when true;
     Z2:int when true;
     Z3:int when (B1 and B2);
```

Exercice 4

```
counter_at_gate (entry, exit : bool) returns (n :int)
let
     n = if entry and exit then 0 -> pre(n)
          else if not entry and not exit then 0 -> pre(n)
          else if entry then (0 -> pre(n)) + 1
          else if exit then (0 -> pre(n)) -1 ;
tel

const max_cars : int;
const min_cars : int = max_cars - 100;
node Gate_controller  (entry1, exit1: bool ;
                       entry2, exit2 : bool ;
                       entry3, exit3 : bool)
returns (red1, green1 : bool;
         red2, green2 : bool;
         red3, green3 : bool)
var  green, red : bool;
     n, n1, n2, n3 : int
let
     red = if green then false
               else true -> not pre(red) ;
     green = if (n > max_cars) then false
               else if (n < min_cars) then true
               else true -> pre(green);

     n = n1 + n2 + n3;

     n1 = counter_at_gate (entry1, exit1);
     n2 = counter_at_gate (entry2, exit2);
     n3 = counter_at_gate (entry3, exit3);

     (green1, red1) = (green, red);
     (green2, red2) = (green, red);
     (green3, red3) = (green, red);

tel.
```

Explications :

- Ici on a décidé d'abord de faire comme si il n'y avait qu'un seul feu global, dont les valeurs sont « green » et « red ». On calcule ces valeurs à partir du nombre de voitures total en ville. On appelle « n » ce nombre. Les deux premières équations donnent les définitions de « red » et « green ».
- Ensuite, on donne la définition de « n », à partir des nombres « n1 », « n2 », « n3 » comptés à chaque porte.

- Ensuite on calcule n1, n2, n3 en utilisant un nœud commun (« counter_at_gate »).
- Enfin, on donne les définitions de « green1 », « red1 »… et constatant que tous les feux doivent avoir le même état, c'est-à-dire l'état « green » et « red » du feu global calculé au début.