Réalisation d'une machine algorithmique
Polytech' Grenoble - UFR-IMAG : RICM1

1 Introduction

Nous allons dans ce TP nous intéresser à la réalisation d'un circuit permettant l'exécution d'un algorithme. Nous nous arrêterons à l'étape de simulation d'une description en Langage Lustre du circuit. Nous ne réaliserons pas pour de vrai le circuit, mais il serait aisé de le faire à partir de cette description à l'aide d'outils de CAO appropriés. Bien sûr cette description doit être réalisée à l'aide des opérateurs logiques et de variables booléennes (portes logiques de base) ; les $ifthenelse$ et variables entières sont interdits.

2 L'algorithme

L'algorithme étudié génère une sinusoïde en utilisant un filtre à réponse impulsionnelle infinie d'ordre 2. Si ce filtre a des paramètres qui le rendent instable, une simple impulsion de départ provoque une oscillation infinie.

L'équation d'un tel filtre est la suivante : $Y[N] = X[N] + a * Y[N-1] + b * Y[N-2]$ avec $Y[-1] = Y[-2] = 0$

Ce filtre est instable si ses pôles sont sur le cercle unité, c'est à dire si $b = -1$ et $0 < a < 2$ (Si b est plus proche de 0, on obtient une sinusoïde amortie).

La valeur de a règle la fréquence de la sinusoïde.

Dans notre cas, $X[N] = 0$, sauf $X[0] = X0$ qui représente l'impulsion de départ (ce qui donne l'ordre de grandeur de l'amplitude la la sinusoïde).

Voici donc l'algorithme à étudier :

Les données :
param1, param2 : des réels ;
X0, arret : des entiers

lexique local :
    i, var0, var1, var2 : des entiers

Algorithme :
    Lire(X0) ;Lire(arret) ;Lire (param1) ;Lire(param2) ;
    i←0 ;
    var0←0 ;
    var1←X0 ;
    tant que i < arret
        debut
            var2 ← param1 * var1 + param2 * var0 ;
            Ecrire(var2) ;
            var0 ← var1 ;
            var1 ← var2 ;
            i ← i+1 ;
        fin

Voici une traduction en langage C :

```
#include <stdio.h>
#include <stdlib.h>
#include<math.h>

main(int argc, char * argv[])
{
    float x0, param1, param2, arret;
    int var0, var1, var2, i;

    if (argc != 5) exit(0);
    sscanf(argv[1], "%f", &x0);
    sscanf(argv[2], "%f", &param1);
    sscanf(argv[3], "%f", &param2);
    sscanf(argv[4], "%f", &arret);

    var0 = 0 ; var1 = x0 ;

    for (i=0 ; i < arret ;  i++)
     {
        var2=  (int)floorf (param1 * var1 + param2 * var0);
/*floorf renvoie la partie entiere d'un reel*/
        printf("%d\t%d\n", i, var2);
        var0= var1 ; var1= var2 ;
      }
}
```

Nous nous intéressons ici à une réalisation de cet algorithme avec les valeurs particulières : $param1 < 2$ et $param2 = -1$.

Faites tourner le programme C avec de telles valeurs et visualisez à l'aide d'un outil approprié, la courbe dont les coordonnées sont délivrées par le programme. (Compilation : `gcc sin.c -o sin`, sous Linux rajouter l'option `-lm`)

Voici le résultat (figure 1) que vous devriez obtenir avec $param1 = 1,99; param2 = -1; X0 = 300; arret = 200$; par exemple.

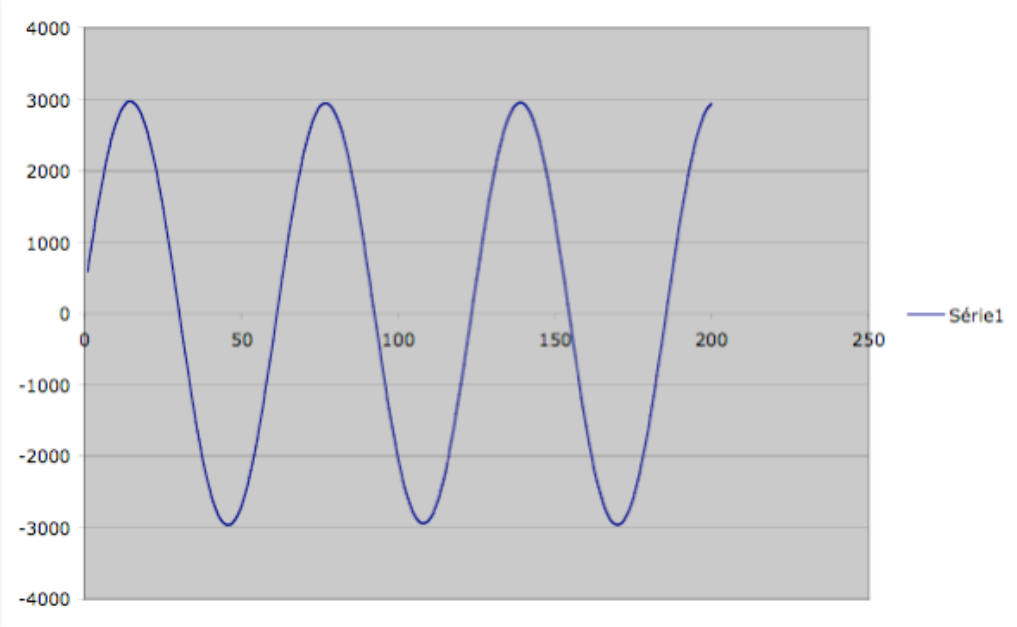

FIG. 1 – Résultat de l'algorithme

3 L'architecture du circuit

Nous allons réaliser ce circuit à l'aide d'une architecture à Partie contrôle/Partie opérative.

3.1 L'interface d'entrée/sortie

La figure 2 donne l'ensemble des entrées/sorties du circuit à réaliser. L'utilisateur du circuit fournit via un bus d'entrée les valeurs des données `param1, X0, Arret`. Pour cela il doit activer l'entrée de chargement correspondante (`chP1, chX0, chAR`) au moment où la valeur est présente sur le Bus d'entrée.

Le circuit délivre sur un bus de sortie les valeurs successives de `var2`. De plus le circuit possède une entrée permettant de cadencer ses calculs `clock` et d'être initialisé `reset`

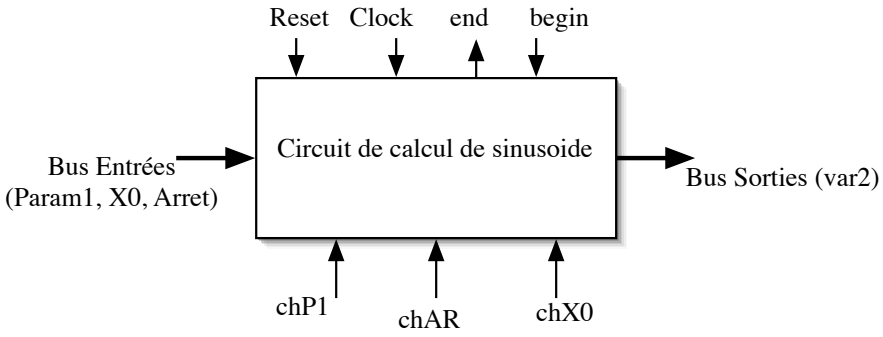

FIG. 2 – Les entrées/sorties du circuit

3.1.1 Fin et début de l'algorithme

Le circuit possède une entrée `begin` particulière permettant de "lancer" l'algorithme à l'aide de la valeur 1 une fois que les données ont été chargées via le bus d'entrée. Pour signifier la fin de l'algorithme le circuit délivre une sortie `end` égal à 1.

3.1.2 Le problème des réels

Nous supposons que `param2` $= -1$. Les variables `var2, var1, var0` sont des entiers, `param1` est un réel. Lors du calcul de $param1 * var1 - var0$, c'est la partie entière du résultat qui est affecté à `var2`.

Pour simplifier l'utilisation de ce réel nous allons le représenter sous la forme $m * 2^{-k}$ avec `k` fixé.

L'utilisateur fournira la mantisse `m` en base 2 de cette approximation de `param1`.

Exemple : On veut fournir la valeur $0,75$ (en base 10) pour `param1`, si on fixe k à 4 sa représentation sera 1100_2. En effet $0,75_{10} = 12_{10}.2^{-4}$.

Pour réaliser le calcul $param1 * var1$ il suffit alors de multiplier `var1` par la mantisse de `param1` puis de décaler de `k` crans à droite le résultat.

3.2 La partie opérative

Transformons l'algorithme afin que seul des opérations à deux opérandes apparaissent.

Les données :
param1 : un réel ;
X0, arret : des entiers
lexique local :

```
    i, var0, var1, var2 : des entiers

algorithme :

    i←0 ; var0←0 ; var1←X0 ;
    tant que i ≠ arret
          debut
                var2 ← param1 * var1 ;
                var2← var2 - var0
                var0 ← var1 ;
                var1 ← var2 ;
                i ← i+1 ;
    fin
```

La figure 3 donne une partie opérative permettant de réaliser tous les calculs et affectations de cet algorithme. Elle est composée des registres `P1, X0, AR, V0, V1, V2, I` contenant les valeurs des variables `param1, X0, arret, var0, var1, var2, i` de l'algorithme. Chacun de ces registres est sensible au front montant de `CK` (l'horloge), il possède une commande de chargement `chNom`. Les registres `I` et `V0` possèdent de plus une commande d'initialisation à 0 (`resetI` et `resetV0`). L'UAL doit permettre d'effectuer la soustraction, l'incrémentation, la multiplication et le passage du premier opérande sans calcul. Il calcul le flag `Z` permettant de délivrer la condition $i = arret$.

Le multiplexeur 2 vers 1 permet de charger au départ les données depuis le bus extérieur.

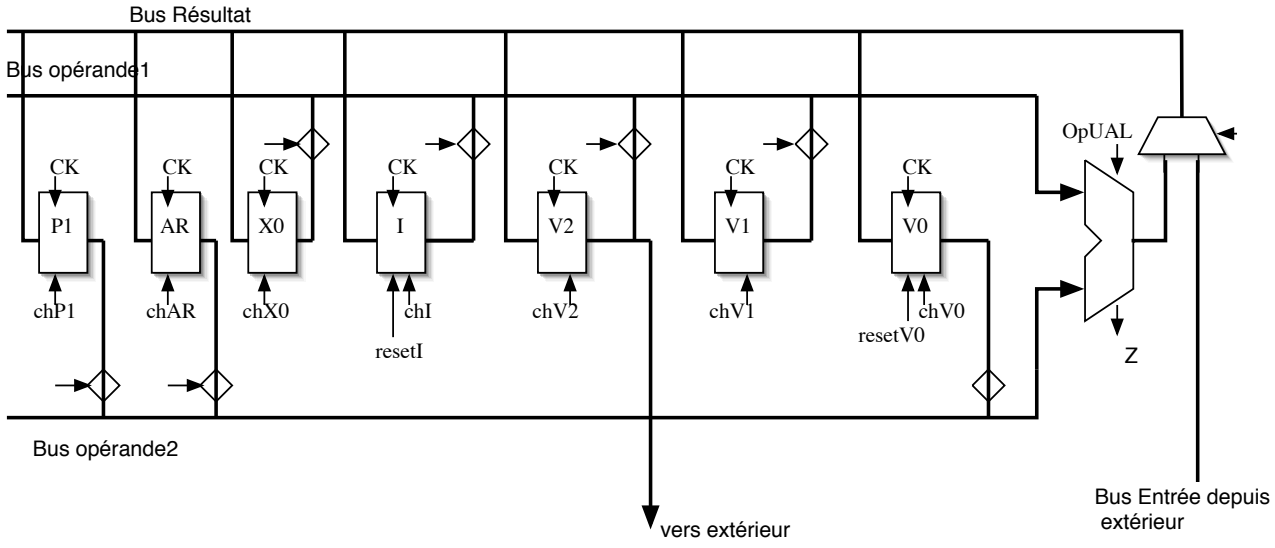

FIG. 3 – La partie opérative du circuit

3.2.1 A faire

1. Vérifiez que cette PO permet bien tous les calculs et affectations apparaissant dans l'algorithme.
2. Fixez le codage de `opUAL` et décrivez en Lustre cet UAL. On repartira de la description de l'UAL précédement faite en TP.
   Le multiplieur n bits est donné (voir dans le placard le fichier `multnbits.lus`).
   Fixer le nombre de bit des variables et la constante `k` intervenant dans la représentation de `param1`. Ne pas oubliez de faire le décalage nécessaire en sortie du multiplieur.
3. Décrivez en lustre le node "registre à n bits" à partir de la bascule donnée (voir placard). Dans les différentes bascules utilisées pour fabriquer les registres, l'entrée `CK` est connectée au signal d'horloge `Clock` du circuit.
   Le chargement ou le non chargement du registre est commandé par l'entrée `CHAR/Enable` des bascules. Par exemple, le chargement effectif du registre `V2` a lieu si l'entrée `chV2` (connectée à l'entrée `CHAR` des bascules) est à 1.
4. Les bus `opérande 1` et `opérande 2` seront réalisés à l'aide de multiplexeurs 4 vers 1 n bits. Fixer les codes des signaux de sélections des deux multiplexeurs utilisés pour les deux bus.
   On pourra utiliser la figure 4 décrivant la PO de façon plus détaillé.
5. Décrivez en Lustre la PO en utilisant les boîtes précédemment créées.

3.3 La partie contrôle

Le graphe de l'automate de contrôle est donné figure 5 ; il tient compte à la fois des communications avec l'extérieur et de l'exécution proprement dite de l'algorithme.

Les bascules de la partie contrôle prennent une valeur à chaque coup de l'horloge `Clock`. Elles ont donc leur signal `CHAR` à 1.

La réalisation de la partie contrôle doit faire apparaître 3 parties : les bascules de mémorisation de l'état, le circuit combinatoire de calcul de l'état suivant, le circuit combinatoire de calcul des sorties. La figure 6 donne une telle organisation.

3.3.1 A faire

1. Donner dans un tableau la valeur des sorties de l'automate (commandes de la partie opérative) déduite des actions apparaissant sur le graphe de l'automate de la figure 6.
2. Après avoir fixé d'un codage 1 parmi N des états de l'automate, donnez le dessin du circuit de la partie contrôle en utilisant des bascules et des portes AND, OR et inverseurs. Comment se fait l'initialisation de ce circuit ?
3. Décrivez en Lustre une réalisation de cet automate .

3.4 Vue d'ensemble du circuit

La figure 6 donne l'architecture générale partie contrôle/partie opérative du circuit.

4 Conseil de réalisation

1. Décider des codes utilisés dans l'UAL et sur les multiplexeurs avant de se lancer dans le lustre. Faire apparaître ces choix sur le dessin de la PO donné dans la figure 4
2. Pour la réalisation de l'UAL, il faut choisir une valeur pour `k` par rapport au nombre de bits `n` des registres. On pourra par exemple prendre `k`= `n`/2.
3. Pour débuguer ne pas hésiter à ajouter en sortie du circuit d'autres variables. En particulier il est intéressant de connaître l'état courant de la partie contrôle.
4. attention au dépassement de capacité de votre circuit ; choisir `X0` en fonction du nombre de bits `n` fixé pour votre circuit.
5. Vérifiez que votre circuit fonctionne en comparant sa sortie à celle du programme C donné. On pourra utiliser le node `entierrel` (donné dans `entier.lus`) qui affiche l'entier relatif égal à la valeur du code en complément à 2 passé en paramètre. On pourra visualiser le registre `V1` au lieu de `V2`, `V2` changeant de valeur 2 fois dans la boucle de l'algorithme.

5 Pour ceux qui veulent aller plus loin

Rajoutez à votre description la réalisation d'une sortie signalant un dépassement des capacités de votre circuit.

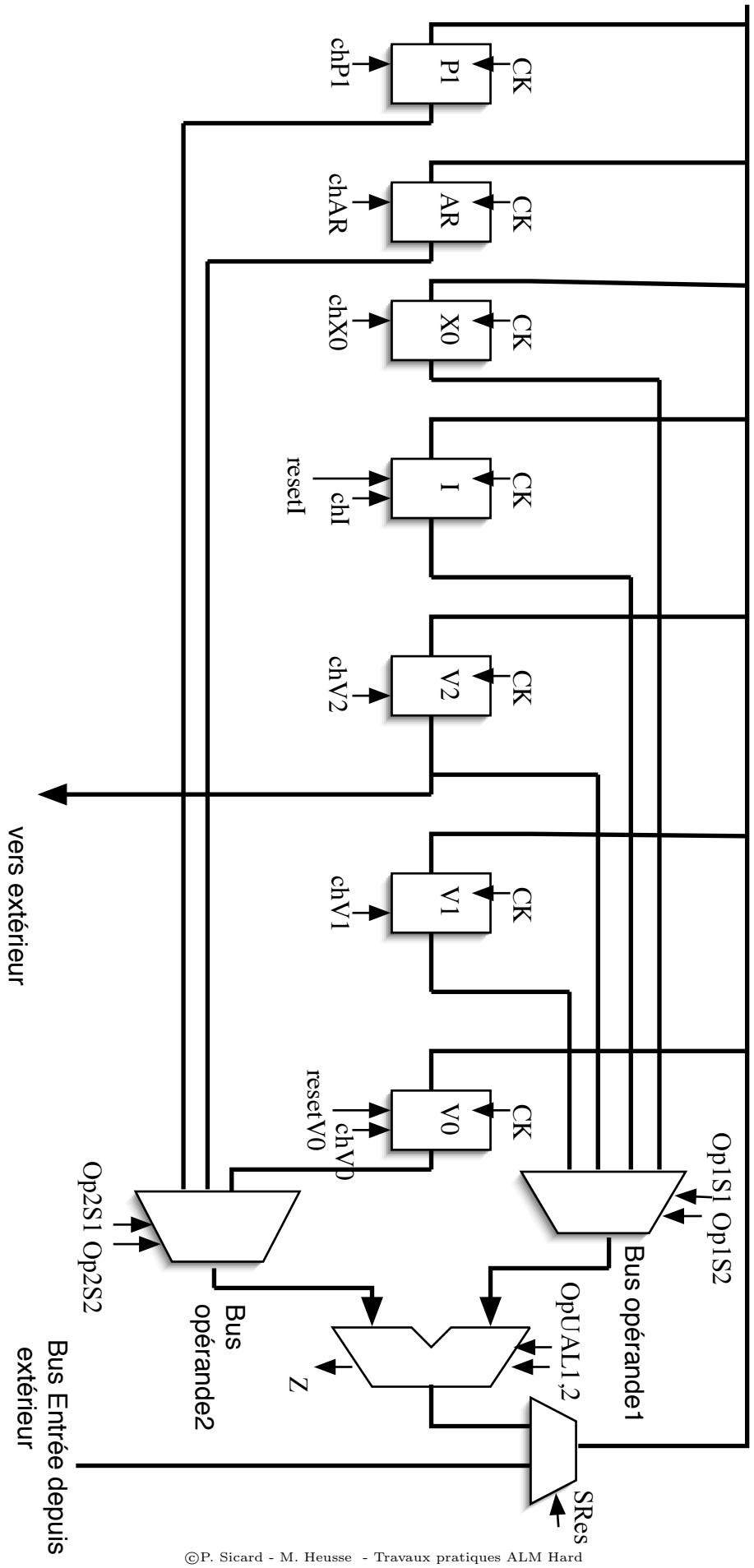

FIG. 4 – La partie opérative du circuit

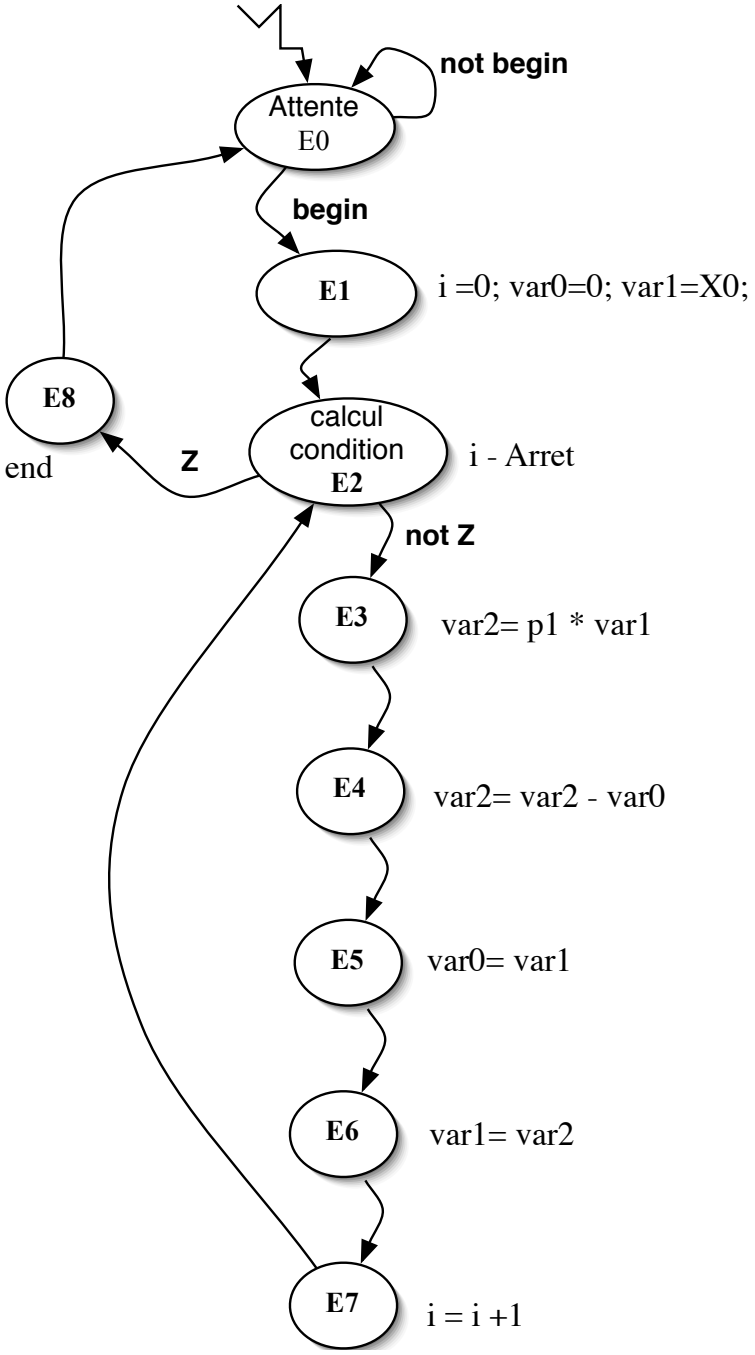

FIG. 5 – La partie contrôle du circuit

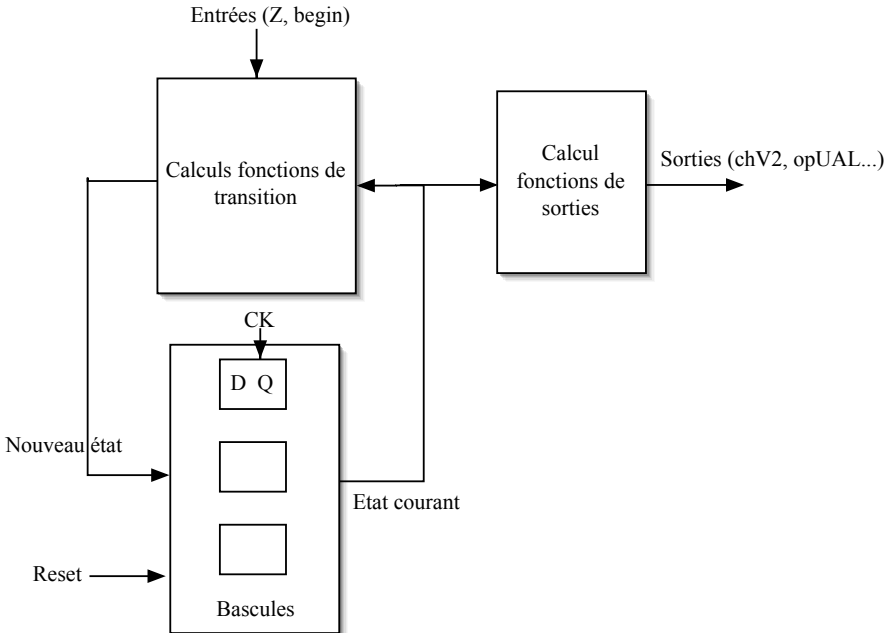

FIG. 6 – Architecture de la partie contrôle

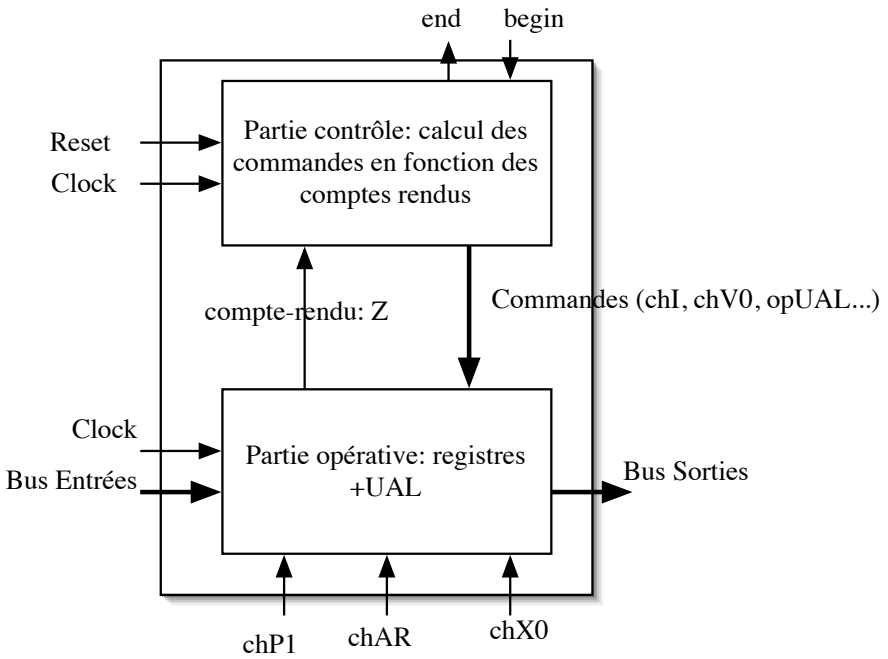

FIG. 7 – Architecture PC/PO du circuit