UTILISATION DU LANGAGE SIGNAL POUR LA SPECIFICATION ET LA SIMULATION D'ALGORITHMES DE TRAITEMENT DU SIGNAL

François DECHELLE* - Yves SOREL**

* TRT, 5, Avenue Réaumur, 92350 Le Plessis Robinson, France

** INRIA, Domaine de Voluceau, BP 105, Rocquencourt, 78153 Le Chesnay Cedex, France

SIGNAL est un langage flots de données, temps-réel synchrone développé à l'IRISA.

Nous avons utilisé SIGNAL pour la spécification et la simulation d'algorithmes de traitement du signal.

Nous proposons une méthode facilitant l'écriture d'un programme SIGNAL.
A l'aide d'exemples (égalisation double échantillonnage, filtre de Kalman parallèle), nous mettons en évidence les particularités du langage et ses avantages par rapport aux langages classiques.

SIGNAL is a real-time synchronous, data-flow programming language developped at IRISA.

We used SIGNAL for specification and simulation of signal processing algorithms.

We propose a method in order to improve the writing of SIGNAL programs. Using examples (double sampling equalizer, parallel Kalman filter), we examine the characteristics of the language and its advantages compared to classical languages.

1 - PRESENTATION DU LANGAGE SIGNAL

La présentation qui suit donne rapidement les caractéristiques essentielles du langage, pour une définition complète on se reportera à [1], [2] et [3].

1.1 SIGNAL est un langage flots de données :

Cette caractéristique permet de décrire facilement le parallélisme.
L'ensemble des calculs à effectuer est représenté par un graphe orienté, dans lequel les arcs sont des chemins de données et les noeuds des opérations.
Lors de l'exécution un noeud de calcul est activé lorsque toutes ses entrées sont prêtes. (règle flots de données).
La notion de variables est remplacée par celle de flot : suite de valeurs.

Un programme SIGNAL est la donnée d'un réseau statique orienté de processus ou "boite noire", connue de l'extérieur par ses ports d'entrée et de sortie.
Un processus est
- soit un processus élémentaire, ou générateur
- soit un réseau statique orienté, construit à l'aide d'opérateurs de connexion appliqués aux processus existants.

Ces opérateurs connectent un port de sortie aux ports d'entrée de même nom .
Ils ont un sens purement topologique, l'ordre dans lequel s'exécuteront les processus est déterminé par la règle flots de données .
Liste des opérateurs de connexion :
PROC1 ; PROC2 sorties de PROC1 connectées aux entrées de PROC2
PROC1 & PROC2 connexion des entrées communes à PROC1 et PROC2
PROC @ X rebouclage, connexion de la sortie de nom X sur l'entrée de nom X
PROC1 | PROC2 sorties de PROC1 (resp. PROC2) connectées aux entrées de PROC2 (resp. PROC1) et connexion des entrées communes
Un opérateur permet le renommage des signaux :
X : XP le signal X est renommé en XP

1.2 SIGNAL est un langage temps-réel synchrone :

Cette caractéristique permet de décrire facilement le comportement temporel d'un algorithme.
Par définition les temps de réponse d'un système temps réel sont bornés.
Dans un langage temps-réel synchrone, on fait l'hypothèse que toutes les actions ont une durée d'exécution nulle.
Un langage synchrone est déterministe, plusieurs exécutions avec la même séquence d'entrées produisent la même séquence de sorties .Ceci simplifie la mise au point des programmes.
A chaque flot est associée une horloge, qui spécifie les instants auxquels les valeurs successives du flot sont disponibles .
Un signal est un couple (flot, horloge).
La notion de temps pour un système est définie avec SIGNAL par des relations entre les horloges des différents signaux du systeme, sans faire référence à une horloge universelle.

Ces relations sont déduites de la sémantique des générateurs, classés en générateurs arithmétiques (ou fonctions) et générateurs temporels.

Les fonctions réalisent les opérations arithmétiques, et imposent que toutes les entrées et toutes les sorties aient la même horloge :
X := A + B + 1 (entrées : A, B; sortie : X)

Ces générateurs temporels ne modifient pas les horloges :
VX := X window N fenêtre glissante sur X de longueur N
event X horloge du signal X

Z := X $ 2 retard de deux échantillons
C := # event X compte les occurences du signal X

Ces générateurs temporels modifient les horloges :
X := A when event B signal A sous-échantillonné par l'horloge de B
X := A cell event B signal A mémorisé à l'horloge de B
X := A default B mélange des signaux A et B, priorité pour A si les deux sont présents.

2 - METHODE DE PROGRAMMATION

La représentation initiale d'un problème de traitement du signal consiste en un schéma bloc, une description textuelle et un ensemble d'équations récurrentes.

La méthode comporte quatre points. Il faut successivement:
1 - déterminer les caractéristiques temporelles de l'algorithme : on associera à chaque signal une horloge et on comparera les différentes horloges entre elles.
2 - déterminer les caractéristiques arithmétiques de l'algorithme : on utilisera pour cela les équations récurrentes
3 - associer calculs et opérations temporelles à des processus
4 - construire le programme en connectant les processus déjà définis.

La méthode décrite ci-dessus est détaillée à l'aide de l'exemple de l'égaliseur double échantillonnage [5].

Point de départ : description de l'égaliseur double échantillonnage :

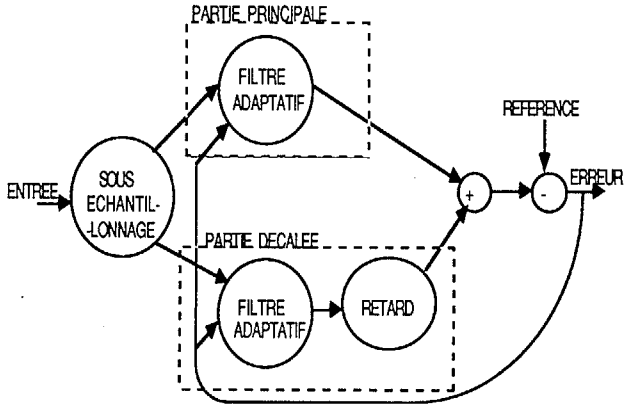


Figure 1 : schéma bloc de l'égaliseur double échantillonnage

Chacune des deux parties travaille à la même fréquence fe. L'entrée X est échantillonnée à deux fois la fréquence fe et on alimente alternativement la partie principale (échantillons pairs) et la partie décalée (échantillons impairs). La longueur M du filtre décalé est inférieure à celle N du filtre principal (avec ce filtre on ne cherche à identifier que la partie centrale de la réponse impulsionnelle), ceci conduit à introduire un retard de longueur (N-M)/2.

Pour chacun des deux filtres adaptatifs on donne les équations récurrentes :

fenêtre glissante sur le signal d'entrée X :

$$V\_X(k) = [\ X(k-N),\ X(k-N+1),\ \ldots,\ X(k)\ ] \quad (1)$$

calcul du filtre

$$Y(k) = \sum Hi(k-1) * V\_Xi(k) \quad (2)$$

calcul de l'élément i du vecteur coefficient H par la méthode du gradient avec un pas constant MU

$$Hi(k) = Hi(k-1) + MU * E(k) * V\_Xi(k) \quad (3)$$

calcul de l'erreur

$$E(k) = YREF(k) - Y(k) \quad (4)$$

où k est l'indice temporel et i l'indice vectoriel.

1 - Caractéristiques temporelles de l'égaliseur double échantillonnage :

Pour les signaux importants on trace un diagramme mettant en évidence les relations temporelles entre leurs horloges (sous échantillonnages). On définit ainsi l'entrelacement global des signaux.

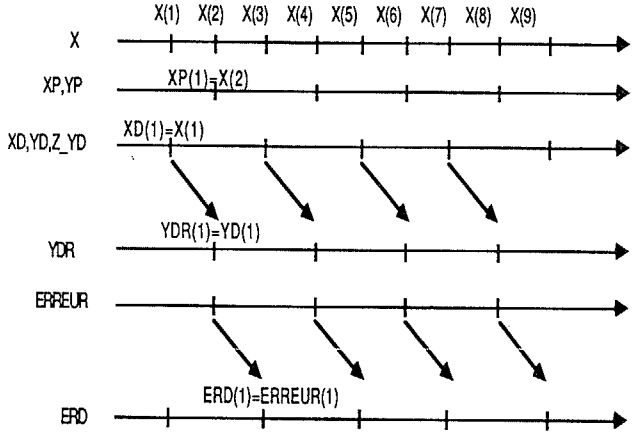


Figure 2 : différentes horloges de l'égaliseur double échantillonnage

La comparaison des différentes horloges permet de construire les processus modifiants les horloges en utilisant les générateurs temporels (when, default, cell). On remarque sur la figure 2 que les deux sorties (partie principale YP, partie décalée Z_YD) doivent être ramenées à la même horloge pour être additionnées. On choisit de calculer l'erreur à l'horloge de la partie principale.
Ceci impose de resynchroniser le signal Z_D (sortie partie décalée) sur le signal YP (sortie partie principale) pour les additionner, et le signal ERREUR sur le signal YD pour adapter les coefficients de la partie décalée.

Dans les équations récurrentes (1), (2), (3) et (4), on distingue les indices temporels (k) des indices vectoriels (i). Ceci permet de construire les processus utilisant les générateurs arithmétiques, les générateurs temporels fenêtre glissante et retard.
L'équation récurrente (3) (qui est du type u(k)=f(v(k-1),u(k-1), ...) se traduit en SIGNAL par le processus ADAPTE où on utilise le générateur temporel $ (retard). A partir du signal H, on produit un signal Z_H avec Z_H(k) = H(k-1).

3 - Liste des processus nécessaires pour décrire l'algorithme :
SEPARE entrée : X sorties : XP, XD ; décompose le signal d'entrée X en deux signaux XP et XD
FILTADAPT entrées : X, ERR sortie : Y, Z_H ; effectue le filtrage adaptatif, il est instancié deux fois avec les renommages (partie principale) X:XP, Y:YP, Z_H:Z_HP, ERR:ERP (partie décalée) X:XD, Y:YD, Z_H:Z_HD, ERR:ERD
RESYNC entrée : IN, SYNC sortie : OUT ; modifie l'horloge d'un signal afin de le synchroniser sur un autre signal, il est instancié deux fois avec les renommages sortie partie décalée IN:ZYD, SYNC:YP, OUT:YDR, erreur IN:ERREUR, SYNC:YP, OUT:ERD

4 - Construction du programme SIGNAL :

On construit le programme SIGNAL en connectant ces processus, on obtient le schéma bloc SIGNAL suivant:

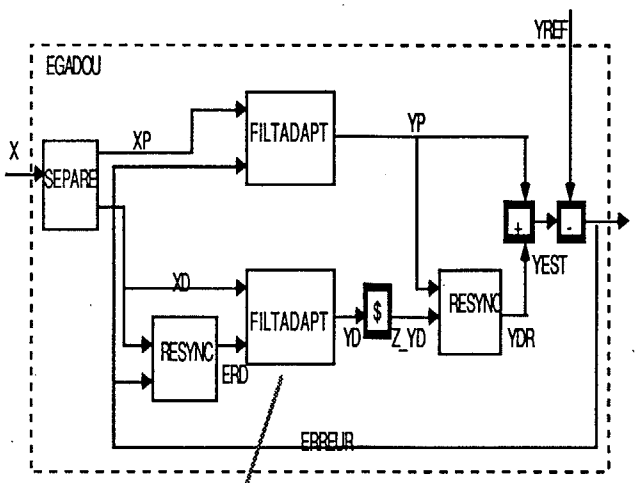


Figure 3a

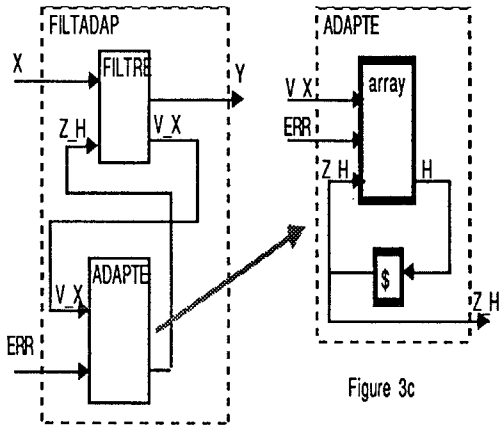


Figure 3b

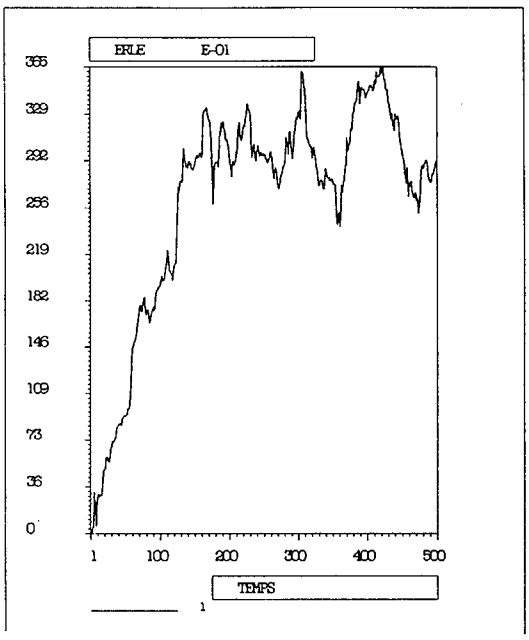
Figure 3c

Figures 3 a, b, c : schéma bloc SIGNAL

Dans les figures 3a, 3b et 3c les processus sont représentés par des boites en trait fin et les générateurs par des boites en trait gras. On passe d'une figure à la suivante selon l'ordre hiérarchique de définition des processus.
On trouvera en Annexe 2 le programme SIGNAL.

3 - SIMULATION avec SIGNAL

Figure 4 : Exemple de l'égaliseur double échantillonnage, rapport signal sur erreur en db



Le compilateur SIGNAL calcule les horloges de tous les signaux du programme et vérifie la cohérence temporelle. Le code produit pour la simulation est du code FORTRAN .

Les signaux d'entrées et de sorties sont automatiquement associés à des fichiers qui sont ensuite utilisés par un logiciel de mise en forme des résultats .

4 - AVANTAGES DU LANGAGE SIGNAL

L'étude d'exemples met en évidence les avantages d'un langages spécifique pour le traitement du signal et les améliorations apportées par rapport aux langages classiques tels que FORTRAN, PASCAL, ADA.
On retiendra les avantages suivants :

- L'aspect temps réel synchrone et la sémantique formelle de SIGNAL [1] permettent une vérification complète du comportement temporel (absence de blocage, synchronisation correcte des signaux) grâce à un mécanisme de calcul d'horloge qui transforme un programme en un système d'équations puis le résoud.
Les opérations temporelles réalisées dans les langages classiques par la gestion d'indices (fenêtre glissante, retard) et de variables booléennes (sous-échantillonnage) s'effectuent directement grâce aux opérateurs spécifiques du langage. L'écriture est donc plus concise et une source d'erreurs fréquentes est supprimée.

- L'aspect flots de données conduit à la fois à une description simple du parallélisme et à une structure modulaire des programmes .
Un processus est une unité syntaxique autonome qui peut être compilée et simulée de manière indépendante .Cette propriété a permis de comparer très facilement la convergence du filtre de Kalman et du filtre de Kalman parallèle (voir annexe 1).
De plus, considérer un processus comme une "boîte noire" accessible de l'extérieur par ses ports d'entrées et de sorties supprime les effets de bords dus aux variables globales et donc une source d'erreurs.

- La mise au point d'un programme SIGNAL est simplifiée. Le compilateur fournit dans un fichier trace toutes les informations nécessaires (en particulier sur les horloges). On change simplement les paramètres d'un programme (longueur de filtre, initialisation des coefficients).
Grâce à la gestion automatique des entrées sorties, on peut conserver sur fichier tout signal interne au programme en le faisant apparaître dans sa liste des sorties.

- On notera également la concision (programme égaliseur double échantillonnage 70 lignes), la lisibilité, la facilité d'écriture à partir des schémas blocs et des équations récurrentes .

5 - PERSPECTIVES

Afin d'assurer une continuité entre la spécification et la simulation de l'application d'une part et son exécution en temps réel sur une architecture multiprocesseurs utilisant des microprocesseurs spécialisés d'autre part [4], nous étudions actuellement:
- une méthode de simulation multimode où l'application est simulée à la fois au niveau fonctionnel et au niveau matériel,
- une bibliothèque de macro instructions en langage assembleur,
- des outils d'aide à la répartition d'une application de traitement du signal sur une architecture multiprocesseurs,
- un exécutil temps réel réparti réalisant un schéma d'exécution conforme aux principes définis dans la sémantique du langage.

BIBLIOGRAPHIE

[1] P. Le Guernic, A. Benveniste
"Real-time, synchronous, data-flow programming : the language SIGNAL and its mathematical semantics". Publications INRIA, Rapport de Recherche n° 533, Juin 1986.
[2] P. Le Guernic, A. Benveniste, P. Bournai, T. Gautier:
"SIGNAL : a data flow oriented language for signal processing". IEEE Transactions on Acoustics, Speech and Signal Processing, Volume ASSP-34(2), April 1986, 362-374.
[3] P. Le Guernic, A. Benveniste, T. Gautier
"Conception synchrone de systèmes à temps réels". Actes des Journées "Systèmes Temps Réel : programmation synchrone". 3-4 Février 1987, INRIA, Rocquencourt.
[4] Y. Sorel, Ph. Wolf
"Outils de mise au point de machines pour le traitement du signal, étude d'un cas : réalisation d'un modem 4800 bps". Dixième Colloque sur le Traitement du Signal et ses Applications, Nice 20-24 Mai 1985.
[5] L. Guidoux
"Egaliseur autoadaptatif à double échantillonnage". L'onde électrique, Vol. 55, pp. 9-13, Janv. 1975.
[6] T. Yahagi
"An adaptive echo canceller using parallel Kalman filters". ICASSP 1986, Tokyo, pp. 965-968.

ANNEXE 1 : Filtre de Kalman parallèle [6].

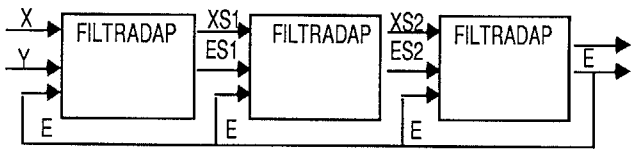
Le filtre à identifier est divisé en 3 sections. Les coefficients de chaque section sont identifiés par l'algorithme du filtre de Kalman. Cette méthode diminue considérablement les temps de calcul, avec une perte de performance acceptable.

Programmes SIGNAL :

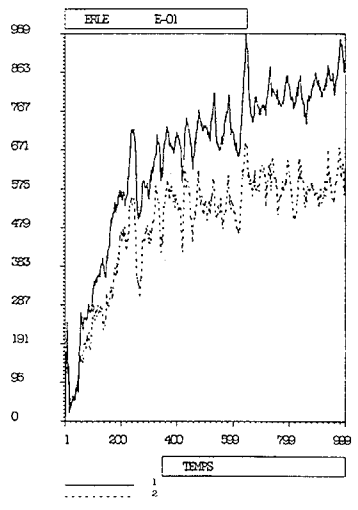
```
KalmanFilter { ... ? real X, Y  ! real E }
=(FILTRADAP (...) ? XE:X,EE:Y,ERR:E ! XS:XS1,ES:E) @E

ParallelKalmanFilter3 { ... ? real X, Y  ! real E }
=(| FILTRADAP (...) ? XE:X,EE:Y,ERR:E     ! XS:XS1,ES:ES1
  | FILTRADAP (...) ? XE:XS1,EE:ES1,ERR:E ! XS:XS2,ES:ES2
  | FILTRADAP (...) ? XE:XS2,EE:ES2,ERR:E ! XS:XS3,ES:E |)
```

ou, sous la forme de schéma bloc SIGNAL :



Rapport signal sur erreur en db, filtre de Kalman (1) et filtre de Kalman parallèle (2) :



ANNEXE 2 : Programme SIGNAL de l'égaliseur double échantillonnage

```
EGADOU {   integer NP,ND ; real MUP,MUD;
           [NP]real H0P; [ND]real H0D
         ? real X, YREF
         ! real ERREUR }

= ( ;  SEPARE ? IN:X ! OUTV:XP, OUTF:XD
    ;  (| FILTADAPT (NP,MUP,H0P) ? X:XP,ERR:ERREUR
                                  ! Y:YP,Z_H:Z_HP
       | ( ;  RESYNC ? IN:ERREUR, SYNC:XD ! OUT:ERD
            ; FILTADAPT (ND,MUD,H0D) ? X:XD,ERR:ERD
                                  ! Y:YD,Z_H:Z_HD
            ;  Z_YD := YD $ ( (NP-ND)/2 )
            ;  RESYNC ? IN:Z_YD, SYNC:YP ! OUT:YDR
            ; )
       | )
    ;  YEST := YP + YDR
    ;  ERREUR := YREF -YEST
    ; ) @ ERREUR !! ERREUR

where
 real XP, XD, Z_YD init 0, YP, YD, YDR, YEST ;
 [NP]real Z_HP init H0P;    [ND]real Z_HD init H0D
process

SEPARE { ? real IN ! real OUTV,OUTF }
= ( ; CPT := # event IN
    ; CPAIR := ( MOD ( CPT, 2 ) = 0 )
    ; ( &  OUTV := IN when CPAIR
       &  OUTF := IN when not CPAIR & )
    ; ) !! OUTV, OUTF
where
 logical CPAIR; integer CPT
end;  % process SEPARE %

RESYNC { ? real IN, SYNC ! real OUT }
= ( ; OUT := IN cell event SYNC
    ; OUT := OUT when event SYNC  ; ) !! OUT
end; % process RESYNC %

FILTADAPT {   integer N; real MU; [N]real H0
            ? real X, ERR
            ! real Y; [N]real Z_H init H0 }
= ( | FILTRE ( N ) ? X:X, Z_H:Z_H ! Y:Y, V_X:V_X
    | ADAPTE ( N, MU ) ? ERR:ERR, V_X:V_X ! Z_H:Z_H
    | ) !! Y, Z_H
where
  [N]real V_X init 0
process

  FILTRE {   integer N
           ? real X; [N]real Z_H
           ! real Y; [N]real V_X }
  = ( ; V_X := X window N
      ; VS0 := 0
      ; array I to N of
          VS := VS + ( V_X * Z_H )
          with VS[0]:VS0, V_X, Z_H end
      ; Y := VS[N] ; ) !! Y, V_X
  where
    real VS0; [N]real VS
end; % process FILTRE %
  ADAPTE {   integer N; real MU
           ? real ERR; [N]real V_X
           ! [N]real Z_H }
  = ( | Z_H := H $ 1
      | array I to N of
          H := Z_H + MU * ERR * V_X
          with Z_H, V_X end |  ) !! Z_H
  where
    [N]real H
end  % process ADAPTE %
end % process FILTADAPT %
end
```