Méthodologies de conversion automatique en virgule fixe pour les applications de traitement du signal

École thématique ARCHI 03
Roscoff 31mars - 4 avril

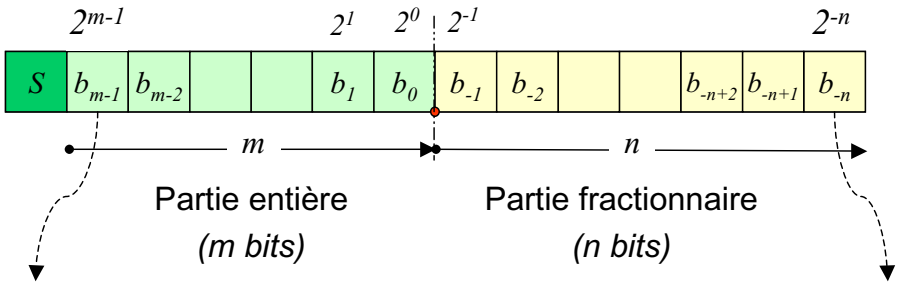
D. MENARD
Équipe de recherche *R2D2* - IRISA/INRIA
ENSSAT - Université de Rennes1

I. Introduction
-
Codage en virgule fixe

Codage en virgule fixe

- **Codage en virgule fixe complément à deux**

$$x = -2^m S + \sum_{i=-n}^{m-1} b_i 2^i$$



Domaine de définition du codage :

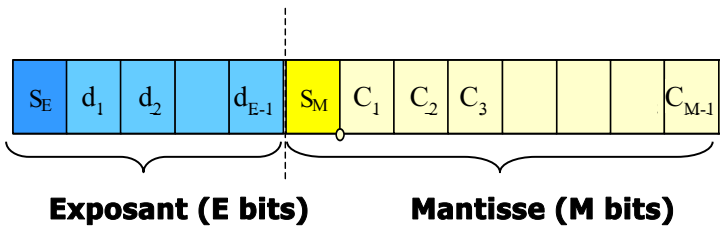
$$D = [-2^m, 2^m - 2^{-n}]$$

Précision du codage
Pas de quantification :

$$q = 2^{-n}$$

Codage en virgule flottante

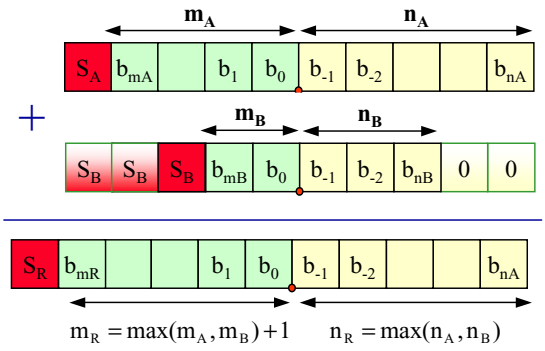
- **Codage virgule flottante**
  - L'exposant associé à la donnée est codée au sein de celle-ci
  - Les données sont composées de deux parties
    - ✓ Exposant
    - ✓ Mantisse
      - ☞ Sa valeur est comprise dans l'intervalle $\left[-1,-\frac{1}{2}\right]\cup\left[\frac{1}{2},1\right]$



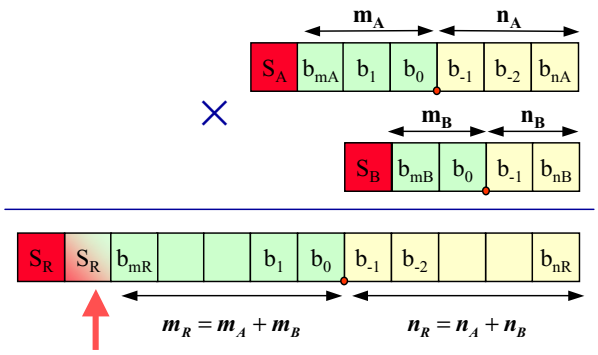
$$x = 2^u (-1)^{S_E} \left(1 + \sum_{i=1}^{M-1} C_i 2^{i-1}\right) \quad \text{avec} \quad u = (-1)^{S_E} \sum_{i=1}^{E-1} d_i 2^i$$

Règles de l'arithmétique virgule fixe

- **Addition** *a+b*
  - *Choix d'un format identique*
  - *Alignement de la virgule*
  - *Extension de bits*



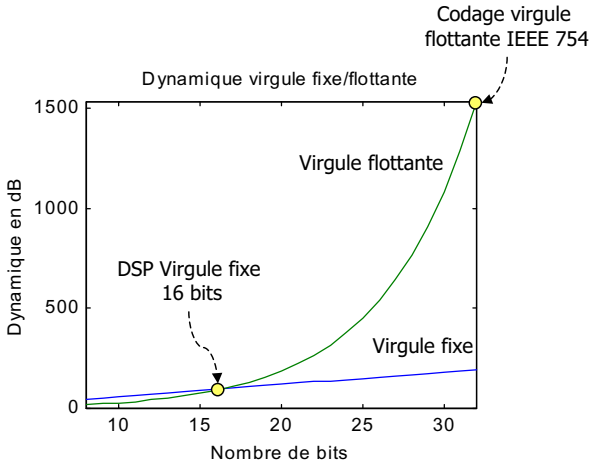
- **Multiplication** *a* × *b*
  - Représentation identique
    - ✓ Doublement du bit de signe



Comparaison virgule fixe / flottante

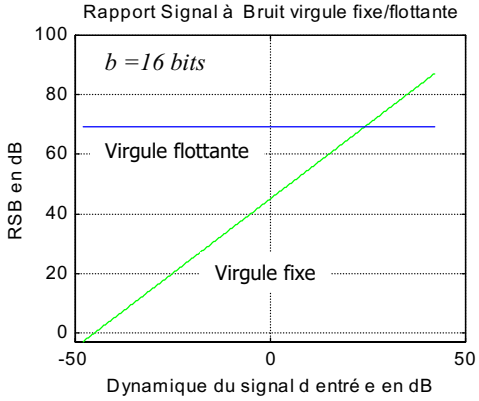
- **Niveau de dynamique**

$$D_{N(dB)} = 20.\log\left(\frac{\max(|x|)}{\min(|x|)}\right)$$



- **Rapport Signal à Bruit de Quantification**

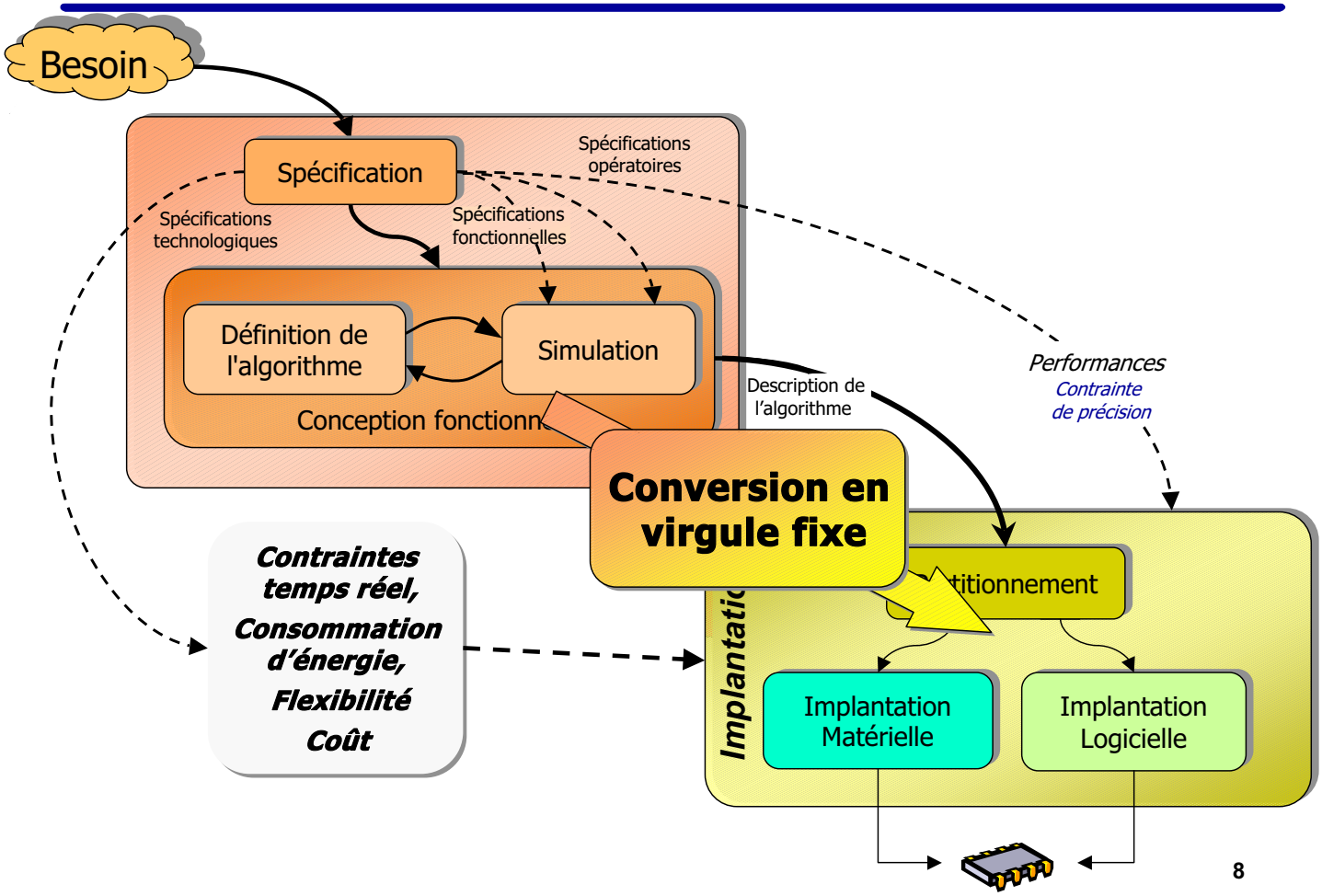
$$\rho_{dB} = 10.\log\left(\frac{P_s}{P_e}\right)$$



Arithmétique & systèmes embarqués

- **Arithmétique virgule flottante**
  - Largeur des données stockées en mémoire : 32 bits (IEEE 754)
  - Opérateurs complexes (gestion de la mantisse et de l'exposant)
  - Temps de développement faible
    - ✓ **L'utilisateur ne doit pas coder les données**
      - ☞ *Applications spécifiques : audionumérique, faible volume*
- **Arithmétique virgule fixe**
  - Opérateurs plus simples
    - ✓ Processeur plus rapide
  - Largeur des données stockées en mémoire : 16 bits
    - ✓ Efficacité énergétique plus importante, consommation moins importante
    - ✓ Processeur moins cher (surface du circuit moins importante)
      - ☞ *Applications grand public*
      - ☞ *DSP virgule fixe : 95% des ventes*

Cycle de développement

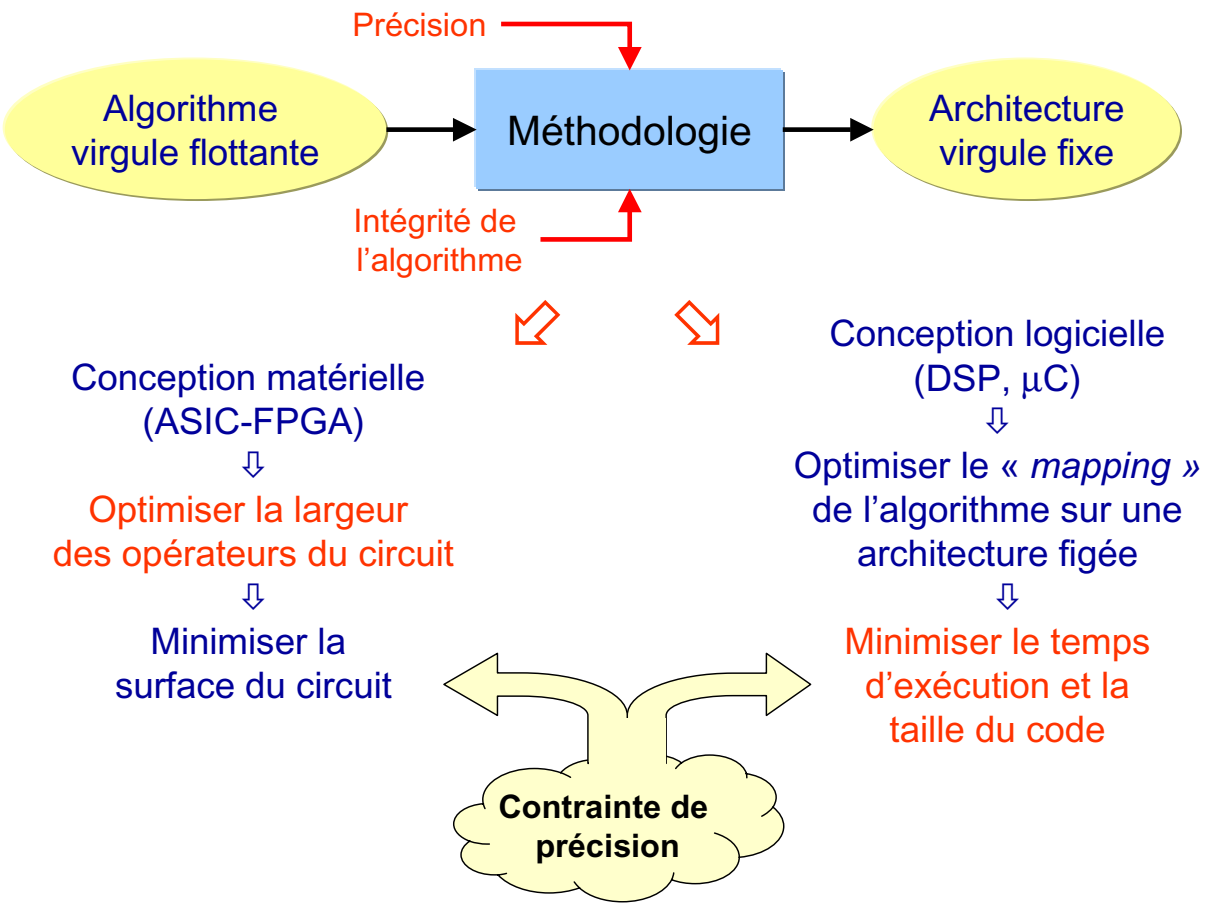


Objectifs du codage des données

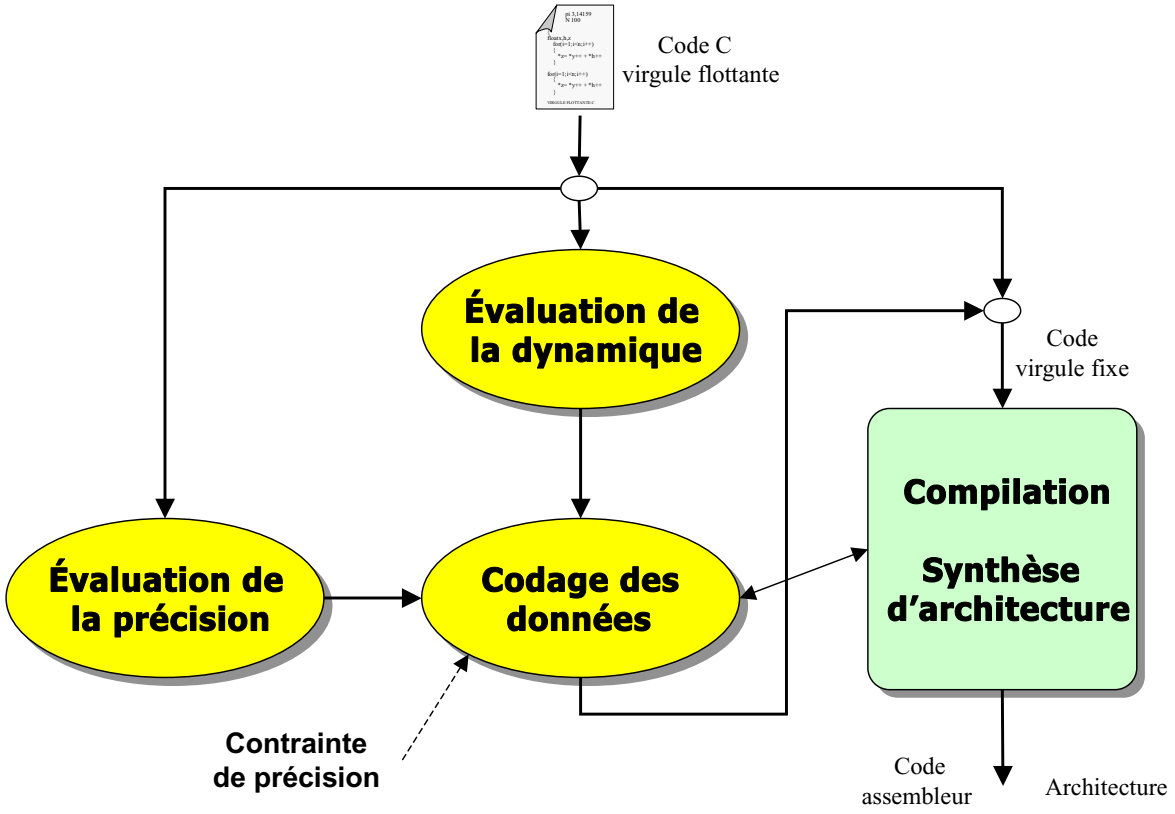
- **Codage en virgule fixe**
  - Définir pour chaque donnée la position de la virgule
    - ☞ Nombre de bits pour les parties fractionnaires et entières
  - Respecter les règles de l'arithmétique virgule fixe

- **Objectifs et contraintes du codage en virgule fixe**
  - Maintenir la fonctionnalité de l'algorithme
    - ✓ Respecter les règles de l'arithmétique virgule fixe
    - ✓ Garantir l'absence de débordement
  - Satisfaire la contrainte de précision
  - Optimiser l'implantation de l'algorithme
    - ✓ Implantation matérielle : minimiser la surface (prix) et la consommation d'énergie
    - ✓ Implantation logicielle : minimiser le temps d'exécution et la taille du code

Objectifs du codage des données



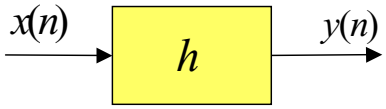
Processus de conversion en virgule fixe



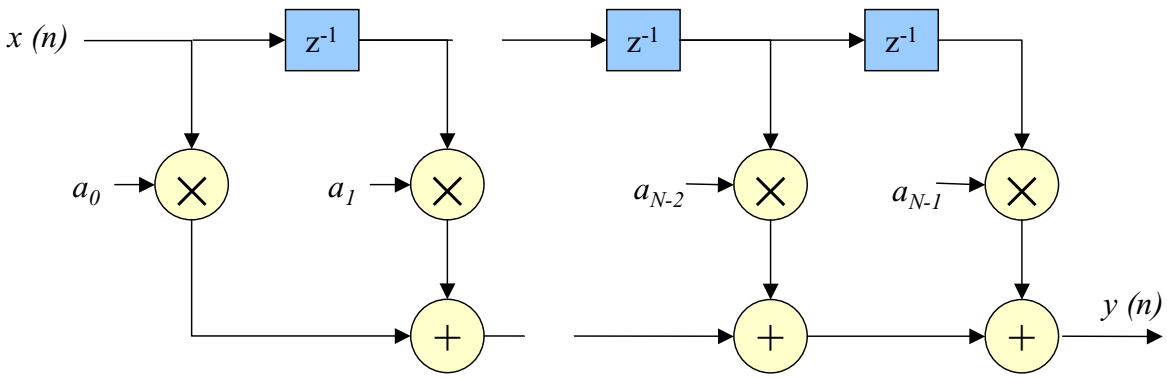
Application *fil rouge* : filtre FIR

- **Équation aux différences**

$$y(n) = \sum_{i=0}^{N-1} a_i . x(n-i) = x(n) * h(n) \quad \text{avec} \quad \begin{cases} h(i) = a_i & \forall i \in [0, N-1] \\ h(i) = 0 & \text{ailleurs} \end{cases}$$



- **Graphe Flot de Signal**



II. Évaluation de la précision des systèmes en virgule fixe

Métrique pour l'évaluation de la précision

- **Erreur de quantification associée à une donnée x :**
  - Différence entre la donnée en précision finie (virgule fixe) et la donnée en précision infinie (valeur exacte)

$$b_x = x_{\text{précision finie}} - x_{\text{précision infinie}}$$

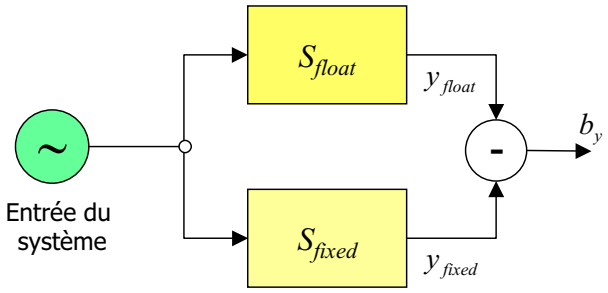
- **Propriété de l'erreur de quantification**
  - Variable aléatoire (ergodique et stationnaire)
    - ☞ Caractérisée par sa puissance (moment d'ordre 2) P_{b_x}
- **Métrique d'évaluation de la précision**
  - Rapport Signal à Bruit de Quantification

$$RSBQ = \frac{P_y}{P_{b_y}}$$

    - ☞ P_y : puissance du signal
    - ☞ P_{b_y} : puissance du bruit de quantification

Méthodes basées sur la simulation

- **Principe**
  - Détermination de la puissance du bruit de quantification à partir de la simulation du système en virgule fixe (y_{fixed}) et en virgule flottante (y_{float})
    - ✓ La sortie en virgule flottante est considérée comme la référence
      - ☞ Hypothèse valide si l 'erreur liée à l 'arithmétique virgule flottante est négligeable par rapport celle liée à la virgule fixe
        - ✓ **La largeur des données en virgule fixe doit rester faible**



$$P_{b_y} = \frac{1}{N_{pts}} \sum_{n=0}^{N_{pts}} \left(y_{float} - y_{fixed} \right)^2$$

Méthodes basées sur la simulation

- **Mise en œuvre : utilisation de librairies pour émuler l'arithmétique virgule fixe**
  - Utilisation de classe C++ : systemC, gFix [Kim 98]
    - ✓ Temps de simulation élevés
  - Utilisation des caractéristiques de la machine hôte pour accélérer la simulation en virgule fixe
    - ✓ Utilisation de types optimisés : pFix [Kim 98]
    - ✓ Génération d'un code optimisé : FRIDGE [Ked 01]
      - ☞ Réduction des temps de simulation
      - ☞ Augmentation du temps nécessaire pour générer le code utilisé pour la simulation

☞Temps d'optimisation du format des données très élevé [Sun 95]
- ☞ Une nouvelle simulation en virgule fixe est requise dès que le format d'une donnée est modifié

Méthodes basées sur la simulation

- **Exemples de temps de simulation**
  - Filtre IIR d'ordre 4, (PC pentium 100 MHz) [Sun 95]

Type	Flottant	gFix	pFix	VHDL	SPW
Temps de simulation (s)	2.19	340	16.3	181	60
Rapport fixe / flottant	1	155	7.4	82.6	27.4

  - Comparaison des temps de simulation en virgule fixe et en virgule flottante pour 6 applications [Ked 01]
    - ✓ FIR, DCT, IIR, FFT, auto-corrélation, produit de matrice

Type	Flottant	SystemC	SystemC précision limitée	Code optimisé
Rapport fixe / flottant	1	540	120	3.6

Méthodes basées sur la simulation

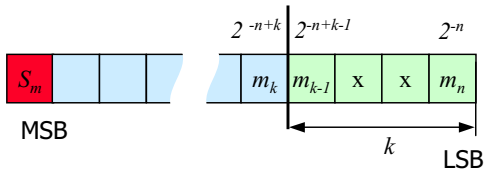
- **Adaptation de la méthode CESTAC à la virgule fixe**
  - Détermination du nombre de bits significatifs au niveau de la sortie de l'application
- **Méthode CESTAC**
  - Estimation de l'erreur d'arrondi liée à l'arithmétique virgule flottante à partir de quelques réalisations de l'application (3)
  - Hypothèse : l'erreur en sortie suit une loi de probabilité Gaussienne
    - ✓ Utilisation du test de Student pour déterminer l'intervalle de confiance de l'estimation de la moyenne de l'erreur
      - ☞ Déduction du nombre de bits significatifs à partir de l'intervalle de confiance

Méthodes analytiques

- **Méthode proposée par Toureille** [Tou99]
  - Détermination de l'expression analytique du RSBQ
    - ✓ Détermination de l'expression de la puissance du bruit
  - Propagation des moments du bruit au sein d'un GFD
    - ✓ Définition d'un modèle de propagation des moments du bruit pour chaque type d'opérateur
      - ☞ Hypothèses simplificatrices
      - ☞ Traitement uniquement des structures non-récursives
- **Développement d'une nouvelle méthode analytique**
  - Estimation précise de la puissance du bruit de quantification
  - Traitement des structures linéaires récursives

Modèles de bruit de quantification

- **Modélisation du bruit généré lors d'un changement de format :**
  - Erreur de quantification *b* :
    - ✓ Variable aléatoire discrète uniformément répartie



$$\mu_b = \frac{-q}{2}\left(1-2^{-k}\right)$$

$$\sigma_b^2 = \frac{q^2}{12}\left(1-2^{-2k}\right)$$

avec $q = 2^{-(n-k)}$

(pas de quantification après changement de format)

- **Modèle de propagation du bruit :**
  - Addition : $b_z = b_x + b_y$
  - Multiplication : $b_z = b_x \times s_y + s_x \times b_y + b_x \times b_y$

Systèmes linéaires

- **Système linéaire, entrée x, sortie y :**
  - Équation aux différences

$$y(n) = \sum_{k=0}^{K-1} a_k . x(n-k) + \sum_{l=1}^{L-1} c_l . y(n-l)$$

☞ Système non récursif si $\forall i,\ c_i = 0$

  - $h(n)$ réponse impulsionnelle du système linéaire

$$y(n) = \sum_{j=0}^{+\infty} h(j).x(n-j) = h(n) * x(n)$$

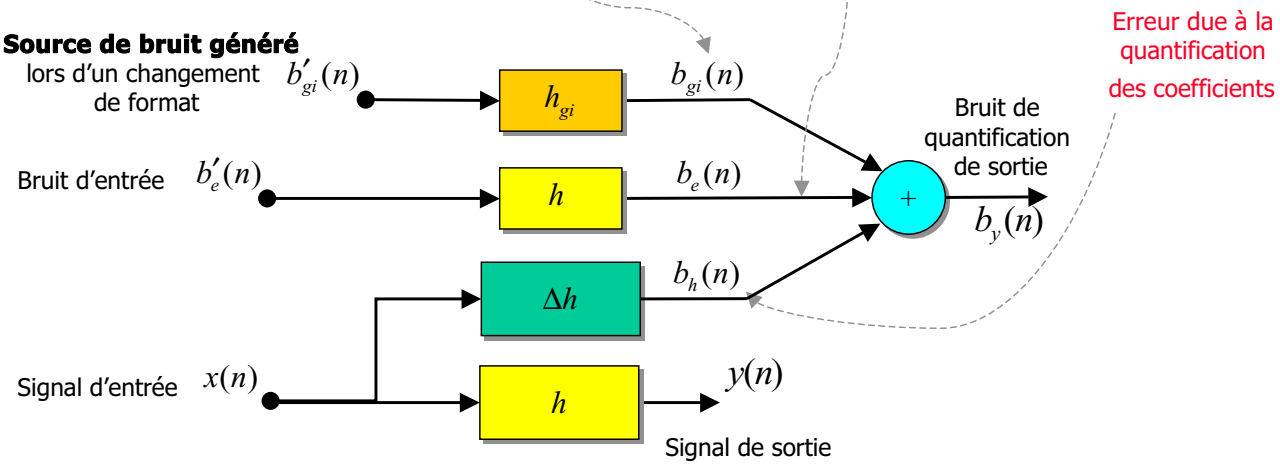
  - $H(z)$ fonction de transfert du système linéaire

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{K-1} a_k . z^{-k}}{\sum_{l=1}^{L-1} c_l . z^{-l}}$$

Modélisation d'un système linéaire

- **Expression du bruit de quantification**

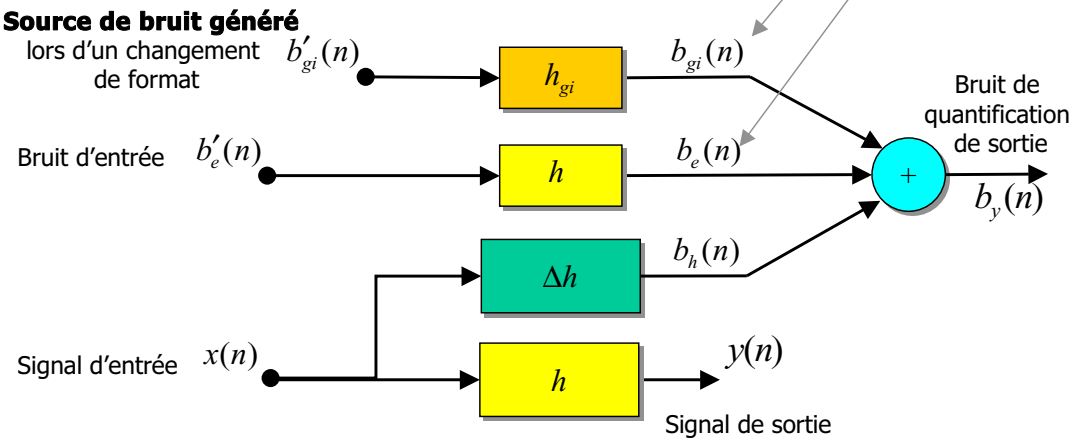
$$b_y = \hat{y}(n) - y(n) = \sum_{i=0}^{N_g-1} h_{gi}(n) * b_{gi}(n) + h(n) * b'_e(n) + \Delta h(n) * x(n)$$



Puissance du bruit de quantification b_y

- **Puissance du bruit de quantification** [Men02b]

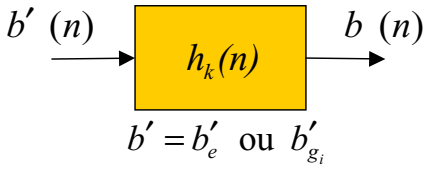
$$E\left(b_y^2\right) = \underbrace{\sum_{i=0}^{N_g-1} \mu_{b_{gi}}}_{\text{RCC TM}} + \mu_{b_e}^2 + \sum_{i=0}^{N_g-1} \sigma_{b_{gi}}^2 + \sigma_{b_e}^2$$



Puissance du bruit de quantification b_y

- **Puissance du bruit de quantification** [Men02b]

$$E\left(b_y^2\right) = \underbrace{\sum_{i=0}^{N_g-1} \mu_{b_{gi}}}_{\text{RCC TM}} + \mu_{b_e}^2 + \sum_{i=0}^{N_g-1} \sigma_{b_{gi}}^2 + \sigma_{b_e}^2$$

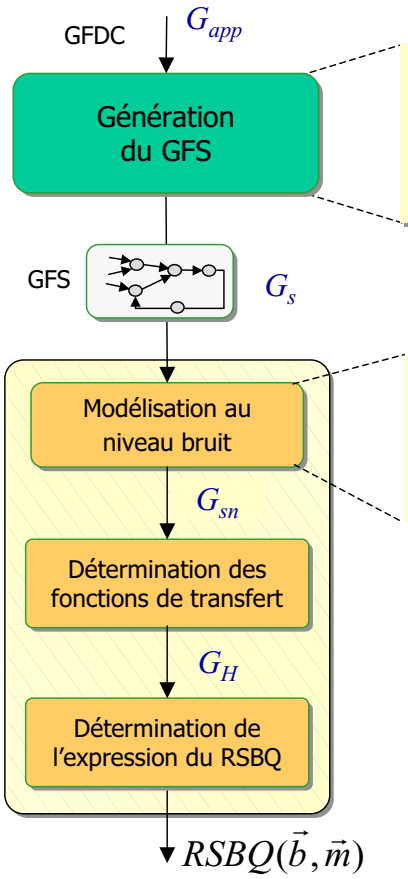


Propriété : les bruits b' sont des bruits blancs (les échantillons ne sont pas corrélés d 'un point de vue temporel)

$$\mu_b = \left(\mu_{b'} H\left(e^{j0}\right)\right)$$

$$\sigma_b^2 = \frac{\sigma_{b'}^2}{2\pi} \int_{-\pi}^{\pi} \left|H\left(e^{j\Omega}\right)\right|^2 d\Omega = \sigma_{b'}^2 \underbrace{\sum_{m=0}^{+\infty} |h(m)|^2}$$

Synoptique de la méthodologie



Génération du Graphe Flot de Signal

- Élimination des structures de contrôle
- Insertion des informations temporelles (opérations de retard)

Modélisation du système au niveau bruit

- Insertion des sources d'erreur
- Modélisation des données et des opérateurs au niveau bruit

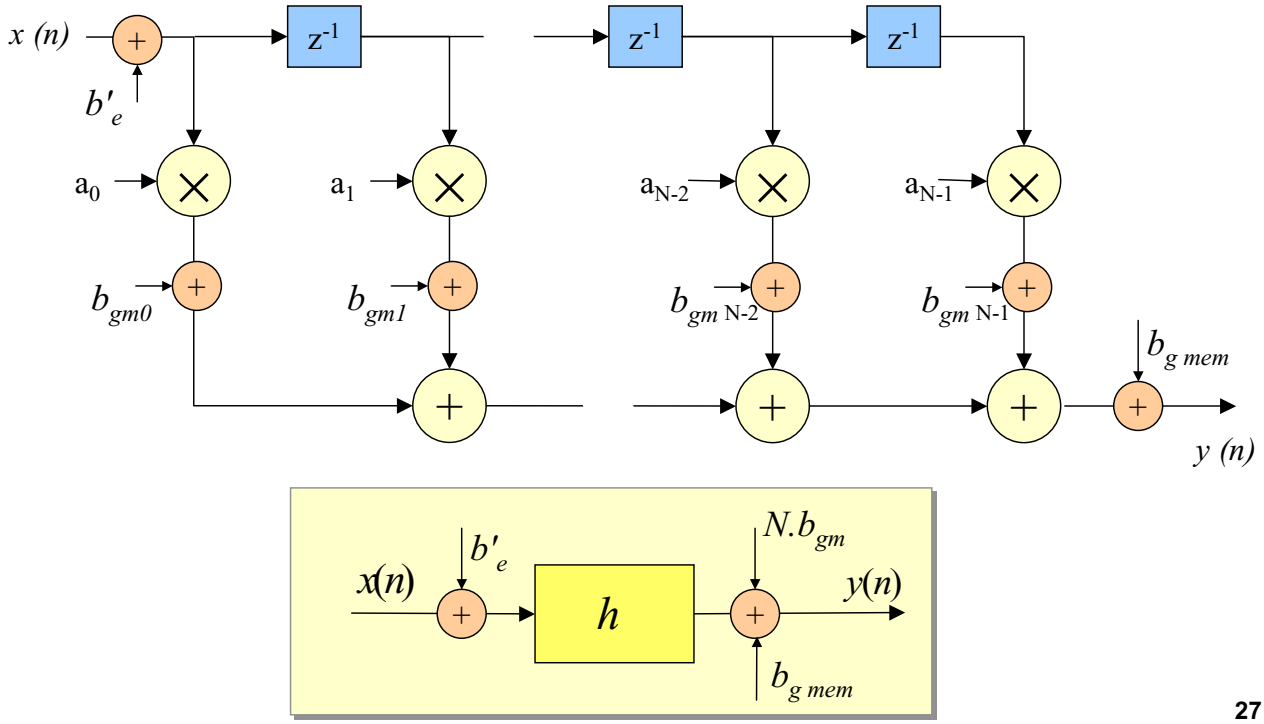
[Men02a]

Détermination des fonctions de transfert

- **1. Démantèlement des circuits**
  - **Objectif** : transformation du Graphe Flot de Signal en graphes acycliques (DAG)
- **2. Détermination des fonctions linéaires**
  - **Objectif :** spécification du système par un ensemble de fonctions linéaires
- **3. Détermination des fonctions de transfert partielles**
  - **Objectif :** spécification du système par un ensemble de fonctions de transfert partielles
- **4. Détermination des fonctions de transfert globales**
  - **Objectif** : détermination des fonctions de transfert entre la sortie et chaque entrée

Exemple du filtre FIR

- **Spécification du filtre au niveau bruit de quantification**
  - ☞ Présence d'une source de bruit potentielle en entrée ou en sortie de chaque opération



Exemple du filtre FIR

- **Expression du bruit en sortie du système**

$$b_y(n) = b_e(n) + b_{g.mem}(n) + \sum_{i=0}^{N-1} b_{gm.i}(n)$$

$$b_y(n) = \sum_{l=0}^{N-1} a_l\, b'_e(n-l) + b_{g.mem}(n) + \sum_{i=0}^{N-1} b_{gm.i}(n)$$

- **Fonction de transfert globale du système**

$$\begin{cases} B_y(z) = H(z)B'_e(z) + B_{g.mem}(z) + \sum_{i=0}^{N-1} b_{gm.i}(z) \\ Y(z) = H(z)X(z) \end{cases}$$

Exemple du filtre FIR

- **Expression de la variance du bruit en sortie du filtre FIR**

$$\sigma_{b_y}^2 = \sigma_{b_e}^2 + \sigma_{b_{g.mem}}^2 + \sum_{i=0}^{N-1} \sigma_{b_{gm.i}}^2$$

$$\sigma_{b_y}^2 = \sigma_{b'_e}^2 \sum_{m=-\infty}^{+\infty} |h(m)|^2 + \sigma_{b_{g.mem}}^2 + \sum_{i=0}^{N-1} \sigma_{b_{gm.i}}^2$$

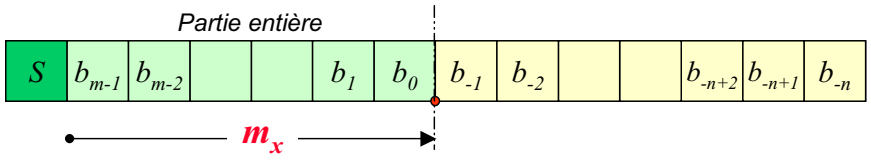
$$\sigma_{b_y}^2 = \frac{q_e^2}{12} \underbrace{\sum_{m=0}^{N-1} a_m^{\,2}}_{\alpha_h} + \frac{q_{g.mem}^2}{12} + N.\frac{q_{mi}^2}{12}$$

$$\sigma_{b_y}^2 = \frac{1}{12}\left(\alpha_h.2^{-2n_e} + 2^{-2n_{gmem}} + N.2^{-2n_{mi}}\right)$$

III. Évaluation de la dynamique des données

Objectifs

- **Estimation du domaine de définition $[x_{min}, x_{max}]$ de chaque donnée x en vue d'en déduire la position de la virgule m_x**

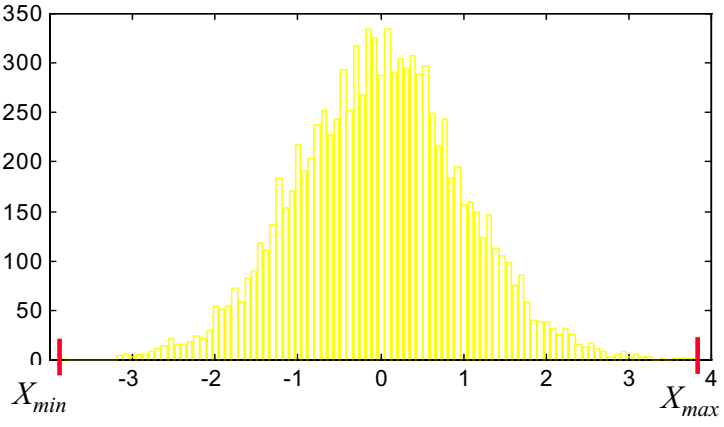


$$m_x = \left\lceil \log_2 \left(\max \left(\left| x_{\min} \right|, \left| x_{\max} \right| \right) \right) \right\rceil$$

- **Critères de qualité pour l'estimation**
  - Précision : minimiser l'erreur d'estimation
    - ✓ Éviter la présence de bits non utilisés au niveau des bits les plus significatifs de la donnée
  - Qualité : garantir l'absence de débordement

Méthodes basées sur la simulation

- **Détermination de la dynamique d'une donnée à partir de ses paramètres statistiques**
  - Simulation de l'algorithme et collecte des échantillons
  - Détermination des paramètres statistiques et/ou des valeurs minimales et maximales
- **Détermination de la dynamique à partir des valeurs minimales X_{min} et maximales X_{max} obtenues** [Aam 01]



Méthodes basées sur la simulation

- **Méthode proposée par Kim** [Kim98]
  - Distribution uni-modale $(-1.2 < k_{x_i} < 5)$ et symétrique $s_{x_i} \approx 0$

$$\tilde{x}_{i\max} = |\mu_{x_i}| + (k_{x_i} + 4)\sigma_{x_i}$$

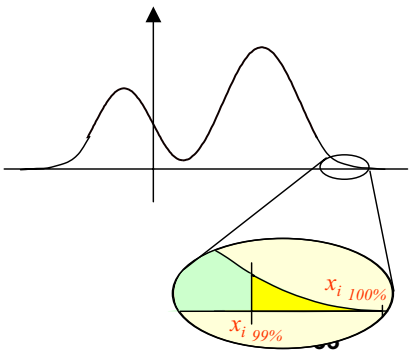
$$k_{x_i} = \frac{E[(x_i - \mu_{xi})^4]}{\sigma_{x_i}^4} - 3 \quad kurtosis$$

$$s_{x_i} = \frac{E[(x_i - \mu_{xi})^3]}{\sigma_{x_i}^3} \quad skewness$$

  - Distribution multi-modale ou non symétrique

$$\begin{cases} |x_{i\min}| = |x_{i_{0\%}}| + r_R |x_{i_{0\%}} - x_{i_{1\%}}| \\ |x_{i\max}| = |x_{i_{100\%}}| + r_R |x_{i_{100\%}} - x_{i_{99\%}}| \end{cases}$$

$x_{i_{99\%}}$ défini tel que $P(x_i < x_{i_{99\%}}) = 0.99$



Méthodes analytiques

- **Propagation de la dynamique des entrées au sein de l'application**
  - Utilisation des résultats de l'arithmétique d'intervalle [Kea 96]

Opérations	*min (z)*	*max (z)*
z=x+y	*min(x)+min(y)*	*max(x)+max(y)*
z=x-y	*min(x)-max(y)*	*max(x)-min(y)*
z=x×y	*min(E)*	*max(E)*

E =(min(x)min(y),min(x)max(y), min(y)max(x),max(x)max(y))

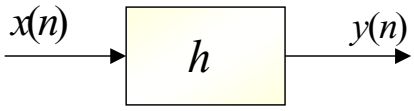
✓ Traitement des structures non-récursives

✓ Estimation pessimiste

☞ Absence de prise en compte de la corrélation entre les données

Méthodes analytiques

- **Systèmes linéaires : utilisation de normes**



- Normes L1

$$y_{\max 1} = \max_n \left(|x(n)| \right) . \sum_{m=-\infty}^{\infty} |h(m)|$$

☞ Systèmes linéaires non-récursifs : résultats identiques à ceux obtenus avec l'arithmétique d'intervalle

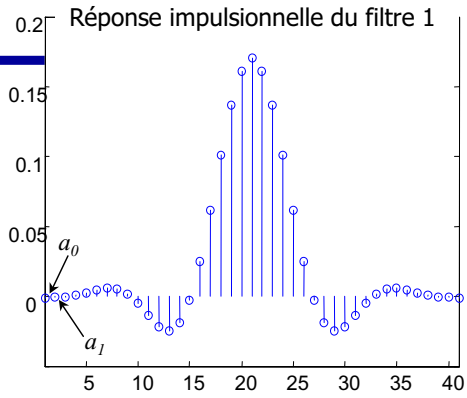
- Norme Chebychev

$$y_{\max 2} = \max_n \left(|x(n)| \right) \max_\omega \left(|H(\omega)| \right)$$

☞ Signal d'entrée du type $x(n) = \cos(\omega.n.T)$

Exemple filtre FIR

- **Entrée** $x \in [-1,1]$
- **Filtre 1 : 41 cellules**
- **Filtre 2 : 32 cellules**



Méthodes	*Filtre 1*		*Filtre 2*	
	y_{max}	m_x	y_{max}	m_x
Méthodes analytiques				
Arith. Intervalle, Norme L_1	1.35	1	6.26	3
Norme Chebychev	1.004	1	4.13	3
Méthodes statistiques				
Chirp	1.06	1	4.2	3
Bruit blanc gaussien	0.43	-1	1.77	1
Bruit blanc uniforme	0.8930	0	4.2	3

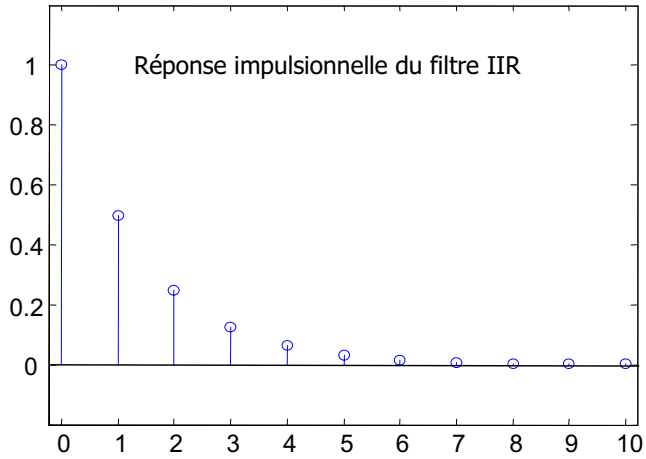
Exemple filtre IIR

- **Équation aux différences**

$$y(n) = x(n) + \frac{1}{2} y(n-1)$$

- **Réponse impulsionnelle**

$$h(n) = \left(\frac{1}{2}\right)^n$$



- **Dynamique de y**

$$\max_n\left(|y(n)|\right) = \max_n\left(|x(n)|\right) \cdot \sum_{m=-\infty}^{+\infty} |h(m)| = \sum_{m=0}^{+\infty} \left|\frac{1}{2}\right|^m = \frac{1}{1-1/2} = 2$$

Comparaison des méthodes

	Méthode statistique	Méthode analytique
Précision	Erreur d'estimation faible	Méthode conservatrice (estimation dans le pire cas)
Qualité	Pas de garantie sur l'absence de débordement Fonction de la représentativité des signaux	Garantie sur l'absence de débordement
Structures traitées	Toutes	Structures linéaires et non-linéaires non-récursives
Connaissances nécessaires	Signaux d'entrée	Domaine de définition des entrées

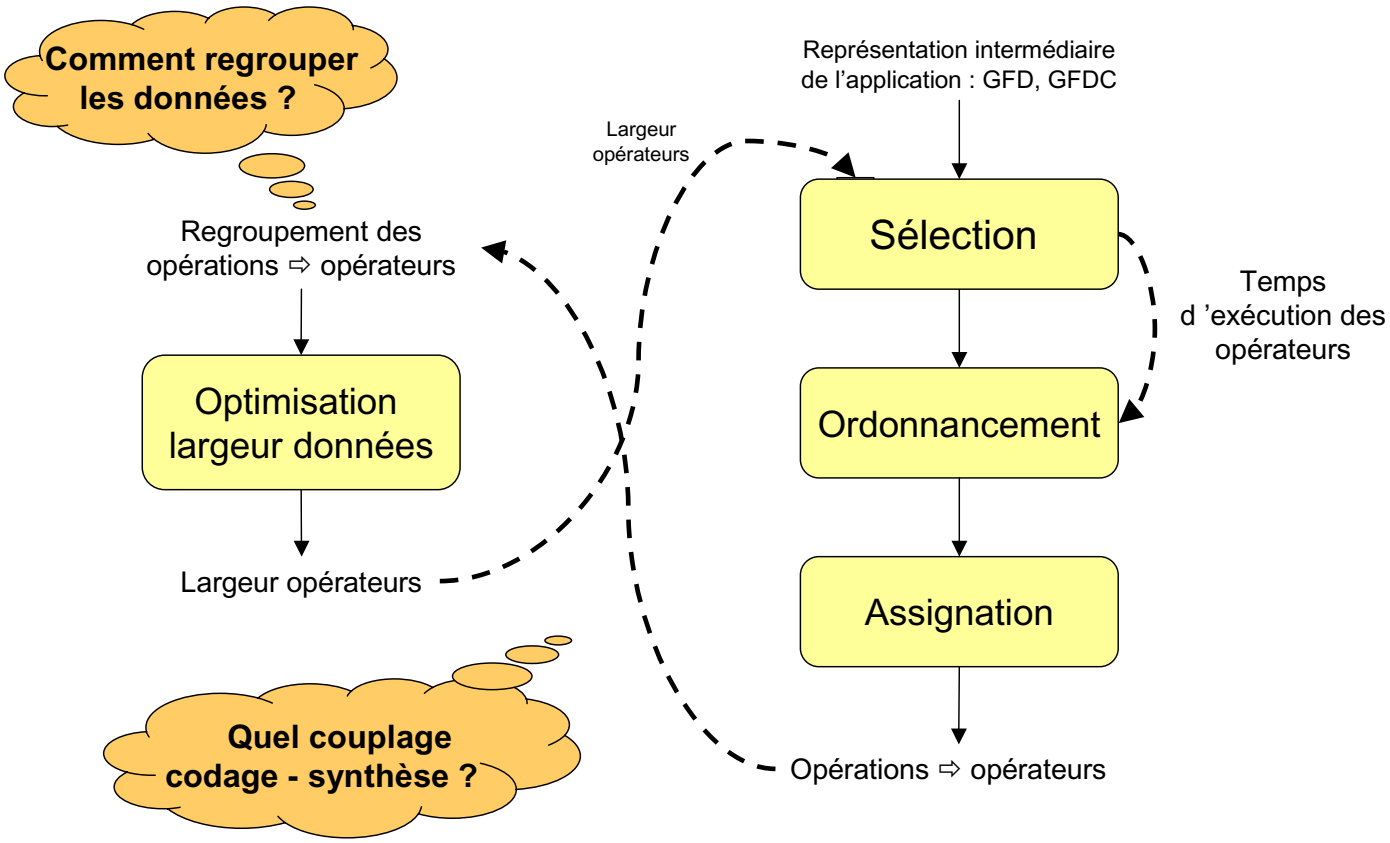
IV. Codage des données en virgule fixe pour une implantation matérielle

Objectifs

- **Implantation matérielle dans un ASIC ou un FPGA**
  - Définition de l'architecture du circuit
    - ✓ Possibilité d'adapter la largeur des opérateurs aux contraintes de précision
  - Objectifs de l'implantation
    - ✓ Minimiser le coût
      - ☞ Optimiser la surface du circuit
    - ✓ Minimiser la consommation d'énergie
      - ☞ Optimiser la largeur des opérateurs
- **Objectif du processus de conversion en virgule fixe**
  - Minimiser la surface du circuit sous contrainte de précision

$$\underset{\vec{b}\in Z^{+}}{\mathrm{Min}}\left(S(\vec{b})\right) \quad \text{tel que} \quad RSBQ(\vec{b}) \geq RSBQ_{\min}$$

Lien entre le codage et la synthèse



Niveau de regroupement des données

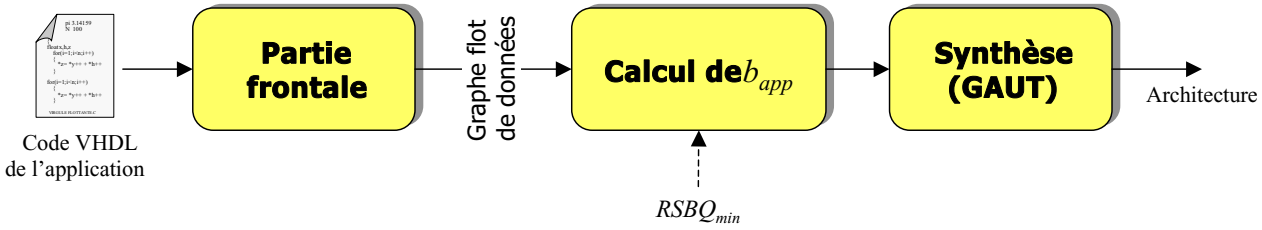
- **Algorithme** : toutes les données ont la même largeur
  - Le processus d'optimisation de la largeur des données peut être réalisé avant la synthèse d'architecture [Mar 01]
    - ✓ Qualité de l'implantation en terme de surface ?
- **Bloc** : toutes les données d'un bloc ont la même largeur
    - ✓ Limitation du nombre de variables pour l'optimisation
  - Regroupement dirigé par
    - ✓ L'analyse du graphe de l'application [Sun 95]
    - ✓ La synthèse : couplage du codage et de la synthèse [Kum 01]
- **Donnée** : chaque donnée à sa propre largeur
  - Codage puis synthèse de l'architecture [Kum 98], [Ked 98]
    - ✓ Affectation d'opérations de largeur différente sur un même opérateur [Con 01]

Méthode GAUT (LESTER - Lorient)

- **Regroupement au niveau de l'algorithme**
  - Largeur identique pour toutes les données (b_{app}) et utilisation d'un seul format (virgule fixe cadrée à gauche)
    - ✓ Utilisation d'une méthode analytique pour évaluer P_b
    - ✓ Relation simple entre la puissance du bruit P_b et b_{app}

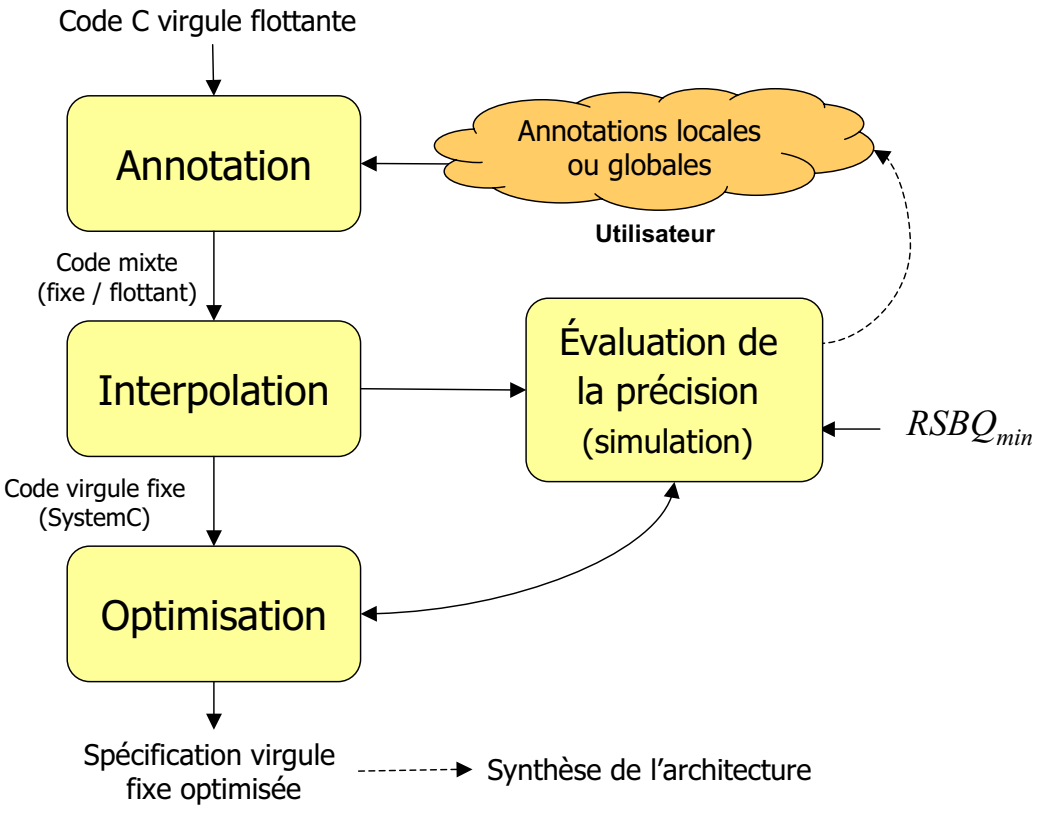
$$P_b = Aq^2 = A.2^{-2(b_{app}-1)} \quad \blacktriangledown \quad b_{app} \geq \frac{1}{2}\log_2\left(\frac{A}{P_s}10^{\frac{1}{10}RSBQ_{\min}}\right)+1$$

- **Séparation des processus de synthèse et de codage**



Méthode FRIDGE (IISPS - Aix la chapelle)

- **Regroupement au niveau donnée**



Les différentes phases de la méthode

- **Annotation**
  - Définition du format de quelques données en virgule fixe
- **Interpolation**
  - Détermination du format virgule fixe des données non-annotées par propagation de règles
    - ✓ Partie entière (m_x) : utilisation de l'arithmétique d'intervalle
    - ✓ Partie fractionnaire (n_x) : utilisation des règles $n_z = f_{Ops}(n_x, n_y)$
- **Optimisation**
  - Minimisation de la largeur des données sous contrainte de précision globale $RSBQ_{min}$
    - ✓ Utilisation d'une contrainte de précision locale :
      - ☞ Δ_{RSBQ} dégradation maximale autorisée du RSBQ en sortie d'un opérateur liée à la réduction de la largeur de la sortie de l'opérateur

Algorithme d'optimisation

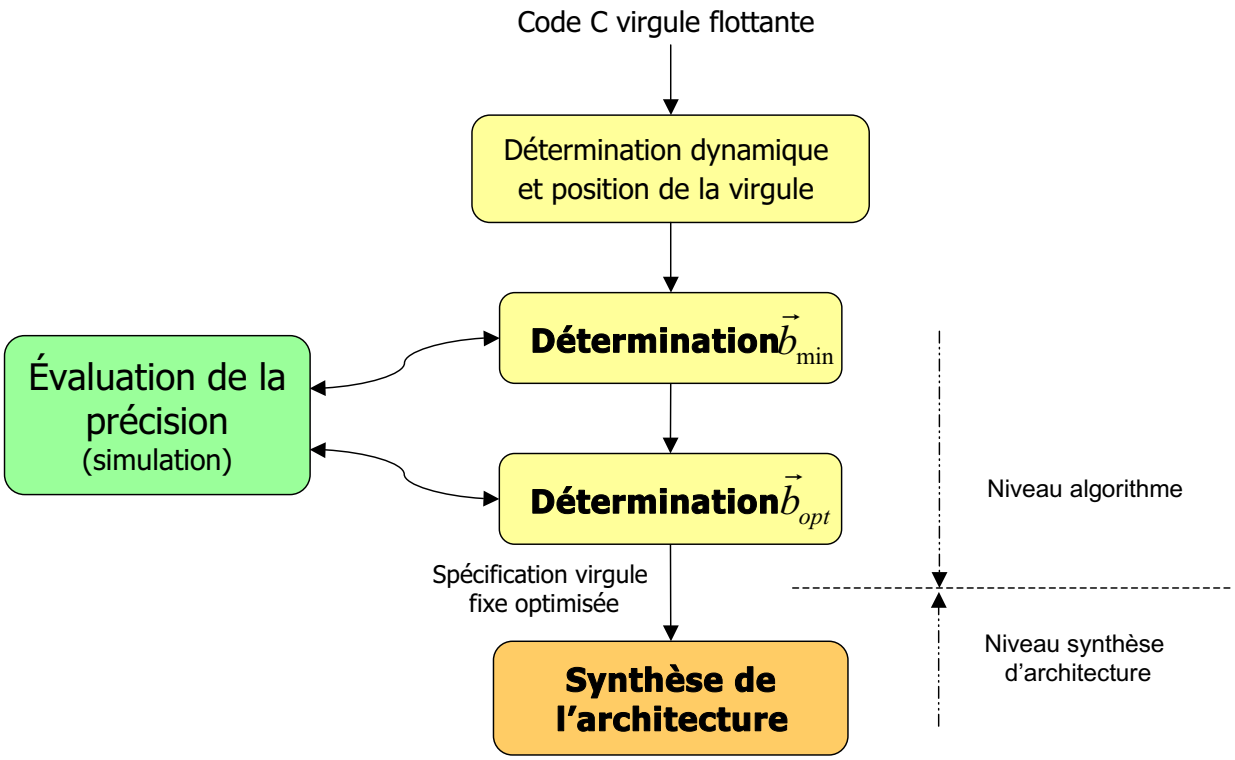
```
Tant que (RSBQ(b) > RSBQmin)   // contrainte de précision globale
{
  ΔRSBQ = ΔRSBQ + ε            // modification de la contrainte de précision locale
  Pour chaque (xi)
  {
    Tant que ( ΔRSBQ(bxi) < ΔRSBQ )
    {                          // pour chaque donnée diminution de la largeur tant
      bxi = bxi -1             // que la contrainte de précision locale est satisfaite
      Calcul(ΔRSBQ(bxi))       // évaluation analytique de ΔRSBQ(xi)
    }
  }
    Calcul(RSBQ(b))            // évaluation par simulation de RSBQ(b)
}
```

Méthode FPO (Université de Séoul)

- **Différentes méthodes proposées**
  - Regroupement au niveau donnée [Kum 98]
  - Regroupement au niveau bloc par analyse du graphe flot de signal de l'application [Sun 95]
    - ✓ Regroupement réalisé par l'utilisateur
    - ✓ Regroupement automatique
      - ☞ Données connectées par un délai ou un multiplexeur
      - ☞ Entrées et sortie d'un additionneur
  - Regroupement au niveau bloc : couplage des processus de codage en virgule fixe et de synthèse [Kum 01]
    - ✓ Regroupement des opérations réalisées sur un même opérateur

Méthode FPO

- **Regroupement au niveau donnée** [Kum 98]



Les différentes phases de la méthode

- **Détermination de la dynamique (méthode statistique) et de la position de la virgule des données**

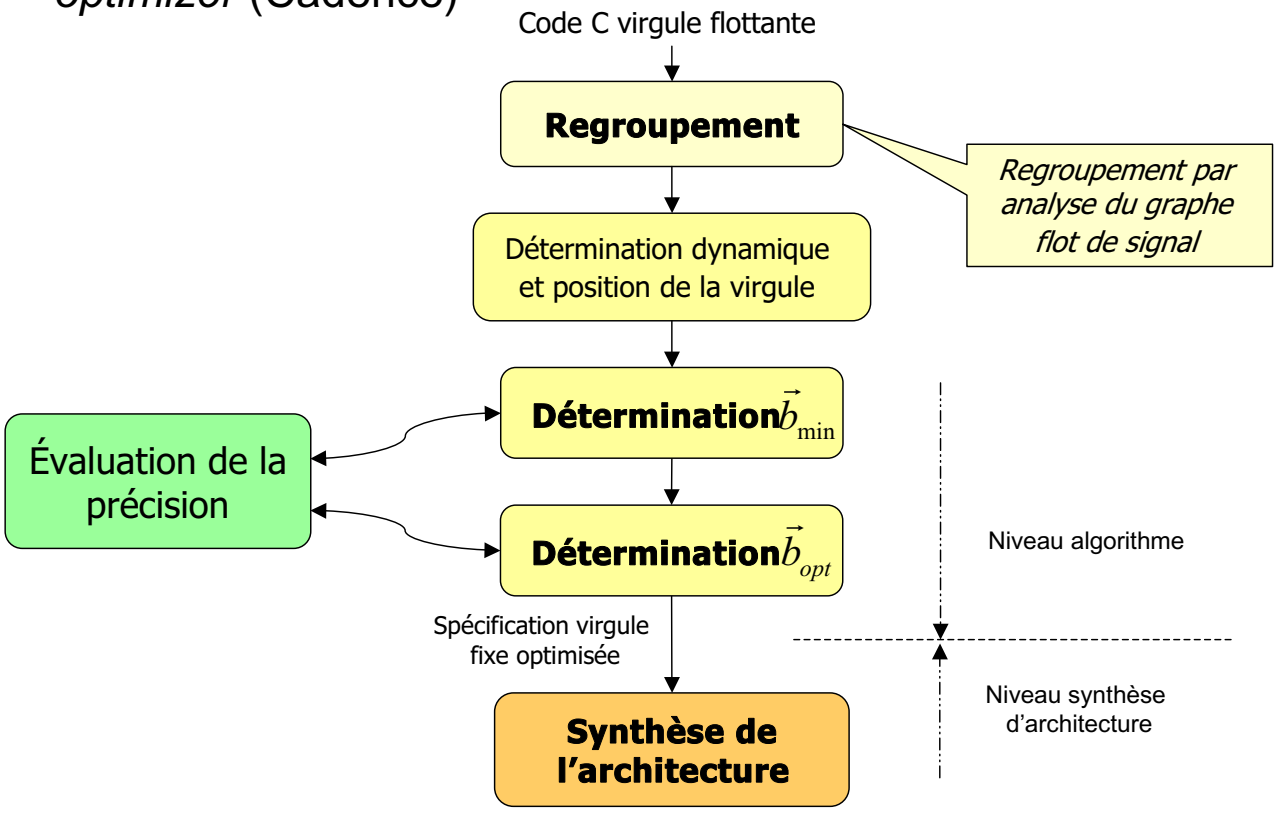
- **Détermination de la largeur minimale de chaque donnée $b_{i.min}$ définie telle que**

$$\min_{b_i \in Z^+}(b_i) \quad \begin{cases} RSBQ(\vec{b}) > RSBQ_{\min} \\ b_j = B_{\max} \quad \forall j \neq i \end{cases}$$

- **Détermination de la largeur optimisée des données**
  - ☞ Au départ la largeur des données est fixée à leur valeur min $b_{i.min}$
  - Heuristique en deux phases :
    - ✓ Incrémentation de la largeur de **toutes** les données tant que la contrainte de précision n'est pas satisfaite
    - ✓ Décrémentation de la largeur des données, **une par une**, tant que la contrainte de précision est satisfaite

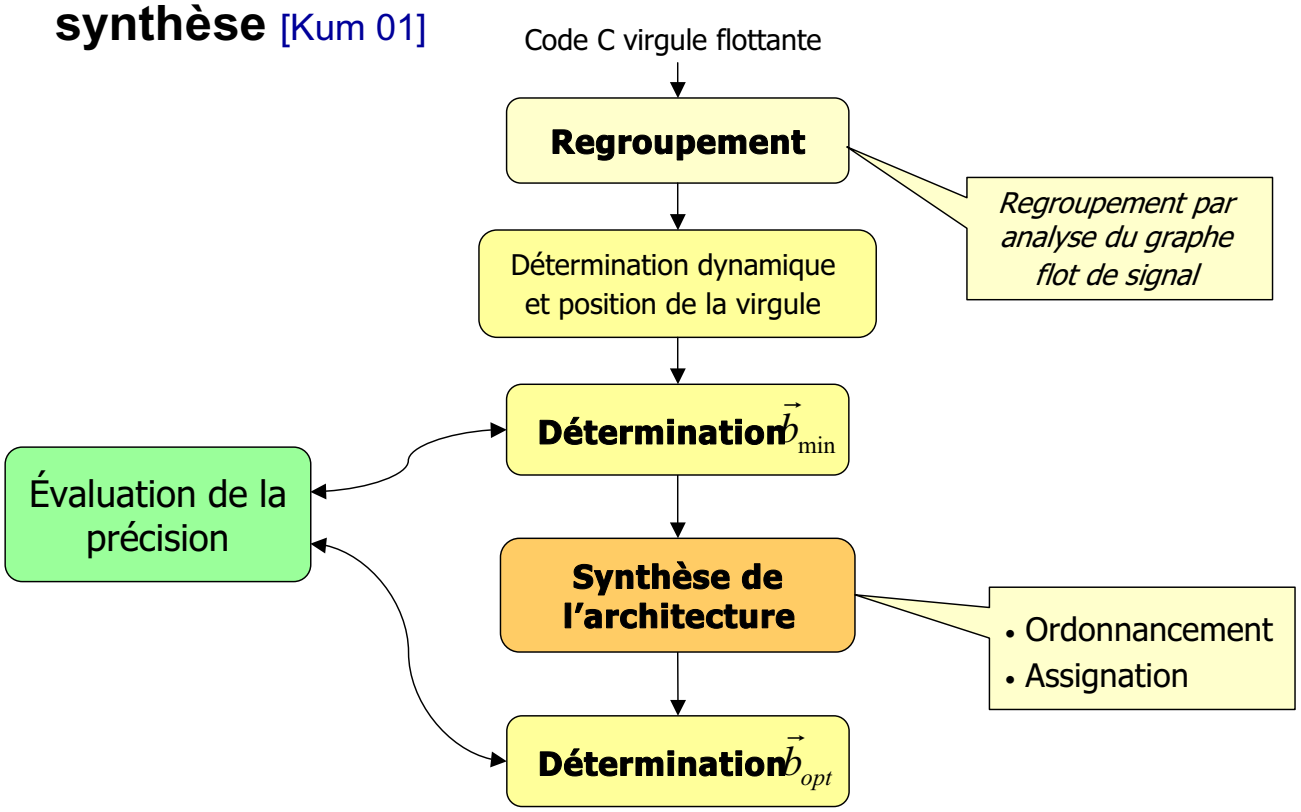
Méthode FPO

- **Regroupement au niveau bloc** [Sun 95] ⇨ *Fixed-point optimizer* (Cadence)



Méthode FPO

- **Regroupement au niveau bloc et couplage codage synthèse** [Kum 01]



Synthèse de l'architecture

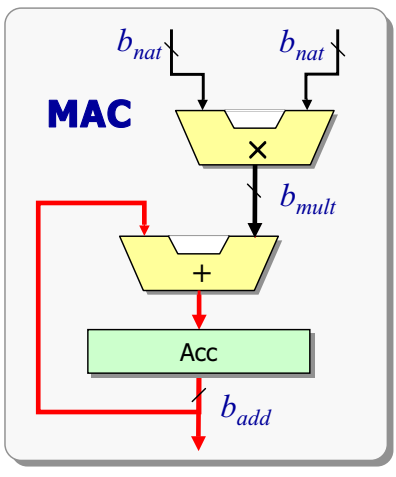
- **Ordonnancement par liste**
  - Une liste pour chaque opérateur de largeur différente
  - Traitement en priorité des opérations de largeur élevée
  - Assignation d'une opération à un opérateur de largeur plus élevée si celui-ci est libre

- **Processus itératif pour l'ordonnancement et l'optimisation de la largeur des opérateurs**
  - 1. Estimation du nombre minimal d'opérateurs nécessaires
  - 2. La largeur des opérateurs est fixée à la valeur maximale
  - 3. Ordonnancement de l'application
  - 4. Réduction de la largeur d'un des opérateurs tant que la contrainte de temps est satisfaite lors de l'ordonnancement

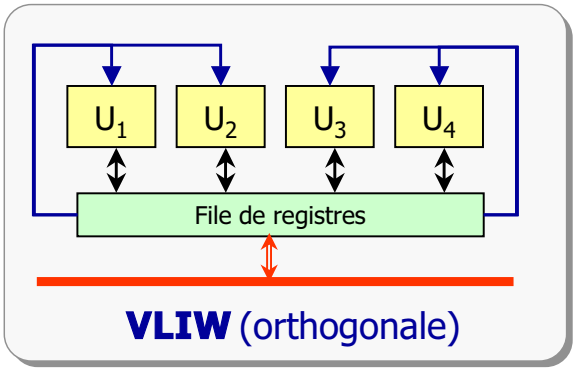
V. Codage des données en virgule fixe pour une implantation logicielle

Architecture des DSP

- **Architecture conventionnelle**
  - Structure multiplication-accumulation MAC
  - Registres dédiés aux opérateurs
  - Connections registre-opérateur optimisées
  - Jeu d'instructions spécialisé



- **Architecture parallèle**
  - Unités fonctionnelles indépendantes
  - File de registres
    - ✓ Tous les registres sont connectés à tous les opérateurs
      - ☞ Utilisation de clusters
  - Parallélisme au niveau instruction
    - ✓ Jeu d'instructions VLIW



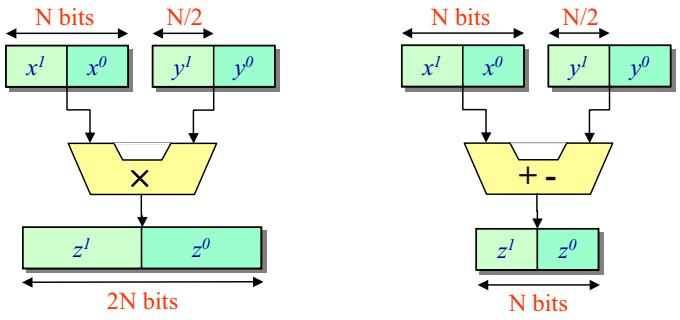
Architecture des DSP

- **Largeur naturelle du processeur** (b_{nat})
  - DSP: largeur fixe : 16 ou 24 bits
  - Cœur de DSP et ASIP : largeur paramétrable
    - ☞CD2450 (Clakspur) : 16 à 24 bits;
    - ☞EPICS (philips) : 12, 16, 18, 20, 24 bits;
    - ☞PalmCore (VLSI / DSP Group): 16, 20, 24 bits

- **Largeur des données au sein de l'unité de traitement**
  - Instructions classiques
    - ✓ Calcul d'une multiplication-addition sans perte de précision
    - ✓ Bits de garde au niveau de l'accumulateur
  - Instructions double précision
    - ✓ Augmentation de la précision : données stockées en mémoire en double précision
    - ✓Augmentation des temps de calcul

Architecture des DSP

- Instructions SWP (Sub-Word Parallelism)
  - ✓ Utilisation du parallélisme au niveau des données
  - ✓ Réduction du temps d'exécution



- **Loi de quantification**
  - Troncature : par défaut
  - Arrondi : quelques DSP

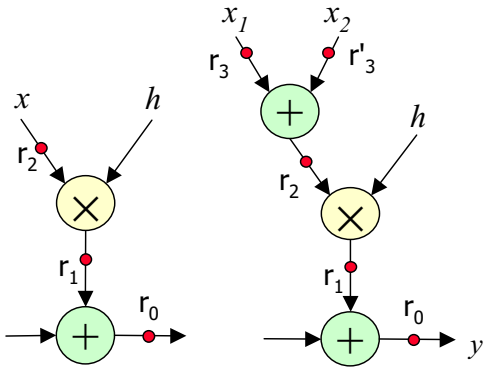
Architecture des DSP

- **Capacités de recadrage**
  - Registres à décalage spécialisés
    - ✓Décalages en sortie du multiplieur (C50 : -6, 0, 1, 4)
  - Registres à décalage en barillet

- **Coût d'un recadrage avec un registre en barillet**
  - Architectures conventionnelles
    - ✓ Le coût du recadrage est fonction de sa position

Recadrage	Coût du recadrage (cycles)			
	TMS320C54x		OakDSPCore	
	FIR	FIR Sym	FIR	FIR Sym
r_0	1	1	1	1
r_1	2	1	3	2
r_2	5	3	4	1
r_3	-	2	-	2

FIR Sym : filtre FIR symétrique



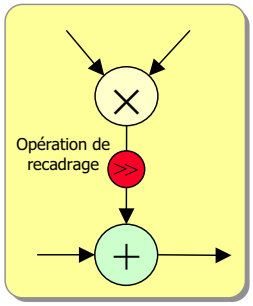
FIR FIR Symétrique

Architecture des DSP

- Architecture VLIW
  - ✓ Sur-coût C_s lié au décalage en sortie du multiplieur pour différentes applications [Men02d]
    - ☞ t_{wos} : temps d'exécution **avec** l'opération de recadrage
    - ☞ t_{ws} : temps d'exécution **sans** l'opération de recadrage

$$C_s = \frac{t_{wos} - t_{ws}}{t_{wos}}$$

Filtre	*IPC*	C_s *(%)*
Filtre FIR réel (traitement par échantillon)	**7.5**	**47**
Filtre FIR réel (traitement par bloc)	**6**	**22**
Filtre FIR réel symétrique (échantillon)	**7.2**	**47**
Filtre FIR réel symétrique (bloc)	**7.4**	**35**
Filtre FIR complexe (échantillon)	**6.5**	**45**
Filtre FIR complexe (bloc)	**4.875**	**0**
Filtre IIR d'ordre 2	**2.8**	**18**



TMS320C62x : $IPC_{max} = 8$

Interaction génération de code - codage

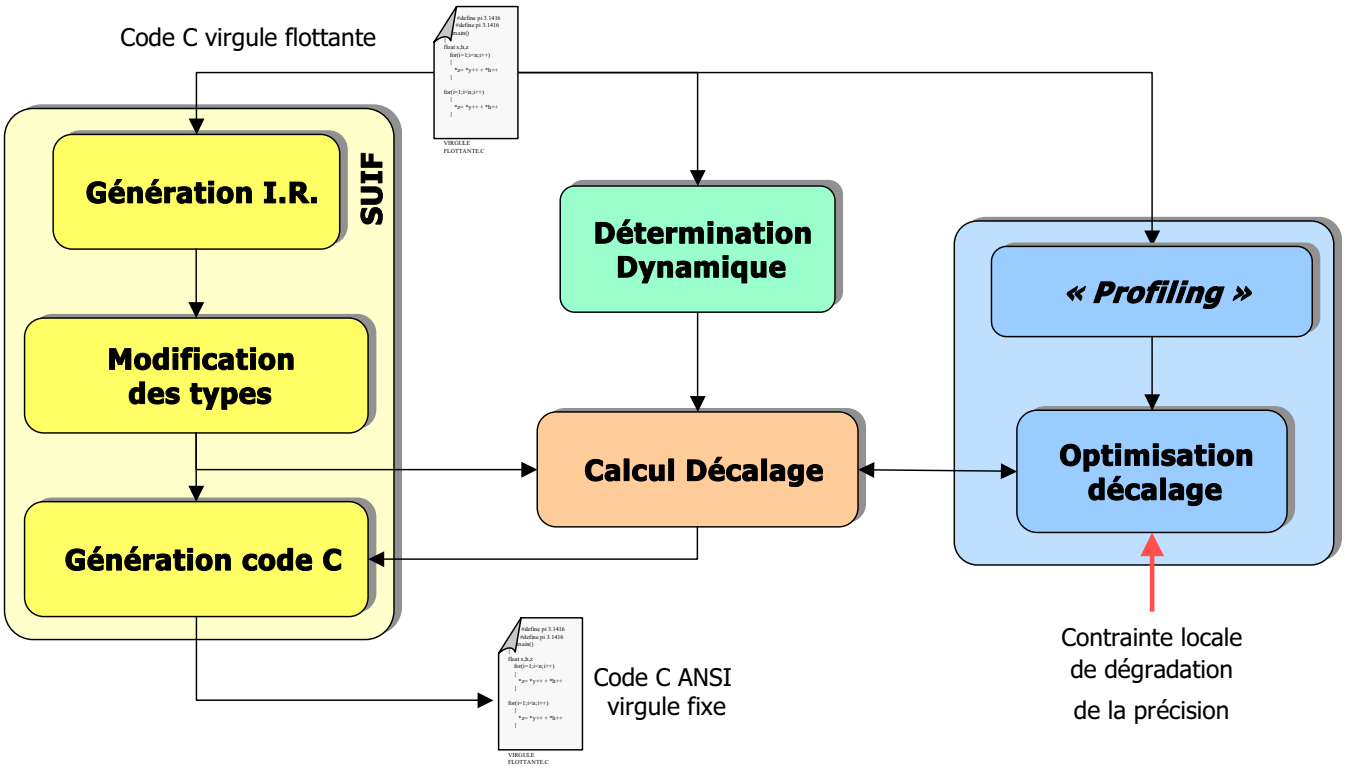
- **Sélection d'instructions**
  - Le choix de l'instruction est lié à la largeur des opérandes
    - ✓ Le type des données doit être déterminé avant la sélection d'instructions
- **Allocation et assignation de registres**
  - Renvoi de données intermédiaires en mémoire (*spilling*)
    - ✓ Optimisation de la largeur des données renvoyées en mémoire
      - ☞ Compromis précision - temps de transfert
- **Ordonnancement**
  - DSP avec parallélisme au niveau instruction : *le coût d'une opération de recadrage est lié à la manière dont les instructions sont ordonnancées* [Men02d]
    - ✓ Optimisation du placement des opérations de recadrage lors de l'ordonnancement des instructions

Objectifs

- **Implantation logicielle dans un DSP ou un ASIP**
  - Développement du code
    - ✓ Architecture figée
    - ✓ Architectures diverses
  - Objectif de l'implantation
    - ✓ Minimiser le temps d'exécution et la taille du code
- **Objectif du processus de conversion en virgule fixe**
  - Minimiser le temps d'exécution du code sous contrainte de précision
    - ✓ Optimiser la largeur des donnée : sélectionner les instructions (classiques, double-précision, SWP) permettant de minimiser le temps d 'exécution
    - ✓ Optimiser le placement des opérations de recadrage en vue de réduire le temps d'exécution

Méthode « *Autoscaler for C* »

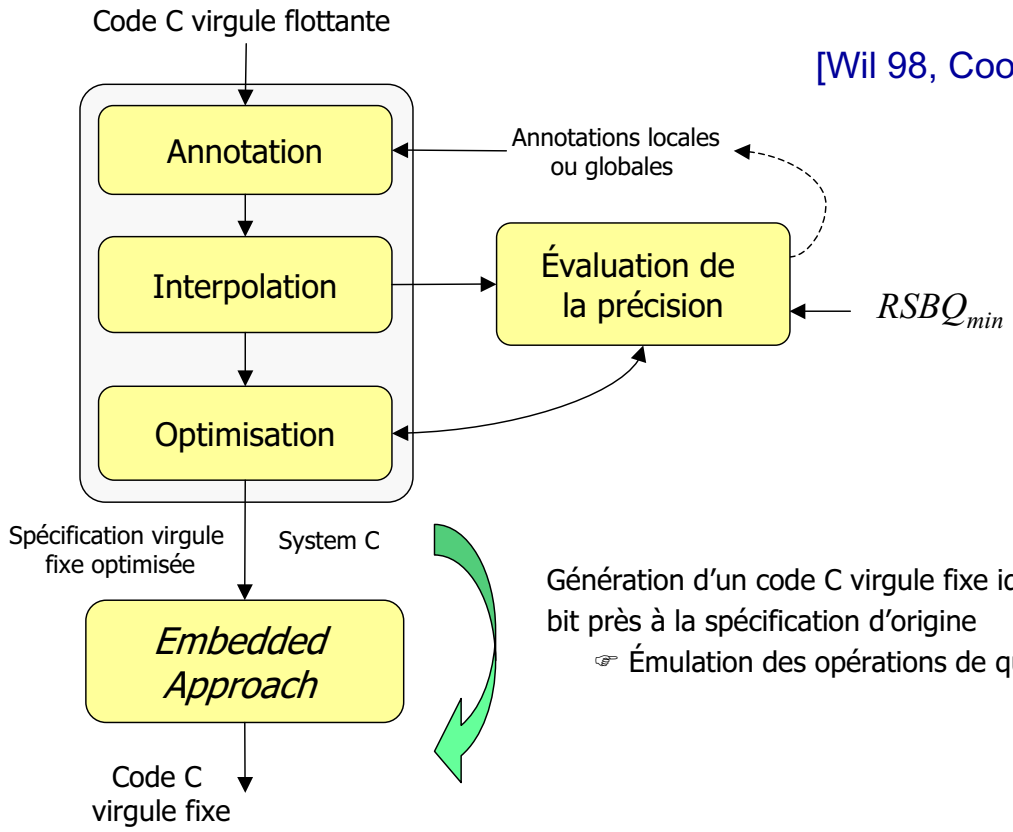
- **Université de Séoul** [Kum 00]



Les différentes phases de la méthode

- **Évaluation de la dynamique**
  - Méthode statistique
- **Détermination des recadrages**
  - Pour chaque expression du code, résolution d'un système d'équations linéaires
- **Optimisation des recadrages**
  - Minimisation de la fonction de coût des recadrages
    - ✓ Réduction du coût : égalisation du format de certaines données
      - ☞ Définition d'une dégradation maximale autorisée de la précision de chaque donnée
    - ✓ Le temps d'exécution d'un recadrage t_r de d_r bits est défini a priori en fonction du type de registre à décalage :
      - ☞ Présence d'un registre à décalage en barillet : t_r = 1 cycle
      - ☞ Absence de registre à décalage en barillet : t_r = d_r cycles

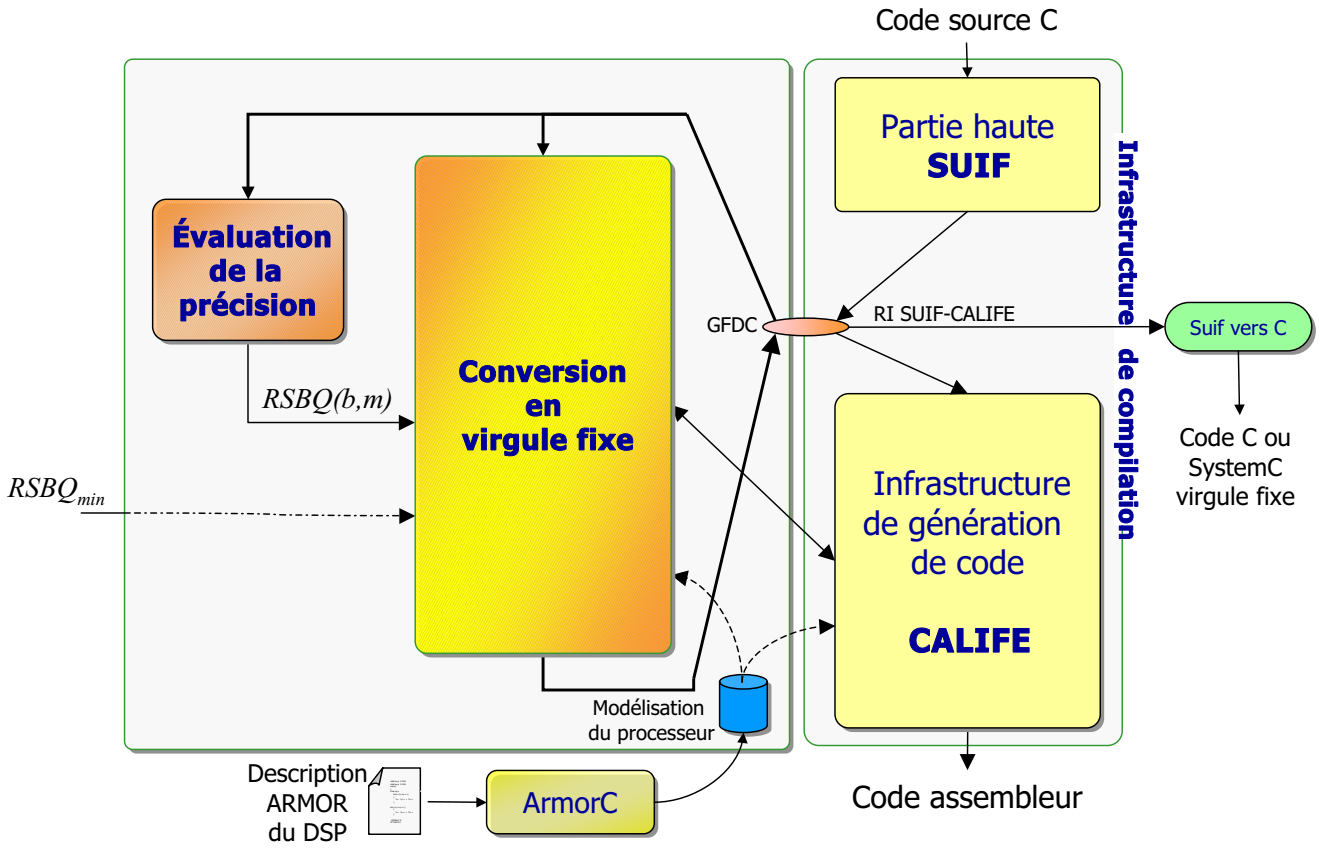
Méthode FRIDGE



Objectifs de nos travaux de recherche

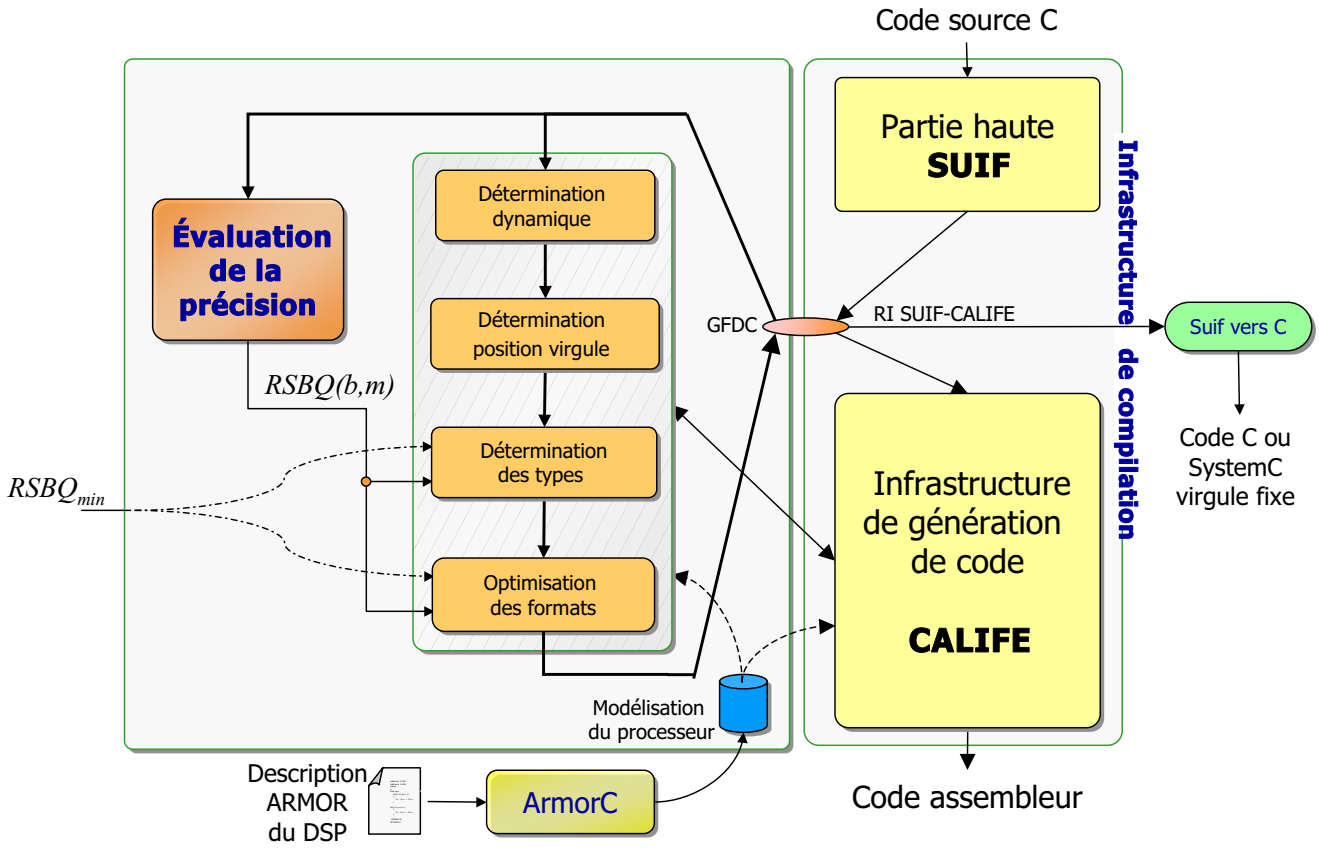
- *Développement d'une méthodologie de compilation d'algorithmes spécifiés en virgule flottante pour les processeurs programmables en virgule fixe sous contrainte de précision globale*
  - Évaluation analytique de la précision du système (RSBQ)
  - Obtention d'une spécification en virgule fixe optimisée
    - ✓ Prise en compte de l'architecture du processeur cible
    - ✓ Couplage des processus de génération de code et de conversion en virgule fixe
  - Optimisation de l'implantation sous contrainte de précision
    - ✓ Minimisation du temps d'exécution du code généré
      - ☞ **Choix du type des données**
      - ☞ **Placement des opérations de recadrage**

Synoptique de la méthodologie



[Men02c, Men02e]

Synoptique de la méthodologie



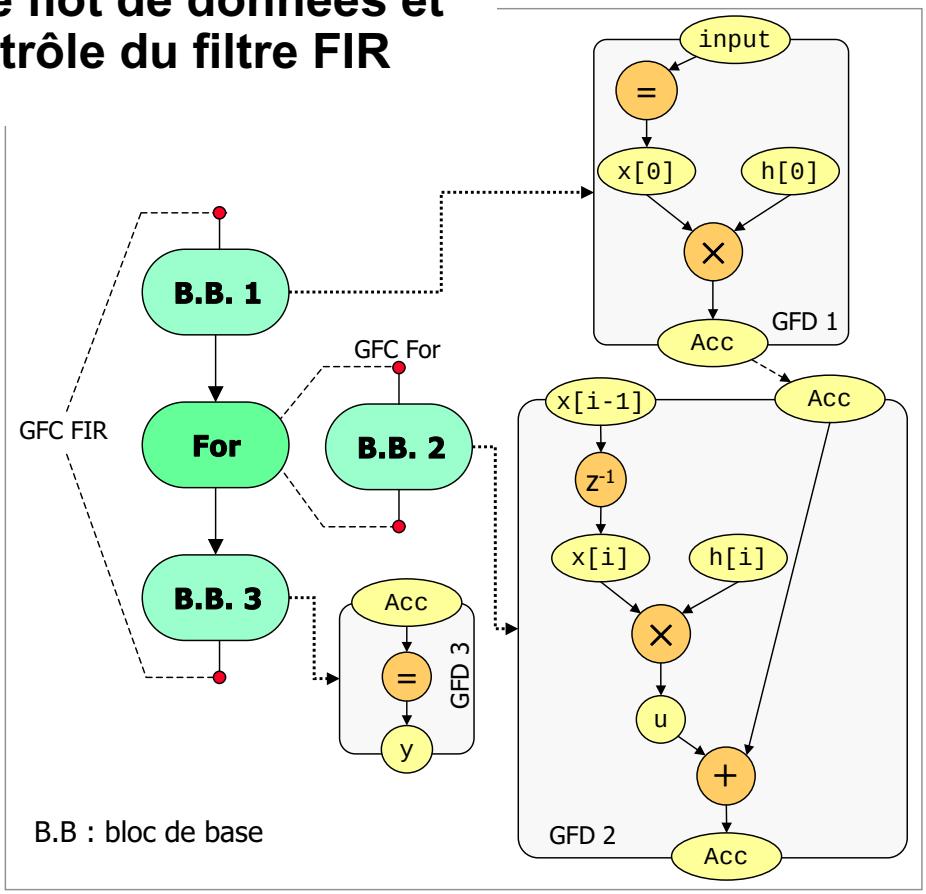
Exemple filtre FIR

- **Code C du filtre FIR**

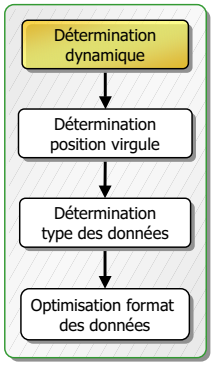
```
float h[32] = {-0.029710,                    -0.0497370,  ..., 897773053,
0.98000000,   0.8977730, ...,                -0.04973706,  -0.0297105, 0.0};

void main()
{
float x[32];
float input;
float yn;
float acc;
int i;

  x[0] = input;

  acc = x[0]*h[0];

  for(i=31; i>0; i--)
  {
    acc = acc + x[i]*h[i];

     x[i] = x[i-1];

  }
  yn = acc;
```


Exemple filtre FIR

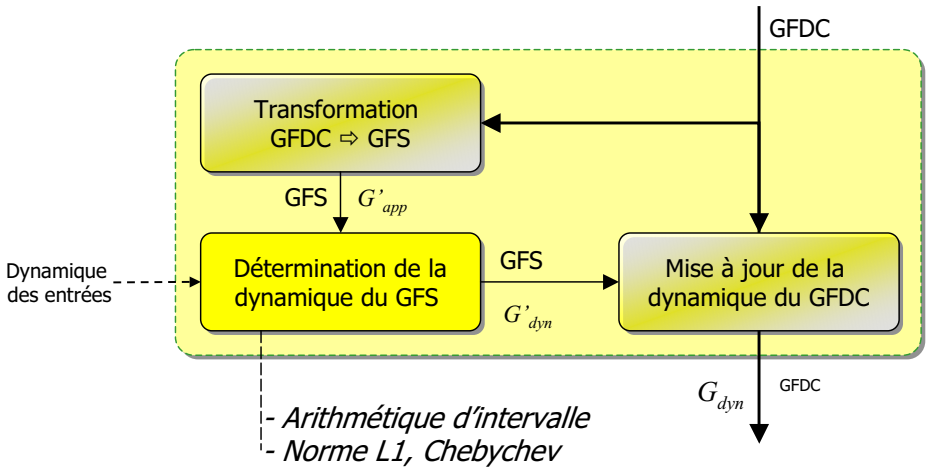
- **Graphe flot de données et de contrôle du filtre FIR**



Détermination de la dynamique

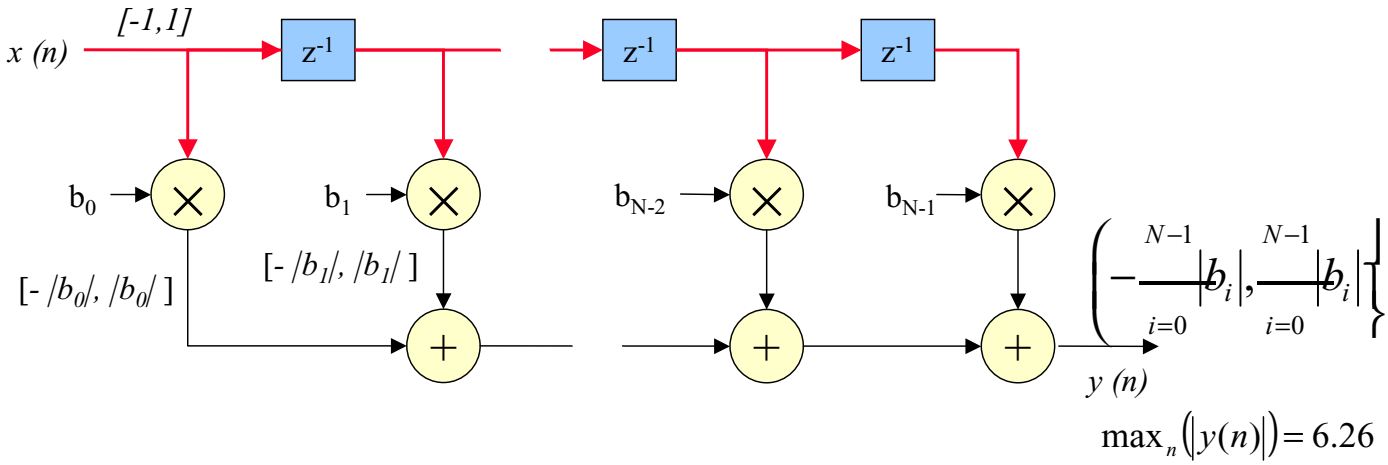


- **Détermination du domaine de définition des données**
- **Technique implantée**



Exemple filtre FIR

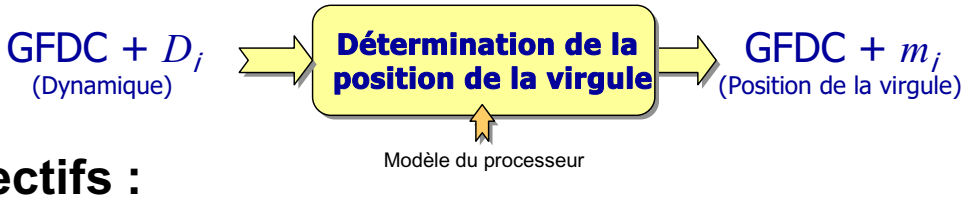
- **Propagation de la dynamique des entrées au sein du graphe flot de signal représentant l'application**

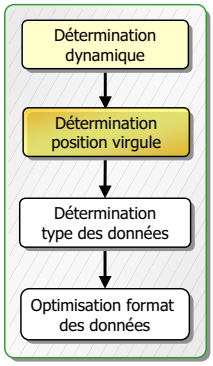


- **Normes L1**

$$\max_n(|y(n)|) = \max_n(|x(n)|) \cdot \sum_{m=-\infty}^{\infty}|h(m)| = \sum_{m=0}^{N-1}|b_i| = 6.26$$

Détermination de la position de la virgule





- **Objectifs :**
  - Détermination de la position de la virgule (m_i)
  - Insertion des opérations de recadrage
- **Technique :**
  - Propagation de la virgule par parcours des graphes flots de données (GFD)
    - ✓ Nécessité d'éliminer les circuits présents au sein des GFD
    - ✓ Définition de règles pour les données et les opérations

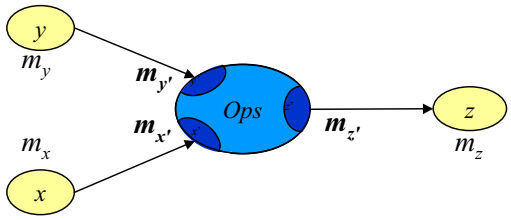
☞ Prise en compte de la présence de bits de garde

Détermination de la position de la virgule

- Données

$$m_x = \left\lceil \log_2 \left(\max \left(\left| x_{\min} \right|, \left| x_{\max} \right| \right) \right) \right\rceil$$

- Opérations :



✓ Multiplication

$$m_{z'} = m_x + m_y + 1$$

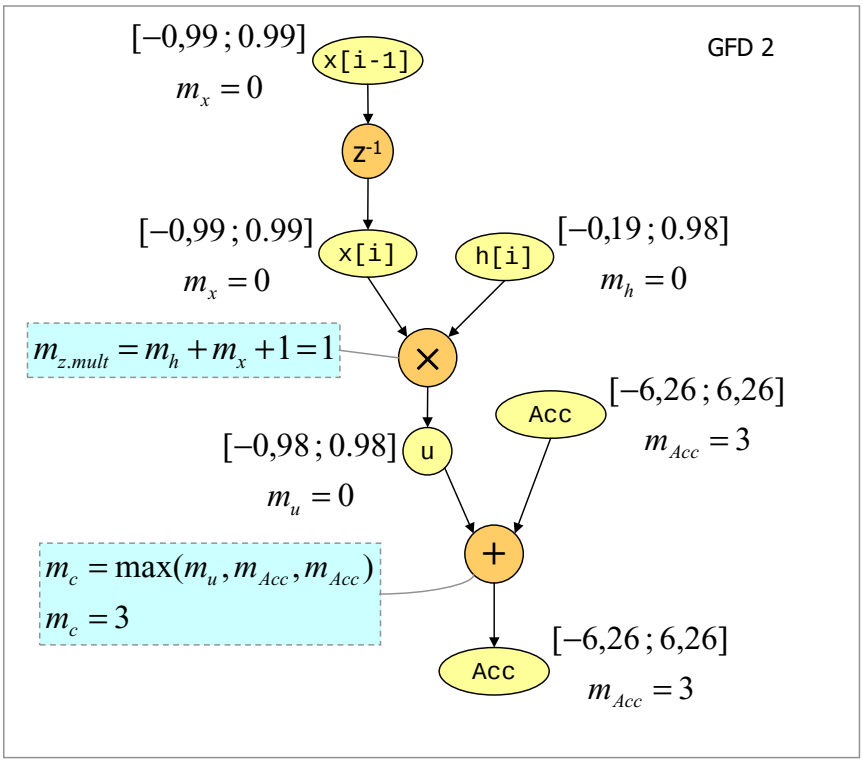
✓ Addition **(sans bits de garde)**

☞ Définition d'un format commun

$$m_c = \max(m_x, m_y, m_z)$$

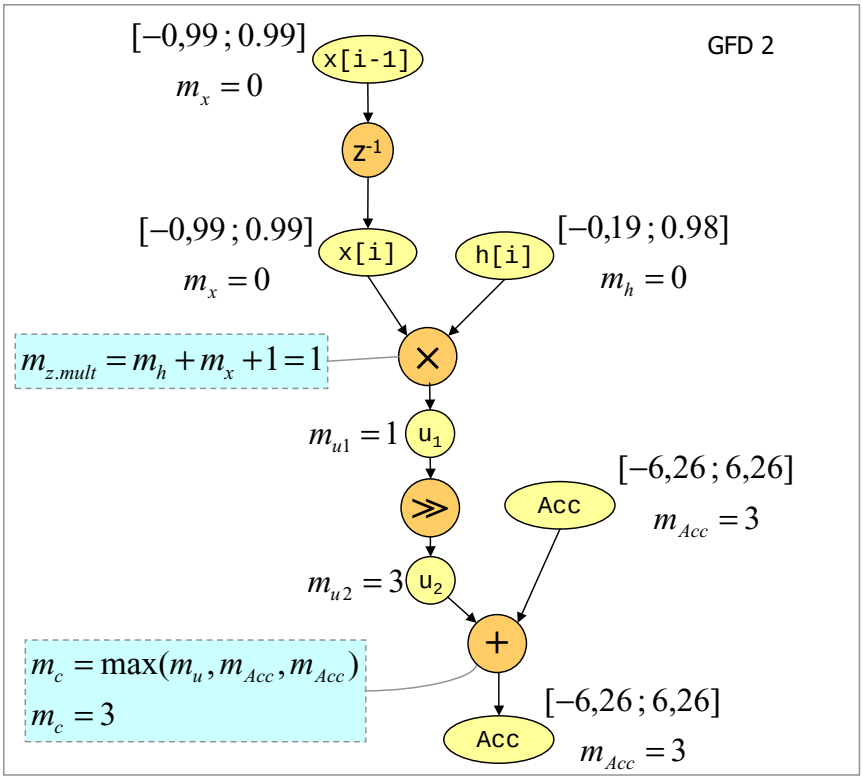
Exemple filtre FIR

- **Détermination de la position de la virgule au sein du GFD représentant le cœur de la boucle FOR**

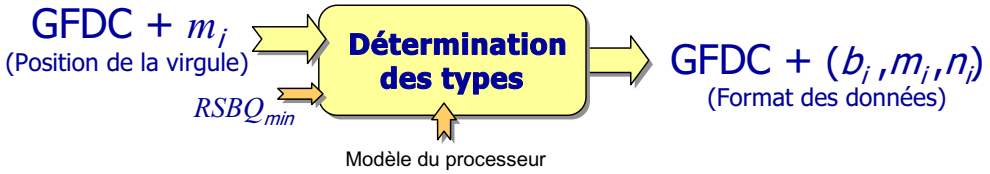


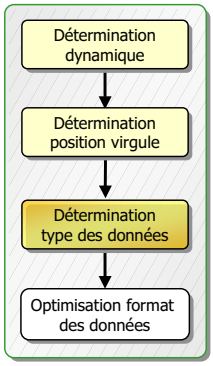
Exemple filtre FIR

- **Insertion des opérations de recadrage au sein du GFD représentant le cœur de la boucle FOR**



Détermination du type des données





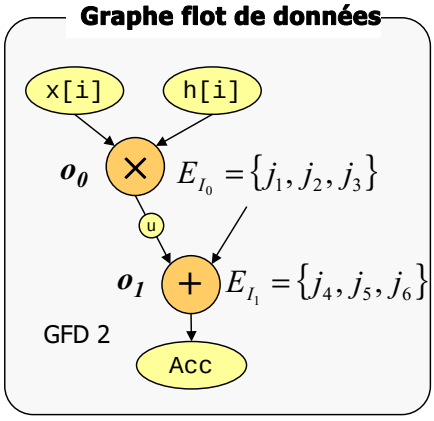
- **Détermination de la largeur des données**
  - Prise en compte des différents types manipulés par le DSP :
    - ☞ Instructions classiques
    - ☞ Instructions double précision
    - ☞ Instructions SWP
- **Sélection de la suite d'instructions permettant de**
  - Minimiser le temps d'exécution global
  - Satisfaire la contrainte de précision ($RSBQ_{min}$)

$$\min_{\vec{b}\in B}\left(T(\vec{b})\right) \quad \text{tel que} \quad RSBQ(\vec{b}) \geq RSBQ_{\min}$$

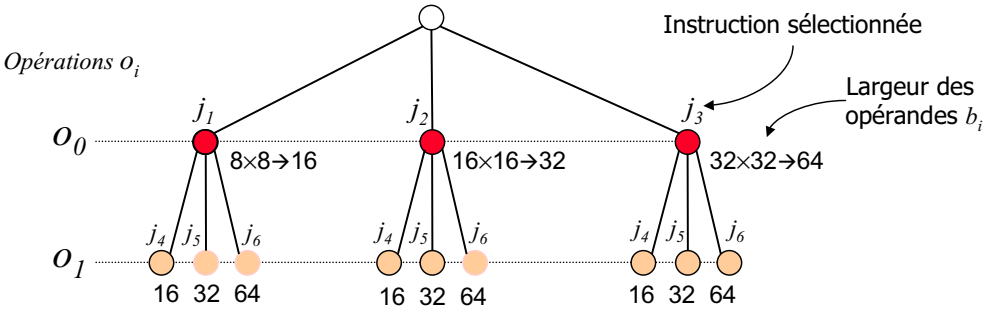
Exemple opération MAC

- **Jeu d'instructions du processeur**

Instruction j_k	Fonction γ_k	Temps d'execution t_k	Largeur des opérandes E/S b_{e1}	b_{e2}	b_s
j_1	MULT	0.25	8	8	16
j_2	MULT	0.5	16	16	32
J_3	MULT	1	32	32	64
j_4	ADD	0.25	16	16	16
j_5	ADD	0.5	32	32	32
j_6	ADD	1	64	64	64

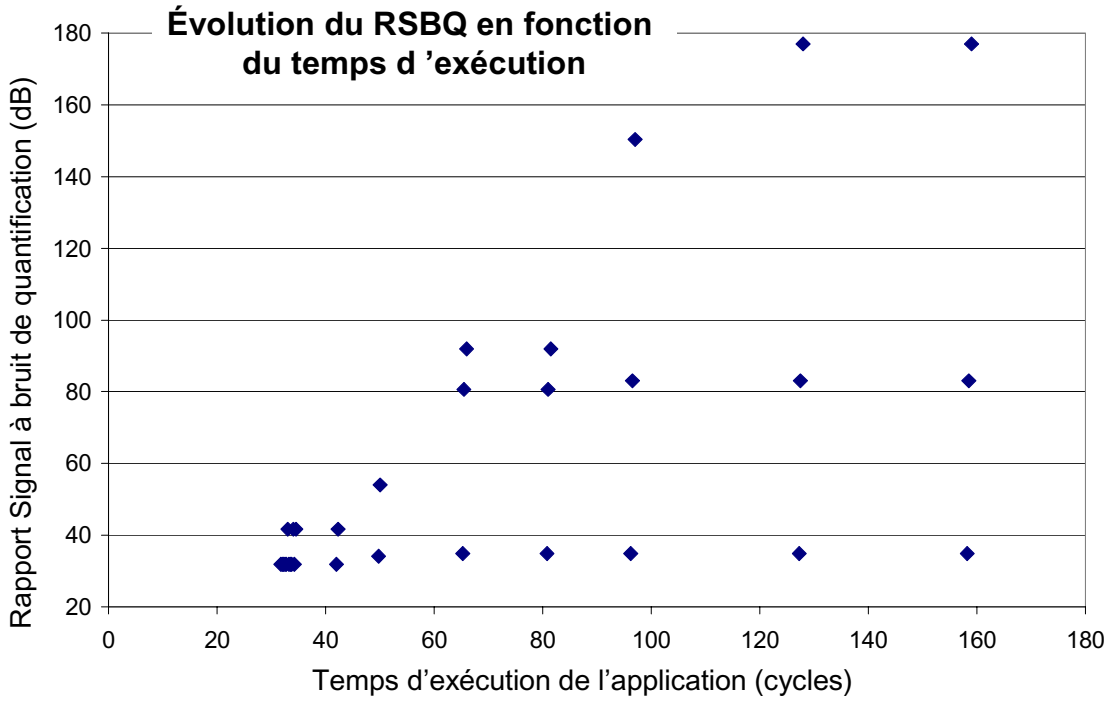


- **Modélisation des solutions sous forme d'arbre**

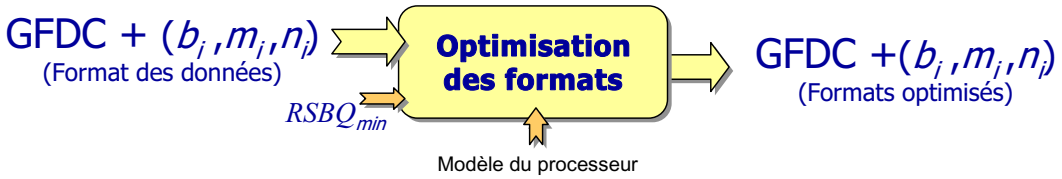


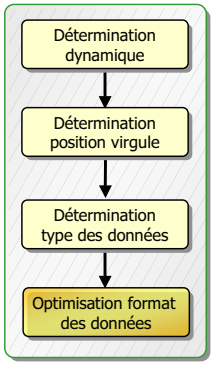
Exemple filtre FIR

- **Espace de recherche**
  - 6 variables (opérations),
  - 3 valeurs possibles par variable



Optimisation du format des données

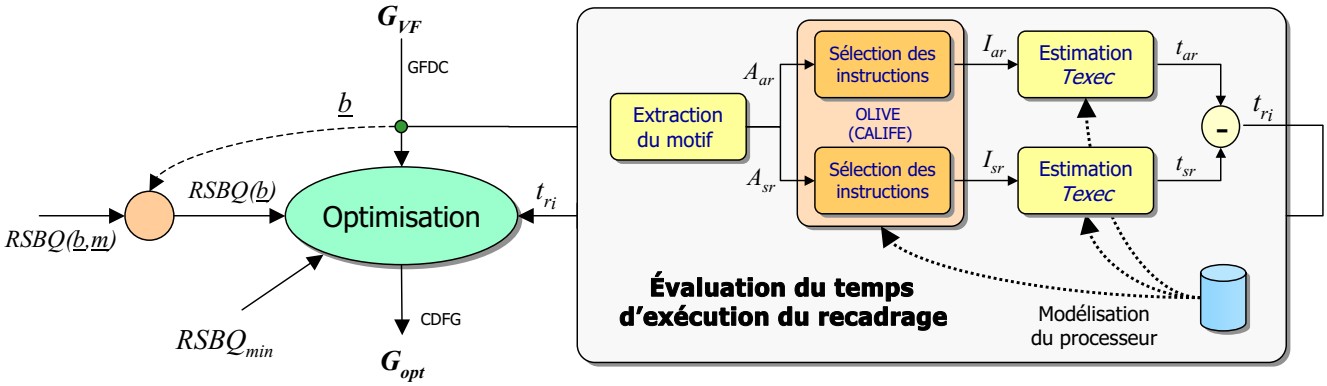




- **Détermination de la position optimale des opérations de recadrage**
  - Déplacement des opérations de recadrage pour réduire le temps d'exécution du code
    - ✓ Minimisation du temps d'exécution tant que la contrainte de précision est respectée
- **Deux types d'approche**
  - DSP sans parallélisme au niveau instruction (ILP)
    - ✓ Optimisation avant le processus de génération de code
  - DSP avec parallélisme au niveau instruction
    - ✓ Optimisation en parallèle à la phase d'ordonnancement
      - ☞ Estimation du coût probable des opérations de recadrage

DSP sans ILP

- **Coût d'une opération de recadrage r_i :** $c_{r_i} = n_i \times t_{r_i}$
- **Processus d'optimisation itératif**
  - Déplacement du recadrage r_0 ayant le coût maximal
  - Évaluation de la précision (RSBQ)
  - Validation du déplacement :
    - ✓ si $RSBQ(m) < RSBQ_{min}$ alors
      - ☞ r_0 est replacée à sa position optimale et n'est plus déplacée



Exemple filtre FIR

- **Alternatives pour le déplacement des opérations de recadrage :**
  - Vers l'entrée du filtre
  - Vers les coefficients du filtre
- **Déplacement vers l'entrée du filtre**
  - Avant optimisation
    - ☞ Temps d'exécution des recadrages : 32 cycles
    - ☞ RSBQ : 80,6 dB
  - Après optimisation
    - ☞ Temps d'exécution des recadrages : 1 cycle
    - ☞ RSBQ : 75.57 dB

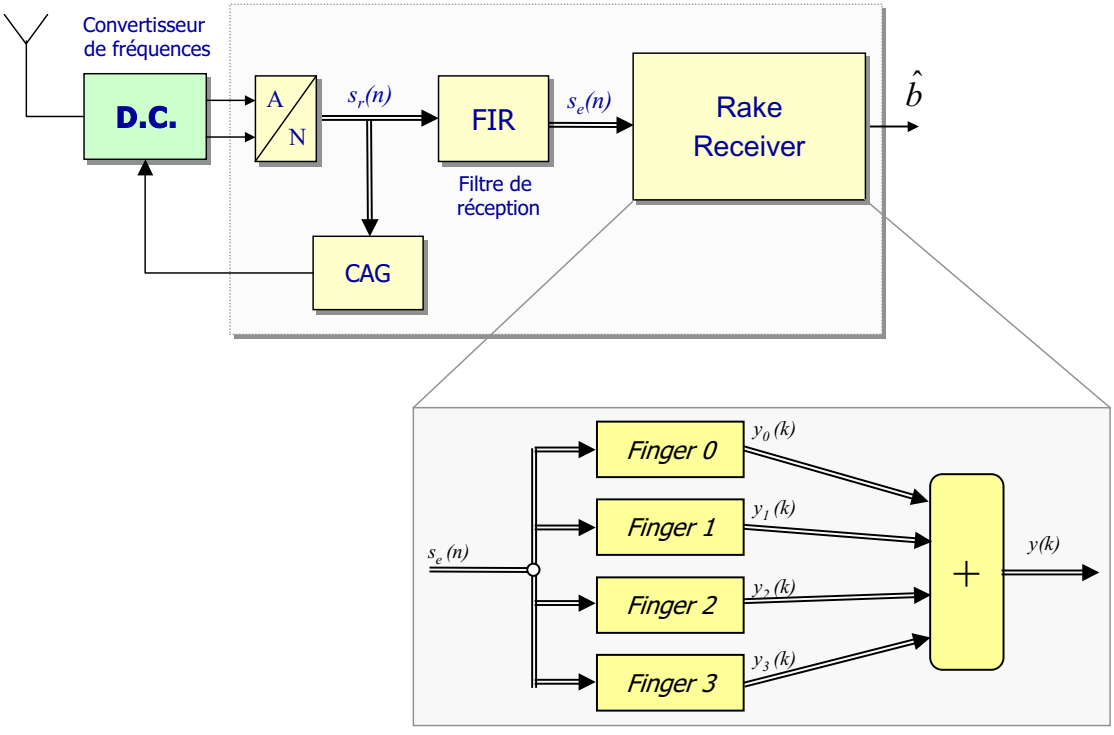
Exemple filtre FIR

- **Code C du filtre FIR après déplacement de l'opération de recadrage vers l'entrée**

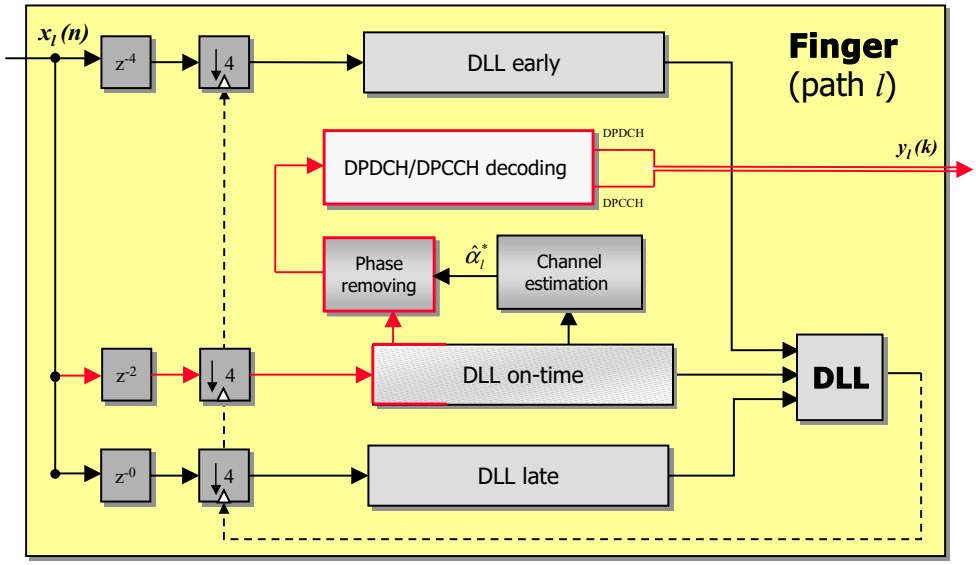
```
short h[32]={-973,..., 29418, 32112, 29418,...-973};
short input;

void main()
{
short x[32];
short y;
int acc;
int i;

    x[0] = input >> 2;
    acc = x[0] * h [0];

    for (i=31; i>0; i--)
    {
        acc = acc + x[i] * h[i];
        x[i] = x[i - 1];
    }
    y = (int)(acc);
}
```


Systèmes de radio de 3ème génération

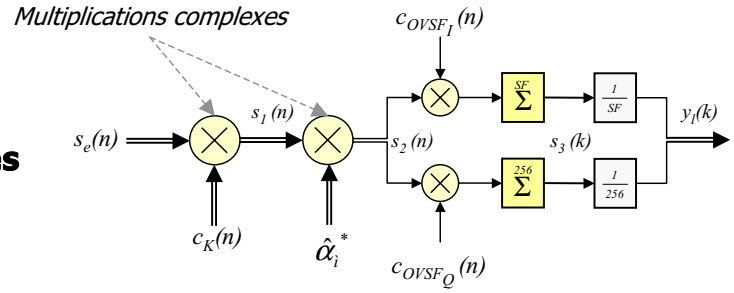
- **Récepteur pour les systèmes de radio-communication de 3ème génération**



Synoptique d'un *finger*



Décodage des données



Spécifications

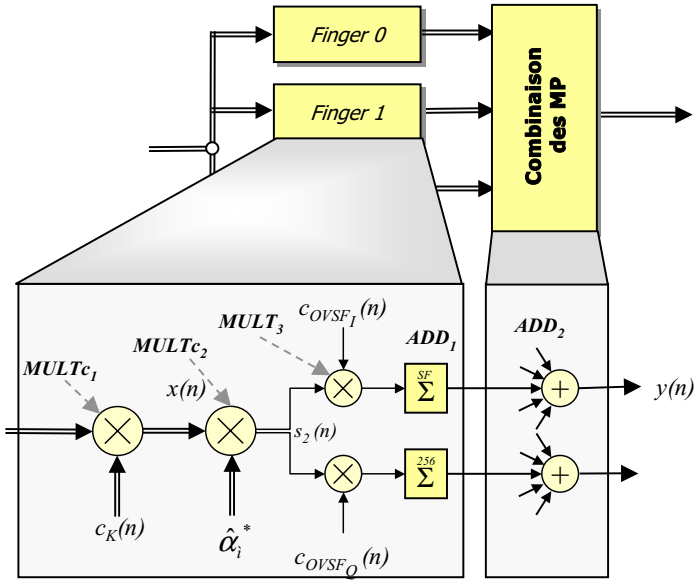
- **Rake Receiver :**
  - 4 fingers
  - Décodage des données
- **Code C de la partie décodage des données d'un *finger***

```
for (i = 0; i < SF; i++)
{
    y0I[i] = x0I[i] * CksI[i] - x0Q[i] * CksQ[i];
    y0Q[i] = x0I[i] * CksQ[i] + x0Q[i] * CksI[i];

    Acc0I = Acc0I + ( y0I[i]*a0I[0] -   y0Q[i]*a0Q[0] )* Co[i];
    Acc0Q = Acc0Q + ( y0I[i]*a0Q[0] +   y0Q[i]*a0I[0] )* Co[i];
}
```

- **Évaluation de la précision :**
  - Graphe flot de signal de l'application : 480 opérations
  - 280 sources de bruit potentielles

Dynamique - position de la virgule

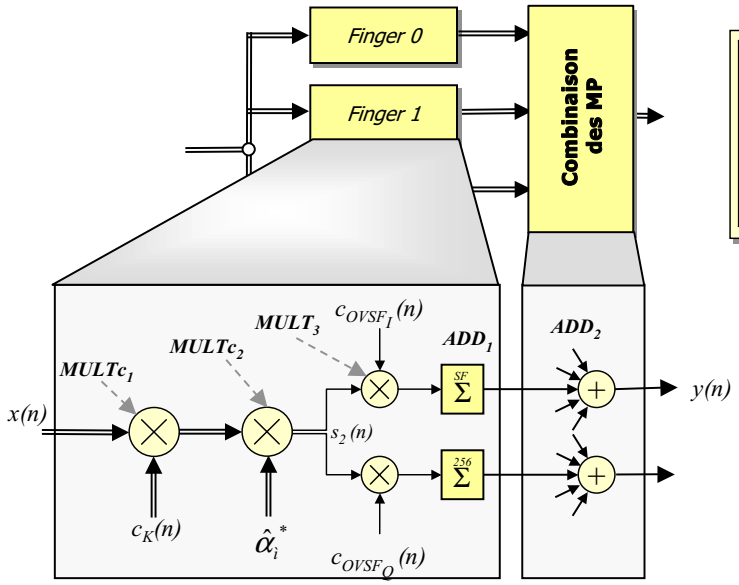
- **Facteur d'étalement SF = 8**



	Dynamique et position de la virgule								
		$MULTc_1$		$MULTc_2$		$MULT_3$	ADD_1	ADD_2	
	x	*mult*	*add*	*mult*	*add*	*mult*	*add*	*Add*	*y*
D_x	0.99	0.99	1.98	1.96	3.92	3.92	31.36	125.45	125.45
m_x	0	2	1	2	2	4	5	7	7

Largeur des données

- **Spécification de la largeur des données pour une contrainte de RSBQ de 12,5 dB**

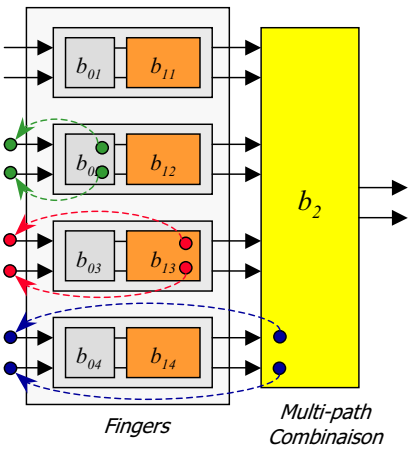


T_{exec} (cycles)	
Instructions classiques	Instructions SWP
3133 = ./ 2.64 ➔	*1185*

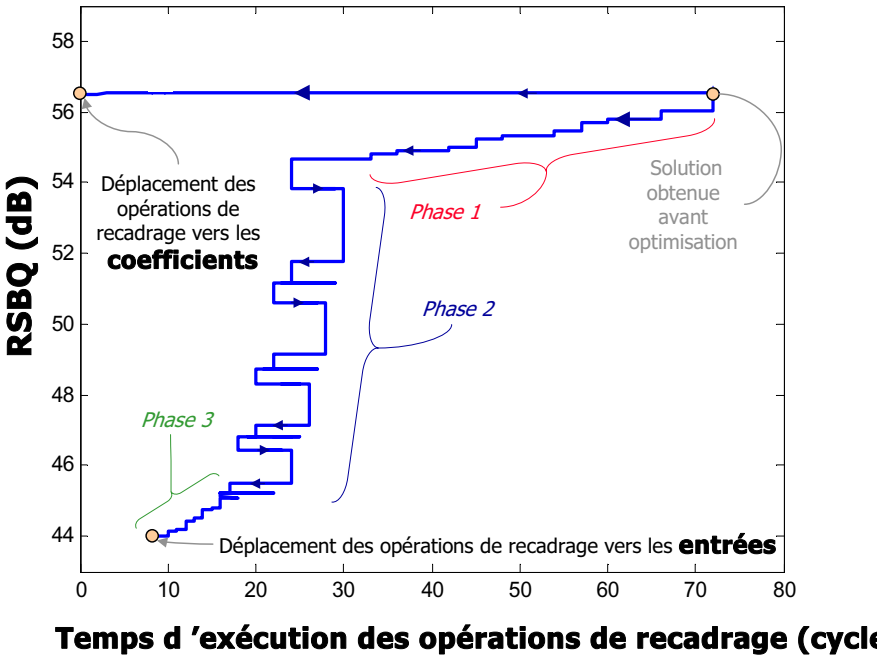
Types des opérations et données (bits)								
	$MULTc_1$		$MULTc_2$		$MULT_3$	ADD_1	ADD_2	
x	*mult*	*add*	*mult*	*add*	*mult*	*add*	*add*	*y*
8	8×8→16	16+16→16	16×16→32	16+16→16	16×16→32	16+16→16	16+16→16	16

Optimisation des recadrages

- **Implantation du rake receiver au sein d'un DSP 16 bits**
  - Structure MAC classique sans bit de garde
  - Présence d'un registre en barillet



Structure des blocs de base de l'application



Conclusion

- **Évaluation de la dynamique**
  - Méthodes statistiques
    - ✓ Estimation précise mais pas ne garantissant pas l'absence de débordement
  - Méthodes analytiques
    - ✓ Estimation pessimiste garantissant l'absence de débordement
- **Évaluation de la précision**
  - Méthodes basées sur la simulation :
    - ✓ Augmentent le temps d'exécution du processus d'optimisation
      - ☞ Mise en œuvre d'heuristiques pour réduire l'espace de recherche
  - Méthodes analytiques :
    - ✓ Toutes les structures ne sont pas prises en comptes
      - ☞ Définition de nouvelles méthodes

Conclusion

- **Implantation matérielle**
  - Nécessité de couplage entre les processus de conversion en virgule fixe et de synthèse d'architecture
  - Mise en œuvre d'un algorithme efficace d'optimisation sous contrainte

- **Implantation logicielle**
  - Nécessité de prise en compte de l'architecture
  - Nécessité de couplage entre les processus de conversion en virgule fixe et de génération de code

✓ Optimisation du format sous contrainte de précision globale

☞ Relation entre la contrainte de précision et les critères de qualité de l'application

Bibliographie (1)

[Aam00] T. Aamodt, Floating-point To Fixed-point Compilation and Embedded Architectural Support, Master Thesis, University of Toronto, January 2001

[Cac02] D. Cachera, T. Risset *Advaces in Bit Width Selection Methodology*. Proceedinngs of the IEEE Internationnal Conference on Application-Specific Systems, Architectures, and Processirs (ASAP 02), Jul 02.

[Coo01] M. Coors, H. Keding, O. Luthje and H. Meyr, *Integer Code Generation For the TI TMS320C62x*, ICASSP-01, May 2001, Sate Lake City, US.

[Con99] G. Constantinides, P. Cheung, W Luk, *Truncation noise in fixed-point SFG*, IEE Electronics Letters, 35(23), November 1999.

[Con01] G. Constantinides, P. Cheung, W Luk, Heuristic Datapath Allocation for Multiple Wordlength Systems, DATE 2001, Mars 2001.

[Kea96] R. Kearfott, *Interval Computations: Introduction, Uses, and Resources*, Euromath Bulletin, vol 2 (1), 1996, p95-112.

[Ked01] H. Keding , M. Coors, O. Luthje, H. Meyr, *Fast Bit-True Simulation*, Design Automation Conference 2001, (DAC-01), Jun 01, Las Vegas, US.

[Ked98a] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: A Fixed-Point Design And Simulation Environment. Design, Automation and Test in Europe 1998 (DATE-98), Mar 98.

[Ked98b] H. Keding and F. Hurtgen and M. Willems and M. Coors, *Transformation of Floating-Point into Fixed-Point Algorithms by Interpolation Applying a Statistical Approach*, 9th International Conference on Signal Processing Applications and Technology, ICSPAT'98, 98.

[Kim98] S. Kim, K. Kum, S. Wonyong. *Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs,* IEEE Transactions on Circuits and Systems II, 45(11), Nov 98.

[Kum00] K. Kum, J.Y. Kang and W.Y. Sung, *AUTOSCALER for C: An optimizing floating-point to integer C program converter for fixed-point DSP* ,IEEE Transactions on Circuits and Systems II, pp 840-848, September 2000.

[Kum01] K. Kum, and W.Y. Sung, *Combined Word-length Optimization and High-level Synthesis of Digital Signal Processing Systems*,IEEE Transactions on Computer Aided Design of circuits and Systems II, pp 921-930, 20(8), August 2000.

Bibliographie (2)

[Liu71] B. Liu. Effect of Finite Word Length on the Accuracy of Digital Filters - A Review (Invited Paper). IEEE Transaction on Circuit Theory, 18(6), Nov 71.

[Mar01] E. Martin, C. Nouet, JM. Toureilles. Conception optimisée d'architectures en précision finie pour les applications de traitement du signal.Traitement du Signal 2001, Vol 18 (1), 2001.

[Men01] D. Menard, O. Sentieys, *Influence du modèle de l'architecture des DSPs virgule fixe sur la précision des calculs*, Dix huitième colloque GRETSI sur le traitement du signal et des images, Toulouse, Sep 01.

[Men02a] D. Menard and O. Sentieys. *Automatic Evaluation of the Accuracy of Fixed-point Algorithms*, IEEE/ACM Conference on Design, Automation and Test in Europe 2002 (DATE-02), Paris, Mar 02.

[Men02b] D. Menard and O. Sentieys. *A methodology for evaluating the precision of fixed-point systems*. International Conference on Acoustics, Speech and Signal Processing 2002 (ICASSP 2002), Orlando, May 02.

[Men02c] D. Menard, T. Saidi, D. Chillet, O. Sentieys. *Implantation d'algorithmes spécifiés en virgule flottante dans les DSP virgule fixe*. 8ème Symposium en Architectures nouvelles de machines (SYMPA), Hammamet, Tunisie, Apr 02.

[Men02d] D. Menard, P. Quemerais, O. Sentieys. *Influence of fixed-point DSP architecture on computation accuracy*. XI European Signal Processing Conference (EUSIPCO 2002), Toulouse, Sep 02.

[Men02e] D. Menard, D. Chillet, F.Charot, O. Sentieys. *Automatic Floating-point to Fixed-point Conversion for DSP Code Generation*. ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems 2002 (CASES 2002), Grenoble, Oct 02.

[Men02f] D. Menard, T. Saidi, D. Chillet, O. Sentieys. *Implantation d'algorithmes spécifiés en virgule flottante dans les DSP virgule fixe*. Sélectionné pour numéro spécial de TSI Architectures des systèmes embarqués.

[Men03a] D. Menard, M. Guitton, S. Pillement, O. Sentieys. Design and Implementation of WCDMA Platforms Challenges and Trade-offs. Accepted for the International Signal Processing Conference, Dallas, April 03.

[Par87] T.W. Parks and C.S. Burrus, *Digital Filter Design*, 1987, Jhon Willey and Sons Inc

Bibliographie (3)

[Sun95] W. Sung, K. Kum. *Simulation-Based Word-Length Optimization Method for Fixed-Point Digital Signal Processing Systems.* IEEE Transactions on Signal Processing, 43(12), Dec. 1995.

[Tou99] J. Toureilles. *Conception d'architectures pour le traitement du signal en précision finie*. PhD thesis, Université de Rennes I, Jan 99.

[Wil97] M. Willems and V. Bursgens and H. Meyr, *FRIDGE: Floating-Point Programming of Fixed-Point Digital Signal Processors* Internartionnal Conférence On Signal Processing Applications and Technology, (ICSPAT'97), 1997.