# Chapitre VIII- La validation fonctionnelle des services.

**Objectif du chapitre :**

- Présenter le mécanisme utilisé pour valider fonctionnellement les services et les schémas d'instructions.

Le chapitre VII montre comment SEP valide l'utilisation des ressources par les services de modules et les schémas d'instructions. Chaque service est construit par agencement d'unités fonctionnelles pour réaliser une fonction plus complexe. Un service détermine complètement les chemins de contrôle et les instants d'exécution des services.

La validation fonctionnelle est alors possible en comparant l'expression algébrique de la fonction réalisée avec l'expression algébrique de la fonction souhaitée.

La manipulation d'expressions algébriques par un ordinateur est complexe, en particulier, la comparaison de deux expressions algébriques peut être très coûteuse car les formes normales sont parfois difficiles à construire. C'est pourquoi nous nous proposons d'utiliser des graphes de décision dont le domaine des valeurs est l'ensemble des entiers relatifs Z pour représenter les expressions algébriques : les Word-Level Decision Diagram (WLDD).

La section 1- introduit les WLDD et leurs particularités qui permettent une manipulation adaptée aux traitements informatisés. La section 2- illustre leur utilisation sur l'exemple de l'unité de calcul présenté au chapitre VII.

## *1- Les WLDD.*

Les graphes de décisions binaires (BDD) ont été introduits en 1978 par Akers pour représenter de façon compacte les fonctions booléennes, c'est-à-dire les fonctions de $B^n \rightarrow B$. Puis en 1986, Bryant propose les OBDD[1] une représentation canonique des BDD ainsi que des algorithmes pour calculer efficacement les opérations booléennes sur ces structures de données.

Depuis, les BDD ont été adaptés à la représentation des fonctions de $B^n \rightarrow Z$. Cette classe de graphes s'appelle WLDD, il en existe plusieurs formes qui sont plus ou moins adaptées suivant les applications et les systèmes étudiés.

La forme qui nous intéresse est la forme *Kronecker Multiplicative Binary Moment Diagram* (*K*BMD*) **[DBR96]**. En effet, il existe des algorithmes efficaces pour le calcul de la plupart des opérations que nous utilisons dans SEP, de plus, elle permet de représenter efficacement la plupart des commandes utiles pour la description d'architectures matérielles **[HoD99]**.

### 1-1. Définitions.

graphe

Un graphe est un ensemble de sommets S reliés par des arêtes $\alpha : SxS \rightarrow B$. Soient deux sommets *s1* et *s2*, $\alpha(s1, s2) = vrai$ si il existe une arête entre *s1* et *s2*, $\alpha(s1, s2) = faux$ sinon. Si $\alpha$ n'est pas commutative, on dit que le graphe est orienté. Les arêtes sont alors appelées des arcs.

[1] Ordered Binary Decision Diagram.

Dans un graphe orienté si *α(s1, s2) = vrai* alors *s1* est le prédécesseur de *s2* et *s2* le successeur de *s1*.

Dans un graphe orienté, un chemin est une suite de sommets reliés par des arcs. Un circuit est un chemin qui contient plusieurs fois le même sommet. Un graphe est dit acyclique s'il ne contient aucun circuit. Un sommet sans successeur est appelé feuille, un sommet sans prédécesseur est appelé racine. La taille d'un graphe est son nombre de sommets, sa hauteur est la longueur du plus long chemin entre la racine et une feuille.

Le degré d'un sommet est une valeur entière qui représente son nombre de successeurs.

arbre

Les arbres sont des graphes orientés acycliques tels que :

- ils possèdent une et une seule racine ;
- tous les autres sommets sont atteints à partir de la racine par un chemin unique.

Les arbres binaires complets sont des arbres dont tous les sommets exceptés les feuilles ont un degré égal à 2.

Avec une définition récursive, les successeurs d'un sommet d'un arbre sont eux-mêmes des arbres appelés sous-arbres.

### 1-2. Introduction aux BDD.

Les BDD sont des graphes orientés sans circuit, basés sur la décomposition de Shannon des fonctions booléennes f.

<u>Décomposition de Shannon[2] :</u> Soit, $\mathbf{f} : B^n \rightarrow B$, $(x_0, x_1, \ldots, x_{n-1}) \mapsto v$

**f** peut se décomposer en 2 sous-expressions $f_i^1 = f(x_0, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ et $f_i^0 = f(x_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ telles que $f = \overline{x_i} \bullet f_i^0 \oplus x_i \bullet f_i^1$.

$f_i^1$ et $f_i^0$ sont des fonctions de $B^{n-1} \rightarrow B$.

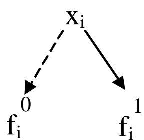


**Figure 1 – forme générale d'un BDD.**

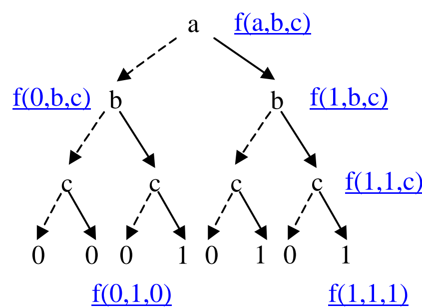


**Figure 2 – arbre de la fonction f=c • (b+a).**

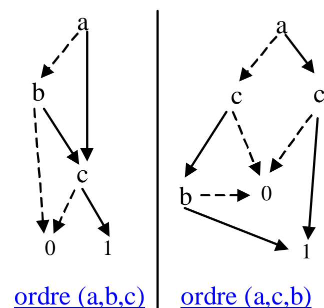


**Figure 3 – BDD de f=c • (b+a).**

Une fonction booléenne **f** peut alors être représentée par un arbre binaire complet dont la définition récursive est telle que :

- $x_0$ est la racine de l'arbre ;
- les deux successeurs de la racine sont les sous-arbres de représentation des fonctions $f_0^1$ et $f_0^0$ (cf. Figure 1).

[2] B dénote l'ensemble des booléens, 1 et 0 représentent respectivement les valeurs booléennes vrai et faux, l'opération *et* est représentée par la multiplication • tandis que l'opérateur *ou* est représenté par l'addition +.

Remarquons alors que les feuilles portent nécessairement la valeur 0 ou 1.

L'arbre ainsi défini d'une fonction de n variables a $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ sommets pour représenter $2^n$ valeurs possibles, une telle représentation n'est donc pas très intéressante (cf. Figure 2). Les sommets qui représentent des expressions identiques peuvent être regroupés pour former un graphe. La taille des graphes est très souvent[3] proportionnelle au nombre de variables (cf. Figure 3). En fait, la taille des graphes dépend très fortement de l'ordre dans lequel les variables sont considérées, le tout est de trouver l'ordre pour lequel la taille est minimale. L'ordre des variables est un point essentiel, des algorithmes de ré-ordonnancement dynamique basés sur des heuristiques existent.

### 1-3. Les extensions existantes.

Il existe plusieurs façons d'étendre ce modèle, nous ne parlerons que de celles utilisées pour la structure de données que nous utilisons.

La première extension possible consiste à considérer des fonctions de $B^n \rightarrow Z$. Le domaine de valeurs des graphes et sous-graphes est Z, nous n'utilisons alors plus des opérateurs booléens mais des opérateurs arithmétiques de multiplication, soustraction ou addition. La décomposition de Shannon peut alors être ré-écrite de la façon suivante :

$f = (1-x_i) \times f_i^0 + x_i \times f_i^1$ *(décomposition de Shannon notée S)*

Une deuxième extension consiste à changer la méthode de décomposition. Deux décompositions couramment utilisées comme alternative à la décomposition de Shannon sont les décompositions positive et négative de Davio par rapport à une variable $x_i$ :

$f = f_i^0 + x_i \times (f_i^1 - f_i^0) = f_i^G + x_i \times f_i^D$ *(décomposition positive de Davio notée pD).*

$f = f_i^1 + (1-x_i) \times (f_i^0 - f_i^1) = f_i^G + (1-x_i) \times f_i^D$ *(décomposition négative de Davio notée nD).*

L'arbre qui représente la fonction f de $B^n \rightarrow Z$ a comme racine un sommet portant la variable $x_0$, ses sous-arbres représentent des fonctions de $B^{n-1} \rightarrow Z$ indépendantes de $x_0$. Dans le cas de la décomposition positive (respectivement négative) de Davio, le premier sous-arbre – arc représenté en pointillé – représente la fonction $f_i^G = f_i^0$ (respectivement $f_i^G = f_i^1$) et le deuxième sous-arbre représente $f_i^D = f_i^1 - f_i^0$ (respectivement $f_i^D = f_i^0 - f_i^1$).

Il est également possible de mélanger dans le même graphe les trois types de décompositions, dans ce cas, les sommets sont marqués par le type de la décomposition utilisée { S, pD, nD }. La famille des WLDD ainsi obtenue est appelée *Binary Moment Diagram* (BMD).

Enfin, une dernière extension consiste à affecter une valeur aux arcs, il existe alors plusieurs familles de graphes. Les graphes EVBDD [4] mono-valués par une valeur $a \in Q$, la fonction représentée est $a +$ **f**, où **f** est l'expression représentée par le sous-arbre destination de l'arc valué par a. Les graphes *BMD [5] mono-valués par une valeur $m \in Q$, la fonction représentée est $m \times$ **f**, où **f** est l'expression représentée par le sous-arbre destination de l'arc valuée par m.

[3] Ceci est un résultat empirique admis.
[4] Edge-Valued Binary Decision Diagram.
[5] Multiplicative Binary Moment Diagram.

Lorsque les arcs sont valués par une paire (a,m) ∈ $Q^2$, la fonction représentée a comme expression a + m × **f**. L'arbre de la figure ci-contre représente la fonction **a** + **m** × **f**. Ces structures de BMD dont les arcs sont valués par cette paire sont appelées K*BMD.

(a,m)
**f**

K*BMD

Ainsi, les K*BMD permettent de représenter en un seul et même graphe un mot de bits alors que les BDD ne peuvent représenter que des bits. De plus, la taille des K*BMD qui représentent les opérations arithmétiques élémentaires sur les mots est linéaire par rapport à la taille des K*BMD des expressions sur lesquelles l'opération a été effectuée **[HoD99]**.

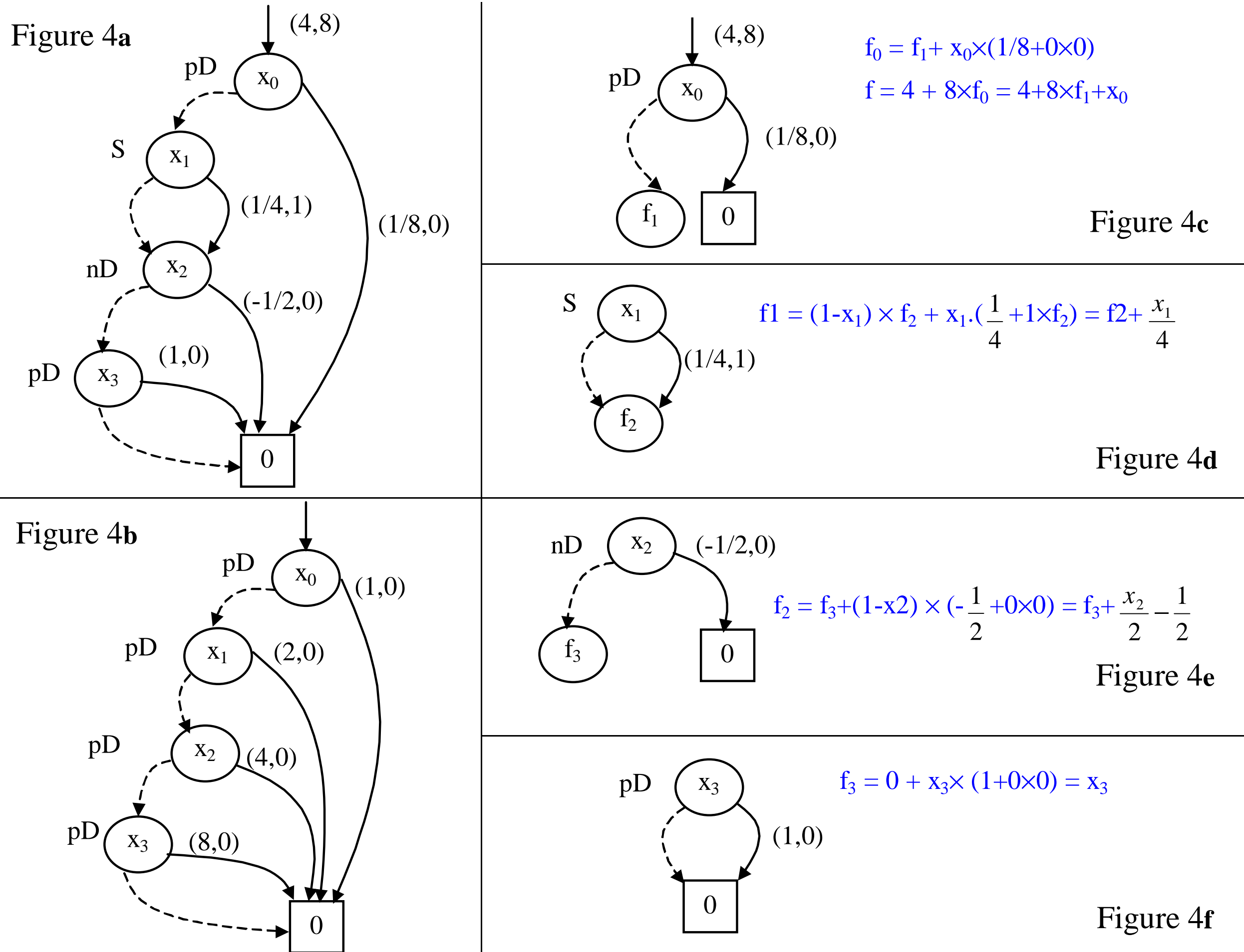


**Figure 4 – K*BMD de $x_0+2.x_1+4.x_2+8.x_3$.**

La Figure 4 présente le K*BMD de la fonction $x_0+2.x_1+4.x_2+8.x_3$ de $B^4 \rightarrow Z$, c'est-à-dire en fait un mot de 4 bits non signé. Le graphe représenté est un graphe mixte, il utilise les trois décompositions (Shannon, positive Davio, négative Davio) simultanément (cf. Figure 4*a*). La Figure 4*b* montre un graphe équivalent en utilisant seulement la décomposition positive de Davio. Par la suite, nous n'utiliserons que cette décomposition qui est plus facile à lire. Les décompositions mixtes permettent d'obtenir des graphes plus petits mais les résultats sont indépendants des décompositions utilisées ainsi que de l'ordre dans lequel ces décompositions ont été appliquées.

Les Figure 4*c*,*d*,*e* et *f* correspondent aux différentes composantes de l'arbre de la Figure 4*a*, elles illustrent la lecture d'un K*BMD. Les arcs dont la valeur n'est pas représentée sur les figures ont en réalité la valeur *(0,1)*, cette simplification permet d'améliorer la lisibilité du dessin.

## *2- Application aux services de l'unité de calcul.*

Il est facile de donner un équivalent en K*BMD pour un BDD. En effet, il suffit d'utiliser toujours une décomposition de Shannon, de valuer avec (0,1) – qui signifie 0+1×… – tous les arcs qui se terminent sur des sommets qui ne sont pas des feuilles, de valuer avec (0,1) tous les arcs qui se terminent sur la feuille 0 et de modifier les arcs se terminant sur la feuille 1 pour qu'ils se terminent sur la feuille 0 avec la valeur (1,0) – qui signifie 1+ 0×0 –. Cette correspondance n'est pas du tout évidente pour les autres formes de WLDD, c'est ce qui fait l'intérêt de cette forme plutôt que d'une des autres extensions possibles.

De plus, les K*BMD peuvent aisément représenter les mots de bits.

La section 2-1 montre comment les K*BMD peuvent être aisément utilisés dans SEP, la section 2-2 présente les différentes fonctions élémentaires à réaliser sur les K*BMD pour la modélisation des composants SEP standards. Enfin, la section 2-3 illustre ces propos sur un exemple.

### 2-1. Le principe.

Comme il a déjà été dit, il s'agit de comparer l'expression algébrique réalisée par un service à l'expression algébrique de la fonction que l'on souhaitait réaliser. Les WLDD ont une forme canonique qu'il est aisé d'obtenir, la comparaison de deux fonctions représentées par des WLDD est linéaire en la taille des graphes ; il suffit de comparer sommet par sommet et arc par arc. En revanche, les comparaisons d'expressions algébriques sont assez compliquées à réaliser.

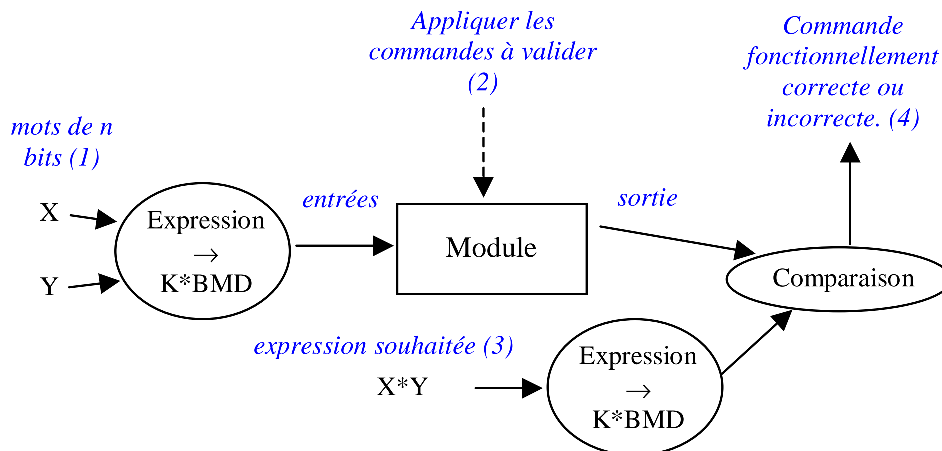


**Figure 5 – Principe du mécanisme de validation fonctionnelle dans SEP.**

La Figure 5 illustre le mécanisme de validation proposé. Pour chaque module et chaque commande à valider, il faudra réaliser les quatre étapes suivantes :

- construire les K*BMD à partir de mots de n bits, ces K*BMD constitueront les entrées du module ;
- appliquer la ou les commandes à valider, cela peut être un appel de service de module, une combinaison d'appels de services élémentaires, ou l'exécution d'un schéma d'instructions ;

- formuler l'expression de la fonction que l'on souhaite réaliser avec ces commandes et la convertir en K*BMD ;
- le résultat obtenu sur une sortie du module est alors un K*BMD, il suffit de le comparer avec le K*BMD de la fonction que l'on souhaitait réaliser pour savoir si les commandes effectuées par rapport à une architecture donnée permettent d'obtenir la fonction souhaitée.

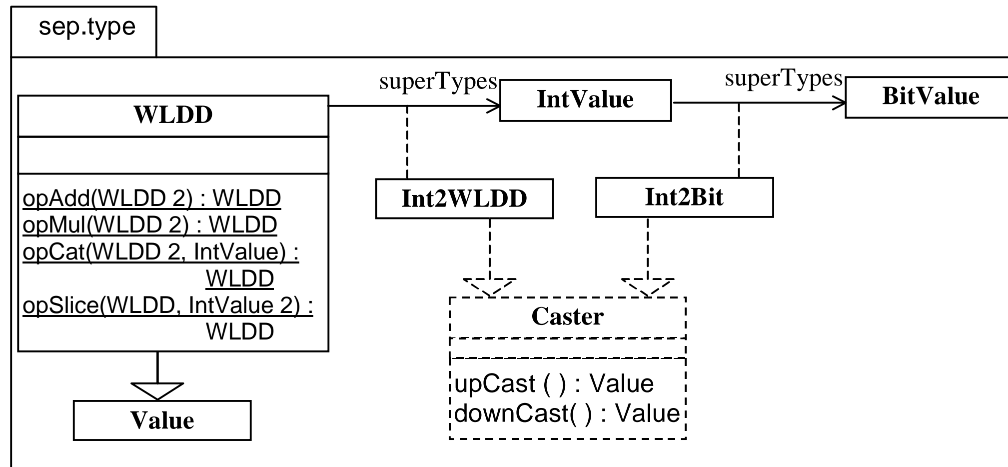


**Figure 6 – Définition du type WLDD.**

Le chapitre VI montre le mécanisme de liaison dynamique mis en place afin de choisir les opérateurs arithmétiques adaptés en fonction du type des données manipulées. C'est ce mécanisme de SEP qui est utilisé pour permettre la manipulation de K*BMD.

Le type WLDD est ajouté dans l'arborescence des types de SEP (cf. Figure 6). La classe WLDD hérite de la classe *Value*, ainsi les composants SEP pourront émettre et recevoir des K*BMD par l'intermédiaire des bus et des signaux de commandes. De plus, le type *WLDD* est défini comme un sous-type de *IntValue*. Tout ce qui peut être fait avec des entiers peut l'être avec des WLDD. Les opérations de base sur les entiers ou les *BitValue* sont redéfinies afin d'implémenter les algorithmes de manipulation des K*BMD. Un K*BMD qui représente un entier n est en fait un graphe réduit à une feuille (cf. Figure ci-contre).

(n,0)
0

Dès lors, les opérateurs adaptés à la structure de K*BMD seront utilisés automatiquement sans aucune modification de l'architecture. Lorsque les entrées d'un module sont des entiers, les opérateurs sur les entiers sont utilisés ; lorsque les entrées du même module sont des K*BMD, les opérateurs sur les K*BMD sont utilisés. Le type des sorties dépend du type déclaré par les opérateurs.

Les opérations caractéristiques nécessaires pour la modélisation d'architectures matérielles doivent être adaptées à la structure de K*BMD. La section 2-2 présente quelques unes de ces opérations et leur implémentation en K*BMD afin de donner une idée de la complexité d'utilisation de cette structure de données.

### 2-2. Les opérations élémentaires.

Pour simplifier le propos nous n'utiliserons que des K*BMD avec une décomposition positive de Davio.

L'implémentation de ce mécanisme pour la validation des services de l'unité de calcul présentée au chapitre VII nécessite l'implémentation pour un K*BMD de toutes les opérations

arithmétiques utilisées dans le module de multiplication (MU) et dans le module arithmétique et logique (ALB), ces opérations opèrent sur des mots de bits. Il faut implémenter les opérations classiques d’une unité arithmétique et logique – nous ne traiterons que l’*addition* et le *et logique* –, l’opération de multiplication, la sélection de n bits consécutifs à partir du m-ième et la concaténation de mots de bits.

Les autres composants – multiplexeurs et registres – n’effectuent aucune modification sur les données, ils se contentent le moment venu de les transmettre ou de ne pas les transmettre.

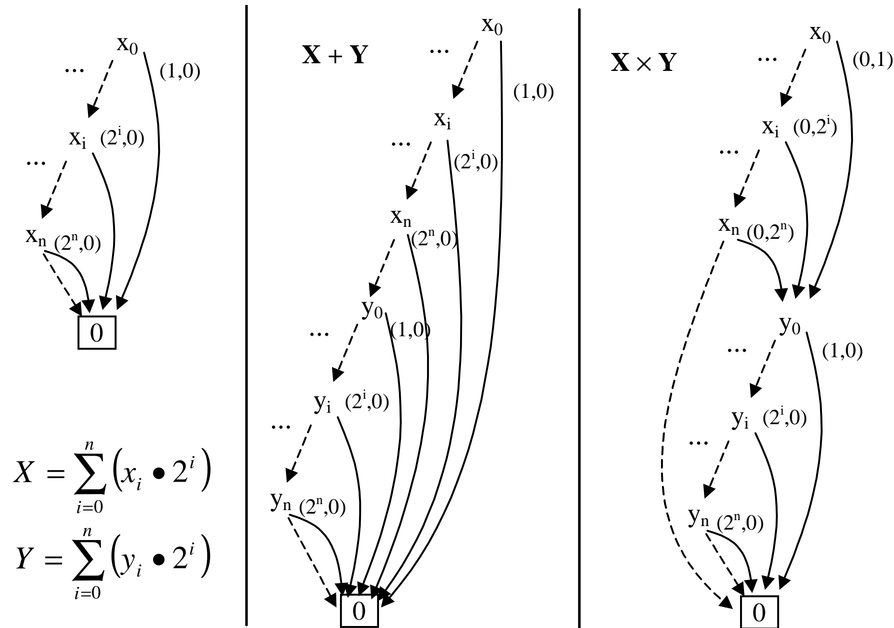


**Figure 7 – Addition et multiplication de deux mots de n+1 bits.**

Les cas les plus simples sont les cas de l’addition et de la soustraction. En effet, si f et g sont des fonctions de $B^n \rightarrow Z$ alors leurs décompositions positives de Davio sont $f = f_i^0 + x_i \times (f_i^1 - f_i^0)$ et $g = g_i^0 + x_i \times (g_i^1 - g_i^0)$. On a alors $f+g = f_i^0 + x_i \times (f_i^1 - f_i^0) + g_i^0 + x_i \times (g_i^1 - g_i^0)$, c’est-à-dire $(f_i^0 + g_i^0) + x_i \times ((f_i^1 - f_i^0) + (g_i^1 - g_i^0))$.

Si on appelle $f_G$ (respectivement $g_G$) la fonction représentée par le premier sous arbre de f (respectivement g) et $f_D$ (respectivement $g_D$) la fonction représentée par le deuxième sous-arbre de f (respectivement g). On a alors $(f+g)_G = f_G + g_G$ et $(f+g)_D = f_D + g_D$.

Pour construire l’arbre de la fonction f+g il suffit donc d’appliquer récursivement l’opérateur addition aux sous-arbres de f et de g. Le K*BMD correspondant à l’addition de deux mots de (n+1) bits est donné par la Figure 7.

Un calcul identique peut être fait pour la soustraction.

Calculons maintenant la décomposition positive de Davio de la fonction $f \times g$ par rapport à la variable $x_i$. On a $f \times g = (f_i^0 + x_i \times (f_i^1 - f_i^0)) \times (g_i^0 + x_i \times (g_i^1 - g_i^0))$. Si $x_i=0$ alors $(f \times g)_i^0 = f_i^0 \times g_i^0$, si $x_i=1$ alors $(f \times g)_i^1 = f_i^1 \times g_i^1$. De ce fait, $f \times g = f_i^0 \times g_i^0 + x_i \times (f_i^1 \times g_i^1 - f_i^0 \times g_i^0)$.

Ainsi, $(f \times g)_G = f_G \times g_G$ et $(f \times g)_D = (f_D + f_D) \times (g_D + g_D)$.

Le K*BMD correspondant à la multiplication de deux mots de n+1 bits est donné par la Figure 7.

La multiplication d'un K*BMD par une constante C se fait à temps constant, puisqu'il suffit de multiplier les coefficients m de l'arc entrant de la racine par la constante C. En effet, $(a + m \times f) \times C = ( a \times C + m \times C \times f)$.

La concaténation de deux mots de bits a et b est alors $a+2^n \times b$ où n est le nombre de bits utilisés pour représenter le mot a. Il s'agit alors d'une multiplication par une constante et d'une addition.

La sélection de n bits à partir du m^ième^ est une opération complexe pour les WLDDs. Cette opération que l'on appellera *slice* est définie pour un mot a de $k \geq n+m$ bits de la façon suivante : $slice(a, n, m) = ( a / 2^m ) \% 2^n$.

Les opérations de modulo et de division sont assez difficiles à réaliser, cependant en utilisant les propriétés que $(f+g)\%h = (f\%h + g\%h)\%h$ et $(f \times g)\%h = (f\%h \times g\%h) \% h$, la taille du K*BMD résultant est souvent linéaire surtout si *f* et *g* sont indépendantes de *h*.

### 2-3. Un exemple.

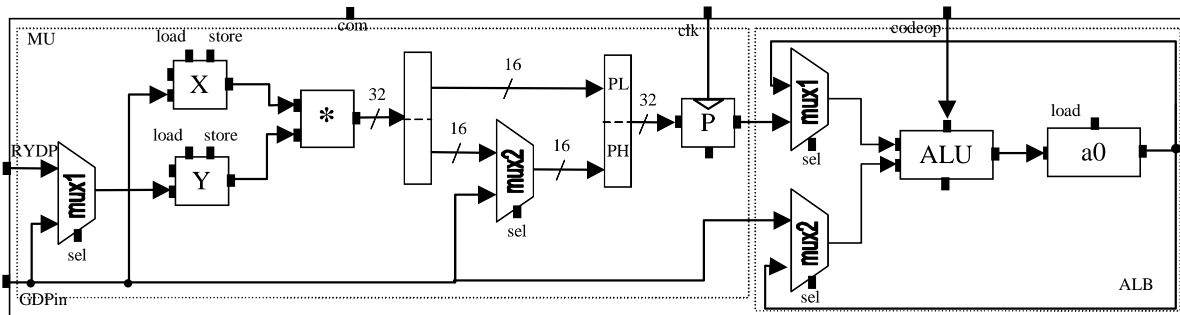


**Figure 8 – Unité de calcul simplifiée.**

Prenons l'exemple de l'unité de calcul représentée par la Figure 8. Pour obtenir ce schéma et afin de simplifier notre propos, les unités fonctionnelles non utilisées ont été supprimées dans l'unité de calcul présentée au chapitre VII.

Il s'agit alors de valider le service **codeop=Add ; mulacc_a0 ; clk** où clk correspond à l'activation du registre p synchrone sur l'horloge du système. Il faut vérifier que cette unité de calcul effectue correctement une opération complète de multiplication-accumulation en deux cycles d'horloge.

Pour cela, le module *unité de calcul* est instancié, la valeur de l'accumulateur a0 est fixée avec un K*BMD qui représente un mot de 36 bits $A0 = \sum_{i=0}^{35} \left(a_i \bullet 2^i\right)$, les deux entrées RYDP et $GDP_{in}$ prennent les valeurs $RYDP = \sum_{i=0}^{35} \left(rydp_i \bullet 2^i\right)$ et $GDP_{in} = \sum_{i=0}^{35} \left(gdp_i \bullet 2^i\right)$. Le service **codeop=Add ; mulacc_a0** est appelé sur le module, le registre p est alors activé pour simuler un cycle d'horloge (**clk**). Le K*BMD obtenu en entrée de l'accumulateur est comparé au K*BMD obtenu à partir de l'expression $A0+RYDP \times GDP_{in}$ qui est une expression de la fonction que l'on souhaite obtenir.

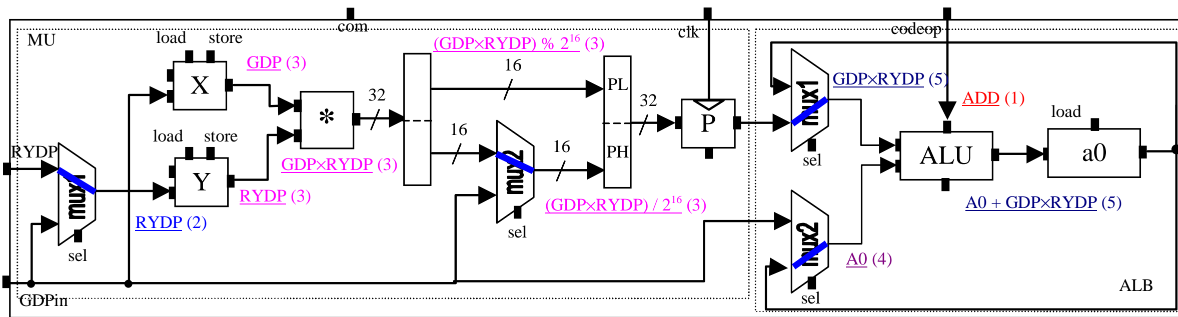


**Figure 9 – Scénario de validation.**

Dans le cas de l'architecture présentée, la comparaison indique que le service est correct. Pour s'en convaincre, regardons instant par instant le comportement de chaque composant (cf. Figure 9).

Premier instant : le code opération *Add* est positionné sur l'ALU.

Deuxième instant : *MU.mux1.sel<=0 || MU.mux2.sel <= 0 || ALB.mux1.sel=2 || ALB.mux2.sel=0.* L'expression RYDP est propagée en entrée du registre Y.

Troisième instant : *MU.Y.com=store || MU.X.com=store*. Les registres X et Y sont chargés, les expressions RYDP et GDP sont multipliées, le résultat est décomposé en sa partie haute *(GDP*RYDP)/16* et sa partie basse *(GDP*RYDP)%16*, puis recomposé par concaténation.

Quatrième instant : *ALB.a0.load.* L'accumulateur a0 est activé, l'expression a0 est propagée en entrée de l'ALU.

Cinquième instant : le registre p est activé, l'expression *GDP×RYDP* est propagée vers l'ALU. Le résultat calculé est alors *A0 + GDP×RYDP*.

En conclusion, le résultat obtenu manuellement ci-dessus en raisonnant sur des expressions algébriques est obtenu automatiquement dans une phase de simulation de SEP en utilisant des K*BMD. L'intégration d'un tel mécanisme dans SEP ne nécessite aucune modification de SEP, puisqu'il suffit de définir un nouveau type dans l'arbre de sous-typage et d'implémenter un paquetage de gestion des K*BMD.

L'implémentation d'un paquetage de gestion de K*BMD n'a actuellement pas été traitée.