- **(2)** l'étape de la construction (Constructive Search).

L'étape de recherche consiste à identifier dans la littérature et les travaux existants, certains opérateurs qui peuvent mutualiser ou partager des ressources communes (tant au niveau algorithmique qu'architectural). La construction est la prochaine étape dans la conception du CO, qui suit l'étape « Existing Search», elle consiste donc à la construction de CO générique qui devra alors réaliser les traitements ciblés [62]. Par conséquent, le premier objectif de la technique de l'opérateur commun est de rechercher les points de similarité entre les différentes architectures étudiées pour définir précisément un cahier des charges concernant les caractéristiques principales de l'architecture générique à réaliser. Suivant les structures présentées précédemment de l'unité de désétalement, le Radix 2/4/8 pour le FFT-SDF, et la cellule Cordic, on remarque l'existence de trois principales similarités [4]:

- La première similarité qui est la principale, est que les trois structures ont la même forme « structure Butterfly ». Cette remarque est apparente à travers les figures (III.1, III.3 et III.4), ainsi les équations qui décrivent leurs fonctionnements (les équations (1), (2), (3), (4) et l'équation (14)). Cette similarité est bien illustrée dans la figure III.12 (pour la partie réelle seulement de FFT), où on peut voir qu'il existe dans les trois fonctions, une opération d'addition et de soustraction pour deux opérandes (avec une légère modification pour le Cordic).
- La seconde similarité est la présence des multiplexeurs 2:1 entre l'unité de désétalement et le processeur élémentaire (PEx) de FFT-SDF. Ces multiplexeurs peuvent être partagés entre les deux fonctions.
- La troisième similarité est la présence de registres entre l'unité de désétalement et la cellule Cordic.

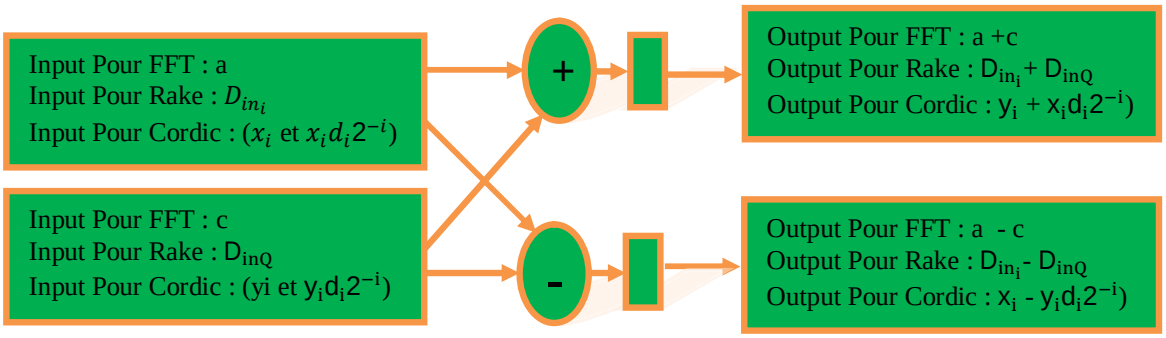


Figure. III.12. Structure Butterfly Commun.

Par contre les blocs qui ne peuvent pas se partager entre les trois architectures sont les deux registres à décalage effectuant la division par 2 pour x et y qui se trouvent seulement dans la cellule Cordic, et les deux inverseurs de signe qui se trouvent seulement dans l'unité de désétalement.

Après avoir identifié les points de similarité entre les trois architectures étudiées, nous essayons de proposer une architecture générique unique capable de prendre en charge les trois fonctions par les mêmes ressources. Notre architecture proposée se compose d'additionneurs, de soustracteurs, de multiplexeurs et de registres comme des blocs fonctionnels partagés entre les trois fonctionnalités [4]. La figure III.13 représente notre architecture universelle proposée pour l'unité de désétalement, le processeur élémentaire PE de FFT, et la cellule Cordic.

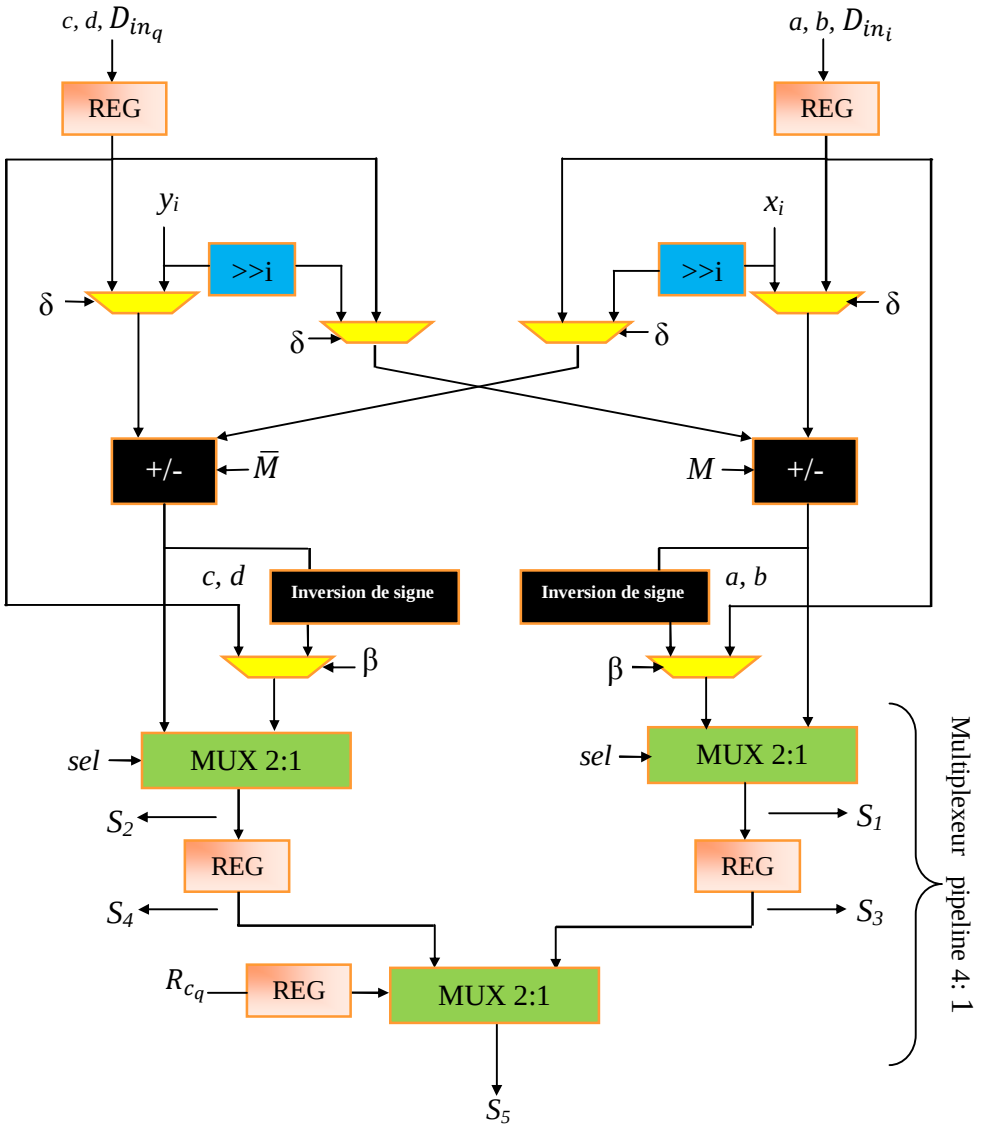


Figure. III.13. L'opérateur universel proposé pour l'unité de désétalement, processeur élémentaire PE de FFT, et la Cellule Cordic [4].

Elle est composée de deux blocs d'addition/soustraction, deux soustracteurs pour l'inversion de signe (inversion du signe dans la figure III.13), deux décaleurs (registres à décalage de couleur bleu), registres de données et des multiplexeurs 2:1. Il faut noter que les blocs additionneur/soustracteur sont capables de traiter différents types de données

(quantification) de la FFT, Cordic et de l'unité de désétalement [4]. En effet, il y a une différence dans le codage (le nombre de bits utilisés) des échantillons entre les différentes applications [18].

Dans cette architecture RTL proposée, β et δ sont des paramètres booléens (implémentés par un seul bit de sélection) qui permettent la configuration de la structure générique en utilisant des multiplexeurs (colorés en jaune dans la figure III.13) pour basculer entre le calcul de: PE-FFT, l'unité de désétalement et la cellule Cordic. Le signal de commande (SEL) est commun entre les trois fonctionnalités, et il peut avoir trois valeurs en fonction du mode choisi: R_{c_i} (signal de sélection pour le désétalement), F_{FFT} (signal de sélection pour la FFT), et δ (utilisée pour la sélection du calcul Cordic). M est un signal de commande permettant de fonctionner comme un additionneur ou un soustracteur. Enfin le signe (d_i) utilisé pour la cellule Cordic, est commandé par la représentation binaire de l'angle θ [93], ceci nous permet de supprimer l'accumulateur d'angle et donc de réduire l'occupation de surface de notre architecture générique.

La fonction de l'architecture proposée (voir figure III.13) peut être comprise, en considérant les équations ci dessous comme une synthèse de calculs obtenus à partir des équations (1), (2), (3), (4) et l'équation (14):

$$\begin{cases} \alpha = \left((a,b,D_{in_i}).\delta + (1-\delta).x_i\right) + M.\left(\left(c,d,D_{in_q}\right).\delta + (1-\delta).y_i.2^{-i}\right) \\ \alpha' = \left((a,b,D_{in_i}).\delta + (1-\delta).y_i\right) + \bar{M}.\left(\left(c,d,D_{in_q}\right).\delta + (1-\delta).x_i.2^{-i}\right) \end{cases} \quad (16)$$

$$\begin{cases} k = \beta.(a,b) + (1-\beta).Inversion\ sign\ (\alpha) \\ k' = \beta.(c,d) + (1-\beta).Inversion\ sign\ (\alpha') \end{cases} \quad (17)$$

$$\begin{cases} S_1 = sel.\alpha + (1-sel).k \\ S_2 = sel.\alpha' + (1-sel).k' \end{cases} \quad (18)$$

$$\begin{cases} S_3 = S_1.Z^{-1} \\ S_4 = S_2.Z^{-1} \end{cases} \quad (19)$$

où Z^{-1} représente le retard de registre.

$$S_5 = R_{c_q}.S_3 + \left(1 - R_{c_q}\right).S_4 \quad (20)$$

$$M = \delta + d_i \rightarrow \bar{M} = not(M) \quad (21)$$

$$sel = \left(R_{c_i}.(1-\beta) + F_{FFT}.(\beta)\right)(\delta) + (1-\delta) \quad (22)$$

L'architecture proposée permet de commuter entre les modes suivants:

- **Le mode FFT.** Ce mode est sélectionné par les paramètres de configuration 'β' = 1 et 'δ' = 1. Selon les équations développées précédemment (pour la partie réelle comme illustration), nous avons les valeurs des paramètres et les signaux de

configuration suivants: $M = \delta = 1 \rightarrow \bar{M} = 0$; $sel = F_{FFT}$ (signal de sélection pour la FFT); $\alpha = a + c$; $\alpha' = a - c$; $k = a$; $k' = c$; $S_1 = F_{FFT}.\alpha + (1 - F_{FFT}).k$ (vers le reste du PE_X) et $S_2 = F_{FFT}.\alpha' + (1 - F_{FFT}).k'$ (vers le buffer de retard). où $(a \,\&\, c)$ représentent les données d'entrées, et S_1 et S_2 sont les sorties de PEx dans le mode FFT. Nous pouvons obtenir le résultat de la partie imaginaire $(b \,\&\, d)$ par le même mode. La figure III.14 (a) illustre l'implémentation du mode FFT et les ressources utilisées (de couleur bleu). Le mode FFT nécessite deux opérations d'addition/soustraction. Dans ce mode également, l'architecture peut être: un élément de traitement "1" (PE1 sur la figure III.3) si on ajoute une multiplication par « *j* », ou un élément de traitement "2" (PE2 sur la figure III.3) si on ajoute une multiplication par « j » et une multiplication par le facteur de rotation constant $(\frac{\sqrt{2}}{2}.(1 \pm j)$, et enfin, un élément de traitement "3" (PE3 sur la figure III.3).

- **Le mode désétalement :** ce mode est sélectionné par les paramètres de configuration de valeurs 'β' = 0 et 'δ'= 1. Les équations développées précédemment donnent les valeurs des paramètres et les signaux de configuration suivants: $M = \delta = 1 \rightarrow \bar{M} = 0$; $sel = R_{c_i}$ (signal de sélection pour le désétalement, voir le tableau III.1); $\alpha = +D_{in_i} + D_{in_q}$; $\alpha' = +D_{in_i} - D_{in_q}$; $k = Inversion\ sign\ (\alpha) = -D_{in_i} - D_{in_q}$; $k' = Inversion\ sign\ (\alpha') = -D_{in_i} + D_{in_q}$; $S_1 = R_{c_i}.\alpha + (1 - R_{c_i}).k$; $S_2 = R_{c_i}.\alpha' + (1 - R_{c_i}).k'$; $S_3 = S_1.Z^{-1}$; $S_4 = S_2.Z^{-1}$; $S_5 = R_{c_q}.S_3 + (1 - R_{c_q}).S_4 = D_{out_i}$. où $(D_{in_i} \& D_{in_q})$ représentent les données d'entrées, et S_5 représente la première sortie (D_{out_i}) dans le mode désétalement. Nous pouvons obtenir la seconde sortie (D_{out_q}) en utilisant un deuxième MUX 4:1 pipeline comme proposé dans la section 2.1. La figure III.14 (b) représente l'architecture proposée en mode désétalement (de couleur bleu). Le mode désétalement nécessite deux opérations d'addition / soustraction et deux inversions de signe (deux inverseurs de signe).
- **Le mode cellule Cordic :** ce mode est sélectionné si 'δ' = 0 et lorsque le signal de commande *sel* est égal à 1. Les équations développées précédemment donnent les paramètres et les signaux de configuration de valeurs suivantes: $M = d_i \rightarrow \bar{M} = \bar{d}_i$; $sel = 1$; $\alpha = x_i + d_i.y_i.2^{-i}$; $\alpha' = y_i + \bar{d}_i.x_i.2^{-i}$; $S_1 = sel.\alpha + (1 - sel).k$; $S_2 = sel.\alpha' + (1 - sel).k'$; $S_3 = S_1.Z^{-1}$ (vers la cellule Cordic suivante); $S_4 = S_2.Z^{-1}$ (vers la cellule Cordic suivante); Où $(x_i \& y_i)$ représentent les données d'entrée, et S_3 et S_4 sont les sorties de la cellule de traitement dans le mode Cordic. La figure III.14 (c) illustre la mise en œuvre du mode cellule Cordic et les ressources utilisées (de couleur bleu). Comme présenté dans la figure III.14 (c), le mode Cordic nécessite deux opérations d'addition/soustraction et nécessite deux décaleurs (deux registres à décalage).

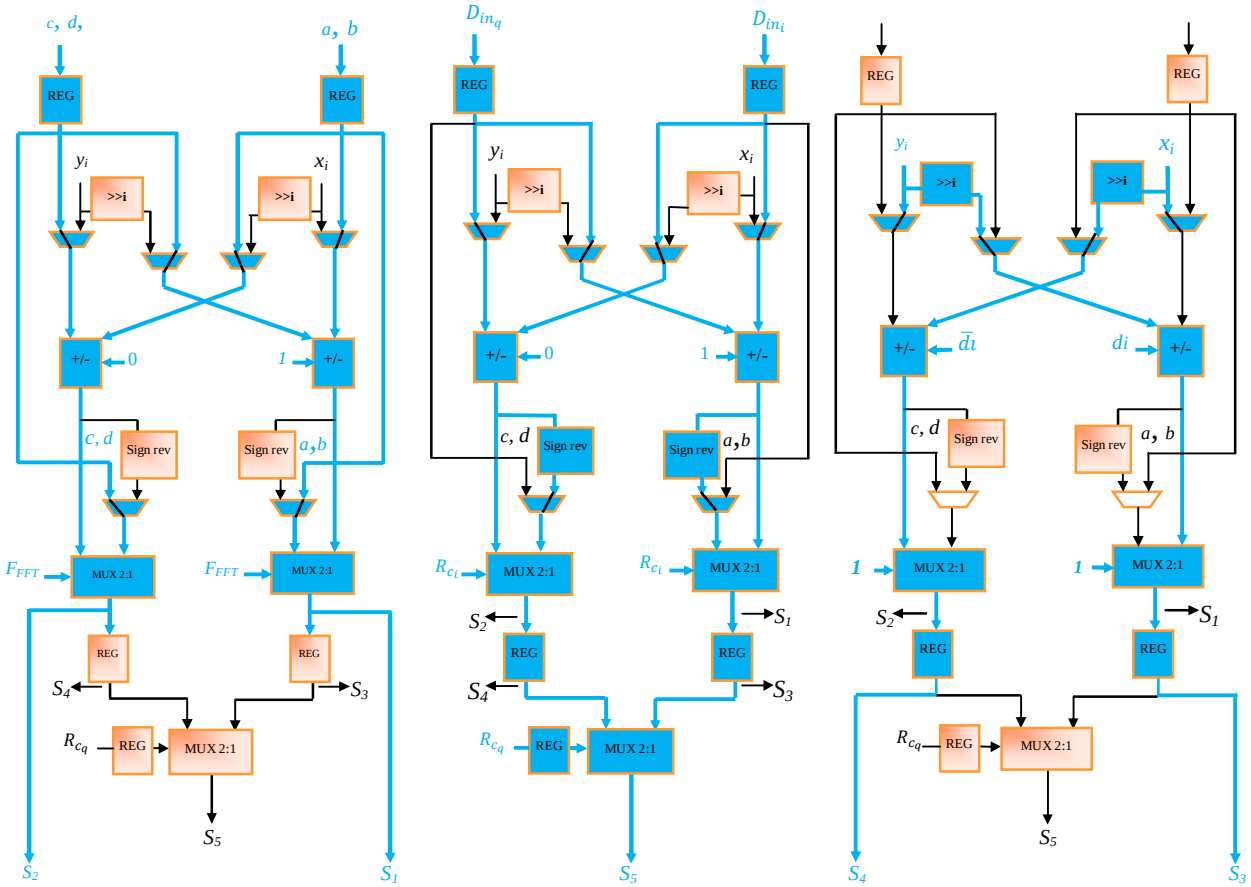


Figure. III.14. Les trois modes de fonctionnement de la cellule proposée et ses ressources: (a) Mode FFT, (b) Mode désétalement, (c) Mode cellule Cordic.

6. Implémentation sur FPGA, résultats et comparaisons de la performance

La description RTL de l'architecture proposée a été implémentée sur un FPGA Virtex-V de Xilinx [94] en utilisant la description structurale VHDL (VHSIC Hardware Description Language). L'outil ISE 10.1i de Xilinx [51] a été utilisé pour cette implémentation numérique permettant d'obtenir les ressources logiques nécessaires et les performances associées. Les résultats de synthèse de notre implémentation après le processus ''place and route'' de l'outil ISE 10.1i, sont détaillés dans le tableau III.2 pour différentes longueurs (tailles) de mots.

Plus précisément, le tableau III.2 spécifiées, la fréquence de fonctionnement maximale et la consommation de ressources en termes de ''logic Slice'' ou ''Slice Flip-Flops'' (Slice registers) et les LUTs (Look Up Tables) pour différentes longueurs-mots. Ces résultats montrent que l'architecture générique proposée peut être implémentée facilement et

efficacement de façon optimisée sur la technologie FPGA tout en réduisant la complexité de l'architecture [4].

Table III.2. Comparaison de l'implémentation de l'architecture proposée avec l'architecture Velcro sur FPGA Virtex-V (XC5VLX50T).

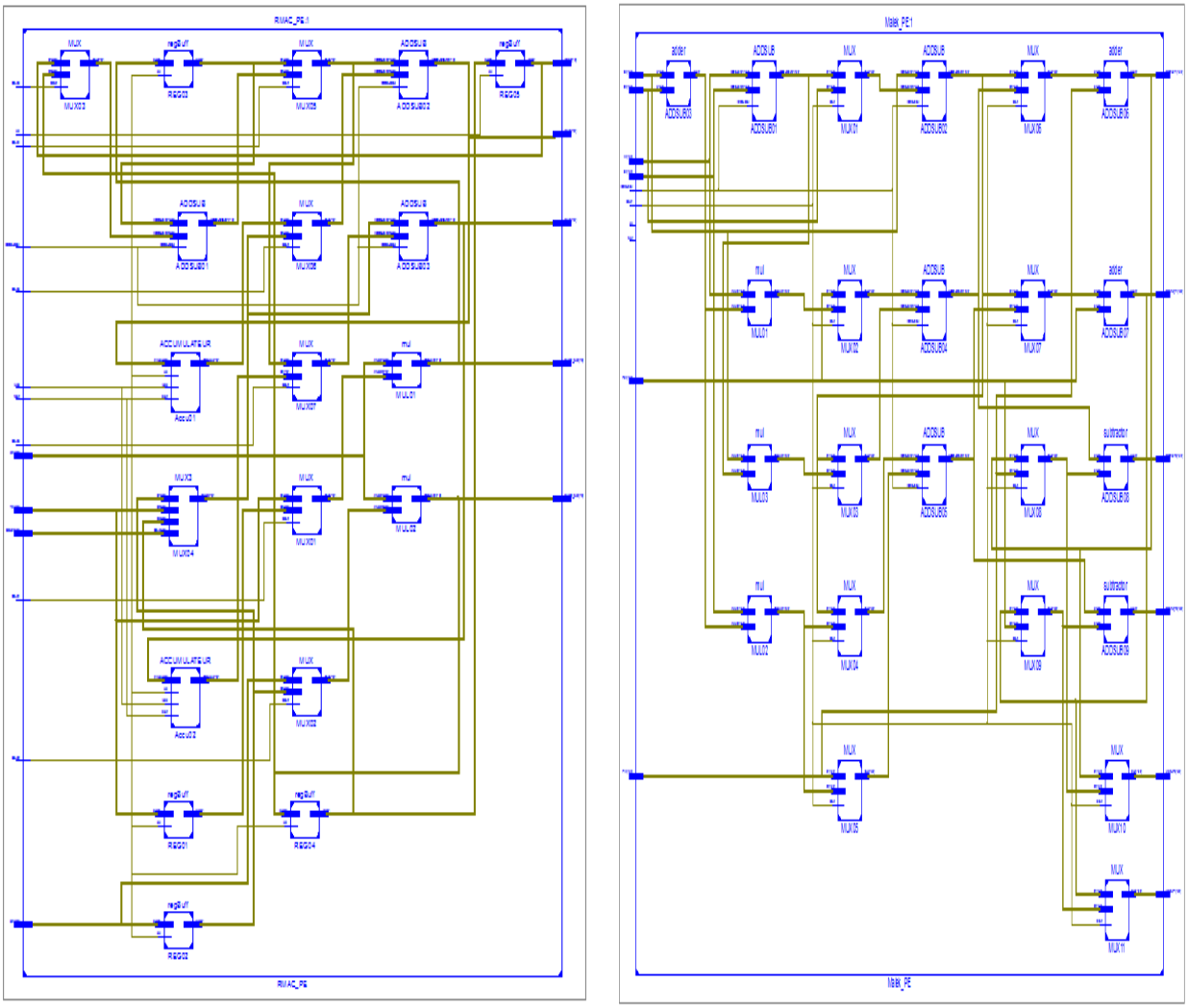
	Longueur de mots (*n*)					
	8 bits		12 bits		16 bits	
Paramètre	Architecture proposée	Velcro	Architecture proposée	Velcro	Architecture proposée	Velcro
Nombre de Slice Registers (19200)	32	16	48	24	64	32
Nombre de Slice LUTs (19200)	99	123	114	205	150	273
Fréquence maximale (MHz)	329.826	Unité de Désétalement : 376.605 Cellule de Cordic : 602.518 Cellule de FFT: 472.903	308.780	Unité de Désétalement : 308.447 Cellule de Cordic : 571.200 Cellule de FFT: 453.391	300.341	Unité de Désétalement : 300.026 Cellule de Cordic : 542.977 Cellule de FFT: 435.426

Cette réduction de la complexité est souhaitable pour effectuer des opérateurs pour l'unité de désétalement, le papillon FFT, et la cellule Cordic. En outre, notre architecture présente un bon compromis entre la vitesse de fonctionnement et la consommation des ressources logiques. Par exemple le résultat de l'implémentation pour une longueur de mot de 8 bits nécessite seulement 99 Slices CLB, aucun multiplicateur ou blocs DSP ou des RAM, avec une fréquence de fonctionnement maximale de 329,8 MHz.

Pour une évaluation réelle de la performance et la consommation en ressources, nous comparons les implémentations de notre architecture générique avec la technique Velcro [21] et avec quelques travaux similaires disponibles dans la littérature qui sont basés sur la technique des opérateurs communs et les architectures reconfigurables [3] [95]. Ces travaux similaires disponibles comportent certaines fonctionnalités communes avec celle proposée. Cependant la comparaison entre les opérateurs communs et/ou les cellules reconfigurables qui sont basées sur des architectures différentes et/ou ont été implémentées sur différentes technologies n'est pas simple. Comme l'implémentation dans la technologie FPGA n'a pas été introduite dans les travaux de [3] et [95] ([3] utilise un DSP et [95] utilise un ASIC), nous proposons de refaire ces travaux avec une implémentation sur FPGA de type Virtex 5. Ce qui nous permet de faire directement la comparaison de nos résultats avec ces architectures précédentes de façon équitable et appropriée. Nous avons également utilisé les mêmes métriques de la comparaison (consommation des ressources et la fréquence maximale) pour différentes précisions (longueurs de mots). La figure III.15 présente les schémas RTL pour les travaux précédents présentés dans [3] et [95] respectivement. La figure III.16 présente le schéma RTL de notre architecture proposée.

La méthode "Velcro" est la méthode conventionnelle pour créer un terminal multistandard comme nous avons dit dans le chapitre I. Dans notre cas, c'est la juxtaposition de trois architectures; une architecture pour l'unité de désétalement, une architecture pour le processeur élémentaire (FFT) et une architecture pour la cellule Cordic, et la "reconfiguration" est simplement réalisée par un simple switch pour basculer d'une architecture à une autre. H. Lange et al. [95], ont proposé un élément de traitement (PE) reconfigurable pour les algorithmes qui se basent sur la multiplication-accumulation ou multiplication-addition, le RMAC_PE (Reconfigurable Multiply-Accumulate-based Processing Element). Il se compose de deux multiplicateurs, trois additionneurs/soustracteurs, deux accumulateurs et plusieurs registres de données. Cette architecture montre une bonne flexibilité qui permet la réalisation de différentes classes d'algorithmes comme FFT/IFFT, filtrage FIR et la multiplication matrice-vecteur [95]. M. Naoues et al. [3] ont suivi la même approche que dans [95], ils ont proposé un opérateur commun pour le FFT et l'algorithme de Viterbi (FFT/Viterbi Common Operator). Le CO

FFT/Viterbi est composé de deux multiplicateurs, deux additionneurs/soustracteurs, des additionneurs, des soustracteurs, multiplexeurs et plusieurs registres de données.



a) **b)**

Figure**.** III.15**.** Schémas RTL de l'implémentation sur FPGA Virtex V de : a) RMAC [95] et b) CO [3].

Le CO FFT/Viterbi [3] définit trois configurations: 1^{ere} config: FFT Radix-2 papillon, 2eme config: unité ACS (Add Compare Select) pour l'algorithme de Viterbi, effectue le calcul des métriques de chemin et la sélection du chemin survivant, 3eme config: unité BMC (Branch Metric Calculation) pour l'algorithme de Viterbi, effectue le calcul des métriques de branche pour chaque état du treillis [3].

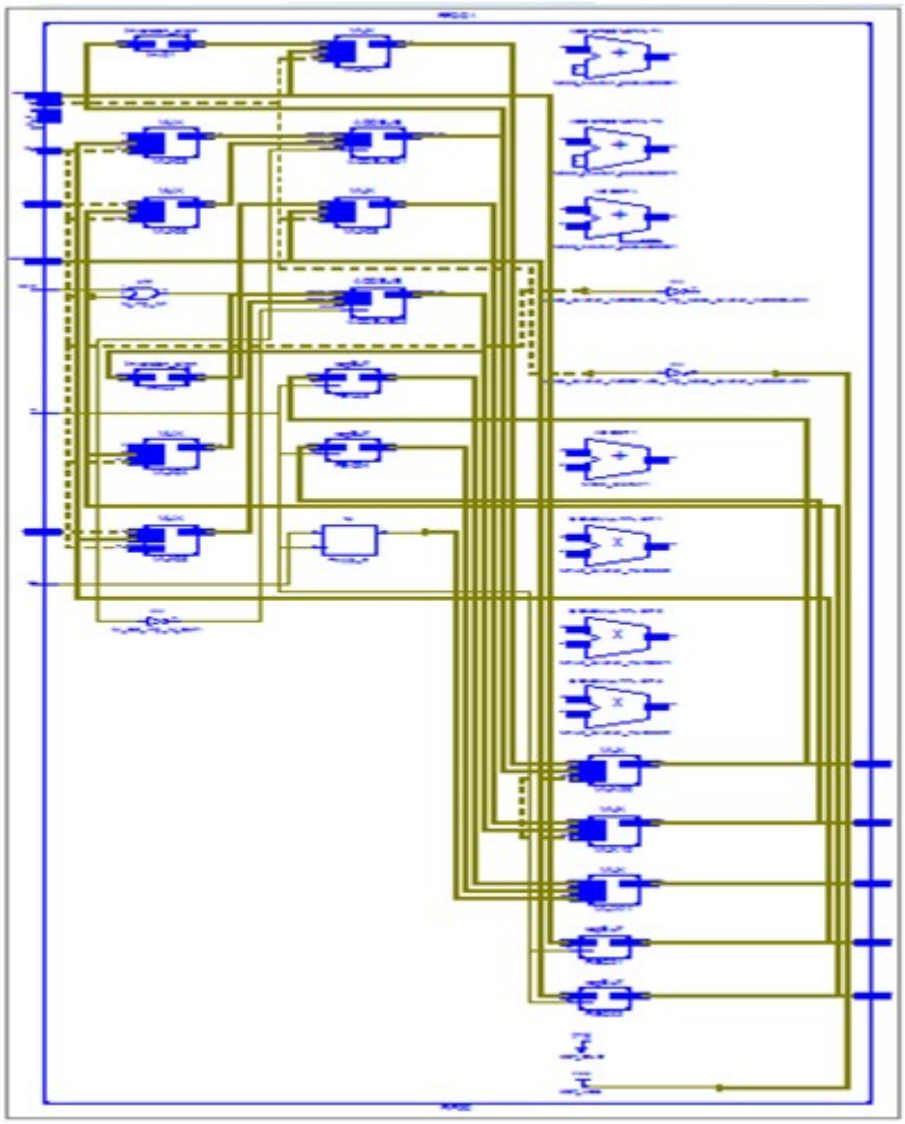


Figure. III.16. Schémas RTL de l'implémentation sur FPGA Virtex V de l'architecture proposée [4].

Les résultats de la comparaison entre l'architecture proposée et l'architecture Velcro et entre l'architecture proposée et les travaux similaires sont donnés dans le tableau III.2 et le tableau III.3 respectivement pour différentes longueurs-mots (n).

Comme le montre le tableau III.2, si on compare avec l'architecture Velcro, notre architecture fonctionne au minimum avec une fréquence de 300 MHz et offre un gain de surface allant de 5,7%, 29,25% et 29,83% pour une longueur de mots de 8, 12 et 16 bits respectivement. Par rapport à l'approche Velcro, l'architecture proposée présente une consommation de surface FPGA raisonnable, ce qui rend possible d'envisager une implémentation complète d'un processeur de réseau pour les systèmes WCDMA et OFDM.

Table III.3. Comparaison de l'implémentation sur FPGA Virtex-V (XC5VLX50T) de l'architecture proposée avec les travaux précédents [3] [95].

Paramètre			Longueur (tailles) de mots *n*		
			8 bits	12 bits	16 bits
Fréquence maximale (MHz)	Nombre de Slice LUTs (19200)	Nombre de Slice Registers (19200)			
329.826	**99**	**32**	Architecture proposée	Architecture proposée	Architecture proposée
142.835	**88**	**56**	RMAC [95]	RMAC [95]	RMAC [95]
240.941	**40**	**48**	FFT/Viterbi CO [3]	FFT/Viterbi CO [3]	FFT/Viterbi CO [3]
308.780	**114**	**48**			
141.002	**132**	**84**			
235.771	**60**	**72**			
300.341	**150**	**64**			
139.216	**176**	**112**			
230.819	**80**	**96**			

Bien que cette réduction de la complexité, n'est pas plus grande, et la fréquence maximale est inférieure à une architecture matérielle dédiée (Velcro). Le grand avantage de l'architecture proposée est la plus grande flexibilité, et son utilisation dans une architecture régulière. Cette flexibilité augmente le parallélisme du traitement en partageant des ressources communes entre les algorithmes, ce qui conduit à améliorer la vitesse de calcul [1][71].

De même, si on fait la comparaison avec le processeur RMAC [95] qui partage avec notre architecture proposée la fonction Cordic et une partie de la fonction FFT, notre architecture présente un gain significatif en termes de surface de FPGA d'environ 9%, 33,3% et 25,69% pour une taille de mots de 8, 12 et 16 bits, respectivement comme le montre le tableau III.3. La fréquence maximale de l'architecture proposée est également supérieure à la fréquence maximale de l'architecture RMAC [95] Ainsi, les résultats de l'implémentation démontrent clairement que notre architecture proposée peut fournir une réduction de la surface logique totale nécessaire dans le FPGA. Par conséquent, par rapport à l'architecture Velcro et l'architecture proposée dans [95], nous montrons l'intérêt de l'architecture adoptée, qui propose une solution permettant d'optimiser les ressources matérielles embarquées. La raison principale est que notre architecture est basée sur une structure RTL pipeline. Comme le montre le tableau III.3, bien que la surface logique soit faible par rapport à l'architecture FFT/Viterbi CO [3], notre architecture proposée qui effectue plus de fonctionnalités (FFT, Cordic, désétalement) améliore les performances en augmentant la vitesse de fonctionnement de 36,89%, 30,96% et 30,11% pour une longueur de mots respectivement de 8, 12 et 16 bits. Cette comparaison montre également que l'opérateur proposé offre un bon compromis entre la fréquence de fonctionnement, la consommation de la surface de FPGA et la flexibilité. En outre, notre architecture fournit une réutilisation efficace des ressources. Par exemple, dans le mode de désétalement, le pourcentage d'utilisation des ressources, est de 90% des ressources totales (figure III.14 (b)). De même, dans les modes Cordic et FFT, le pourcentage d'utilisation des ressources, est de 60% des ressources totales de l'opérateur (figure III.14 (a) et (c)), respectivement. Avec cette réutilisation efficace de ressources et la réduction de la consommation de surface de FPGA obtenu, nous attendons une consommation raisonnable de la puissance [3] [4].

En outre, le principal avantage de la conception proposée est la versatilité; au lieu de trois architectures différentes pour l'unité de désétalement, la cellule Cordic et le processeur élémentaire (PE), l'architecture proposée est capable de faire toutes ces fonctionnalités par la même structure, avec une utilisation optimale de la surface logique de FPGA et ainsi une consommation de puissance raisonnable. Cette réduction de la consommation en ressource et en puissance permet d'envisager une parallélisation éventuelle en multipliant le nombre des opérateurs communs, ce qui conduit à améliorer la fréquence

de fonctionnement. Il est ainsi possible avec cette réduction de la consommation en ressource de réutiliser notre opérateur commun par blocs dans le cadre d'un processeur de réseau complet, pour un récepteur Rake, un module de FFT-SDF, et un module Cordic [4].

7. Discussion

* Tout d'abord, si nous considérons le coût de l'implémentation sur DSP (Digital Signal Processing) et ASIC (Circuit Application-Specific Integrated) sur lesquels sont basés les travaux précédents [3] et [95], nous pouvons faire plusieurs observations et remarques. Comme nous avons dit dans le deuxième chapitre, le DSP est flexible, mais il manque d'une puissance de calcul suffisante. Si on considère la surface (zone) de silicium, la performance de FPGA est améliorée par rapport aux DSP existants. Cependant, par rapport aux ASIC, le FPGA nécessite entre cinq et dix fois plus d'espace de silicium au même nombre de portes équivalent (gates count), avec une vitesse inférieure (En moyenne, les FPGA sont environ 7,2 fois plus lent qu'un ASIC), et une consommation de puissance encore grande (En moyenne, un FPGA consomme en puissance 12 fois plus qu'un ASIC) [96]. Néanmoins, le facteur clé pour la réduction de cet écart (gap) entre le FPGA et l'ASIC, est la disponibilité des blocs hétérogènes dans les FPGA modernes comme la mémoire et les multiplicateurs, où ces blocs peuvent réduire cet écart (gap) de surface de façon significative. En effet, l'écart de la surface, le retard du chemin critique (critical-path delay) et la consommation de puissance dynamique sont potentiellement réduits à cinq (divisés par cinq). Mais en général, ce coût n'est pas très important si on considère les autres avantages de FPGA, à savoir la flexibilité et la capacité de faire de grandes modifications des fonctions sans aucun changement de la partie matérielle. En outre, le parallélisme peut être exploité pour obtenir une grande vitesse de fonctionnement dans le FPGA. En dehors de la flexibilité et les performances, le FPGA a un autre avantage: en effet les différentes précisions (longueurs ou tailles de mots), peuvent être facilement logées à différents nœuds dans le système de telle sorte que le traitement requis, soit exactement réalisé [4].

* Nous pouvons comparer les fonctions accomplies par les solutions proposées dans [3], [95] et notre solution dans le cas de l'implémentation dans la même technologie FPGA. En conséquence, H. Lange et al [95], ont proposé un élément de traitement reconfigurable (RMAC_PE) pour les algorithmes basé sur la multiplication-accumulation ou multiplication-addition. Ce processeur montre une bonne flexibilité qui permet la réalisation de différentes classes d'algorithmes comme le FFT/IFFT, le filtrage FIR de valeur réelle ou complexe, les multiplications matrice-vecteur et le Cordic-PE. La bonne flexibilité dans le RMAC_PE est obtenue au détriment de la consommation de surface. En revanche, notre approche proposée dans ce travail, fournit une meilleure fréquence de

fonctionnement avec une consommation de surface faible au détriment de la flexibilité par rapport à l'architecture de [95], car notre approche à première vue est limitée à trois fonctionnalités; unité de désétalement, élément de traitement FFT-SDF et la cellule Cordic. Cela est dû au fait que la technique des opérateurs communs développée fournit des blocs logiques spécifiques adaptés aux fonctionnalités implémentées. Par conséquent, par rapport à [95], notre architecture est flexible, et montre une bonne efficacité en surface et en puissance. M. Naoues et al [3], ont suivi également la même approche que [95], mais en limitant leur approche pour deux fonctionnalités; le Radix-2 FFT butterfly et l'algorithme de Viterbi. L'architecture de décodeur de Viterbi peut être divisée en trois unités (figure III.17).

1- l'unité BMC (Branch Metric Calculation) qui calcule les métriques de branche associée à chaque transition du treillis,

2- l'unité d'addition et sélection (ACS) qui calcule les métriques accumulées et sélectionne le chemin survivant entrant pour chaque état du treillis,

3- l'unité de gestion de survivant (SMM) qui stocke la décision prise par l'unité ACS dans le but de fournir le chemin le plus probable à la sortie du décodeur.

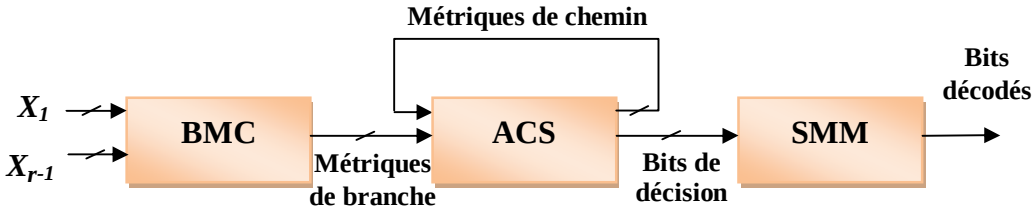


Figure III.17 – Structure globale d'un décodeur de Viterbi

En ce qui concerne l'unité BMC, on peut remarquer qu'il existe une similitude (similarité) apparente avec l'unité de désétalement dans notre architecture générique [4]. L'opération de l'unité ACS nécessite les métriques de branche BM_{00}, BM_{01}, BM_{10} et BM_{11}. Ces mesures consistent à de simples opérations de comparaison qui peuvent être obtenues par l'unité BMC, comme représentée dans l'équation (23) [3] [4].

$$\begin{cases} bm_{00} = +E_1 + E_2 \\ bm_{01} = +E_1 - E_2 \\ bm_{10} = -E_1 - E_2 \\ bm_{11} = -E_1 + E_2 \end{cases} \qquad (23)$$

Selon les équations (23), l'unité BMC utilise de simples opérations d'addition/soustraction et leurs inverses, donc c'est la même conception (design) qui se trouve dans l'unité de désétalement (voir section 2). Par conséquent, nous pouvons en déduire que notre opérateur commun proposé, est capable facilement de prendre en charge le

fonctionnement de l'unité BMC [4]. De même, avec l'unité ACS, celle ci nécessite deux opérations d'additions et une opération de soustraction comme illustré dans [3], notre architecture proposée est également capable de prendre en charge l'opération de l'unité ACS, avec une légère modification de la configuration. Ce qui nous conduit à dire aussi que notre architecture, a plus de fonctionnalités donc plus de flexibilité avec un gain de surface par rapport à [3].

En ce qui concerne l'algorithme de FFT, [3] et [95] ont proposé des architectures qui implémentent le Radix-2 FFT butterfly composés d'additionneurs et multiplicateurs, tandis que notre approche est spécifique pour la méthode de FFT-SDF (Fast Fourier Transform-Single path Delay Feedback). Ici aussi, il faut noter que notre architecture peut facilement prendre la fonction de Radix-2 FFT butterfly avec une légère modification. D'abord d'après Cooley-Tukey [97], la représentation habituelle de la FFT radix-2, est le diagramme en treillis qui partitionne récursivement une FFT en éléments de calculs intermédiaires également nommés papillons. L'algorithme FFT radix-2 met à jour les échantillons intermédiaires selon l'équation 24 qui illustre les opérations réalisées par un papillon FFT Radix-2 Décimation-en-Temps (DIT) où les $y_k^{[i]}$sont les échantillons à l'entrée du papillon et W_n^k les « Twiddle factors » de la FFT.

$$\begin{cases} y_{k+1}^{[0]} = y_k^{[0]} + y_k^{[1]} \times W_n^k \\ y_{k+1}^{[1]} = y_k^{[0]} - y_k^{[1]} \times W_n^k \end{cases} \tag{24}$$

Si
$$\begin{cases} y_k^{[1]} = a + jb \\ W_n^k = c + jd \\ y_k^{[0]} = e + jf \end{cases} \tag{25}$$

Donc l'équation 24 devient :

$$\begin{cases} y_{k+1}^{[0]} = y_k^{[0]} + y_k^{[1]} \times W_n^k = e + (ac - bd) + j[f + (ad + bc)] \\ y_{k+1}^{[1]} = y_k^{[0]} - y_k^{[1]} \times W_n^k = e - (ac - bd) + j[f - (ad + bc)] \end{cases} \tag{26}$$

Pour adapter notre opérateur commun au calcul de radix 2 FFT butterfly représenté par l'équation 26 on utilise le multiplicateur complexe pipeline proposé dans le paragraphe 3.4 (initialement proposé pour l'architecture FFT-SDF représentée dans la figure III.9). Le nouvel opérateur commun adapté pour le calcul de radix 2 FFT est représenté dans la figure III.18.

L'architecture de multiplicateur complexe pipeline (figure III.9) permet d'obtenir les résultats de la multiplication de $y_k^{[1]} \times W_n^k = (a + bj)(c + dj) = (ac - bd) + j(bc + ad) = R1 + jR2$. Puis notre opérateur commun calcule les résultats finaux comme décrit dans l'équation 26. Cette nouvelle fonctionnalité ajoutée à notre opérateur

commun (mode **radix 2 FFT butterfly)** est sélectionnée par les même paramètres de contrôle que dans le mode FFT-SDF, c'est-à-dire par 'β' = 1 et 'δ' = 1. Donc, selon les équations développées précédemment (pour la partie réelle comme illustration), nous avons les valeurs des paramètres et les signaux de configuration suivants: $M = \delta = 1 \rightarrow \bar{M} = 0;\ \alpha = e + R1;\quad \alpha' = e - R1;\quad S_6 = \alpha = e + (ac - bd)\ et\ S_7 = \alpha' = e - (ac - bd)$. Où $(e\ \&\ R1)$ représentent les données d'entrées, et S_6 et S_7 sont les sorties dans le mode Radix 2 FFT. Nous pouvons obtenir le résultat de la partie imaginaire $(f\ \&\ R2)$ par le même mode.

La Figure III.18 illustre l'implémentation du mode FFT radix 2 et les ressources utilisées (de couleur bleu). Ce mode nécessite un multiplicateur complexe pipeline et deux opérations d'addition/soustraction.

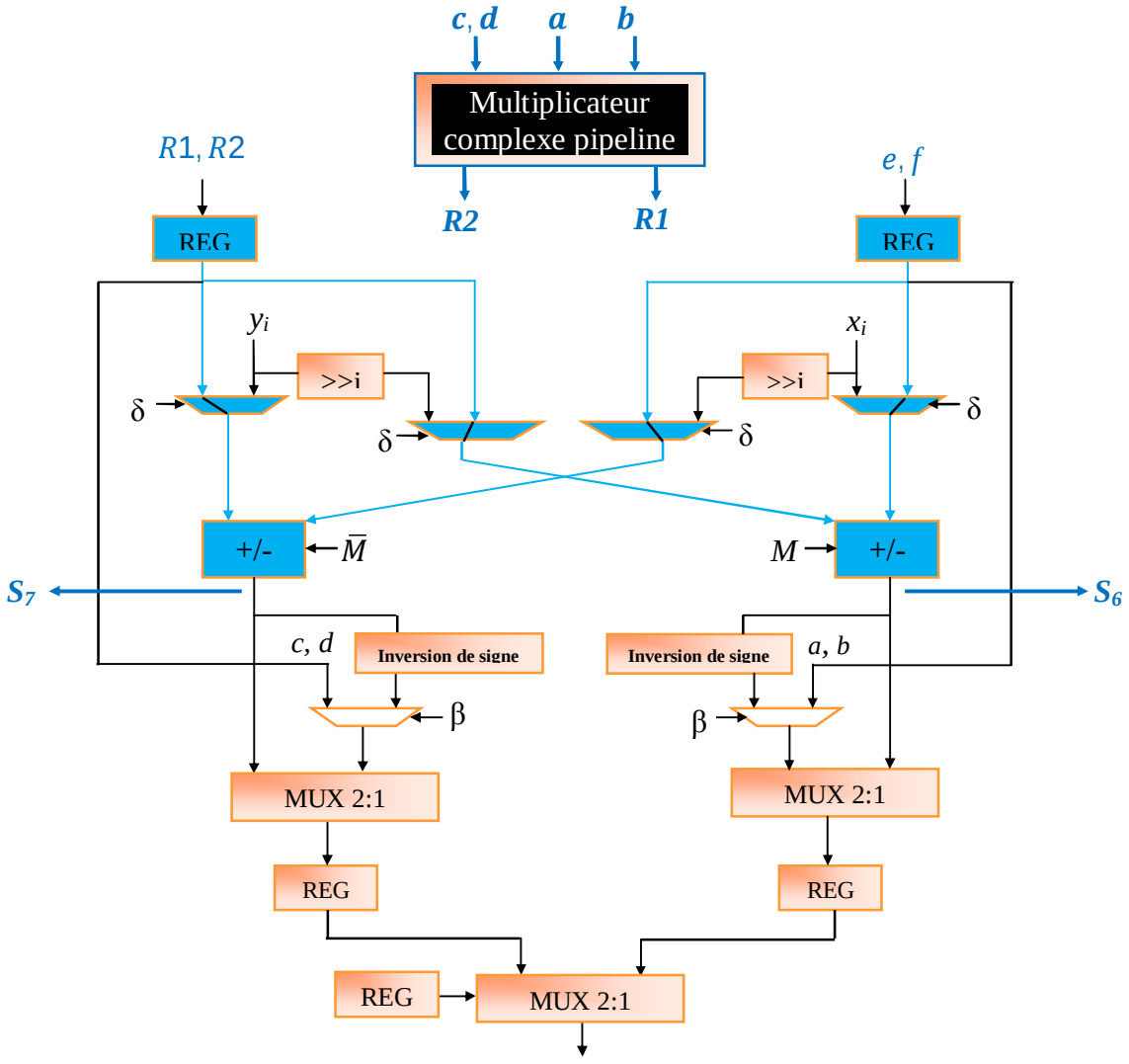


Figure. III.18. L'opérateur universel proposé pour l'unité de désétalement, processeur élémentaire PE de FFT papillon, la Cellule Cordic, et le Radix 2-FFT.

Cela nous permet de dire que notre architecture peut prendre aussi le calcul de radix 2 FFT sans aucun changement ni pour les paramètres de contrôle ni pour la structure de notre opérateur commun proposée dans la figure III.13. Néanmoins, on profite seulement de notre architecture pipeline pour la réalisation de la multiplication complexe. En plus, le Radix 2 butterfly représente une partie clé pour l'accomplissement des autres architectures des butterfly comme le radix 4 et le radix 8.

Cela nous permet de dire aussi qu'avec cette nouvelle fonctionnalité, notre opérateur commun apporte plus de flexibilité. En fait, il est devenu très convenable à implémenter non seulement les architectures FFT-SDF mais aussi d'autre architectures pipeline comme le FFT-MDC (qui est l'architecture la plus adaptée aux systèmes MIMO-OFDM), les architectures parallèles et les architectures à base de mémoire.

Revenons maintenant à l'unité ACS, celle-ci comme nous avons dit nécessite deux opérations d'additions et une opération de soustraction [3]. On peut encore profiter de l'architecture proposée pour l'opérateur commun (figure III.13) et pour le multiplicateur complexe pipeline (figure III.9) pour pendre en charge la fonctionnalité de l'unité ACS. L'opération de l'unité ACS peut se résumer par les équations suivantes [3]:

$$\begin{cases} S_8 = E_3 + E_4 \\ S_{10} = E_3 + E_4 - E_1 - E_2 = S_8 - S_9 \\ S_9 = E_1 + E_2 \end{cases} \quad (27)$$

Où les E_i *et* S_i sont les entres/sorties de cette unité.

L'architecture proposée pour la multiplication complexe est utilisée maintenant pour faire le calcul de $S_8 = E_3 + E_4$ et $S_9 = E_1 + E_2$ comme représenté dans la figure III. 19, puis l'opérateur commun fait le calcul final $S_{10} = S_8 - S_9$. Pour que cette nouvelle fonctionnalité ajoutée à notre opérateur commun fonctionne, il suffit de sélectionné 'δ' = 1 comme représenté dans la figure III.19. Cette figure montre aussi les ressources utilisées en couleur bleu pour l'unité ACS.

Là aussi on peut dire que notre architecture peut prendre le calcul nécessaire pour l'algorithme de Viterbi (unité BMC et unité ACS) sans aucun changement de la structure de notre opérateur commun proposée. Comme pour le FFT Radix 2 on a réutilisé l'architecture pipeline proposée pour le multiplicateur complexe.

L'architecture pipeline pour le multiplicateur complexe qui a été proposée initialement pour le radix-2/4/8 est devenue un opérateur commun pour d'autres fonctionnalités comme le Viterbi. Cela ouvre la voie aussi sur l'utilité de la cohabitation entre les différents opérateurs communs, dans le but de maximiser l'utilisation de ces opérateurs dans une implémentation SDR et ainsi d'obtenir d'autres fonctionnalités.

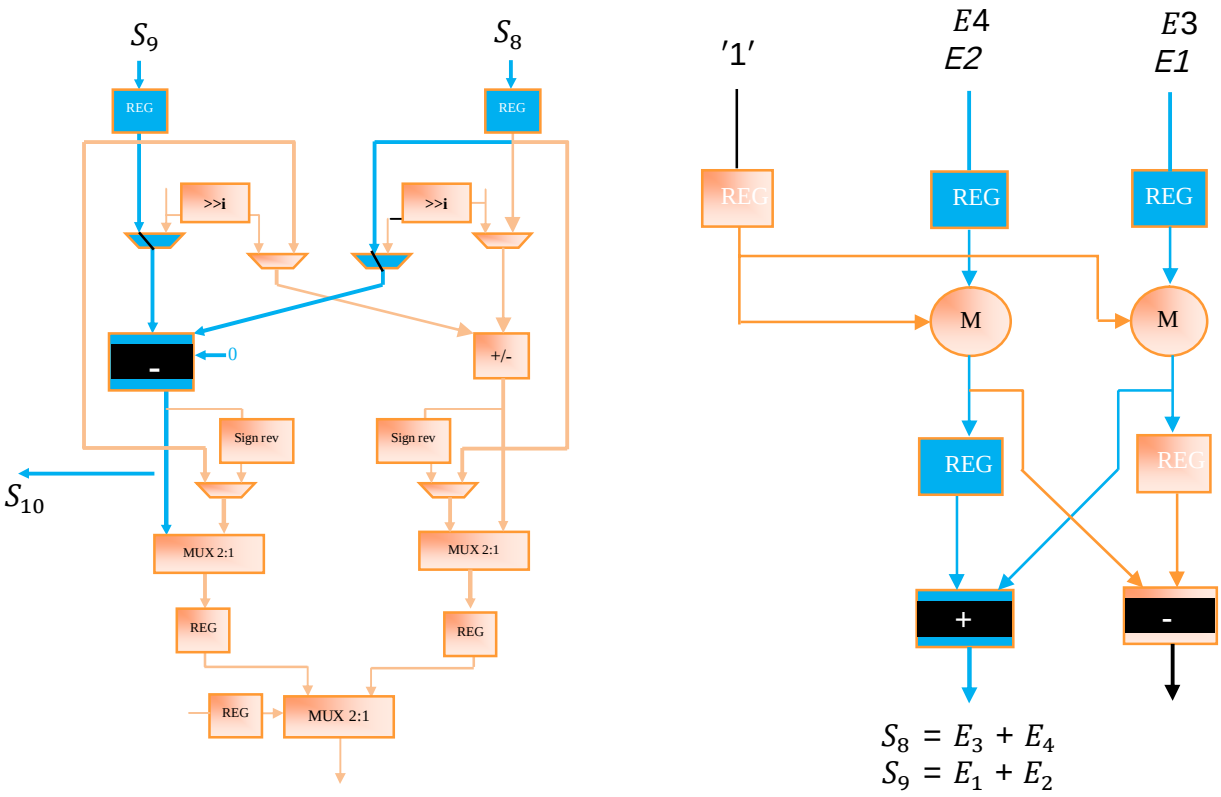
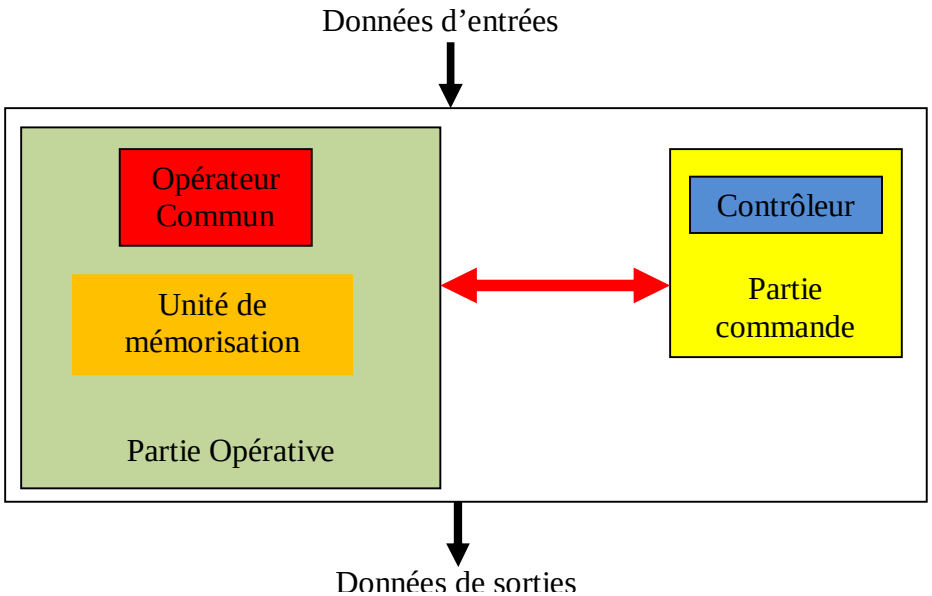


Figure. III.19. L'opérateur universel proposé adapté pour l'unité ACS de Viterbi.

* **S**i on combine la technique de l'opérateur commun avec la technique de synthèse architecturale qui est basée sur la réutilisation des opérateurs en ajoutant un contrôle sur le moment différent, nous pouvons également comparer notre approche versatile à la méthode basée sur le partitionnement temporel (temporal partitioning). Plus précisément, la synthèse architecturale est basée sur la réutilisation d'opérateurs à différents instants pour exécuter l'ensemble de l'algorithme [98][99][100]. Cette réutilisation est mise en œuvre par l'addition supplémentaire de ressources de contrôle. L'approche qui est basée sur le partitionnement temporel, permet de minimiser la taille du réseau (array size) d'un FPGA, en utilisant la reconfiguration dynamique (la reconfiguration dynamique consiste en l'exécution fractionnée et successive d'un algorithme par reconfiguration successive de la partie matérielle configurable) [101]. Dans le partitionnement temporel, il est nécessaire de replier temporellement différentes parties de l'algorithme à implémenter sur la même surface logique FPGA. Ceci va augmenter l'efficacité en silicium par le fonctionnement à la fréquence maximale autorisée sur la plus petite surface possible, laquelle satisfait la contrainte temps réel, mais au détriment de la consommation de puissance et parfois d'un accroissement du temps d'exécution.

Bien que le partitionnement temporel permette d'éviter un surdimensionnement des ressources nécessaires pour une application et reflète les contraintes de conception

imposées par la technologie cible, notre approche de combinaison implémente plusieurs algorithmes par les mêmes contraintes tout en maintenant la consommation de puissance [4]. La figure III.20 illustre un exemple de la mise en œuvre de la technique des opérateurs communs par la synthèse architecturale.



Figue. III.20. Illustration de la mise en œuvre de la technique des opérateurs commun par la synthèse architecturale.

En effet, l'approche de l'opérateur commun (OC), nous permet d'exécuter plusieurs fonctionnalités en utilisant les mêmes ressources par un simple téléchargement des paramètres. Avec l'approche de **synthèse architecturale (SA),** on est capable de réutiliser l'opérateur commun à différents instants. Par conséquent, la combinaison de ces deux approches peut apporter les avantages suivants:

- minimiser encore le nombre de partitions temporelles nécessaires pour une application exécutée en reconfiguration dynamique, et donc apporter plus de souplesse et faciliter l'approche de la Reconfiguration Partielle d'un FPGA.
- atteindre une meilleure utilisation des ressources et atteindre la meilleure Adéquation-Algorithme-Architecture (AAA) [102][103].
- cette combinaison nous permet aussi de réaliser une implémentation optimale qui s'adapte par rapport à l'environnement et qui satisfait les contraintes (temps réel, zone logique, consommation de puissance, etc…). Ceci permet de répondre aux critères d'intégration et d'embarquabilité.

Donc, notre approche proposée de la mise en œuvre des opérateurs communs basés sur la synthèse architecturale peut être utile pour la conception d'une application générique de communications multi standard [4].

* En outre, il est nécessaire de prendre en compte la consommation de puissance qui est un paramètre important dans les systèmes embarqués. En effet, dans les communications sans fil, les mobiles sont limités par leurs batteries d'alimentation. Par conséquent, la consommation de puissance est un facteur important qu'on ne peut pas négliger facilement. Dans notre cas d'étude, cette estimation de la consommation dépend de la technologie cible et l'environnement de travail comme la fréquence de travail et les ressources nécessaires pour chaque partie de la fonction commune [4]. En effet, les FPGAs sont devenus un choix intéressant par rapport aux ASICs, en raison d'un délai de mise sur le marché (time-to-market) plus court, une conception faible coût et la diminution de l'écart (gap) en termes de consommation de puissance. Ceci est dû grâce aux nouvelles générations des FPGAs de faible puissance et aux nouvelles techniques de routage [104],[105]. Bien que les ASICs ont été largement utilisés pour la conception des stations de base cellulaire, mais avec de longs cycles de conception, un coût élevé et fixe et le manque de flexibilité, ils deviennent impropres pour le prototypage et ne sont pas adaptés à un marché qui est en évolution [106].

* Un autre critère important est la complexité de la reconfiguration, la figure III.13 ((a), (b) et (c)) montre que l'architecture proposée peut fournir une commutation rapide entre les algorithmes et réduire la complexité de la reconfiguration par rapport aux travaux de [95].

8. Conclusion

Dans ce chapitre, on a proposé une nouvelle architecture générique et versatile pour l'unité de désétalement, le processeur élémentaire (PE) de FFT (en utilisant la méthode SDF), et la cellule Cordic. L'architecture proposée est basée sur la technique des opérateurs communs. Cette architecture est évolutive, scalable et peut être facilement adaptée pour traiter des mots de longueurs plus grandes. Pour démontrer l'efficacité de l'architecture proposée, nous avons fait la comparaison avec la technique Velcro et quelques opérateurs communs et des cellules reconfigurables disponibles dans la littérature. Toutes les architectures ont été implémentées sur la technologie FPGA Virtex-V pour une comparaison équitable. L'approche proposée améliore de façon importante le temps de traitement et montre une faible consommation de surface. En effet, les résultats de l'implémentation prouvent clairement que notre architecture proposée peut fournir une réduction de la surface logique totale nécessaire dans le FPGA, alors qu'elle fonctionne à une fréquence d'horloge supérieure à 300 MHz. En outre, l'implémentation de

l'architecture proposée nécessite peu de ressources matérielles par rapport à la technique Velcro et aux travaux précédents qui portent une certaine fonctionnalité commune. En effet, elle est capable de fournir un gain de surface logique (% des slices) sur FPGA (Xilinx Virtex- XC5VLX50T) de 5,7% , 29,25 et 25,69% pour un mots de longueurs de 8, 12 et 16 bits, respectivement. Ce gain de surface peut être amélioré, si on combine notre architecture qui est basée sur la technique de l'opérateur commun avec la technique de synthèse architecturale.

Ainsi, la conception de notre approche présente de bonnes capacités d'évolution, où on a montré que notre architecture peut facilement prendre en charge d'autres fonctionnalités. Ainsi, notre opérateur commun est défini pour effectuer des opérations élémentaires de traitement du signal indépendamment de la fonction qui l'exécute.

Enfin, et afin d'explorer des nouvelles architectures efficaces pour les terminaux mobiles, notre approche présente des performances attrayantes et peut être utilisée comme un opérateur commun dans les plus importantes fonctions pour plusieurs normes de transmission sans fil.