SYNTHÈSE LOGIQUE UTILISANT UN COMPILATEUR DE CELLULES COMPLEXES

Vincent BEAUDENON Alain GREINER

Département ASIM, laboratoire LIP6, Université Pierre et Marie CURIE
4, place Jussieu, 75252 Paris Cedex 05

{Vincent.Beaudenon, Alain.Greiner}@lip6.fr

RÉSUMÉ : Une des étapes essentielles dans la conception des Circuits Intégrés VLSI est la *Synthèse Logique*. Un outil de *Synthèse Logique* prend en entrée une description comportementale et génère en sortie un réseau de porte logiques interconnectées entre elles. Nous proposons une méthode de *Synthèse Logique* utilisant un *Compilateur de Cellules* qui génère dynamiquement les cellules au moment de la synthèse, de façon à éviter d'utiliser une bibliothèque de cellules pré-existantes.

Mots-clefs : Synthèse Logique de Circuits Intégrés VLSI, Compilateur de Cellules, Réseau Booléen.

I – INTRODUCTION

Un outil de *Synthèse Logique* prend en entrée une description comportementale, représentée par un *Réseau Booléen*, et génère en sortie un schéma en portes logiques interconnectées entre elles. En règle générale, un outil de *Synthèse Logique* utilise une bibliothèque de cellules pré-définies et pré-caractérisées pour le procédé de fabrication choisi. Chaque cellule réalise une Fonction Logique simple (AND, OR, XOR etc. à deux trois ou quatre entrées). Cette bibliothèque de cellules est fournie par le fabricant du circuit, et constitue la *«Cible»* de l'outil de *Synthèse Logique* qui doit décomposer les *Expressions Booléennes* attachées aux nœuds du *Réseau Booléen*, pour les «réécrire» en utilisant les cellules disponibles dans la bibliothèque.

Le LIP6 a développé un *Compilateur de Cellules Complexes* appelé C4 [1][3]. Cet outil prend en entrée une *Expression Booléenne* multi-niveaux (c'est à dire comportant un nombre quelconque d'opérateurs OR et AND et un nombre quelconque de niveaux de parenthésage de sous-expressions) et fournit en sortie le schéma en transistors de la cellule correspondante, ainsi que le dessin des masques permettant de graver le silicium.

En utilisant un *Compilateur de Cellules Complexes*, un outil de *Synthèse Logique* n'est plus limité à l'utilisation d'une biblitohèque de cellules prédéfinies. Sa principale tâche consiste à décomposer les *Expressions Booléennes*, attachées aux *nœuds* du *Réseau Booléen*, en sous-expressions plus simples, telles que chacune de ces sous-expressions soit directement synthétisable par le compilateur de cellules. Les deux principaux critères d'optimisation sont la surface de silicium occupé par le circuit synthétisé, et le temps de propagation entre une entrée et une sortie du circuit.

II – LE COMPILATEUR C4

II–1 – Présentation

Le compilateur C4 (Complex CMOS Cells Compiler) prend en entrée une expression booléenne multi-niveaux et fournit en sortie les différentes *vues* de la cellule qui réalise la fonction booléenne correspondante. Le dessin des masques respecte un gabarit imposé, de façon à ce que les cellules générées par C4 puissent être utilisées par n'importe quel outil commercial de placement/routage.

Pour ce faire, l'expression booléenne doit être de polarité définie : Si la polarité est *positive* l'expression ne comporte que des operateurs OR et AND. Si la polarité est *négative* l'expression comporte un seul inverseur à

son premier niveau. Par exemple $not(AND(a,(OR(AND(b,c),d))))$, dont les figures 1 et 2 donnent le schéma en transistors et le dessin des masques associés.

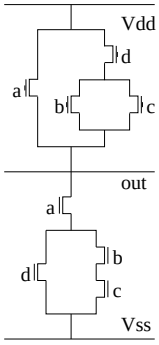

FIG. 1 – Réseau de Transistors généré par C4.

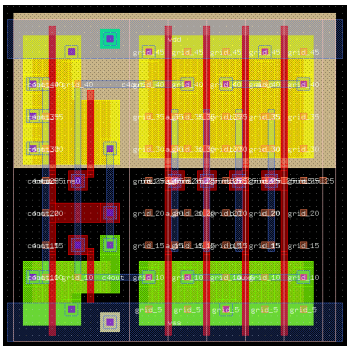

FIG. 2 – Dessin des Masques généré par C4.

L'avantage de cet outil est sa capacité à engendrer un grand nombre de cellules avec un coût de maintenance négligeable. De plus le compilateur C4 s'appuie sur une technique de dessin symbolique [2] qui garantit que les cellules générées peuvent être fabriquées dans n'importe quel procédé de fabrication CMOS.

Il faut cependant définir un modèle de caractérisation des cellules qui permettra d'orienter le processus de synthèse.

II–2 – Modèle de caractérisation

Une spécificité de C4 est que chaque cellule comporte en sortie un buffer (inverseur ou non-inverseur) qui garantit un comportement électrique acceptable, même pour une cellule très complexe . En contrepartie, cet élément induit un retard qui doit être pris en compte dans le modèle.

Les temps de propagation dépendent de l'entrée qui commute ainsi que du type de transition (front montant ou front descendant) :

– Pour une cellule non inverseuse :

$$TP_i^{UU} = T^{DU} + n_i \times T_N \qquad TP_i^{DD} = T^{UD} + p_i \times T_P$$

– Pour une cellule inverseuse :

$$TP_i^{UD} = T^{DD} + n_i \times T_N \qquad TP_i^{DU} = T^{UU} + p_i \times T_P$$

Dans ces formules, les paramêtres structurels n_i et p_i correspondent aux nombres de transistors N et P en série sur la branche contrôlée par l'entrée i. On détermine pour chaque procédé de fabrication visé les 6 paramètres technologiques T^{UU},T^{DD},T^{UD},T^{DU},T_N et T_P par simulation (*SPICE*).

Soit une cellule dont les entrées sont E_i et la sortie S. En s'appuyant sur ce modèle, il est possible de déterminer l'instant le plus tardif d'une transition sur la sortie (transition montante $U(S)$, ou transition descendante $D(S)$), à partir des informations correspondantes sur les entrées :

– Pour une cellule non inverseuse :

$$U(S) = max_i(U(E_i) + TP_i^{UU}) \qquad D(S) = max_i(D(E_i) + TP_i^{DD})$$

– Pour une cellule inverseuse :

$$U(S) = max_i(D(E_i) + TP_i^{DU}) \qquad U(S) = max_i(U(E_i) + TP_i^{UD})$$

III – SYNTHÈSE LOGIQUE À L'AIDE DE C4

Le problème posé est la transformation d'un réseau booléen en un réseau de cellules générées par C4. Un réseau booléen est un graphe orienté acyclique. L'idée ici est de traiter successivement chacun des nœuds du réseau booléen, en parcourant ce graphe orienté depuis les entrées primaires du circuit vers les sorties.

III–1 – Principe de décomposition d'un arbre d'opérateurs

A chaque nœud du réseau booléen est attachée une expression booléenne . On ne considère que des expressions booléennes «normalisées» : la polarité est définie (les opérateurs NOT ont été reportés vers les variables booléennes constituant le support de l'expression par application des lois de Morgan), et les constantes booléennes ont été éliminées. Les seuls opérateurs sont des OR et des AND d'arité quelconque.

CalculeTransN(E)
{
Si Littéral(E)
Renvoie 1;
Si Opérateur(E)=OR
Renvoie $\max_{F_i \in Fils(E)} CalculeTransN(F_i)$;
Si Opérateur(E)=AND
Renvoie $\sum_{F_i \in Fils(E)} CalculeTransN(F_i)$;
}

FIG. 3 – CalculeTransN

Une expression booléenne parenthésée multi-niveaux peut être représentée par un arbre d'opérateurs. Chaque nœud non-terminal de cet arbre est lui-même la racine d'un sous-arbre qui représente une sous-expression booléenne. Chaque nœud ne peut avoir comme fils que des feuilles terminales (variables booléennes) ou des nœuds associés à un opérateur booléen de type différent. L'arbre est donc *stratifié* avec une alternance entre les opérateurs AND et OR.

Si on parcourt cet arbre en remontant des feuilles vers la racine, les relations ci-dessous permettent de calculer de façon récursive, pour chaque nœud X de l'arbre les valeurs des paramètres n_X et p_X, qui représentent les nombres de transistors N et P en série qu'on obtient lorsque le sous-arbre correspondant à X est réalisé par une seule cellule C4. Pour calculer n_X on utilise l'algorithme de la figure 3, le même procédé est utilisé pour le calcul de p_X en permutant les opérateurs AND et OR.

Même s'il est en principe possible de réaliser n'importe quel arbre «normalisé» avec une seule cellule C4, il faut en pratique, pour des raisons électriques et topologiques, limiter la complexité des cellules, et en particulier imposer un nombre maximal de transistors en série. On définit pour cela les paramètres N_{max} et P_{max}.

Le principe de décomposition est le suivant : On parcourt l'arbre des feuilles vers la racine, en calculant les n_X et p_X. Dès qu'une des deux conditions $n_X > N_{max}$ ou $p_X > P_{max}$ est vérifiée pour un nœud X de l'arbre, le sous-arbre qui a X pour racine n'est plus réalisable en une seule cellule. (En revanche, chacun des nœuds fils du nœud X peut être réalisé en une seule cellule).

Il faut donc décomposer l'opérateur correspondant au nœud X, en utilisant l'associativité des opérateurs AND et OR, et en regroupant les nœuds fils de X de façon à constituer des sous-arbres dont chacun est réalisable en une seule cellule C4.

On démontre que le nombre de décomposition possible $d(n)$ d'un opérateur d'arité n se calcule ainsi :

$$d(1) = d(2) = 1,$$

$$d(n) = 1 + \sum_{i=1}^{i=n-2} \mathrm{C}_n^i \times d(i) \times d(n-i) + \sum_{i=2}^{i=n-2} \frac{1}{2} \times \mathrm{C}_n^i \times d(i) \times d(n-i)$$

Une recherche exhaustive de la solution optimale étant exclue, nous chercherons des solutions approchées, en utilisant le modèle de caractérisation décrit dans la section II–2.

III–2 – Décomposition Booléenne

Pour un nœud X correspondant à un opérateur OR ou AND possédant n opérandes Y_i, il existe deux grands modes de décomposition :

- La décomposition balancée qui assure un nombre équivalent de portes traversées entre chaque entrée Y_i et la sortie X.
- La décomposition non balancée qui minimise ce nombre pour les entrées Y_i les plus tardives.

Si on connaît les retards associés aux variables constituant les feuilles de l'arbre, on peut calculer, en utilisant le modèle décrit dans la section II–2, les valeurs des retards $U(X)$ et $D(X)$ associés à chaque chaque nœud X. Ce calcul est effectué en même temps qu'on parcourt l'arbre, des feuilles vers la racine, pour calculer les paramètres n_X et p_X de chaque nœud. Cette information sur les retards va permettre de choisir le type de décomposition le plus adapté.

On prend l'exemple d'un opérateur AND : On commence par rechercher le fils Y_0 pour lequel le retard U_0 est minimal. On ajoute une coupure sur Y_0 (c'est à dire de créer une cellule C4 pour le sous-arbre dont la racine

est Y_0) si cela suffit pour satisfaire la contrainte N_{max}. Sinon, on regroupe les fils Y_i de plus faible retard U_i, de telle sorte que $\sum_i n_i \leq N_{max}$, et on on crée un opérateur AND intermédiaire. On reprend ce processus tant que $N > N_{max}$.

On illustre ce principe à l'aide de la figure 4. Sur chaque nœud, on peut lire le nom du nœud, la fonction booléenne réalisée, le nombre de transistors N en série (lorsqu'on décompose un AND le nombre de transistors P n'est pas à prendre en compte) et le délai $U(X)$ (*idem*, les délais ($D(X)$ n'ont pas d'influence) associé. On prend comme contrainte $N_{max} = 4$. Elle n'est pas satisfaite au niveau du nœud q. Les lignes en pointillés indiquent les coupures. Chaque coupure Z induit une nouvelle cellule, et donc la traversée d'un buffer de sortie. Ainsi, on entroure chaque cellule isolée, dont la sortie attaquera une entrée du nœud père ramenant ainsi la valeur des n_i et p_i à 1.

Dans le premier arbre, $U_a < U_b < U_c < U_d$. La première étape de décomposition consiste à réunir les deux fils de plus faibles retards en atteignant la valeur limite de N. Un nouveau nœud, x, est créé. Cependant le nœud X n'est toujours pas réalisable. Il faut donc une seconde étape de décomposition.

Pour la seconde étape, $U_c < U_x < U_d$. Il donc est possible de satisfaire la contrainte N_{max} en ajoutant une coupure sur le fils de plus faible délai (c). Cette fois-ci, on choisira donc la décomposition balancée. L'arbre d'origine est ainsi décomposé en trois sous-arbres synthétisables par C4.

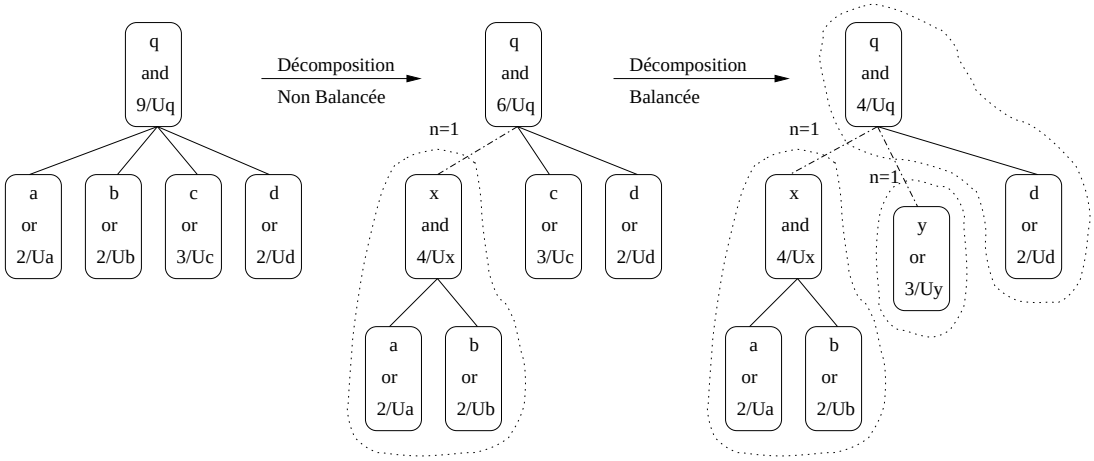

FIG. 4 – Exemple de décomposition Booléenne

IV – CONCLUSION

Le département ASIM du LIP6 a développé un prototype d'outil de Synthèse Logique, C4map [4], qui s'appuie sur les propriétés du compilateur C4, et sur les principes de décomposition énoncés dans la section III–2.

Ces travaux ont montré qu'il était possible d'inclure l'utilisation d'un compilateur de cellules complexes dans une chaîne de synthèse logique. Les principales perspectives de recherche à ce sujet visent à déterminer en quelles mesures une telle méthode peut compléter un outil de synthèse logique utilisant une bibliothèque de cellules pré-caractérisées.

RÉFÉRENCES

[1] N. Dictus, *Synthèse logique des circuits VLSI : Utilisation d'un compilateur de cellules complexes et optimisation des performances temporelles*, thèse UPMC, 1996. graphes de décision

[2] http ://www-asim.lip6.fr/alliance.html

[3] A. Greiner, V. Beaudenon, *Spécification et Réalisation d'un Compilateur de Cellules Complexes*, Stage d'Application ECL, 1999.

[4] A. Greiner, V. Beaudenon, *Synthèse Logique Utilisant un Compilateur de Cellules Complexes*, Stage de DEA, 2001.