Description comportementale en langage C pour la synthèse haut niveau

François Donnet
Laboratoire ASIM/LIP6
Université Pierre et Marie Curie (UPMC)
4, place Jussieu, 75252 Paris cedex 5

E-mail: `francois.donnet@asim.lip6.fr`

Résumé

Cet article présente les avantages d'une description comportementale en langage C pour la synthèse de coprocesseurs dans les systèmes intégrés. Au cours des dernières années, la synthèse de circuits a progressé vers un plus grand niveau d'abstraction, mais les outils existants utilisent souvent des langages d'une très fine granularité dont les caractéristiques sont héritées de la conception VLSI bas niveau. Utiliser un langage plus généraliste et permissif correspond mieux aux besoins du concepteur de modèles haut niveau. Le langage C bénéficie d'une large diffusion et d'une très forte adaptabilité.

1. Introduction

Cet article repose sur l'expérience du simulateur *CASS* -Cycle Accurate System Simulator- prenant en entrée des descriptions C et pouvant les simuler dans un ensemble hardware/software. L'outil *UGH* -User Guided High level synthesis- s'inscrit dans la continuité en prenant en entrée les modèles C simulés et en produisant une architecture au niveau RTL. Ces outils font partie de l'ensemble de CAO `disydent` développé au LIP6.
La première partie traite du mécanisme de Système on Chip. La section deux détaille le fonctionnement de la synthèse. La dernière partie décrit l'entrée C supportée par la synthèse.

2. Vue générale

Le concepteur conçoit en langage C son ou ses modèles, qui ne peuvent communiquer avec l'extérieur que par des lectures et des écritures bloquantes. Il peut choisir de ne synthétiser qu'un seul de ses modèles.

Interface du modèle Les Fifos étant les seules entrées/sorties du modèle, la synthèse peut donc s'affranchir de la partie *déclarations des ports* pour déterminer son interface. Il suffit donc de parcourir les accès externes de la description comportementale pour déterminer un connecteur, son sens et sa taille. Les Fifos ne font pas partie du modèle et ne sont donc pas synthétisées avec le modèle.

Synchronisation entre modèles C'est la principale mission des Fifos dont les états sont pris en compte lors de la simulation et de la synthèse. Les accès externes sont matérialisés par des fonctions C prédéfinies symbolisant une lecture ou une écriture.

3. Synthèse

L'architecture du modèle est un chemin de données contrôlé par une machine à états finis. Le chemin de données contient les registres, la mémoire locale et les opérateurs logiques ou arithmétiques qui sont interconnectés par des bus ou des multiplexeurs.

Formalisation de la description Dans la première phase, le C est compilé en un modèle formel basé sur les réseaux de Pétri temporels . Le réseau de Pétri temporel généré à partir du code C du Plus Grand Diviseur Commun (PGCD) est montré dans la figure 2.

```c
/* tailles des signaux utilisés */
typedef int FIFO_IN32;
typedef int FIFO_OUT32;
typedef int int32;

/* fonctions d'accès aux Fifos pour la simulation et la synthèse */
extern void readFifo ( FIFO_IN32* fifo, int32* valeur );
extern void writeFifo( FIFO_OUT* fifo, int32* valeur );

/* thread posix du PGCD pour la simulation et la synthèse */
void hcf() {
    FIFO_IN32    Din;
    FIFO_OUT32 Dout;
    int32          Ra;
    int32          Rb;

    while( 1 ) {                  /* process */
        readFifo( &Din,   &Ra );
        readFifo( &Dout, &Rb );

        while( Ra != Rb ) {
            if ( Ra < Rb ) Rb = Rb-Ra;
            else           Ra = Ra-Rb;
        }

        writeFifo( &Dout, &Ra );
    }
}
```

FIG. **1. Description C du PGCD.**

Machine à états Les transitions entre les places sont dûes à la séquentialité des instructions du langage C. Les branchements conditionnels dans le réseau de Pétri temporel sont issus des branchements conditionnels de la description. Ces transitions sont soumises à des conditions de garde complémentaires et exclusives.

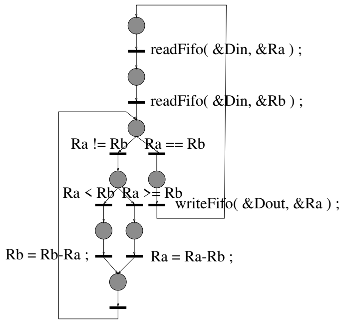

FIG. **2. Le PGCD en réseau de Pétri.**

Chemin de données Le chemin de données est contrôlé par la machine à états grâce aux multiplexeurs présents avant chaque entrée d'instance. Les expressions contenues dans les transitions sont traduites par une suite d'opérateurs arithmétiques ou logiques dans le chemin de données (cf. l'exemple sur le PGCD figure 3). Le chemin de données est construit en fonction de la description C et des contraintes matérielles définies par le concepteur.

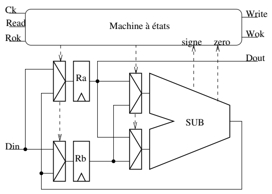

FIG. **3. Synthèse du PGCD.**

4. Sous ensemble du langage C

4.1. Les types

On peut considérer qu'il existe deux types de variables : les variables entières symbolisant des registres et les tableaux représentant une RAM.

Entiers A l'instar du VHDL, il est nécessaire de pouvoir définir des types au bit près. Ceci afin de ne pas générer des opérateurs plus grands que nécessaire. Via la définition de deux types génériques, il est possible de couvrir toutes les tailles de vecteurs : les entiers signés *int* et les entiers non-signés *uint*. Le nombre de bits de la variable est indiqué textuelement dans le nom du type.

Ports Les Fifos ne sont pas assimilées à des variables puisqu'ils représentent les ports du modèle. La création d'un type de port particulier se fait grâce aux types génériques *FIFO_IN*, *FIFO_OUT* et *FIFO* représentant respectivement un port entrant, sortant et à double sens.

Tableaux Seuls les tableaux à une dimension peuvent avoir une réalité matérielle. Ils peuvent représenter des bancs de registres ou des RAMs. On pourrait cependant agréger les dimensions pour n'en faire plus qu'une seule.

Pointeurs Les pointeurs en tant que variables n'ont aucun sens et sont donc interdits. Seul l'opérateur de référencement est autorisé en paramètre de fonction. Il signale que le paramètre est de type *inout*, c'est à dire que la fonction crée un effet de bord sur ce paramètre.

4.2. Les instructions

Une instruction C est une primitive ou une expression suivie d'un point virgule.

Primitives Tous les branchements du C s'expriment dans le réseau de Pétri par des sauts conditionnels vers une place destination. Les boucles C peuvent toutes se traduire par un branchement conditionnel. Les primitives de branchement sont donc toutes synthétisables : while , do , goto, switch case, break, continue, return, if else.

Expressions D'un point de vue séquentiel, toute les opérations dans une expression sont faites dans un même cycle. Le chemin de données issu des expressions, doit donc comporter suffisament de matériel pour pouvoir régler cette simultanéité. Pour cette raison, les effets de bord interdépendants dans une expression ne donnent pas de résultat prédictible. Une pré ou post-incrémentation est remplacée par une simple affectation placée respectivement avant ou après l'expression.

Fonctions La fonction principale possède une boucle infinie qui symbolise le process. Les autres fonctions sont la plupart du tant destinées à être incluses dans la fonction principale. Elles servent uniquement à structurer le programme. Il existe aussi des fonctions prédéfinies. Les principales sont les fonctions d'accès aux ports externes (*writeFifo*, *readFifo*). Elles prennent en paramètres une Fifo et un registre source ou destination. D'autres fonctions servent ainsi de boîtes noires auxquelles la synthèse associe des opérateurs précis.

Conditions Les opérateurs logiques synthétisés sont des opérateurs bit à bit, ils ne sont pas équivalents aux opérateurs booléens du C. On compare l'expression booléenne à la constante zéro pour ramener l'expression sur 1 bit. En effet sur 1 bit, les opérateurs booléens et bit à bit ont le même comportement. De plus, les opérateurs booléens binaires sont préemptifs, il n'exécute la deuxième opérande que si c'est nécessaire. S'il y a un effet de bord dans la deuxième opérande, le hardware n'aura pas le même comportement que le software puisque l'effet de bord sera toujours effectif quelque soit la valeur de la première opérande. Il faut alors séparer les deux conditions en deux tests distincts.

5. Conclusions

L'outil de synthèse *UGH* et de simulation *CASS* créent un environnement C alternatif au VHDL. Le lien entre simulation et synthèse s'en trouve donc renforcé. Des travaux sont menés actuelement sur un modèle synthétisable décrivant un décodeur de Huffman.