RENPAR'14 / ASF / SYMPA
Hamamet, Tunisie, 10 – 13 avril 2002

CoDeNios: Un Outil de Codesign pour le Processeur Nios

Yann Thoma et Eduardo Sánchez

Laboratoire de Systèmes Logiques
Ecole Polytechnique Fédérale de Lausanne,
IN - Ecublens,
1015 Lausanne - Suisse
{yann.thoma , eduardo.sanchez}@epfl.ch

Résumé
Le codesign, consistant en une séparation des tâches entre le matériel et le logiciel, pose différents problèmes, notamment le partitionnement logiciel/matériel, l'interfaçage entre le logiciel et le matériel et le test de tels systèmes. CoDeNios, l'application présentée dans cet article, est un outil d'aide pour les deux premiers problèmes: le partitionnement et l'interfaçage. A partir d'un simple programme écrit en C pour le processeur Nios, un développeur peut effectuer un partitionnement au niveau des appels de fonctions. L'interface matériel/logiciel est automatiquement générée, évitant au développeur de se soucier de la création d'un protocole. Dans ce document, nous allons tout d'abord présenter brièvement le processeur Nios, puis le système conçu pour assurer une communication efficace entre les différents processeurs et les modules matériels. Enfin nous analyserons les possibles améliorations de CoDeNios.

Mots-clés : codesign, FPGA, système multi-processeurs, processeur Nios

---

1. Introduction

Le codesign [1] consiste en la conception globale d'un système, avec répartition des tâches entre le matériel et le logiciel afin d'optimiser les performances. La réalisation de tels systèmes pose, de par la complexité accrue par la conception conjointe, pose de nouveaux problèmes, notamment:

- Le partitionnement logiciel/matériel [2], qui consiste à choisir quelles parties d'une tâche seront calculées par logiciel, et lesquelles le seront par des modules matériels.
- L'interfaçage entre le matériel et le logiciel, visant à faire communiquer le processeur avec le matériel, c'est-à-dire à lancer un calcul et passer des paramètres.
- Les tests servant à valider les modèles mettant en jeu une partie logicielle et une matérielle, décrites dans 2 langages différents (C et VHDL par exemple). En effet, en plus du fonctionnement correct de chaque module, la synchronization de l'ensemble doit être assurée. Dans ce cadre, de nouveaux langages dérivés de C sont apparus: SystemC [3] et HandleC [4], permettant la description logicielle et matérielle d'une manière homogène.

Dans l'optique de faciliter la réalisation d'applications faisant intervenir un processeur ainsi que des modules matériels, CoDeNios a été créé. Cet outil logiciel est basé sur le processeur Nios, qui est proposé par Altera comme un soft core configurable (taille des instructions, nombre de registres, ajout de périphériques, ...) pouvant être inclus dans n'importe quel design. Notre outil facilite le partitionnement matériel/logiciel en proposant à l'utilisateur, à partir d'un programme C écrit pour le processeur Nios, d'implémenter certaines fonctions en matériel. L'interface entre le processeur et les modules matériels est alors entièrement générée et le code C est modifié afin d'appeler les fonctions matérielles. Pour proposer plus de souplesse, une fonction C peut également être calculée par un processeur esclave, lui-même ayant la possibilité de contrôler d'autres modules matériels.
Par convention, dans la suite du document, nous appelerons une fonction calculée par matériel ou par un processeur esclave une *fonction codesign*.

Paramètre	Valeur
type	RISC
pipeline	4 niveaux (5 pour les load/store)
taille des données et des instructions	16 ou 32 bits
nombre de registres	128, 256 ou 512
fréquence	< 50 MHz
place	environ 26000 portes en version 16 bits

TAB. 1 – *Caractéristiques du processeur Nios*

2. Le Processeur Nios

En phase avec les besoins grandissants en applications mêlant logiciel et matériel, Altera propose deux processeurs, que l'on peut ajouter à leurs FPGAs:

– Un processeur **ARM** en hard, couplé à une partie reconfigurable, le tout sur un seul et même circuit.
– Le processeur **Nios** [5], un soft core configurable donnant la possibilité de l'implémenter à l'intérieur de n'importe quel design.

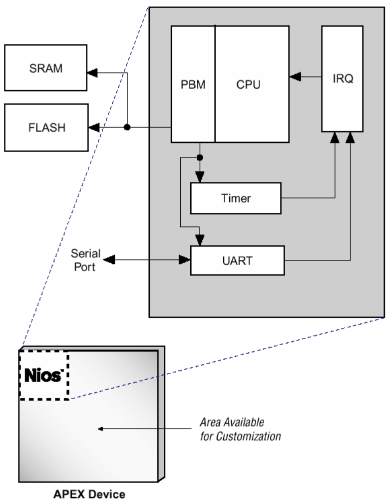

FIG. 1 – *Processeur Nios*

Notre projet utilise la solution logicielle, c'est-à-dire un Nios (cf. figure 1), ainsi qu'un FPGA d'une complexité permettant d'y placer plusieurs processeurs. Le Nios, dont les caractéristiques principales sont décrites dans la table 1, peut être généré en mode 16 ou 32 bits grâce à une interface graphique intuitive. Son nombre de registres est également paramétrable, de même que ses différents périphériques. Ces derniers sont de différentes natures: mémoire RAM/ROM, UART pour gérer une ligne série, timer, PIO, contrôleur de disque IDE, etc. Hormis ces périphériques implémentés par Altera, l'utilisateur a la possibilité d'en ajouter qu'il aura lui-même écrits. Ils sont vus par le Nios comme de simples emplacements mémoires accédés autant en lecture qu'en écriture avec la possibilité de gestion d'interruptions.

Altera propose, à partir du Nios, un kit de développement autour d'une carte comprenant un circuit reconfigurable APEX20KE200, ainsi que divers périphériques tels qu'un écran LCD, deux affichages hexadécimaux, etc. Ce kit rend la création de systèmes composés d'un processeur et de divers modules matériels sur un circuit reconfigurable possible, et même aisée. L'APEX20KE200 contient 200'000 portes et 106'496 bits de mémoire configurables permettant de placer des designs complexes pouvant contenir jusqu'à 3 processeurs Nios ainsi que leurs mémoires RAM. Une ligne série, gérée par un UART, est utilisée par le Nios pour communiquer avec un PC connecté à la carte de développement. Il est alors possible, via un simple terminal, d'envoyer des caractères tapés par l'utilisateur et d'afficher des chaînes de caractères à l'écran.

Le processeur est configurable en 16 ou 32 bits. La version 32 bits prend 20% des cellules disponibles sur l'APEX20KE200, alors que la version 16 bits n'en utilise que 12%. Afin d'avoir la possibilité de placer un maximum de processeurs sur la carte, il a été choisi de travailler avec l'option 16 bits.

Il est à noter que chaque processeur possède une mémoire ROM de démarrage le forçant à attendre que le code à exécuter lui soit envoyé par la ligne série. Ce code est alors stocké dans une RAM qui est la mémoire SRAM de la carte pour le processeur principal, ou un bloc de 1Ko onchip pour les esclaves.

3. CoDeNios

Le partitionnement d'un système en matériel/logiciel vise à accélérer une tâche en profitant de la vitesse du matériel. Il faut alors avoir le moyen d'évaluer le temps d'exécution d'une fonction en matériel ou logiciel. A cette fin, l'outil CoDeNios permet d'effectuer un partitionnement au niveau des appels de fonctions. La difficulté vient ensuite dans la réalisation d'une interface entre logiciel et matériel. En effet, un protocole doit être mis au point afin d'assurer une communication sans faille entre les différents modules. C'est pourquoi CoDeNios, après que l'utilisateur ait défini une partition, génère une interface complète qui implémente un protocole entre les modules. L'utilisateur n'a qu'à écrire un fichier VHDL pour chaque fonction calculée par matériel.

3.1. Interface graphique

Dans le but de rendre l'utilisation de CoDeNios la plus intuitive possible une interface graphique (cf. figure 2) est proposée au développeur. Après l'ouverture d'un fichier C, les fonctions retournant `void` sont listées dans le rectangle représentant le processeur principal. Ensuite, il est possible de déplacer une fonction pour en faire un module hardware. Un simple clic sur celui-ci le transforme en un processeur esclave qui peut alors lui-même contrôler d'autres modules. Pour chaque fonction calculée en-dehors d'un processeur, tous ses paramètres sont listés, et pour ceux passés en C par référence (`int *a`), il est possible de choisir s'ils doivent effectivement l'être par référence (un pointeur permet d'accéder une adresse dans la mémoire partagée) ou par valeur (cas d'un paramètre en sortie, par exemple). Leur sens est également paramétrable en entrée, sortie ou entrée-sortie.

Après que tous les choix aient été faits, des boutons permettent de générer les fichiers VHDL, les fichiers C, lancer la compilation du code C, la synthèse du VHDL, et enfin le placement-routage. Le développeur n'a plus qu'à configurer la FPGA grâce au logiciel Quartus fourni par Altera, puis, grâce encore à un bouton, à lancer l'exécution des programmes dans les différents processeurs. Un terminal s'ouvre alors, instaurant un dialogue entre l'utilisateur et le processeur principal.

3.2. Génération de fichiers

Comme expliqué précédemment, CoDeNios génère tous les fichiers VHDL nécessaires à la communication entre les différents processeurs et les modules matériels. Dans le cas d'une communication de processeur à processeur, l'utilisateur n'a pas à intervenir, le code C étant modifié et le VHDL généré. En revanche, si l'option d'une fonction matérielle a été choisie, il est de son devoir d'écrire son comportement en VHDL. Un fichier lui est généré, contenant la déclaration de l'entité ainsi que le squelette de l'architecture.

L'exemple de la fonction PGCD (cf. figure 3), calculant le plus grand diviseur commun de deux nombres, illustre le type de fichier généré pour le module matériel. La fonction C a été déclarée ainsi:

```
void pgcd(int a,int b,int *c).
```

Le protocole à respecter par le développeur est très simple. Un signal `start` passe à `'1'` pendant un coup d'horloge, indiquant que les opérandes sont chargés. Le module peut commencer à effectuer son traitement. Pour chaque paramètre en sortie, un signal `load_x` est utilisé comme signal de chargement d'un registre externe au module. Lorsque le traitement et terminé et que tous les paramètres de sortie

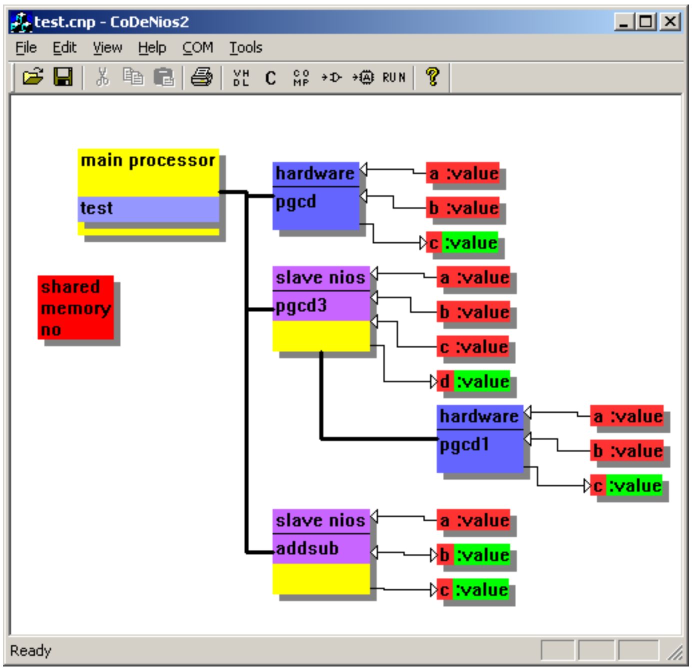

FIG. 2 – *Interface graphique*

sont chargés, le signal `done` doit être placé à `'1'` pendant un coup d'horloge pour signifier au système que le calcul est terminé.

3.3. Parallélisme

Sur le plan des fichiers C, pour chaque fonction devant être calculée par un esclave (processeur ou matériel), son appel est remplacé par la partie du protocole faisant communiquer le processeur avec un contrôleur du système. Poursuivant l'exemple du PGCD, `pgcd(a,b,&c)` sera remplacé par:

```
hcall_pgcd(a,b,&c);hwait();
```

`hcall_pgcd()` ne fait que lancer le calcul de la nouvelle fonction matérielle, tandis que `hwait()` attend qu'elle se termine et récupère les paramètres en sortie. Cette séparation appel-terminaison permet de tirer profit du parallélisme inhérent aux systèmes matériels. En effet, il est possible d'appeler plusieurs fonctions n'ayant pas de dépendances, puis d'attendre qu'elles se terminent toutes. Dès lors, en appelant en premier la fonction la plus gourmande en temps d'exécution, le temps total peut être grandement réduit (cf. figure 4).

3.4. Evaluation du temps d'exécution

Comme présenté précédemment, un des aspects important de CoDeNios est sa capacité à évaluer le temps d'exécution d'une fonction logicielle et matérielle. Pour ce faire plusieurs compteurs sont automatiquement placés dans le système lors de sa génération. Un compteur global 32 bits est directement accessible par le processeur principal. Il est remis à 0 au reset soft de la FPGA et compte de nombre de coups d'horloges passés. Parallèlement, chaque module codesign contient un compteur (également 32 bits) chargé d'évaluer le temps d'exécution de la fonction, sans tenir compte de l'appel ni du passage des paramètres. Sa valeur est récupérée par le processeur maître à la suite des paramètres de sortie.
La valeur du compteur global est accessible via une fonction `void GetTime(time_t *t)` alors que les compteurs des modules le sont par `void GetFuncTime(int FUNCID,time_t *t)`. Elles sont mises à disposition du développeur dans un fichier C généré qui contient également les procédures responsables des appels de fonctions codesigns.

```
library ieee;use ieee.std_logic_1164.all;

entity pgcd is port (
    a_in: in std_logic_vector(15 downto 0);      -- input parameter
    b_in: in std_logic_vector(15 downto 0);      -- input parameter
    c_out: out std_logic_vector(15 downto 0);    -- output parameter
    load_c: out std_logic;                       -- put it at '1' to
                                                -- load the output
                                                -- parameter
    clk:    in std_logic;   -- clock signal
    rst:    in std_logic;   -- reset, '0' active
    start:  in std_logic;   -- '1' during one clock cycle to begin the treatment
    done:   out std_logic   -- put it at '1' during one clock cycle when the
                            -- treatment is finished
); end pgcd;

architecture struct of pgcd is

        type state_type is (s0,s1);
        signal state,n_state: state_type;

begin
        process(state,start)
        begin

            -- default output values
            done<='0';
            c_out<=(others=>'0');
            load_c<='0';
            n_state<=state;

            case state is
            when s0=>
                if start='1' then
                    n_state<=s1;
                end if;
            when s1=>
                done<='1';
                load_c<='1';
                n_state<=s0;
            end case;
        end process;

        process(rst,clk)
        begin
            if rst='0' then
                state<=s0;
            else
                if clk'event and clk='1' then
                    state<=n_state;
                end if;
            end if;
        end process;

end struct;
```

FIG. 3 – *Fichier VHDL généré pour la fonction PGCD*

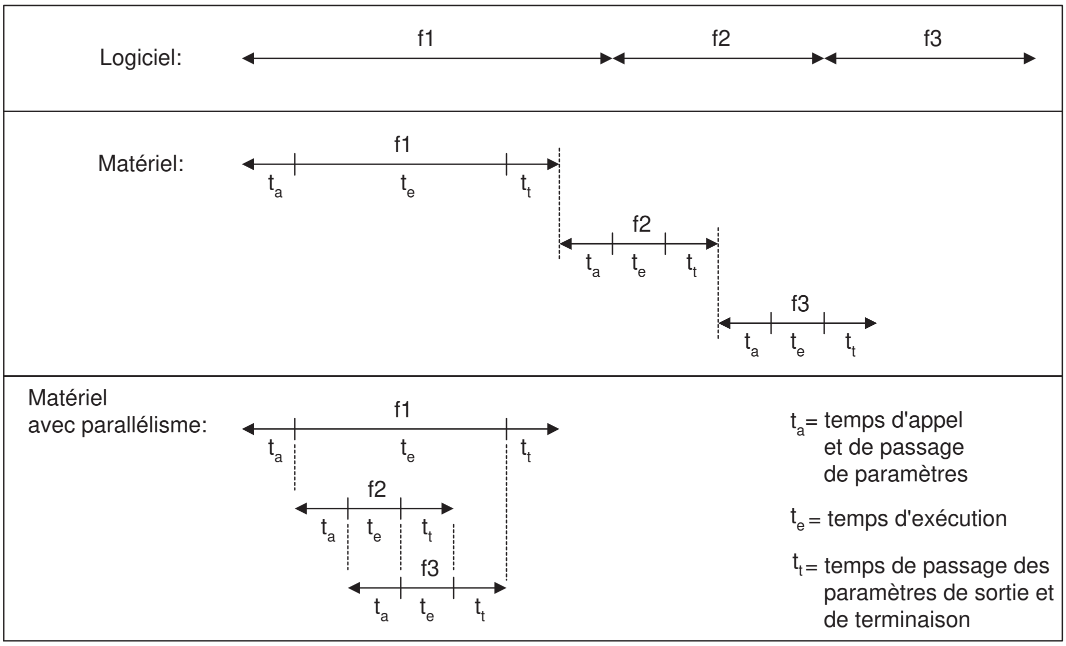

FIG. 4 – *Différents types d'exécutions de fonctions*

3.5. Mémoires

Ayant la possibilité de créer un design multi-processeur, plusieurs mémoires sont utilisées. Le processeur principal place son code exécutable dans la mémoire SRAM de 1Mo présente sur la carte de développement. En revanche, les esclaves ne disposent chacun que d'une RAM de 1Ko implémentée dans la FPGA. Cette limitation est nécessaire car le nombre de blocs de mémoire n'est pas infini (52 blocs de 2056 bits), et une taille plus importante ne permettrait pas de placer plus de 3 processeurs dans un design. Une mémoire additionnelle de 1Ko peut être utilisée afin de passer des tableaux en paramètres des fonctions codesign (en passant un pointeur). Cette mémoire est partagée entre le processeur principal et tous les modules codesign (matériels ou logiciels). La gestion de cette RAM est faite d'un simple arbitrage par priorité où chaque module a une priorité différente.

4. Performances

Le calcul des performances d'un design réalisé avec l'aide de CoDeNios est dépendant de l'implémentation matérielle écrite par le développeur pour chaque module matériel. En effet, le temps d'exécution total dépend du temps de passage des paramètres ainsi que de l'appel de la fonction et du temps d'exécution proprement dit. Le temps de passage des paramètres est très court, puisqu'il ne s'agit que d'une écriture par paramètre d'entrée et d'une lecture pour un en sortie. En revanche, l'appel et l'attente de terminaison du calcul utilisent 113 coups d'horloge. Dès lors l'efficacité des modules matériels est déterminante. Il est évident qu'une simple addition sera plus lente si elle est implémentée en matériel, de par la latence de 113 coups d'horloge, alors que le calcul d'une suite sera sûrement plus avantageux en matériel.

Les procédures de calcul de temps d'exécution fournies à l'utilisateur sont donc déterminantes dans le sens où elles permettent une évaluation de la solution logicielle aussi bien que de la matérielle.

Il faut également noter que la performance d'un système dépend grandement du parallélisme imposé par le développeur. Si plusieurs fonctions peuvent être calculées en même temps, alors le temps d'exécution total peut être grandement réduit (cf. figure 4).

5. Conclusion

Nous avons présenté l'outil de développement d'applications codesign CoDeNios. Cet outil pédagogique aide le développeur à effectuer un partitionnement matériel/logiciel d'un programme C, et génère tout l'interface entre le matériel et le logiciel. Une architecture multi-processeurs également réalisable, évitant à l'utilisateur la tâche d'interfacer les différents processeurs.

CoDeNios, dans sa version actuelle, peut servir à l'enseignement, évitant aux étudiants la lourde tâche de synchronisation entre le matériel et le logiciel, et leur permet de pouvoir rapidement tester des modules matériels. Afin d'évaluer leur travail, ils disposent de fonctions C qui récupèrent les valeurs de différents compteurs. De ce fait, il leur est possible de comparer leur réalisation matérielle avec la solution purement logicielle.

Outre le but pédagogique de CoDeNios, une utilisation industrielle est possible. En effet, grâce à cet outil, la rapidité de création d'un système complet mêlant logiciel et matériel, ainsi que son évaluation aisée, impliquent un temps de développement très faible. Dans cette optique, un outil permettant de générer du VHDL à partir de C est en cours de développement. Il sera capable de transformer un sous-ensemble de C (if, for, while, +, -, *, /) ne travaillant qu'avec des entiers 16 bits en un pipeline matériel. Intégré à CoDeNios il comblera la non-automatisation de l'écriture des modules matériels. Dès lors le processus de développement sera entièrement automatisé à partir des choix du développeur.

Bibliographie

1. G. De Micheli et R. K. Gupta – Hardware-Software Co-Design – in Readings in hardware software co-design. G. De Micheli, R. Ersnt et W. Wolf (Eds.). The Morgan Kaufmann Series in Systems on Silicon, pp.30-44, Juin 2001.
2. A. Kalavade et E. A. Lee – The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling, and Implementation-bin Selection – in Readings in hardware software co-design. G. De Micheli, R. Ersnt et W. Wolf (Eds.). The Morgan Kaufmann Series in Systems on Silicon, pp.293-313, Juin 2001.
3. http://www.systemc.org
4. http://www.celoxica.com
5. http://www.altera.com