SMC - TP1 : Prototypage Virtuel avec SoCLib

Encadrant : Franck Wajsbürt

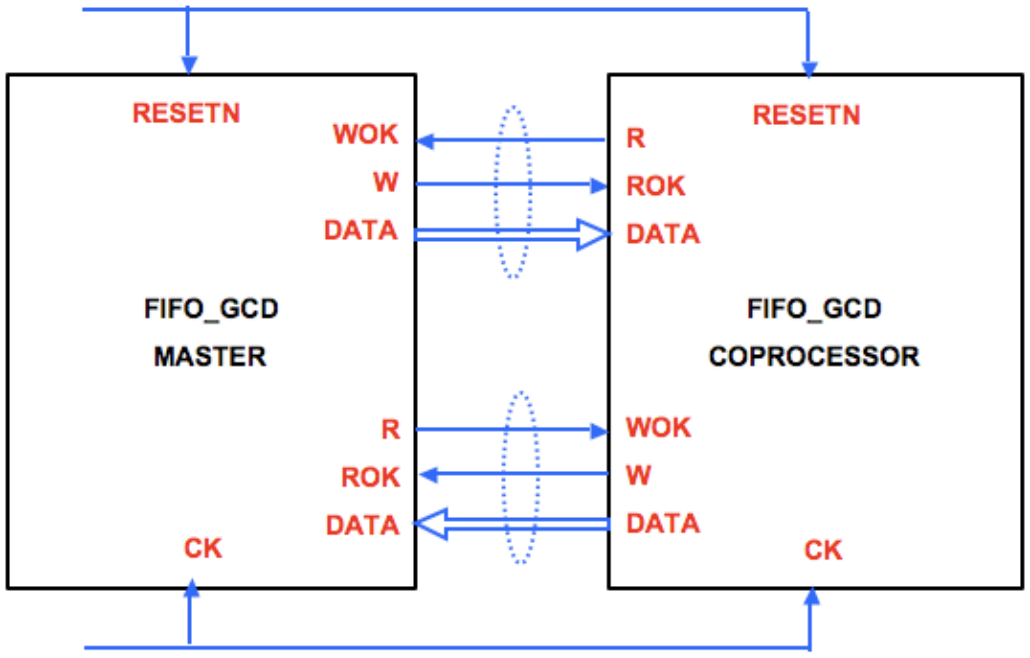
- Wilfrid LYAUTEY - 21310245 - wilfrid.lyautey@etu.sorbonne-universite.fr
- Jorge MENDIETA - 21304469 - jorge.mendieta_orozco@etu.sorbonne-universite.fr
- Victor SPEHAR - 21309680 - victor.spehar@etu.sorbonne-universite.fr

1 Objectif

L'objectif de ce premier TP est d'illustrer les principes de la modélisation SystemC au niveau d'abstraction CABA (Cycle Accurate, Bit Accurate). Ce type de modélisation s'appuie sur la théorie des automates d'états synchrones communicants (CFSM), et permet d'utiliser des techniques d'ordonnancement statique pour accélérer la simulation.

2 Architecture matérielle

L'architecture matérielle à modéliser comporte 2 composants en parallèles : un coprocesseur câblé qui calcule le PGCD et un composant chargé de transmettre les opérandes au coprocesseur et de récupérer le résultat. Les 2 composants communiquent par fifo.



Architecture Matérielle

Le producteur écrit dans la fifo lorsqu'elle n'est pas pleine et le consommateur lit lorsqu'elle n'est pas vide. L'avantage de ce protocole est que les 2 composants se comportent comme des automates de Moore ce qui résulte d'un comportement simple et prévisible.

2.1 Question

Compte-tenu de l'algorithme de calcul du PGCD implémenté par le composant `fifo_gcd_coprocessor`, *que se passe-t-il si un des deux opérandes transmis au coprocesseur a la valeur 0?*

Comment peut-on modifier le composant `fifo_gcd_master` *pour que ceci ne se produise jamais?*

Si un des deux opérandes vaut 0, le coprocesseur boucle indéfiniment et ne produit jamais de résultat. On peut modifier le master et forcer les opérandes à ne jamais valoir 0.

3 Travail à réaliser

3.1 Ecriture du modèle CABA du coprocesseur

En s'inspirant du code fourni pour le composant master, on complète les fichiers `fifo_gcd_coprocessor.h` et `fifo_gcd_coprocessor.cpp`.

3.1.1 Dans `fifo_gcd_coprocessor.h`

- **Modification d'enum** avec les états de l'automate : `READ_OPA`, `READ_OPB`, `COMPARE`, `DECR_A`, `DECR_B`, `WRITE_RES`
- **Modification des registres** : `r_opa`, `r_opb`, `r_fsm`
- **Modification ports** : `p_in`, `p_out`, `p_clk`, `p_resetn`
- **Modification constructor/destructor** :

```
FifoGcdCoprocessor(sc_core::sc_module_name insname);
~FifoGcdCoprocessor();
```

- **Modification private** : `void transition();` et `void genMoore();` Comme l'automate se comporte comme un automate de Moore, on n'a pas besoin de définir de fonctions de type `genMealy()`.

3.1.2 Dans `fifo_gcd_coprocessor.cpp`

- Modification de la fonction `transition()` On gère ici la machine à états du coprocesseur en observant quand la fifo n'est pas vide pour que l'on puisse lire les opérandes et passer au prochain état, le cycle suivant.
- Modification de la fonction `genMoore()`, on gère ici la FIFO du coprocesseur : on lit les opérandes en début et on écrit le résultat à la fin. Dans les autres états on ne lit et n'écrit pas.

3.2 Ecriture du modèle CABA de la top-cell

Dans cette partie, on décrit les interconnexions des différents composants constituant l'architecture matérielle qu'on souhaite simuler.

3.2.1 Dans `tp1_top.cpp`

- Modification des signaux :

```
FifoSignals signal_fifo_m2c("signal_c2m");
```

- Appel des constructeurs avec leurs arguments pour instancier les composants.
- Modification de la netlist pour connecter les signaux du coprocesseur :

```
coproc.p_clk(signal_clk);
coproc.p_resetn(signal_resetn);
coproc.p_in(signal_fifo_c2m);
coproc.p_out(signal_fifo_m2c);
```

- Dans la simulation, il n'y avait pas de temps de début dans `sc_start`, on rajoute donc `sc_start(0);`

3.2.2 Makefile

Enfin on modifie le `Makefile` pour qu'il puisse exécuter le code avec un autre moteur de simulation.

3.3 Simulation

On effectue des simulations en SystemC et en SystemCASS :

- Sur 10 000 cycles :

Type	SystemC	SystemCASS	Delta
real	0m0.013s	0m0.013s	0m0.000s
user	0m0.005s	0m0.008s	+0m0.003s
sys	0m0.005s	0m0.002s	−0m0.003s

Table 1: Comparaison des performances sur 10 000 cycles

- Sur 5 000 000 de cycles :

Type	SystemC	SystemCASS	Delta
real	0m2.295s	0m1.155s	−0m1.140s
user	0m1.923s	0m0.883s	−0m1.040s
sys	0m0.249s	0m0.151s	−0m0.098s

Table 2: Comparaison des performances sur 5 000 000 cycles

Sur un petit nombre de cycles, on n'observe que très peu de différence, mais sur un grand nombre, on observe que le temps est quasiment divisé par 2 avec SystemCASS.

Cela est dû au fait que SystemC utilise une technique d'ordonnancement dynamique (vérification à chaque cycle) qui s'adapte à n'importe quel style d'écriture des modèles SystemC.

SystemCASS, de l'autre côté, exploite les caractéristiques des modèles SoCLib CABA (MAE avec comportement déterministe) pour mettre en œuvre une technique d'ordonnancement statique qui est donc plus rapide.