SMC TP1: "Prototypage Virtuel avec SoCLib"

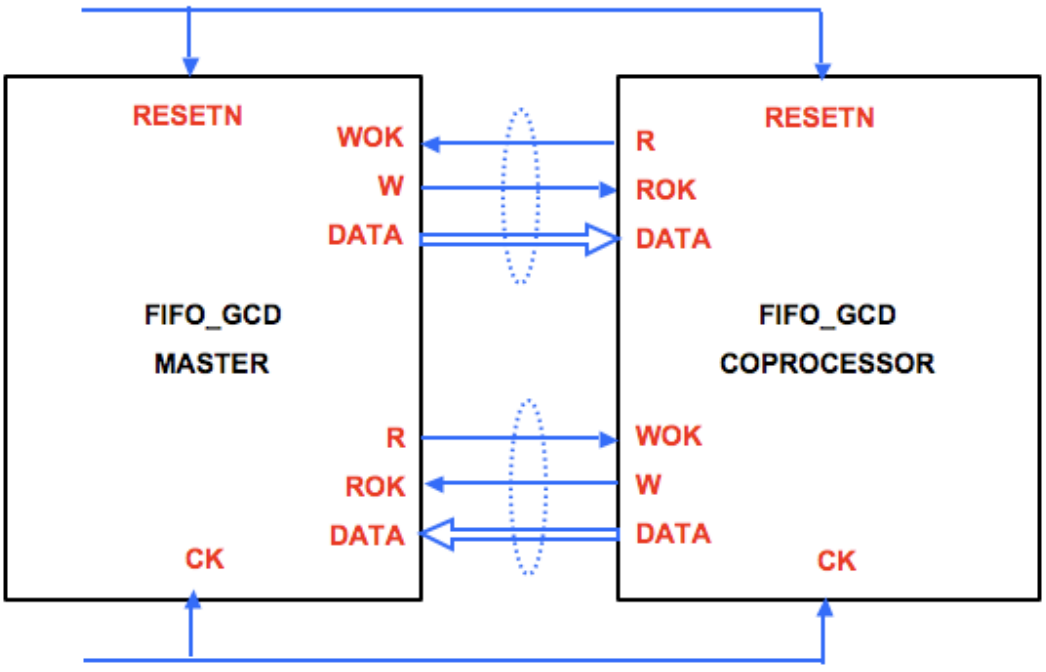
Mendieta Jorge
Liu Owen
Bordas Florent 21308825

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



Architecture Matérielle

Le premier composant est un coprocesseur cablé qui calcule le PGCD de deux nombres entiers positifs A et B, codés sur 32 bits.

Le second composant est chargé de transmettre les valeurs des opérandes A et B au coprocesseur, et de récupérer le résultat.

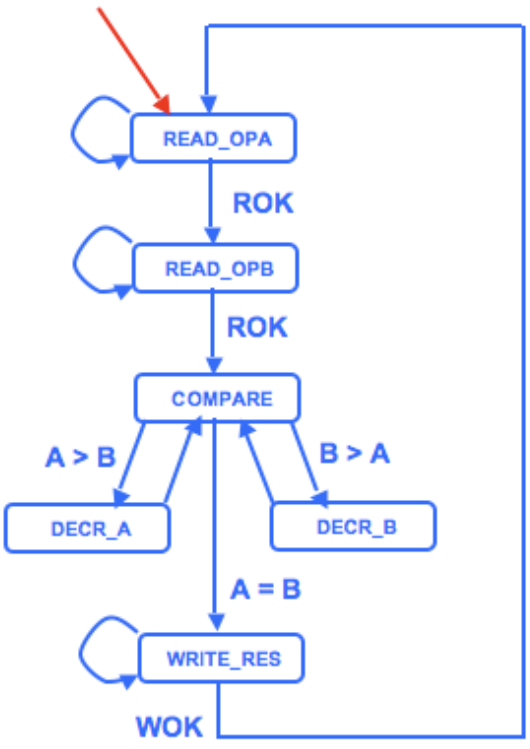
Ces deux composants matériels fonctionnent en parallèle, et communiquent entre eux par des canaux de communication de type FIFO.

2.2 Composant fifo_gcd_coprocessor

Le code utilisé par l'automate cablé qui calcule le PGCD est :

```
uint32_t pgcd( uint32_t opa, uint32_t opb)
{
    while (opa != opb )
        {
        if ( opa > opb ) opa = opa - opb;
        if ( opa < opb ) opb = opb - opa
        }
    return( opa );
}
```

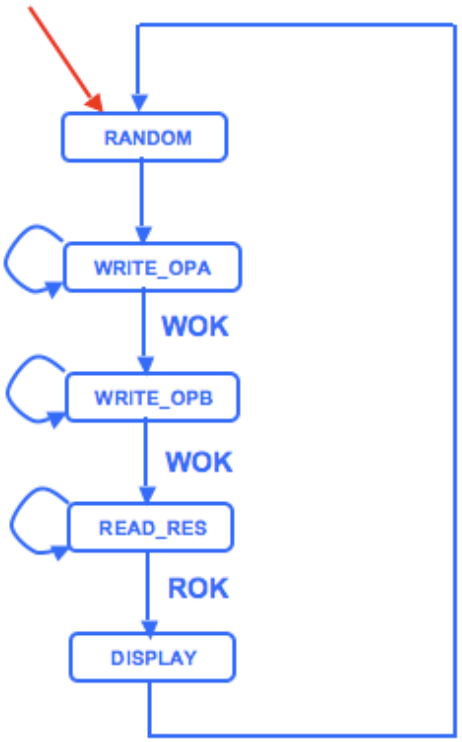
L'automate qui décrit le comportement



FifoGcdCoprocessor FSM

2.3 Composant fifo_gcd_master

Le composant `fifo_gcd_master` est donc un composant matériel paramètrable (un paramètre permettant de contrôler la séquence de valeurs aléatoires), modélisé comme un automate à 5 états :



FifoGcdMaster FSM

Compte-tenu de l'algorithme de calcul du PGCD implémenté par le composant fifo_gcd_coprocessor, que se passe-til si un des deux opérandes transmis au coprocesseur a la valeur 0 ?

Quand un des deux opérandes est **égale a zéro**, le programme tombe sur une boucle infinie (lignes 5 et 6 du fonction `pgcd`) car le `while` ne vérifie pas ce cas là.

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

Pour régler ce problème, nous pouvons modifier la FSM `fifo_gcd_master` en ajoutant une transition depuis l'état **RANDOM** à l'etat **DISPLAY** quand `r_opa` ou `r_opb` sont égales à zéro

$!(r_{opba} \cdot r_{opb})$.

Voici la MAE modifiée :

(TODO : Ajouter image)

3.1 Ecriture du modèle CABA du coprocesseur

Nous avons utilisé les règles d'écriture (http://www.soclib.fr/trac/dev/wiki/WritingRules/Caba) des modèles de simulation au niveau CABA afin de complèter les fichiers

Le fichier header `fifo_gcd_coprocessor.h` manquait des sections dans la classe `FifoGcdCoprocessor` .

```
class FifoGcdCoprocessor
    : public sc_core::sc_module
{
    enum coprocessor_fsm_state_e {
        A COMPLETER
    };

    // Registers
    A COMPLETER

protected:
    SC_HAS_PROCESS(FifoGcdCoprocessor);

public:
    // ports
        A COMPLETER

    // constructor & destructor
        A COMPLETER
private:
    // member functions
        A COMPLETER
};
```

Nous avons complété donc la classe `FifoGcdCoprocessor` en ajoutant les parties manquantes.

```
class FifoGcdCoprocessor
    : public sc_core::sc_module
{
    enum coprocessor_fsm_state_e
    {
        READ_OPA,
        READ_OPB,
        COMPARE,
        DECR_A,
        DECR_B,
        WRITE_RES
    };

    // Registers
    sc_core::sc_signal<uint32_t> r_opa;
    sc_core::sc_signal<uint32_t> r_opb;
    sc_core::sc_signal<int> r_fsm;

protected:
    SC_HAS_PROCESS(FifoGcdCoprocessor);

public:
    // ports
    sc_core::sc_in<bool> p_clk;
    sc_core::sc_in<bool> p_resetn;
    soclib::caba::FifoInput<uint32_t> p_in;
    soclib::caba::FifoOutput<uint32_t> p_out;

    // constructor & destructor
    FifoGcdCoprocessor(sc_core::sc_module_name insname);
    ~FifoGcdCoprocessor();

private:
    // member functions
    void transition();
    void genMoore();

}; // end class FifoGcdCoprocessor
```

Après avoir completé les fonctions manquants du code (`transition()` , `genMoore()` , `genMealy()`) nous avons compilé les modèles `fifo_gcd_master.cpp` et `fifo_gcd_coprocessor.cpp` en utilisant la commande suivante dans la console :

```
++ -Wno-deprecated -fpermissive -std=gnu++0x -I. -I/users/outil/dsx/cctc
```

3.2 Ecriture du modèle CABA de la top-cell

Nous avons défini les arguments des constructeurs, les signaux, la netlist.

- Les signaux :

```
sc_clock signal_clk("signal_clk", sc_time(1, SC_NS), 0.5);
sc_signal<bool> signal_resetn("signal_resetn");
FifoSignals<uint32_t> signal_fifo_m2c("signal_m2c");
FifoSignals<uint32_t> signal_fifo_c2m("signal_c2m");
```

- Components :

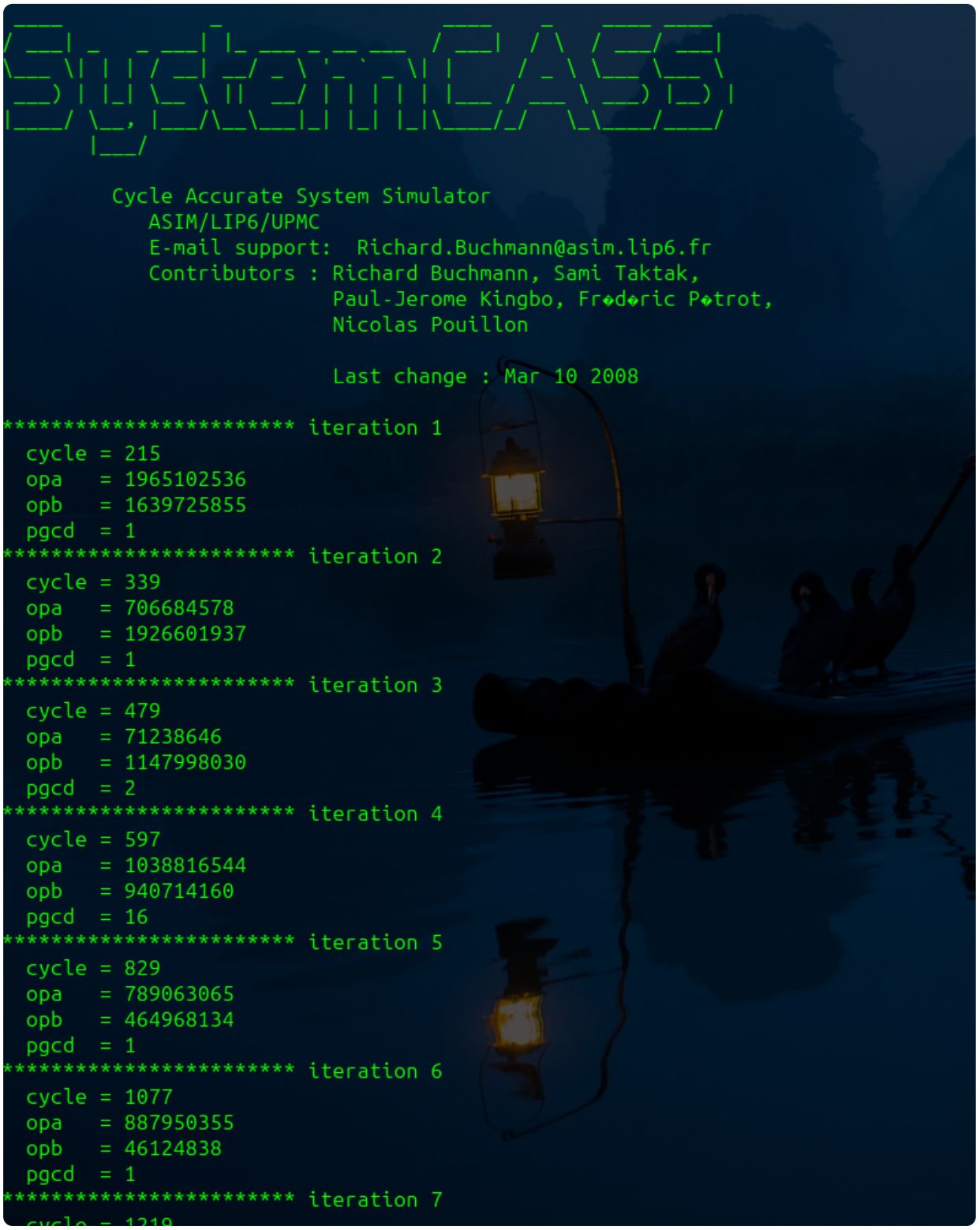
```
FifoGcdMaster master("fifo_gcd_master", seed);
FifoGcdCoprocessor coproc("fifo_gcd_coprocessor");
```

- Net-List :

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

coproc.p_clk(signal_clk);
coproc.p_resetn(signal_resetn);
//Attention ici à ne pas mettre les mêmes arguments pour les fi
//L'ordre est inversé car le composant gcd_copro est l'esclave.
coproc.p_in(signal_fifo_m2c);
coproc.p_out(signal_fifo_c2m);
```

Comparaison et Démonstration



time 100000 cycles

`simulator.x`

real 0.056s

user 0.052s

sys 0.002s

`fast_simulator.x`

real 0.025s