

Modélisation en SystemC

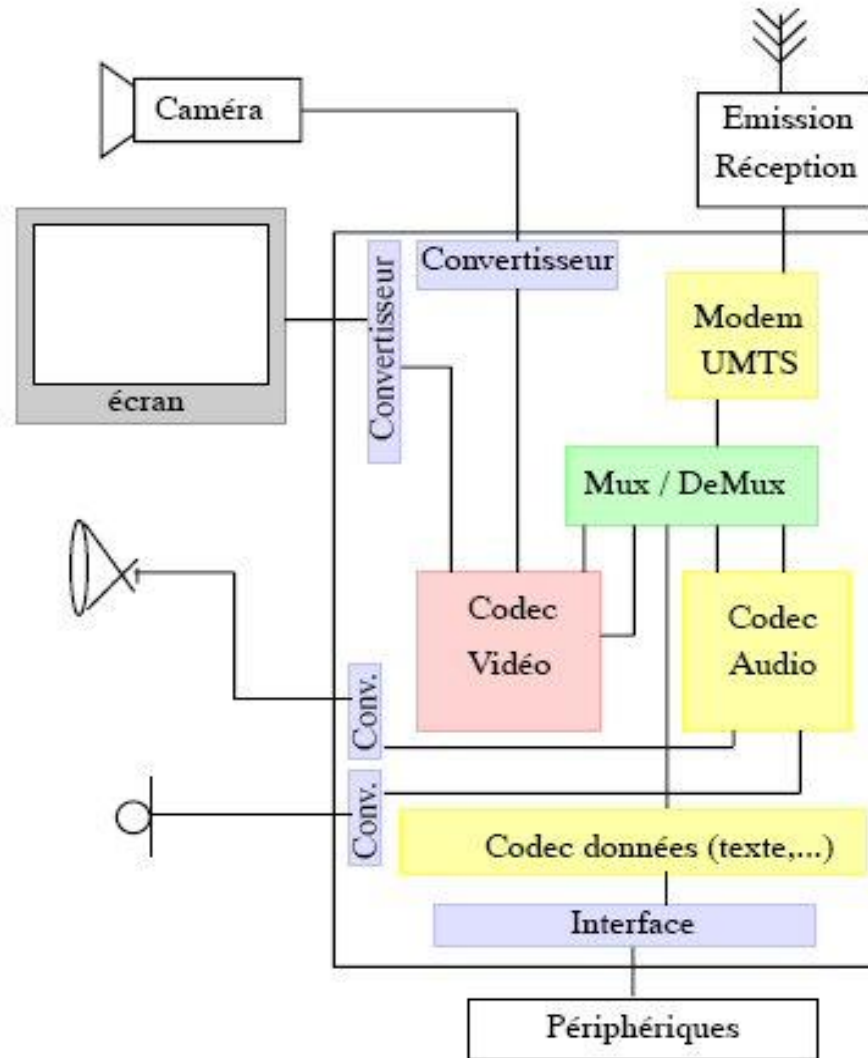
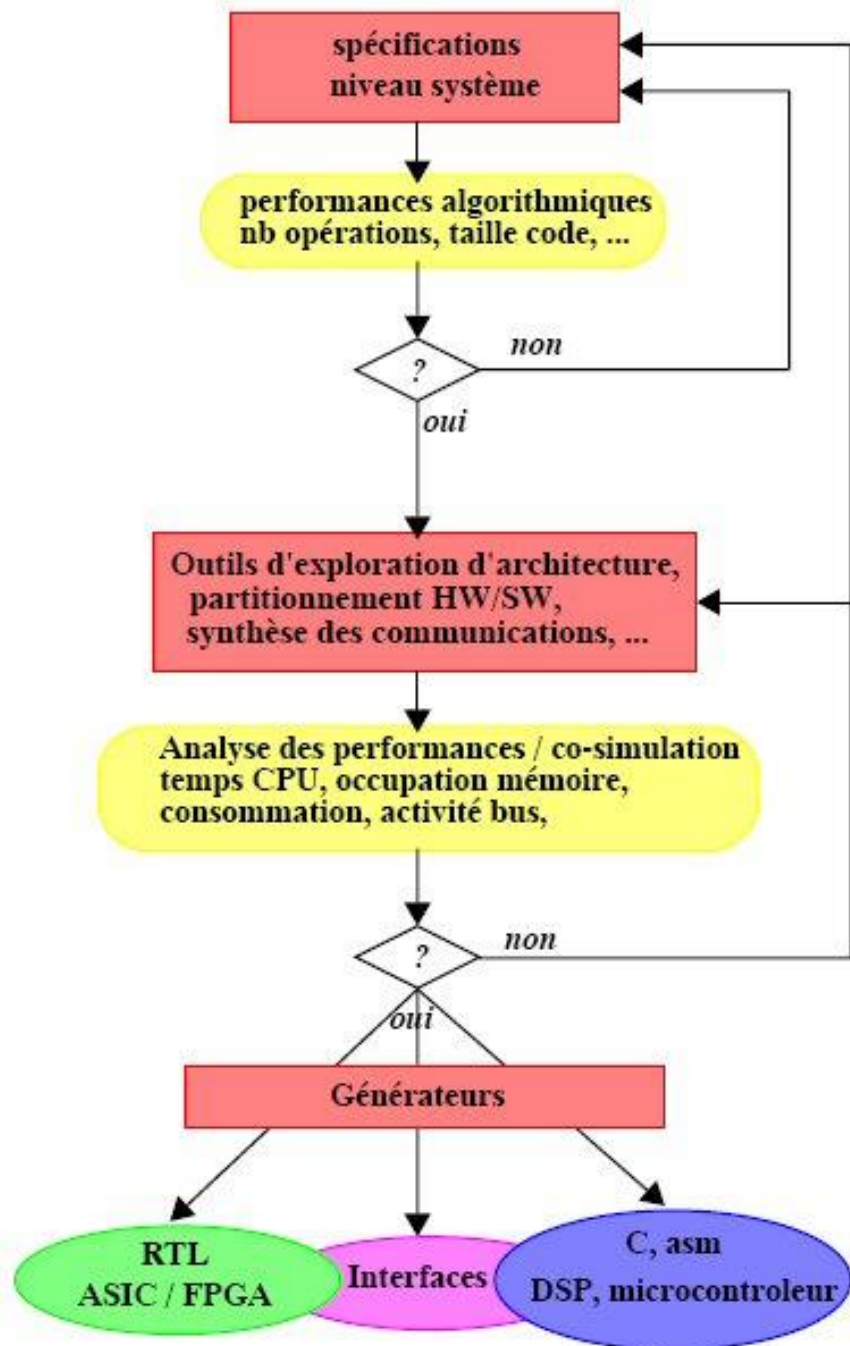


Schéma fonctionnel d'un terminal multimédia (visiophone mobile)



Quelle technologie ?

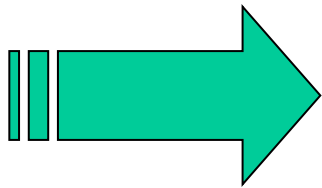
- ASIC
- GPP
- DSP, Microcontrôleur, ASIP
- Reconfigurable : FPGA, Data-Path, ...
- Architecture mixte
- ...



SoC, RSoC, SoPC

Quelle Méthodologie ?

Quel langage ?

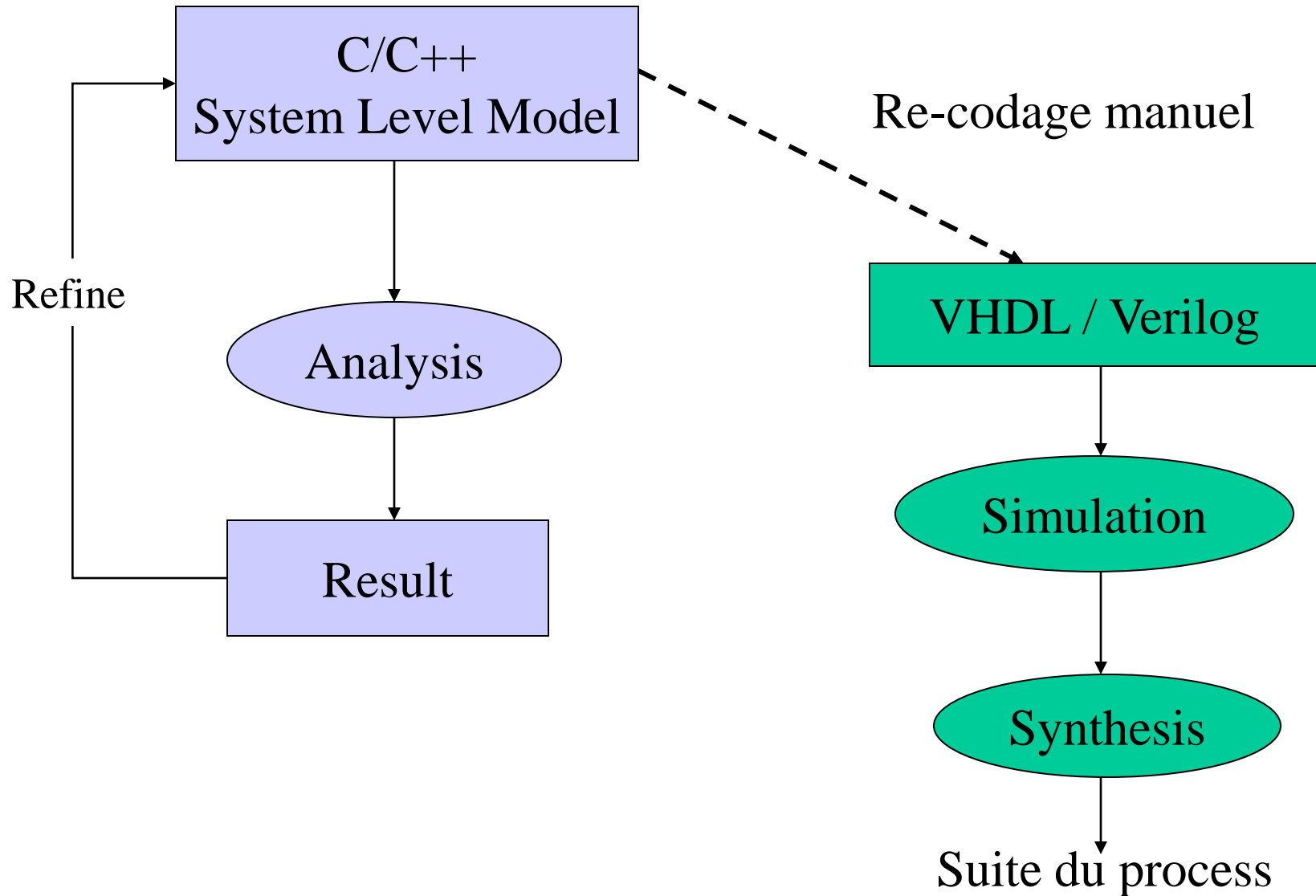


Co-design, un langage ?

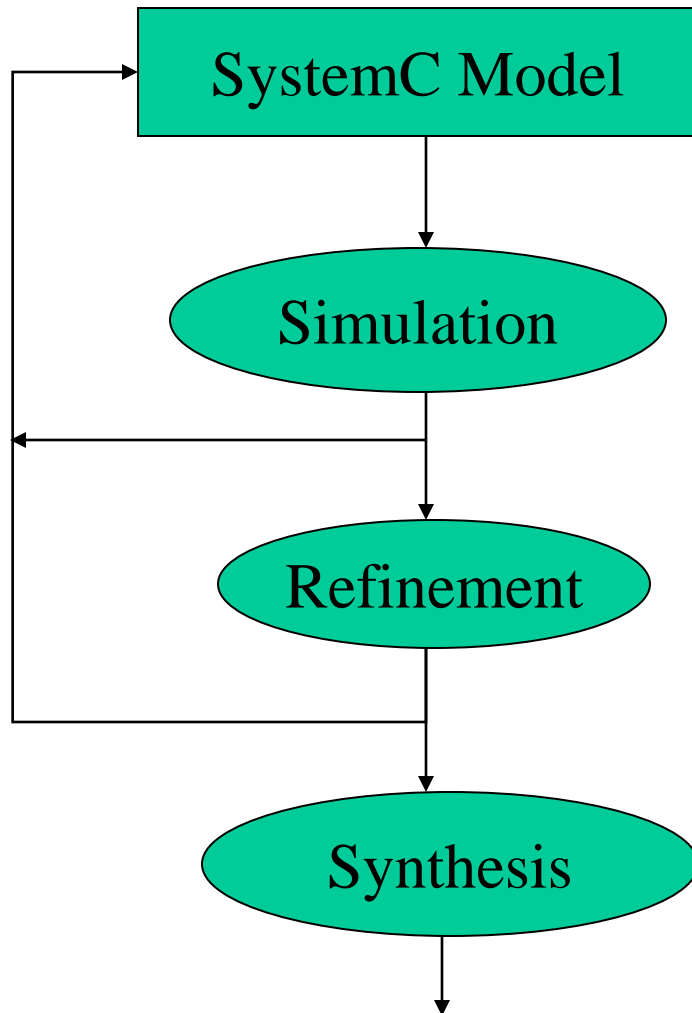
Méthodologie SystemC

- Langages HDL, C, Handel-C, C++, ...
- Interface C - HDL
- SystemC :
 - Basé sur une approche Objet
 - Librairie de classes C++
 - Initiative OpenSource : <http://www.systemc.org>
(9 univ. et + 50 compagnies)
 - Modélisation au niveau **système**
 - Description du Hard et du Soft avec le même langage
 - Un seul langage pour tous les niveaux d'abstraction : C, C++
 - Modèles de simulation au cycle près et au bit près

Flux de conception traditionnel



Méthodologie SystemC



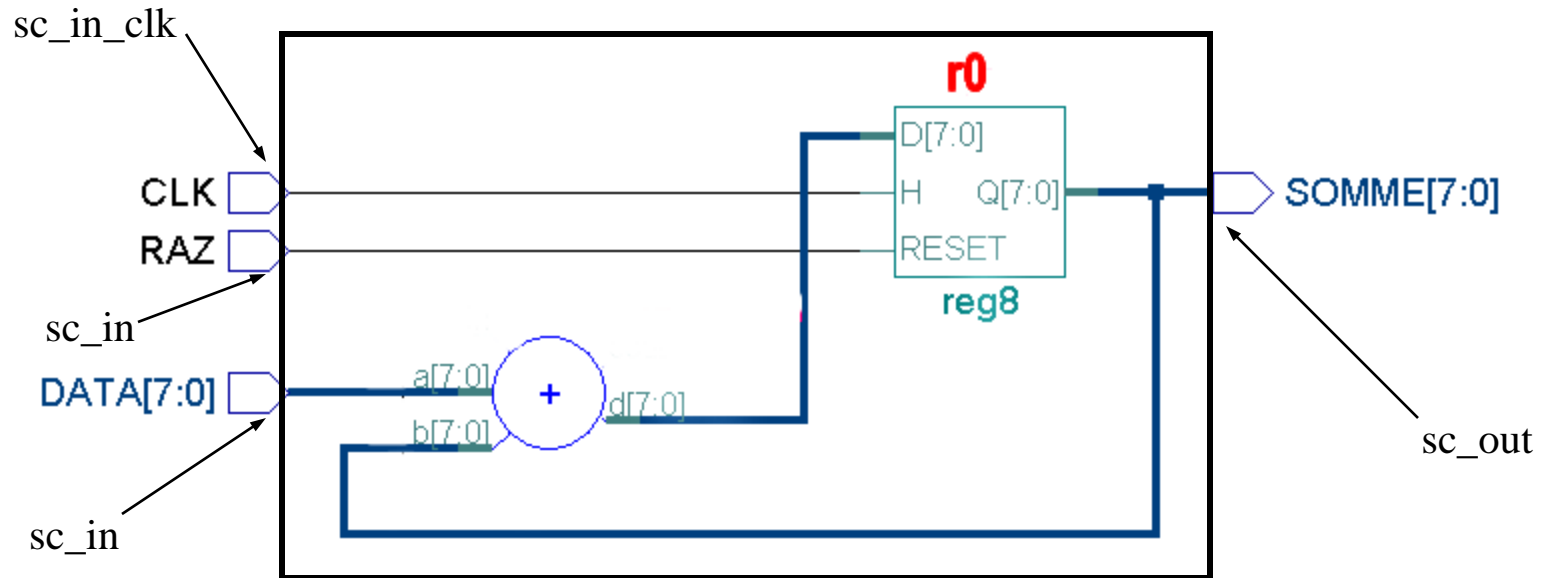
-Méthodologie avec raffinement :
pas de conversion $C \rightarrow HDL$

-Effort minimal pour la synthèse

Approche Objet

- Encapsulation des données ➔ modules
 - C : structures (**struct**)
 - C++ : **class**
 - Membres : **private** et **public**
 - Méthodes : **private** et **public**
 - Constructeur ➔ objets dynamiques
- Polymorphisme et surcharge d'opérateurs

Éléments clés de SystemC (1)



- Modules :
 - Encapsulation hiérarchisée de composants
- Processus :
 - Description du comportement, évènements et liste de sensibilité
- Ports / Signaux :
 - Communications unidirectionnelles (**in**, **out**), bidirectionnelles (**inout**)
 - Grande diversité de types de données

Eléments clés de SystemC (2)

- Types de données
 - C++ (char, int, short, bool, float, ...)
 - SystemC (sc_int<n>, sc_uint<n>, sc_bit, ...)
 - Autres (sc_logic, sc_lv<n>, sc_fixed, ...)
 - Utilisateur
- Processus
 - SC_METHOD
 - SC_THREAD
 - SC_CTHREAD
 - ...

Module (1)

```
SC_MODULE(nom_du_module)
{
    // déclarer les ports du module
    // déclarer les signaux intermédiaires
    // module de construction : SC_CTOR
    // process, sensibilités aux signaux
    // SC_METHOD, SC_THREAD, SC_CTHREAD
    // instancier les composants, décrire les connections, ...
    // effectuer les initialisation des variables et/ou signaux
}
```

Module (2)

- Ports d'E/S
 - Entrée : **sc_in**<type> *nom_du_port*
 - Sortie : **sc_out**<type> *nom_du_port*
 - Bidirectionnel : **sc_inout**<type> *nom_du_port*
- Signaux internes : **sc_signal**<type> *nom_du_signal*
- Déclarer un module

```
type_module *nom_pointeur;  
nom_pointeur = new type_module ("label");
```

```
type_module nom_instance ("label");
```

- Utiliser un module

```
1)- nom_pointeur-> e1(a); ...  
2)- (*nom_pointeur)(a, b, s);
```

```
a)- nom_instance << a << b << s;  
b)- nom_instance.e1(a); ...
```

Exemples (1)

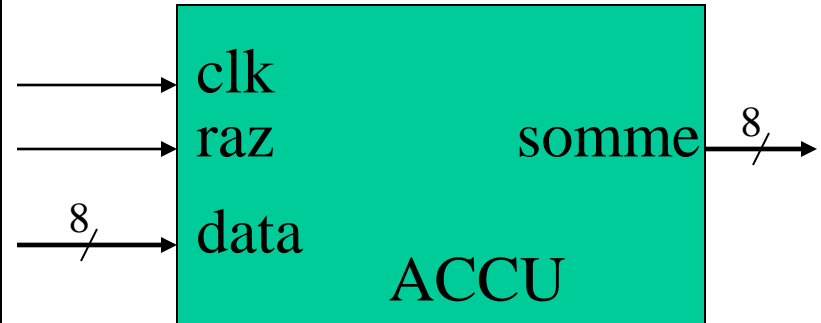
```
SC_MODULE(accu)
{
    ports
    ↑
    sc_in<bool> clk;
    sc_in<bool> raz;
    sc_in<sc_uint<8>> data;
    ↓
    sc_inout<sc_uint<8>> somme;

    Signaux
    ↑
    sc_signal <sc_uint<8>> sum2reg, q2sum;

    dff *r0; // Objet instancié
    void add2x8() // Méthode
    {
        sum2reg=(int)data.read()+(int)somme.read();
    }
    ...
    ...

    Constructeur
    ↑
    ↓

};
```

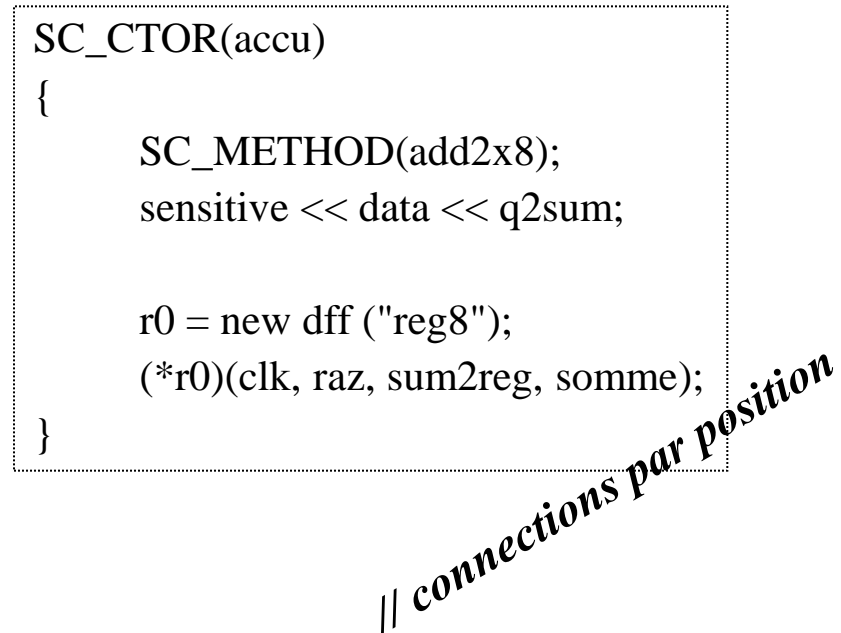
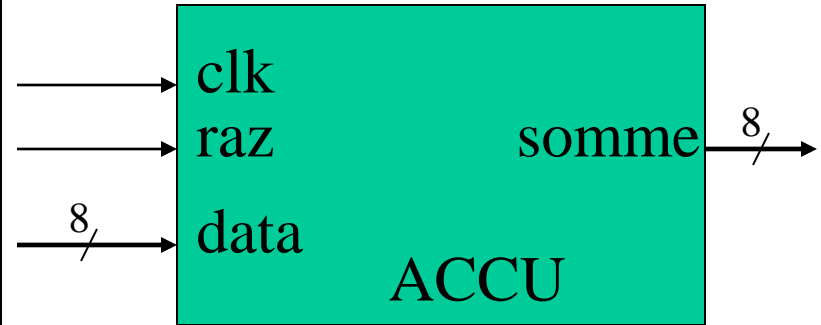
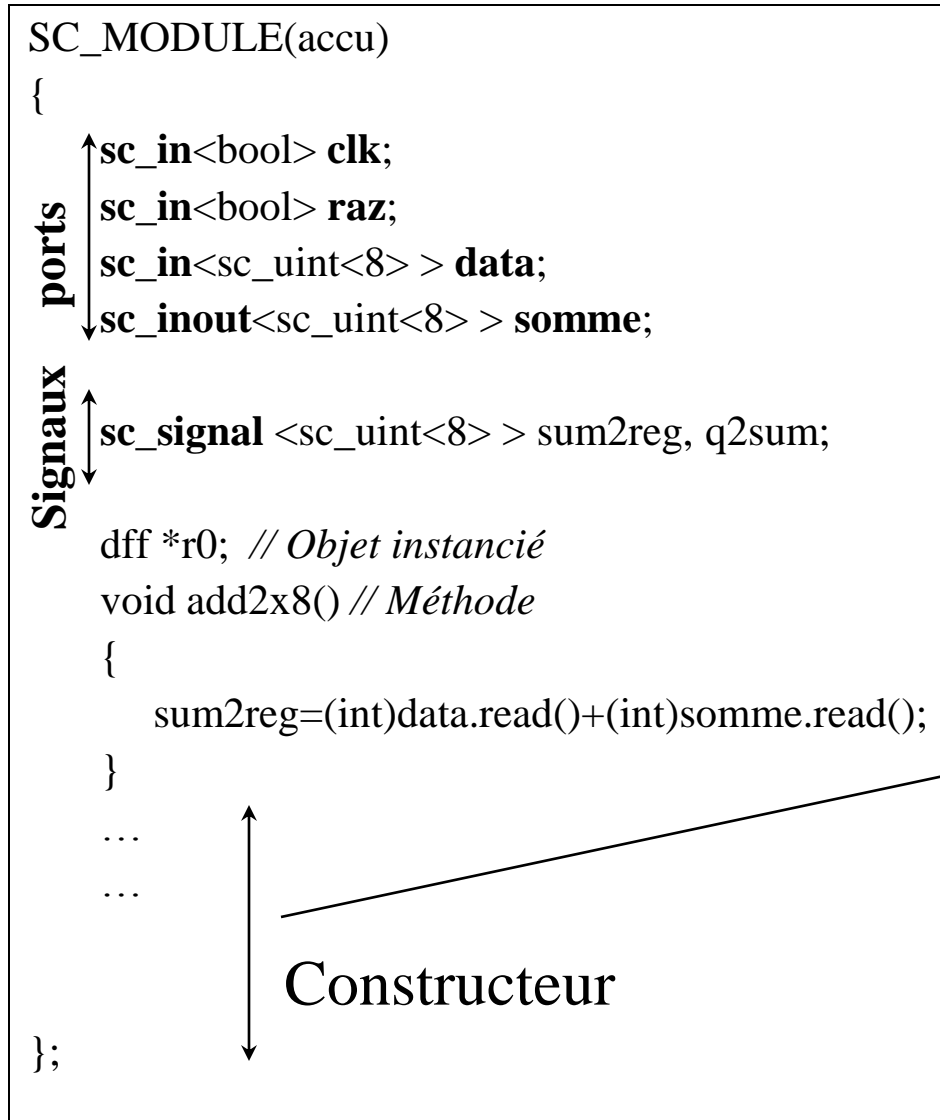


```
SC_CTOR(accu)
{
    SC_METHOD(add2x8);
    sensitive << data << q2sum;

    r0 = new dff ("reg8");
    r0->h(clk);
    r0->reset(raz);
    r0->d(sum2reg);
    r0->q(somme);
}
```

// connections par association

Exemples (1bis)



Exemples (2)

// fichier : accu.h

```
#include "dff.h"
SC_MODULE(accu)
{
    sc_in<bool> clk;
    sc_in<bool> raz;
    sc_in<sc_uint<8>> data;
    sc_out<sc_uint<8>> somme;

    sc_signal <sc_uint<8>> sum2reg, q2sum;
    sc_uint<8> e1, e2;
    dff *r0;
    void add2x8();
    SC_CTOR(accu)
    {
        SC_METHOD(add2x8);
        sensitive << data << q2sum;

        r0 = new dff ("reg8");
        r0->h(clk);
        r0->reset(raz);
        r0->d(sum2reg);
        r0->q(q2sum);
    }
};
```

// fichier : accu.cpp

```
#include "systemc.h"
#include "accu.h"

void accu::add2x8()
{
    e1 = data.read();
    e2 = q2sum;
    sum2reg = e1 + e2;
    somme.write(e2);
}
```

Examples (3)

// fichier : dff.h

```
#include "systemc.h"

SC_MODULE(dff)
{
    sc_in<bool> h;
    sc_in<bool> reset;
    sc_in<sc_uint<8>> d;
    sc_out<sc_uint<8>> q;

    void d2q();

    SC_CTOR(dff)
    {
        SC_METHOD(d2q);
        sensitive_pos << h;
        sensitive << reset;
    }
};
```

// fichier : dff.cpp

```
#include "systemc.h"
#include "dff.h"

void dff::d2q() {
    if (reset) q = 0;
    else q = d;
}
```


TestBenches (1)

Module de test



```
int sc_main(int argc, char *argv[]) {...}
```

- Formats de **Waveform Tracing** supportés : VCD, WIF, ISBD
- Clocks ➔ `sc_clock clk("my_clock" , 20, 0.5);`
- `sc_start(1000);`
- `sc_start(-1); sc_stop();`
- `sc_initialize(); sc_cycle(100);`

TestBenches (2)

// fichier : stimulus.h

```
SC_MODULE(stimulus)
{
    sc_out<bool> reset;
    sc_in<bool> clk;
    sc_out<sc_uint<8>> data_in;

    unsigned cycle;
    void data_gen();

    SC_CTOR(stimulus)
    {
        SC_THREAD(data_gen);
        sensitive_neg << clk;
        sensitive << clk;

        cycle = 0;
    }
};
```

// fichier : stimulus.cpp

```
#include <systemc.h>
#include "stimulus.h"

void stimulus::data_gen()
{
    cycle=0; data_in = 0; reset = true;
    while(true)
    {
        for(cycle=0; cycle < 4; cycle++)
            wait();

        reset = false ;
        for(cycle = 4; cycle < 20; cycle++)
        {
            data_in = cycle - 4;
            cout << "Stimuli : " << cycle << " at time "
                << sc_time_stamp() << endl;
            wait();
        }
        // Fin de simulation
        sc_stop();
        cout << "Simulation of " << cycle-1
            << " items finished"
            << " at time " << sc_time_stamp() << endl;
    }
}
```

TestBenches (3)

// fichier : display.h

```
#include "systemc.h"

SC_MODULE(display)
{
    sc_in<bool> clk;
    sc_in<sc_uint<8>> data_in;
    sc_in<sc_uint<8>> data_sum;

    void affiche();

    SC_CTOR(display)
    {
        SC_METHOD(affiche);
        sensitive_pos << clk;
    }
};
```

// fichier : display.cpp

```
#include <systemc.h>
#include "display.h"

void display::affiche()
{
    cout << "Display : " << data_in << " " << data_sum
        << " at time " << sc_time_stamp() << endl;
}
```

TestBenches (4)

// fichier : tb_accu.cpp

```
#include "systemc.h"
#include "accu.h"
#include "stimulus.h"
#include "display.h"

/*****/
int sc_main (int argc, char *argv[])
{
    // signaux
    sc_clock clk("clk", 100, 0.5);
    sc_signal<bool> reset;
    sc_signal<sc_uint<8> > data_in, data_sum;

    stimulus stimulus1("stimulus1");
    stimulus1.reset(reset);
    stimulus1.data_in(data_in);
    stimulus1.clk(clk);

    accu accu1("accu");
    accu1.raz(reset);
    accu1.clk(clk);
    accu1.data(data_in);
    accu1.somme(data_sum);
```

```
// Display output
display display1 ("display");
display1.clk(clk.signal());
display1.data_in(data_in);
display1.data_sum(data_sum);

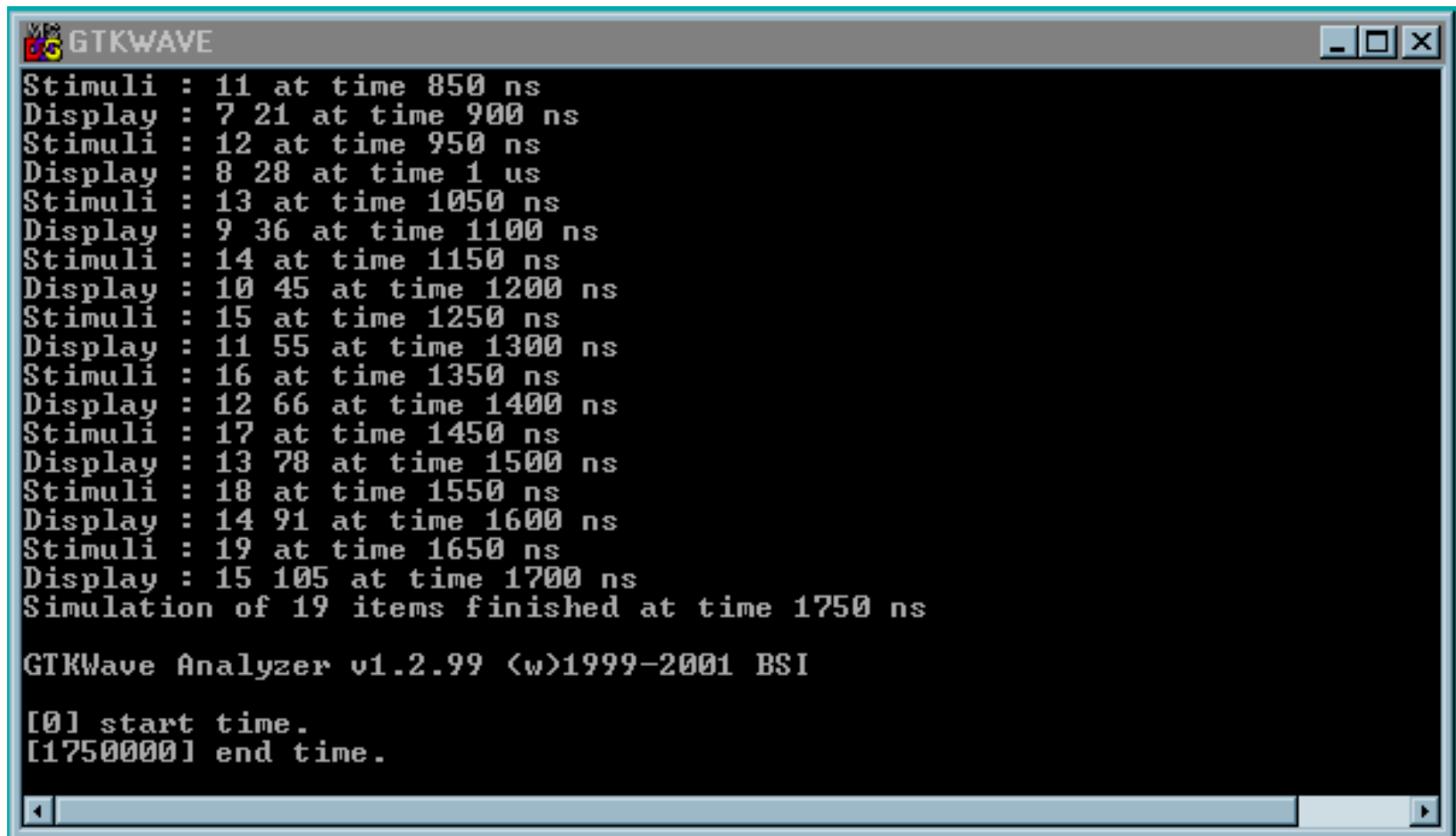
// <Debut TRACE>
sc_trace_file *tf = sc_create_vcd_trace_file("accu");
sc_trace(tf, clk.signal(), "clk");
sc_trace(tf, reset, "RESET");
sc_trace(tf, data_in, "DATA_IN");
sc_trace(tf, data_sum, "DATA_SUM");
// <Fin TRACE>

// Initialize SC
sc_initialize();

sc_clock::start(-1);
sc_close_vcd_trace_file(tf);
system("gtkwave accu.vcd");

cout << "-----" << endl;
cout << "End of fsm simulation..." << endl;
return 0;
}
```

Résultats de la simulation (1)

A screenshot of a GTKWAVE window. The window has a title bar with the text 'GTKWAVE' and standard window control buttons (minimize, maximize, close). The main area is black with white text. The text lists a series of stimuli and display events, each with a number, a time value, and a unit. The stimuli are numbered 11 through 19, and the displays are numbered 7 through 15. The simulation ends at 1750 ns. At the bottom, there is a version string and a copyright notice, followed by two lines of bracketed text: '[0] start time.' and '[1750000] end time.'.

```
GTKWAVE
Stimuli : 11 at time 850 ns
Display : 7 21 at time 900 ns
Stimuli : 12 at time 950 ns
Display : 8 28 at time 1 us
Stimuli : 13 at time 1050 ns
Display : 9 36 at time 1100 ns
Stimuli : 14 at time 1150 ns
Display : 10 45 at time 1200 ns
Stimuli : 15 at time 1250 ns
Display : 11 55 at time 1300 ns
Stimuli : 16 at time 1350 ns
Display : 12 66 at time 1400 ns
Stimuli : 17 at time 1450 ns
Display : 13 78 at time 1500 ns
Stimuli : 18 at time 1550 ns
Display : 14 91 at time 1600 ns
Stimuli : 19 at time 1650 ns
Display : 15 105 at time 1700 ns
Simulation of 19 items finished at time 1750 ns

GTKWave Analyzer v1.2.99 (c)1999-2001 BSI

[0] start time.
[1750000] end time.
```

Résultats de la simulation (2)

