Chapitre I.

3ère partie

Généralité sur systemc

1. Introduction

Avant l'utilisation d'un outil comme systemc, la conception de systèmes numérique se faisait d'une manière complètement indépendante entre les parties matérielles et les parties logicielles. Une tâche qui oblige les concepteurs à partitionner leurs applications entre parties s'exécutant sur les processeurs et les parties qui étaient destinées à être implantées en matériel sur un circuit.

1.1. Les aspects matériels : *Les circuits de l'époque possédaient une quantité relativement réduite de ressources, dans l'ordre de quelques milliers de portes logiques. Par conséquent, ils étaient très faciles de concevoir rapidement ce type de système en utilisant des outils schématiques, ou des langages de description de matériel tels que le «Verilog » ou le « VHDL ». Ces langage sont encore utilisés mais ils sont essentiellement des langages travaillant au niveau RTL (Register transfert level) qui consiste à décrire une architecture à partir d'opérateurs logiques de base et en décrivant le transfert de données entre ces opérateurs et des registres permettant de synchroniser ses opérations.*

1.2. Les aspects logiciels : *Ils consistaient simplement à apporter le code d'une application sur une architecture matériel simple avec des périphériques dont le nombre et la nature étaient connus. Aujourd'hui, les circuits actuels sont devenus très complexes et font intervenir non seulement des microprocesseurs mais également des accélérateurs matériels complexes ainsi qu'une grande variété de périphériques. Ces systèmes sont regroupés sous le terme de « SOC » (system on chip). La conception de ces systèmes est devenue très difficile puisqu'elle fait intervenir en parallèle le développement logiciel et le développement matériel. D'autre part la communication, entre ces différents éléments, est devenue également difficile à modéliser en faisant intervenir par exemple des systèmes d'interconnexion complexes comme des bus, voire des véritables réseaux sur puce. Aujourd'hui il est tout à fait difficile de concevoir ce type de système rapidement et efficacement tellement que les choix auxquels sont confrontés les concepteurs, sont multiples et notamment difficile d'effectuer le partitionnement matériels logiciels comme précédemment car les fonctions sont devenues non seulement très compliquées mais sont également trop nombreuses. Aujourd'hui les circuits comportent des centaines de millions*

de portes logiques. Concevoir un tel circuit, le tester et le vérifier constitue un véritable enjeu.

Pour concevoir ce genre de système il semble donc nécessaire de faire promouvoir le niveau de représentation du système à un niveau beaucoup plus haut c'est-à-dire travailler sur une représentation suffisamment détaillées, permettant d'avoir une idée globale du fonctionnement du circuit tout en ne faisant pas d'hypothèses sur la cible de l'implantation. Cette représentation à haut niveau va notamment permettre aux concepteurs d'explorer les différents choix de conceptions dont ils sont confrontés avant la réalisation du circuit, il va rapidement identifier les solutions inaptes à satisfaire ses exigences et celle qui pourront être préservées. Car ces étapes vont devoir se réaliser le plus rapidement possible de manière à augmenter la productivité globale et concevoir le système idéal dans un minimum de temps. Un premier constat qu'on peut faire à ce niveau-là est que les approches basés sur les langages de description matériels ne sont plus adaptés aujourd'hui. La complexité est devenue telle que travailler au niveau opérateurs logiques, n'est plus viable et requiert énormément de temps. D'autre part les logiciels de description du matériel ne prennent pas en compte les différents logiciels qui devront être testés par d'autres outils spécifiques et complètement indépendants. Il est donc nécessaire d'avoir un outil et une méthodologie associée permettant d'intégrer les aspects matériels et les aspects logiciels en même temps, ce sera précisément le rôle du systemc.

2. Définition :

Le systemc n'est pas un langage de description de matériel à part entière mais une bibliothèque comprenant des classes C++ qui permettent conjointement la modélisation de systèmes logiciels et matériels. Systemc est basé en fait sur des processus concurrents et communiquant entre eux qui vont être capable de décrire la structure ou le fonctionnement des parties matériel et logiciel d'un objet. La bibliothèque systemc est basée sur le langage C++ et possède exactement la même syntaxe et elle conserve toutes les propriétés du C++ tout en ajoutant de nouvelle fonctionnalités. Systemc intègre également la possibilité de simuler la description réalisée puisque il intègre dans son sein son propre simulateur qui est un simulateur événementiel.

3. Objectif du systemc :

- ✓ Faire face à la complexité de la conception du système numérique par le travail avec des modèles de haut niveau d'abstraction.
- ✓ La réutilisation de description : exploiter la description de composants préalablement conçus afin de faciliter la conception de nouveaux composants.

- ✓ La vérification qui consiste à vérifier l'interaction entre les parties logicielles et les parties matérielles du circuit avant leurs productions. Ceci est traité dans une méthodologie appelée TLM (Model bases Technology), associée de très près à systemc.
- ✓ Systemc est libre (open source) et il est facilement accessible et distribué de manière à ce que son utilisation soit la plus large possible.
- ✓ Systemc est mis en œuvre dans tous les conceptions de systèmes numérique aujourd'hui.

4. Apport du systemc par rapport au C++

- ✓ C++ est un langage logiciel qui n'a pas été crée pour la description du matériel. Par exemple la concurrence qui est un principe essentiel pour la description du matériel et qui n'existe pas en C++. La concurrence a pour objective la description du fonctionnement d'un système électronique réel dans lequel toutes les fonctions ou composants s'exécutent simultanément alors que C++ a l'aspect séquentiel.
- ✓ C++ ne possède pas la notion du temps permettant de modéliser le comportement matériel d'un système matériel.
- ✓ Un certain nombre de structure de données sont manquante dans C++ pour décrire finement un circuit matériel, par exemple il n'existe pas des notions pour travailler au niveau des bits ou prendre en considération l'état d'un bus. c'est à dire C++ est pour le software et systemc est à la fois pour le software et le hardware.

En conclusion : Systemc est une bibliothèque de C++ qui possède son propre noyau de simulation basés sur les évènements. Il fournit des modèles pour représenter le temps, des types supplémentaires qui sont dédiés pour la modélisation du matériel ainsi que des types logiques, le concept de signal, les bus, les réseaux…

5. La programmation systemc :

Pour rappel C++ est un langage de programmation orientée objet qui est basé sur le concept de classes d'objets cette notion de classe étant la notion de structure existante dans le langage C (struct). Cette notion est fortement liée à un concept très important en programmation C++ et systemc « encapsulation ». Cette dernière permet de déclarer les membres d'une classe avec des mots réservés : « private », « public » ou « protected ».

Les notions de constructeur et destructeur sont aussi appliqués en systemc de la même manière qu'en C++. Nous verrons plus loin ces deux concepts en détail.

6. Architecture du systemc :

Comme nous l'avons déjà noté systemc est basé sur le langage C++, il est constitué de : les modules (clases), les ports, les processus, les interfaces, et les canaux.

Systemc utilise non seulement les types de données déjà vus en C et en C++ mais présente également des nouveaux types de données capables de modéliser les aspects matériels et logiciels, comme le type logique multivalué et le type time qui permet de prendre en compte le temps en modélisation système (n'existe pas en C++). Les différents éléments du langage communiqueront à travers des canaux élémentaires par les signaux, les moteurs événementiels permettant la simulation des systèmes complexes. Le moteur est destiné à gérer les processus et les événements comme dans un outil de simulation et de description de langage matériel. Ajoutons aussi qu'il y a d'autre bibliothèque pour la vérification des systèmes qui sont indépendantes du systemc mais fournies avec systemc comme « TLM ».

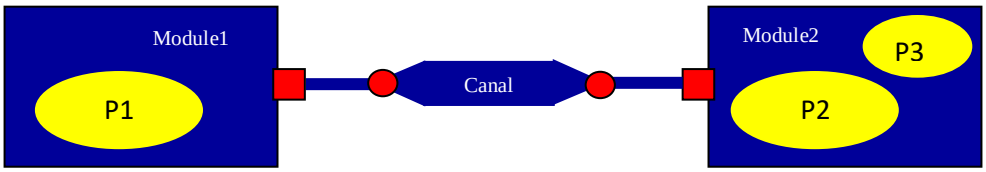


Figure 3.1 : Les différents éléments du langage systemc

P1, P2, P3 sont des fonctions de traitement (processus)

Port.

Interface

Les ports, les canaux et les interfaces sont les moyens de communication. Une des propriétés de systemc est que la modélisation décorrelle complètement les éléments structurels des éléments de communication entre les modules. C'est la notion de séparation des préoccupations qui vise à segmenté un problème en plusieurs parties afin de que chacune entre elle isole et gère un aspect spécifique d'une problématique générale.

6.1. Les types de données :

En systemc tous les types natifs de C++ sont utilisables. En plus il fournit des types de données additionnels :

- ✓ Le type entier : sc_int, sc_uint, sc_bigint, sc_biguint.
- ✓ Bit simple: sc_bit, sc_logic.
- ✓ Vecteur de bits : sc_bv, sc_lv.
- ✓ Point fixe : sc_fixed,sc_ufixed, sc_fix, sc_ufix.

Les types les plus importants sont :

Le type ***sc_bit*** qui représente des données ayant deux valeurs possibles 0 ou 1. Ce type de données ne supporte que les opérateurs logiques connus en C++.

Le type ***sc_logic*** produit par systemc est plus intéressant que ***sc_bit*** car il permet de représenter des données multi-valuées pouvant modéliser un circuit numérique. Il permet non seulement de coder le 0 et le 1 mais possède également la valeur « X » pour indéfini, inconnu ou lorsqu'il y a un conflit entre différents états logiques. Il peut représenter aussi la haute impédance par la valeur « Z ».

Le ***sc_bv*** est un ensemble de sc_bit et ***sc_lv*** est un veteur de sc_logic. Ils peuvent être considérés comme des chaines de caractères en C ou en C++ mais non compatible.

Exemple1: *sc_bv<12>données ;* // déclaration de la variable « données » de longueur 12.

données = ''*101110010001''* ; //initialisation par une chaine.

sc_bv<12> données = false // mettre un zéro dans tous les bits

Exemple2 :

```
sc_lv<8> val ;
if (enable) val= ''100Z0111'';
else val= ''ZZZZZZZZ'';
```

6.2. Les opérateurs

En plus des opérateurs déjà connus en C et C++, systemc offre de nouveaux opérateurs comme:

- ✓ Sélecteur de bits : [ ].
- ✓ Sélecteur de plage de bits : .range(.).
- ✓ La concaténation : (,).
- ✓ La réduction : reduce.

6.3. La simulation événementielle :

Le systemc défini un type spécifique pour représenter le temps de simulation ou les intervalles de temps. Ce type est « sc_time » qui est construit à partir de type double et d'une unité (*sc_time_unit*) :

- ✓ *SC_FS* femtoseconde = 10-15s
- ✓ *SC_PS* picoseconde = 10-12s par défaut
- ✓ *SC_NS* nanoseconde = 10-9s
- ✓ *SC_US* microseconde = 10-6s
- ✓ *SC_MS* milliseconde = 10-3s
- ✓ *SC_SEC* seconde

Exemple : sc_time T(30,SC_FS) c'est une déclaration de la variable temps T qui prend comme valeur initiale 30 femtoseconde

Il défini aussi plusieurs fonctions qui sont reliés au temps pour gérer et synchroniser le fonctionnement du système : sc_time_stamp (temps courant), sc_simultion_time(le temps courant de la simulation par defaut)…

Pour gérer la simulation, le code peut la contrôler par la fonction : « sc_start(argument) » dans la fonction principale sc_main son argument est double plus unité e temps. Cette fonction peut être appelée sans arguments, dans ce cas là la simulation est lancée indéfiniment jusqu'à la fin ou une autre fonction d'arrêt soit appelée « sc_stop » .

Exemple :

```
#include"systemc.h"
int sc_main (int argc,char*argv[])
{sc_time T(22, SC_NS);
 sc_start(T);
 cout<<"current time:"<<sc_time_stamp()<<endl;
sc_start(10,SC_NS);
cout<<"current time:"<<sc_time_stamp()<<endl;
  sc_start(15,SC_US);
  cout<<"current time:"<<sc_time_stamp()<<endl;
  sc_stop;
  cout<<"current time:"<<sc_time_stamp()<<endl;
  return 0:}
```

Résultat:

```
Current time: 22ns
Current time: 32ns
Current time: 15032ns
 Current time: 15032ns
```

6.4. visualisation graphique des résultats:

Dans la majorité des cas, il serait très intéressant de visualiser l'état de la simulation graphiquement. Systemc permet facilement de représenter graphiquement les résultats en fournissant les fonctions nécessaires à la génération de fichiers particuliers dans les formats « vcd » (Value Change Dump). Ce format est un format des échanges standards. C'est un format ASCII utilisé pour enregistrer les événements dans un fichier texte. Celui-ci pourra ensuite être envoyé à un outil qui permet de traduire les informations textuelles en chronogramme :

- sc_trace_file() : pour gérer le fichier de trace
- sc_create_vcd_trace_file(): permet de créer le ficher texte.
- Sc_trace / permet d'ajouter des objets dans la liste des objets à tracer
- close file: permet de fermer le fichier.

Exemple :

```
#include "systemc.h"
int sc_main ( int argc,char*argv[ ] )
{
  sc_trace_file *trace_file;
  trace_file=sc_create_vcd_trace_file("monfichier");
  trace_file->set_time_unit(1,SC_NS);
  bool value=false;
  sc_trace(trace_file,value,"value");
  sc_start(10,SC_NS);   value =true;
  sc_start(10,SC_NS);   value=false;
  sc_start(10,SC_NS);   value =true;
  sc_start(10,SC_NS);   value=false;
  sc_start(10,SC_NS);   value =true;
  sc_start(10,SC_NS);   value=false;
  sc_close_vcd_trace_file(trace_file);
}
```

Résultat :

sc_signal :

En systemc les signaux sont des canaux particuliers que l'on appelle des « canaux atomique » ces signaux sont mis en œuvre par le type sc_signal paramétrables permettant de transporter des objets de natures différentes.

sc_signal<int> x ; entiers

Sc_signal<bool> y ; booléen

sc_signa<sc_int<8> > z ; des variable de sc_int à 8 bits.

Un signal possède deux valeurs une valeur courante et une autre futur qu'il prendra à la fin du cycle de simulation.

Cette classe d'objets met en œuvre trois méthodes dont deux seulement sont accessibles par l'utilisateur.

- Read ( ) : retourne la valeur courante du signal.

- Write ( ) : permet de modifier la valeur future du signal.
- Update ( ) : gérée par le simulateur et permet de recopier la valeur future dans la valeur courante du signal.

Write( ) ⟶ valeur future ⟶ Update⟶ valeur courante ⟶Read ( ).

6.5. sc_clock :

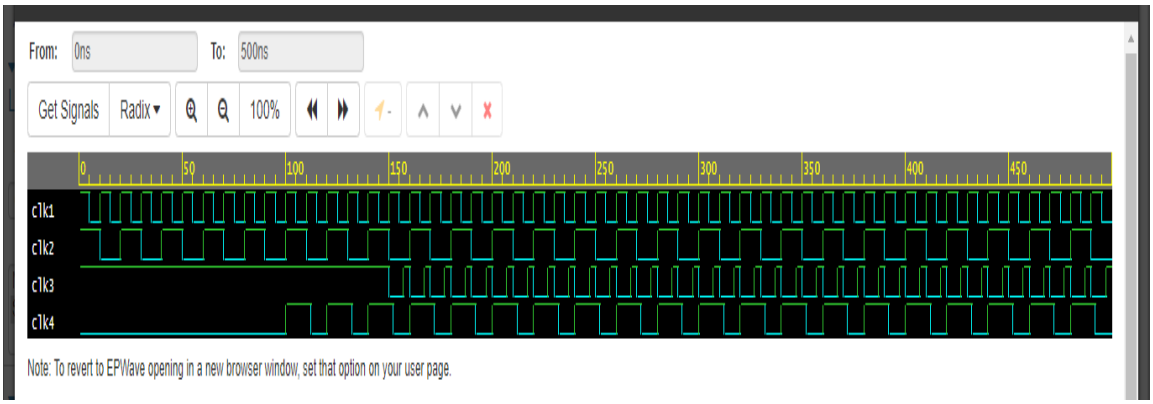
Très utilisé pour modéliser des horloges dans des modèles de circuit matériel. sc_clock est un signal de type sc_signal qui transporte des valeurs booléennes. La construction de ce type d'objet fait intervenir un certain nombre de paramètres : le nom du signal, la période de l'horloge exprimée en valeurs double et l'unité de temps ainsi que le rapport cyclique du signal] 0, 1 [, le quatrième argument précisera le moment du premier front et le dernier est un booléen qui précisera si le premier front est montant ou descendant.

Exemple:

```
#include "systemc.h"
int  sc_main(int argc,char*argv[])
{
  sc_trace_file *trace_file;
  trace_file=sc_create_vcd_trace_file("my_file");
  trace_file->set_time_unit(1,SC_NS);
  sc_clock clk1("clk1",10,SC_NS);
  sc_time T(20,SC_NS);
  sc_clock clk2("clk2",T);
 sc_clock clk3("clk3",10,SC_NS,0.25,150,SC_NS,false);
 sc_clock clk4("clk4",20,SC_NS,0.6,100,SC_NS,true);
  sc_trace(trace_file,clk1,"clk1");
  sc_trace(trace_file,clk2,"clk2");
  sc_trace(trace_file,clk3,"clk3");
  sc_trace(trace_file,clk4,"clk4");
  sc_start(500,SC_NS);

  sc_close_vcd_trace_file(trace_file);
  return 0;
}
```

Résultat:



6.6. SC_MODULE :

Le module est l'élément principal pour construire une description, il s'agit du plus petit conteneur de fonctionnalités qui a le même concept qu'un composant dans les autres langages de description matérielle.

Le module peut aussi bien représenter des composants matériels que des composants logiciels ». Il est composé de :

- ✓ Ports qui permettent la communication avec d'autres modules.
- ✓ Processus qui définissent la fonctionnalité de ce module
- ✓ Ces propres données (données internes).
- ✓ Autres modules permettant ainsi d'avoir une représentation hiérarchique d'un modèle.

Dans la modélisation d'un circuit électronique un module peut représenter un registre, un compteur, un convertisseur, un multiplexeur, une unité arithmétique et logique… qui sont en réalité des composants électronique qui vont communiquer entre eux via des canaux d'interconnexion. Chaque module peut avoir des attribues et des méthodes comme toutes classe en C++.

6.6.1. Déclaration d'un module :

```
SC_MODULE(nom_module){
public :
} ;
```

Comme toutes classes en C++ chaque module possède au moins un constructeur qui a pour rôle de donner naissance au module, ou initialiser toutes les données dans le même module. Il n'a pas de valeurs de retour. Il peut être défini à l'intérieur de la

classe comme il peut l'être à l'extérieur. Le module et le constructeur ont le même nom et la syntaxe se présente ainsi :

```
SC_MODULE(nom_module)
{
public :
SC_CTOR (nom_module)
{ …….. }
}
```

6.7. Les processus

Sont une famille particulière de fonctions ou de procédures disponibles en systemc.

6.7.1. Les SC_METHOD

Les **SC_METHOD** sont du point de vue simulation, des fonctions normales sans variables locales, qui vont être appelées par leur ordonnanceur à chaque notification d'événement de leur liste de sensibilité. Elles constituent les objets de base qui permettent la mise en œuvre de la concurrence en systemc. Ces méthodes sont appelées lors de l'étape d'initialisation (SC_CTOR) et exécute leurs codes jusqu'à la dernière instruction. Lorsque celle-ci atteint la fin du code elle se bloc e attendant sont prochain événement de sensibilité. Par conséquent, il est judicieux de ne pas utiliser de boucle infinie dans les SC_METHOD.

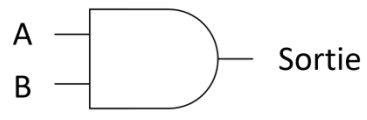
```
SC_METHOD (process_name) ;
sensitive<< signal1<<signal2<<signal3<<… ;
```

« *sensitive* » est une fonction écrite à l'intérieur du constructeur. elle prend en argument un objet de type événement ou un signal de 1 bit. Il est possible de choisir un des fronts montant (.pos()) ou descendant (.neg()), sinon elle est sensible à tous les fronts du signal.

Exemple :

Ecrire un model pour une porte ''ET'' logique en systemc.

```
# include ''systemc.h''
SC_MODULE(porte_ET)
{
sc_in<sc_uint<1> > A, B;
sc_out<sc_uint<1> > sortie ;
sc_in<bool> clk ;
void fonction ( )
{
sortie.write (A.read() & B.read() ) ;
}
SC_CTOR (porte_ET)
{
SC_METHOD (fonction) ;
sensitive<<clk.pos() ;
}
}
```



6.8.2 Les SC_THREAD

Contrairement aux ***SC_MEHOD*** qui sont utilisées pour la modélisation du matériel d'un système ***SC_THREAD*** ont un comportement similaire aux autres ***THREAD*** traditionnelles en logiciel (software). ils sont exécutées en boucle infinie pour un certain nombre d'instructions, ils sont exécutées une seule fois au début de a simulation et plus jamais après. Comme ils sont définies en boucles, elles doivent être mises en veille régulièrement La fonction permettant de suspendre l'exécution (mode veille) du SC_THEAD et la fonction *''wait ( )''* dont les arguments peuvent être un événement, un temps, ou elle peut être sans argument. Elles ont aussi une liste de sensibilité qui spécifie quand les SC_THREAD doivent être réveillées et reprendre l'exécution. Les variables locales gardent leurs valeurs avant e après l'exécution de « *wait* » pour redonner la main à l'ordonnanceur.

Syntaxe :

SC_THREAD(process_name) ;

sensitive<< signal1<<signal2<<signal3<<… ;

6.8. Les interfaces

L'interface est fortement liée au concept de canal. Il permet de séparer le comportement d'un canal spécifique de la façon dont il va être vu de l'extérieur. Une interface contient uniquement le prototype du canal. Autrement dit l'interface contient la liste des fonctions qi doivent être mises en œuvre dans le canal. Le canal mis en œuvre les méthodes qui sont définies dans interface. En général un canal met en œuvre deux méthodes primitives read() et write () alors que l'interface présente simplement les prototypes de ces deux fonctions.

6.9. Les ports

Les ports sont les parties d'un module par lesquels transitent toutes les informations qui entrent ou qui sortent de ce module. Donc un port permet au module d'accéder à l'interface du canal qui lui est connecté. Le port est un objet en systemc défini dans la classe « *sc_port* », il hérite d'une interface particulière, par conséquent il ne peut être relié qu'à un canal dont l'interface est le même.


Port d'entrée


port de sortie


port d'entrée/sortie

Syntaxe :

Sc_port<type_interfae,N> nom_port ;

Le nombre N précise le nombre de canaux auxquels peut être connecté le port.ans le cas où ce ombre n'était pas indiqué, l sera donc « 1 » par défaut.

Exemple :

Sc_port<sc_signal_in_if<int> > X ;
Sc_port<sc_fifo_out_if<char> > X1 ;
Sc_port<sc_fifo_in_if<double> > X2 ;

Exemple: sc_time

```
#include <systemc.h>

int sc_main (int argc, char* argv[]) {
  // sc_set_time_resol... should be called before sc_time
  // variable declration
  sc_set_time_resolution(1,SC_PS);
  // Declare the sc_time variables
  sc_time        t1(10,SC_NS);
  sc_time        t2(5,SC_PS);
  sc_time        t3,t4(1,SC_PS),t5(1,SC_PS);
  // Print all the variables
  cout <<"Value of t1 "<< t1.to_string() << endl;
  cout <<"Value of t2 "<< t2.to_string() << endl;
  cout <<"Value of t3 "<< t3.to_string() << endl;
  cout <<"Value of t4 "<< t4.to_string() << endl;
  cout <<"Value of t5 "<< t5.to_string() << endl;
  // Start the sim
  sc_start(0);
  sc_start(1);
  // Get the current time
  t3 = sc_time_stamp();
  cout <<"Value of t3 "<< t3.to_string() << endl;
  // Run for some more time
  sc_start(20);
 // Get the current time
 t4 = sc_time_stamp();
// Print the variables for new time
 cout <<"Value of t4 "<< t4.to_string() << endl;
 // This is how you do arth operation
 t5 = t4 - t3;
  cout <<"Value of t5 "<< t5.to_string() << endl;
 // This is how we do compare operation
 if (t5 > t2) {
   cout <<" t5 is greated then t2" << endl;
 } else {
   cout <<" t2 is greated then t5" << endl;
 }
 return 1;
}
```

Resultats

```
Value of t1 10 ns
 Value of t2 5 ps
 Value of t3 0 s
 Value of t4 1 ps
 Value of t5 1 ps
 Value of t3 1 ns
```

Value of t4 21 ns
Value of t5 20 ns
t5 is greated then t2

Exemple: SC_MODULE, read,write,wait, SC_CTOR, PORT

```
#include <systemc>
SC_MODULE(MODULE1)
{
  sc_signal<int> s;
  sc_port<sc_signal_out_if<int> > p;
  SC_CTOR(MODULE1) {
    SC_THREAD(selfWrite);
    SC_THREAD(selfRead);
    sensitive << s;
        SC_THREAD(outsideWrite);
  }
  void selfWrite() {
    int val = 1; // init value
    while (true) {
      s.write(val++);
      wait(1, SC_SEC);
    }
  }
  void selfRead() {
    while (true) {
      std::cout << sc_time_stamp() << ": reads from own channel, val=" << s.read() <<
std::endl;
      wait();
    }
  }
  void outsideWrite() {
    int val = 1; // init value
    while (true) {
      p->write(val++);
      wait(1, SC_SEC);
    }
  }
};
SC_MODULE(MODULE2) {
  sc_port<sc_signal_in_if<int> > p;
  SC_CTOR(MODULE2) {
    SC_THREAD(outsideRead);    sensitive << p;
    dont_initialize();
  }
```

```
  void outsideRead() {
    while (true) {
      std::cout << sc_time_stamp() << ": reads from outside channel, val=" << p->read() <<
std::endl;
      wait();    }
  }
};

int sc_main(int, char*[]) {
  MODULE1 module1("module1");
  MODULE2 module2("module2");   sc_signal<int> s;
  module1.p(s);
  module2.p(s);
  sc_start(2, SC_SEC);
  return 0;
}
```

Résultants

```
0 s: reads from own channel, val=1
0 s: reads from outside channel, val=1
1 s: reads from own channel, val=2
1 s: reads from outside channel, val=2
```

Bibliographie

1. Chkouri Mohamed Yacine, Modélisation des systèmes temps-réel embarqués en utilisant AADL pour la génération automatique d'applications formellement vérifiées, thèse de doctorat, université Joseph Fourier, France, 7 Avril 2010
2. Cours Systèmes Embarqués http://echnologuepro.com/cours-systemes-embarques/cours-systemes-embarques-introduction.htm
3. Cours : les actionneurs : http://staff.univ-batna2.dz/sites/default/files/meguellati-mohamed/files/actionneurs.pdf
4. Introduction aux Systèmes Temps Réel – STR http://data0.ek.la/sid/mod_article45964497_4f9db5dc9ad00.pdf?9528

Georges Asch, *Les capteurs en instrumentation indusrielle*, Dunod (ISBN 2100057774) Technique et ingénierie novembre 2017.

5. Cours : Systèmes embarqués et programmation temps réel A. E. benyamina file:///C:/Users/dell/Downloads/C3-Bis%20M1%20IEEP%20PTR.pdf
6. [AADb] SAE. Architecture Analysis & Design Language (standard SAE AS5506), September 2004, available at http ://www.sae.org.
7. 2 . LEI PI Langage de description d'architecture : sémantique et analyse comportementale. Thèse de doctorat de l'université de Toulouse, systèmes informatiques et systèmes embarqués. Juillet 2010 ;
8. Ian Sommerville, Software engineering, Addison Wesley, 2013
9. Robert Oshana; Mark Kraeling, Software Engineering for Embedded Systems, Newnes, June 3, 2013
10. C. Marley, Gestion d'un projet système d'information, Dunod, 2003
11. OMG, "UML 2.2 Superstructure Specification (formal/2009-02-04)." KORDON Fabrice, HUGUES Jérôme, CANALS Agusti, DOHET Alain , Modélisation et analyse de systèmes embarqués (Coll. SEE) , ISBN: 9782746239005 , Hermes Science, 2013