EN325: Modélisation et simulation multi-niveaux avec le langage SystemC

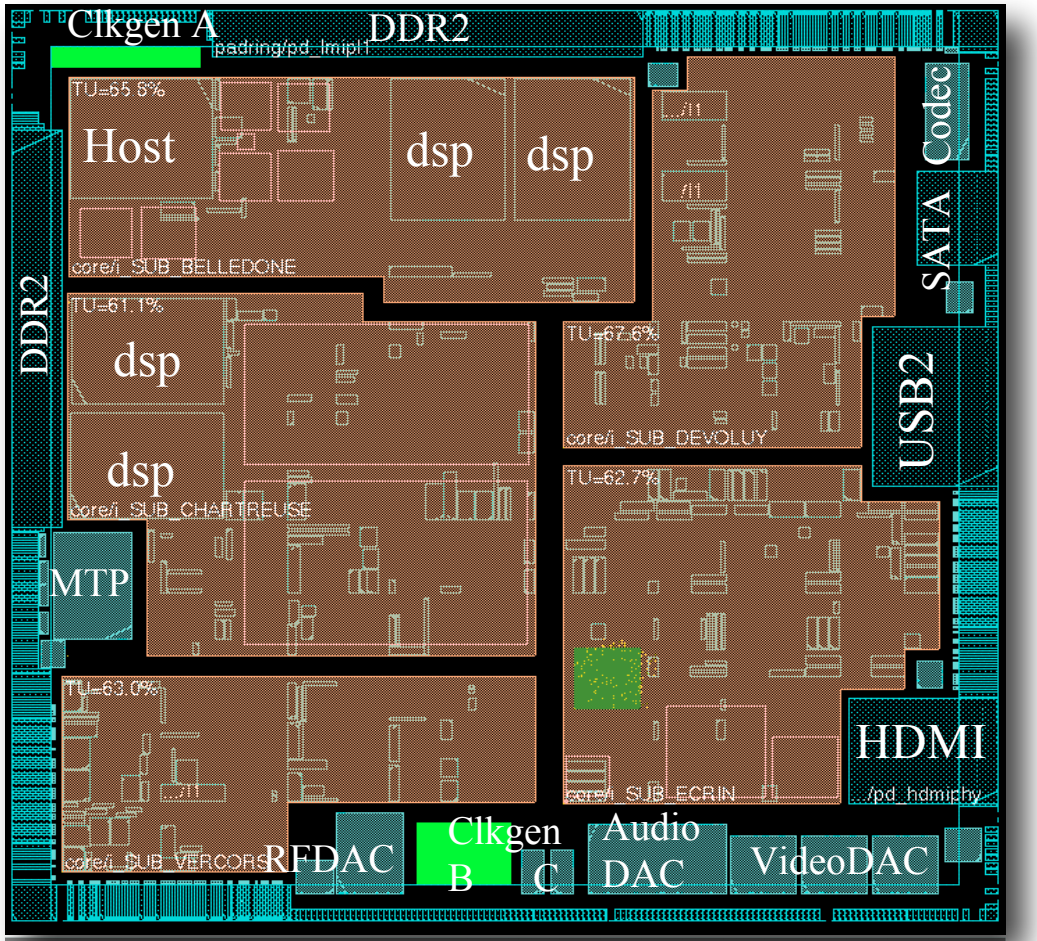
Bertrand LE GAL
[bertrand.legal@ims-bordeaux.fr]

Filière Electronique - 3ème année
ENSEIRB-MATMECA - Bordeaux INP
Talence, France

Introduction à SystemC

Exemple, le décodeur “HD 264” de STMicro

- Circuit dédié au décodage de la TV HD (norme H264)
  - Circuit contenant 150M transistors et 886 pads IOs (~5 GMIPS)
  - 128 sources d'interruption
  - 73 initiateurs et 96 cibles sur les bus,
  - 115 réseaux d'horloge (19 pour les interconnexions),
- Construction du système
  - 4 processeurs (2 DSP vidéo, 1 DSP audio et 1 uP généraliste pour la configuration du système),
  - 36 Soft IPs + 2 Hard IPs
  - 140 memory cuts

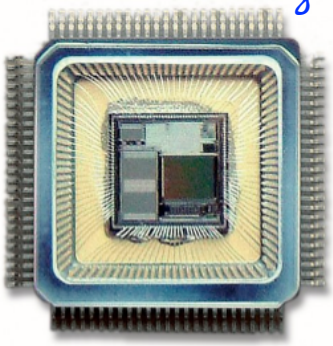


Circuit développé en 200X ?!
Que fait on alors aujourd'hui ...

Flot de développement de niveau système

Spécifications

Circuit numérique



Cahier des charges de l'application

Modélisation du système

Partitionnement logiciel / matériel

Modélisation du matériel

Modélisation des interfaces

Modélisation du logiciel + OS

Raffinement du matériel

Raffinement des interfaces

Raffinement du logiciel + OS

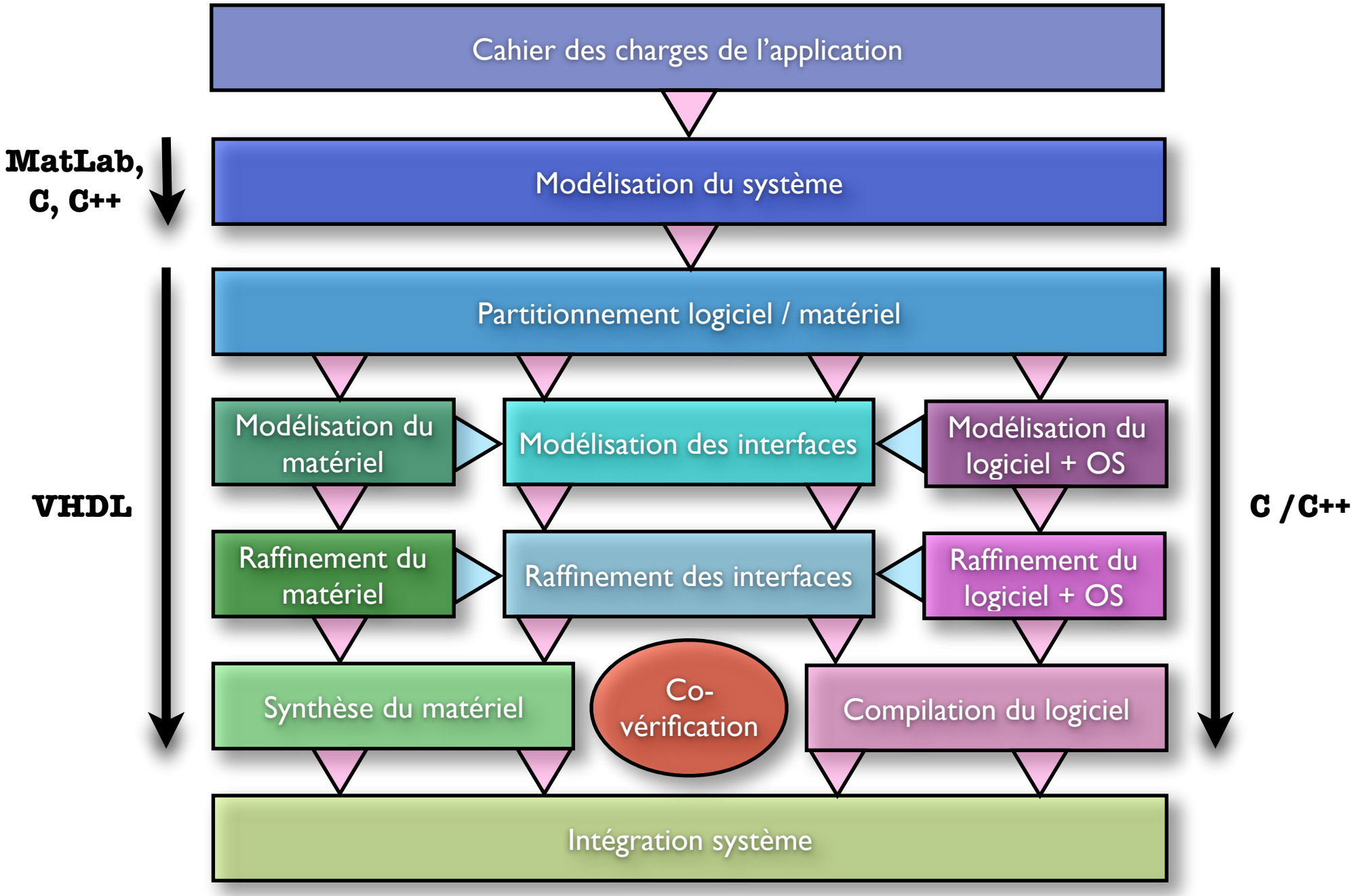
Synthèse du matériel

Co-vérification

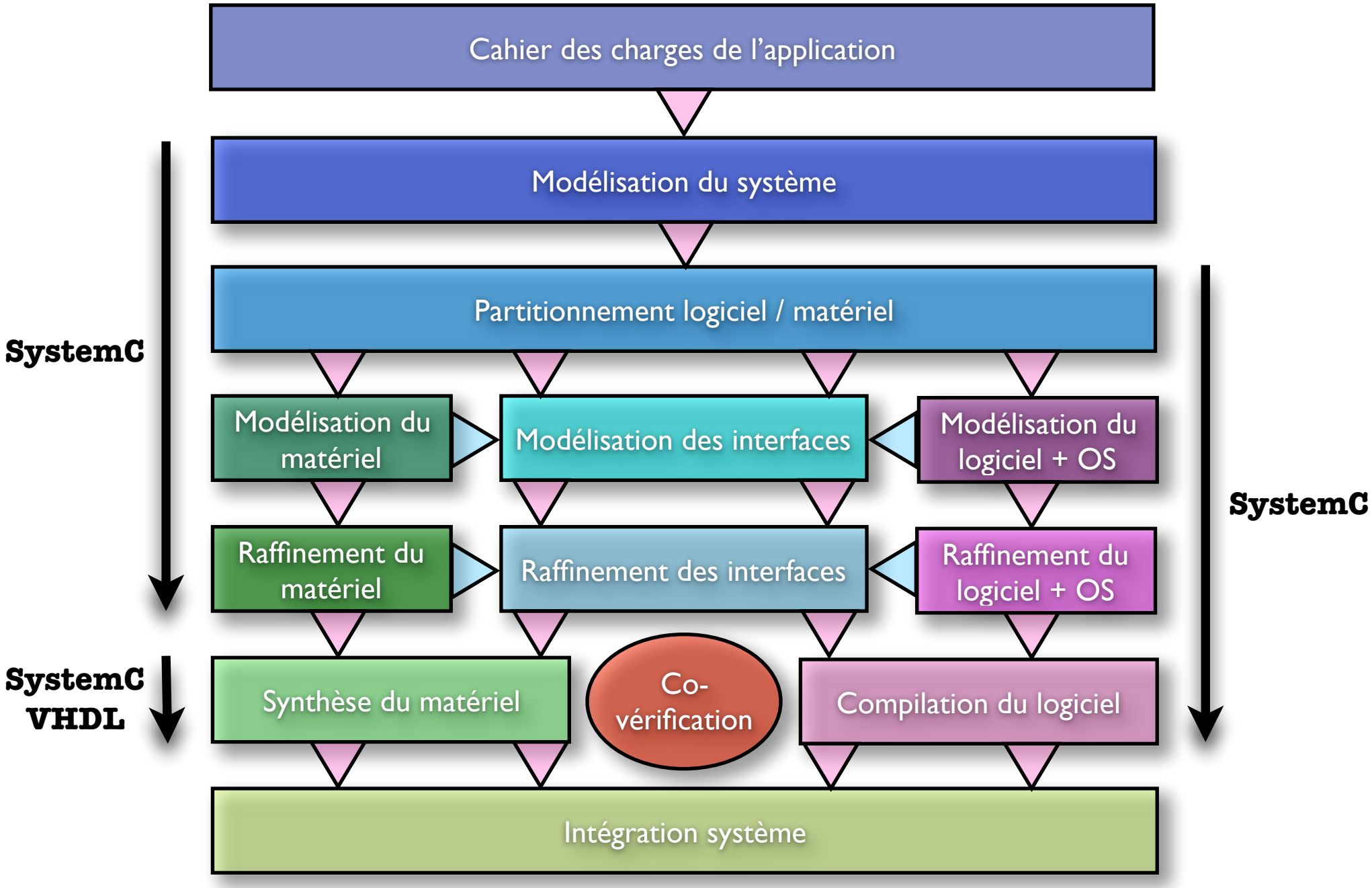
Compilation du logiciel

Intégration système

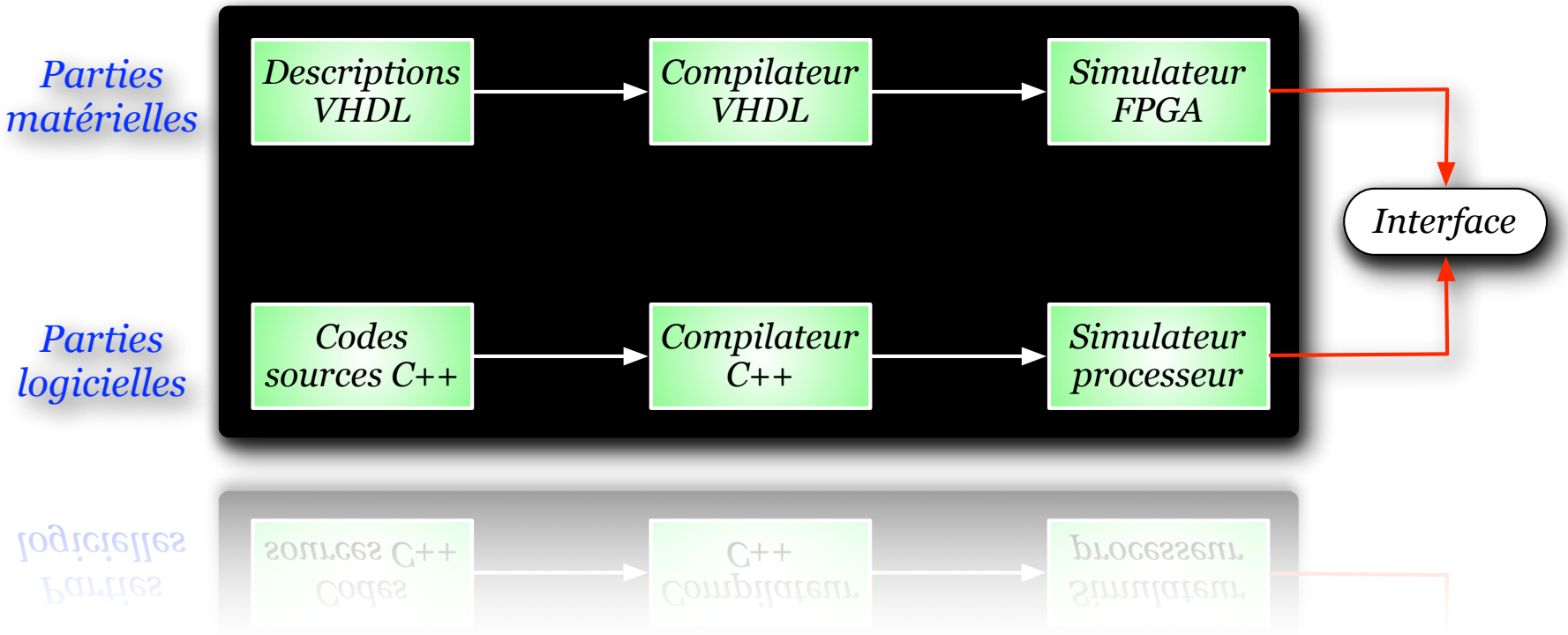
Flot de conception système usuel



Evolution du flot de conception à l'aide de SystemC



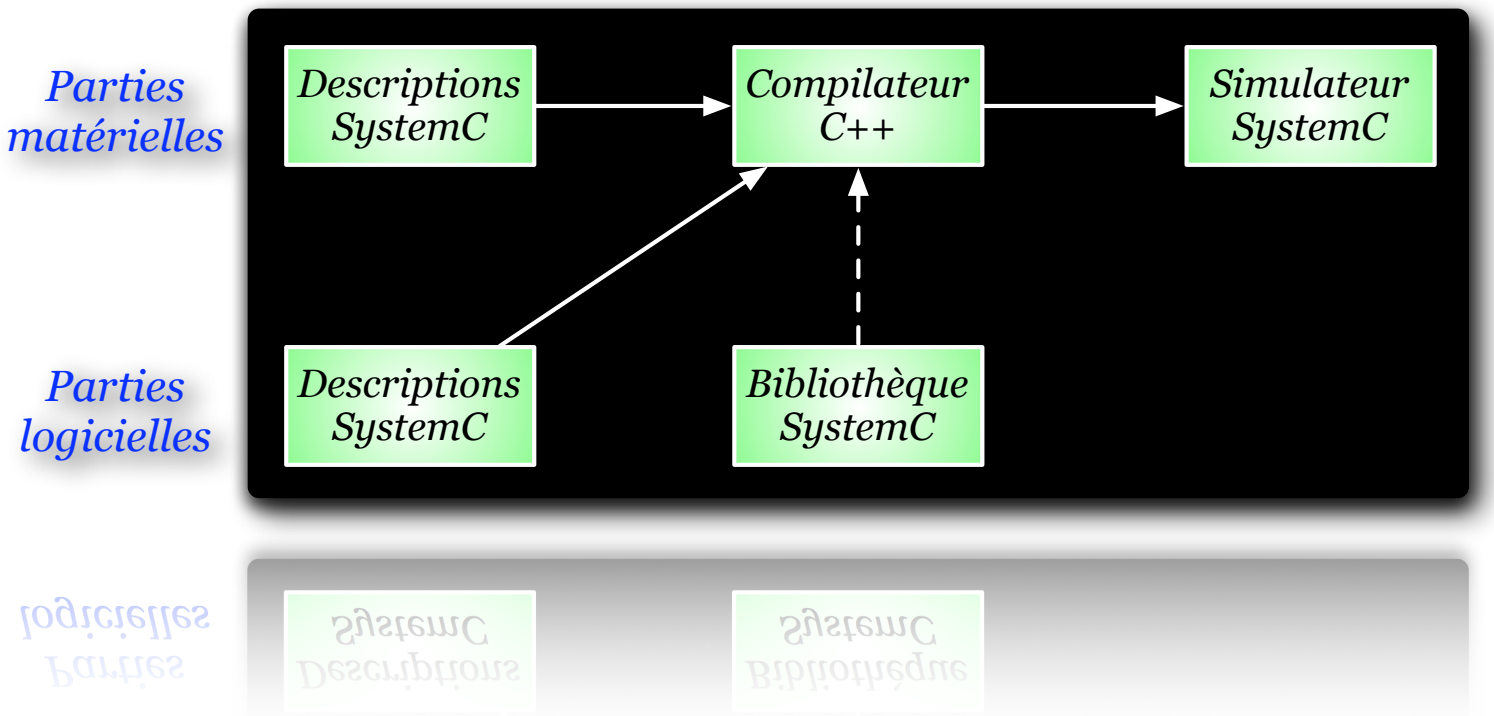
Simulation lors de conceptions conjointes (1)



Lors de la conception d'un système hétérogène, il est nécessaire de disposer de pléthore d'outils différents peu faits pour communiquer ensemble => Problème majeur.

De plus les interfaces de communication (entre outils) ralentissent grandement des simulations déjà longues !

Simulation lors de conceptions conjointes (2)



Le langage SystemC permet de simplifier les étapes de conception conjointe en unifiant les outils de conception.

Dans le même temps il permet de conserver un langage unique pour développer du logiciel et du matériel (moins de changements de langages implique moins d'erreurs de "traduction")

Pourquoi utiliser SystemC ?

- SystemC est un HDL (*Hardware Description Language*) qui a pour objectif de modéliser des systèmes numériques:
  - Le langage permet la modélisation des systèmes matériels,
  - Le langage permet la modélisation des systèmes logiciels,
  - Le langage permet la modélisation des systèmes mixtes (logiciels + matériels)

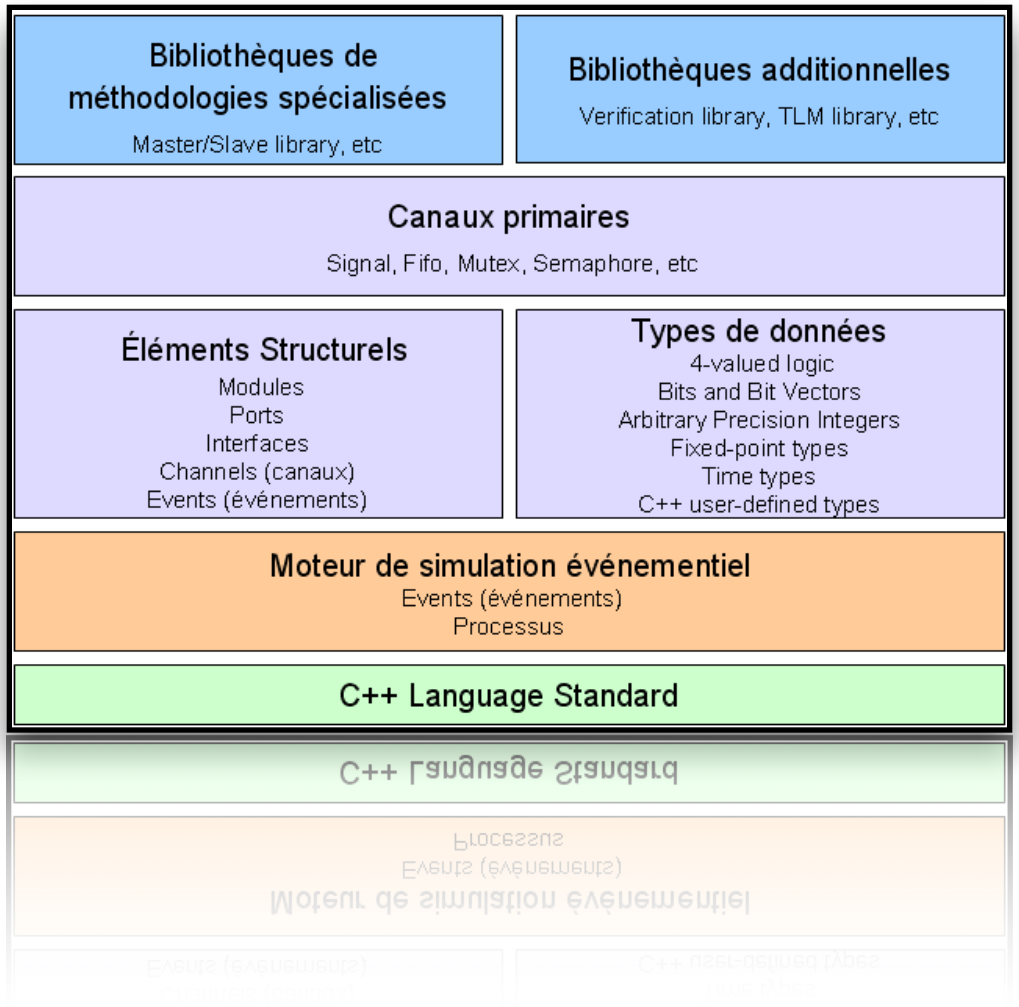
- L'objectif de la modélisation d'un système est:
  - De vérifier l'exactitude du comportement attendu,
  - Analyser et optimiser ses performances,
  - Etudier l'impact de certaines décisions (i.e. découpage logiciel/matériel),
  - De permettre le dimensionnement et optimiser l'implantation.

Introduction à SystemC

- SystemC n'est pas exactement un nouveau langage,
  - Il s'agit d'une bibliothèque objet décrite en langage C++,
  - Un modèle SystemC est donc un programme écrit en C++,
    - Utilise les classes de la bibliothèque logicielle SystemC.
- C'est le fruit de contributions de plusieurs sociétés
  - Synopsys met son outil commercial Scenic dans le domaine libre en 1989 et crée la version 0.9 de SystemC,
  - Frontier Design y apporte sa contribution qui donne lieu à la version 1.0,
  - CoWare apporte sa contribution et aboutit en 2000 à la version 1.1,
- L'OSCI (*Open SystemC Initiative*)
  - Rassemble une multitude de sociétés et laboratoires de recherche.
  - Cette organisation est en charge de diffuser et de rédiger les spécifications de SystemC.

Hiérarchie des classes SystemC

- SystemC est une bibliothèque de classes qui permet de modéliser le comportement de blocs logiciels & matériels,
- SystemC est un ensemble de briques de base utiles pour modéliser un système,
  - ➡ Modéliser des données typées, des canaux de communication (bus, fifos, mutex), etc...
- SystemC intègre d'origine un moteur de simulation événementiel,



A l'heure actuelle... que peut on faire avec SystemC ?

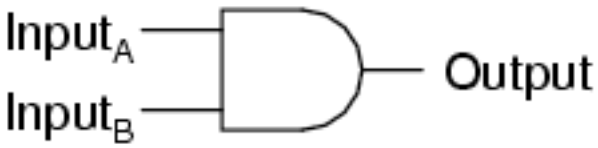
- A l'heure actuelle, SystemC propose une technologie mature permettant de modéliser des systèmes complexes composés de parties logicielles et matérielles,
  - Technique de gestion de processus (fork, join),
  - Technique de gestion de canaux abstraits (TML),
  - Technique de gestion des données au bit près (entières, flottantes, virgule fixe),
  - Bibliothèque de vérification (pour tester les systèmes),
- Le calendrier prévisionnel n'a pas été respecté mais le langage évolue tout en remplissant ses objectifs,
- L'outillage professionnel développé autour de SystemC présente des lacunes surtout à bas niveau (synthèse logique par exemple),

Partie 1
« Initiation au langage SystemC »

Partie 1
«La description des modules»

Premier contact avec le langage SystemC

2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Code source de la porte AND

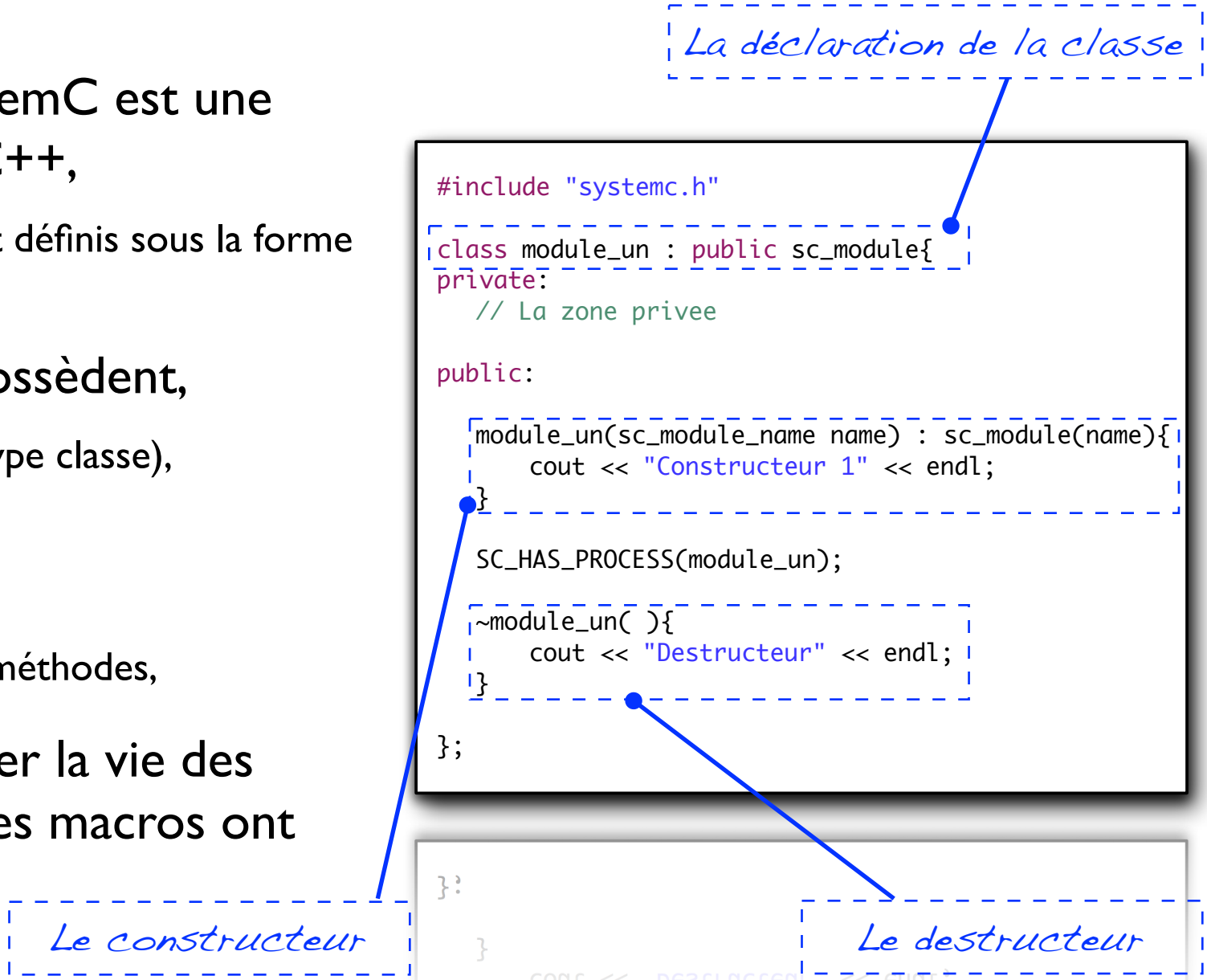
```
#include "systemc.h"

SC_MODULE(Porte_AND)
{
    sc_in<bool> a, b;
    sc_out<bool> s;

    SC_CTOR(Porte_AND)
    {
        SC_METHOD(do_and);
        sensitive << a << b;
    }

    void do_and();
};
```

```
void Porte_AND::do_and(){
    s = a && b;
}
```

Description du comportement

Comment définit on un module en SystemC ?

- Le langage SystemC est une extension du C++,
  - Les modules sont définis sous la forme de classes C++,
- Les modules possèdent,
  - Une définition (type classe),
  - Un constructeur,
  - Un destructeur,
  - Des attributs et méthodes,
- Afin de simplifier la vie des concepteurs, des macros ont été définies.



Une simplification de la syntaxe du langage C++

Description objet laborieuse ! Les constructeur doivent toujours appeler le constructeur du parent.

```
#include "systemc.h"

class module_un : public sc_module{
private:
   // La zone privee

public:

  module_un(sc_module_name name) : sc_module(name){
      cout << "Constructeur 1" << endl;
  }

  SC_HAS_PROCESS(module_un);

  ~module_un( ){
      cout << "Destructeur" << endl;
  }

};
```

L'utilisation des macros définies par SystemC permet de simplifier l'écriture du code sources des modules.

```
SC_MODULE(module_deux)
{
private:
   // La zone privee

public:
  SC_CTOR(module_deux)
  {
  }

  ~module_deux( ){
      cout << "Destructeur" << endl;
  }
};
```

Toutefois, attention aux pièges ?!

```
SC_MODULE(module_deux)
{
private:
   // La zone privee

public:
   SC_CTOR(module_deux)
   {
   }

   ~module_deux( ){
       cout << "Destructeur" << endl;
   }
};
```

Ce constructeur correspond à : module_deux(sc_module_name name)

```
void ma_fonction( ){
   module_deux  m1();
   module_deux  m2("un_nom");
   module_deux *m4 = new module_deux("???");
   delete m4;
}
```

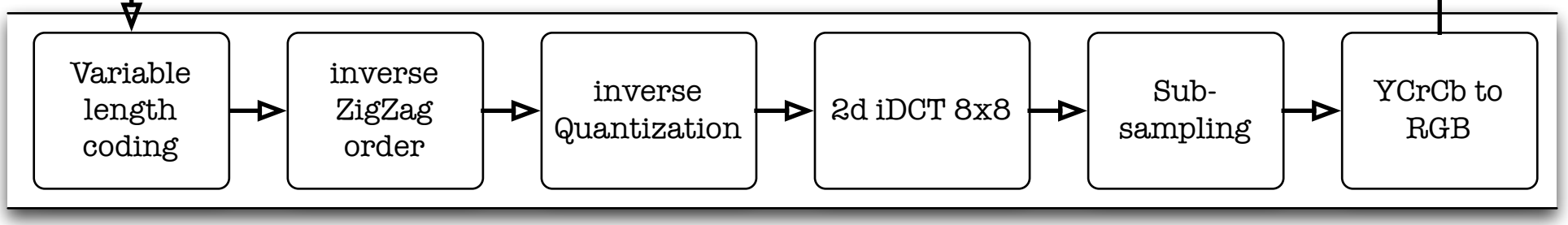
L'instanciation d'un objet en SystemC requiert un nom pour l'instance crée (le simulateur s'en sert pour les messages d'erreur).

Donc vous ne créez jamais d'objets à l'aide du constructeur vide (sans argument) !

Les 2 types de composants élémentaires

Un système se décrit à l'aide

de plusieurs modules interconnectés !



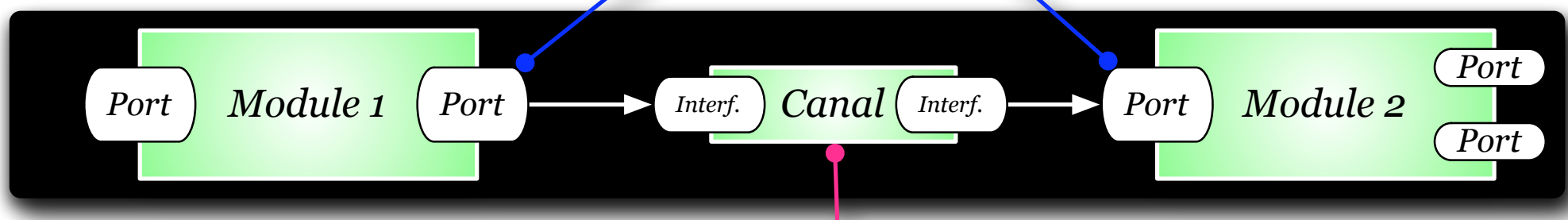
En SystemC il y a 2 types d'éléments principaux,

=> Les éléments structurels (les modules)

=> Les canaux de communication (liens entre les modules)

Terminologie utilisée dans le langage SystemC

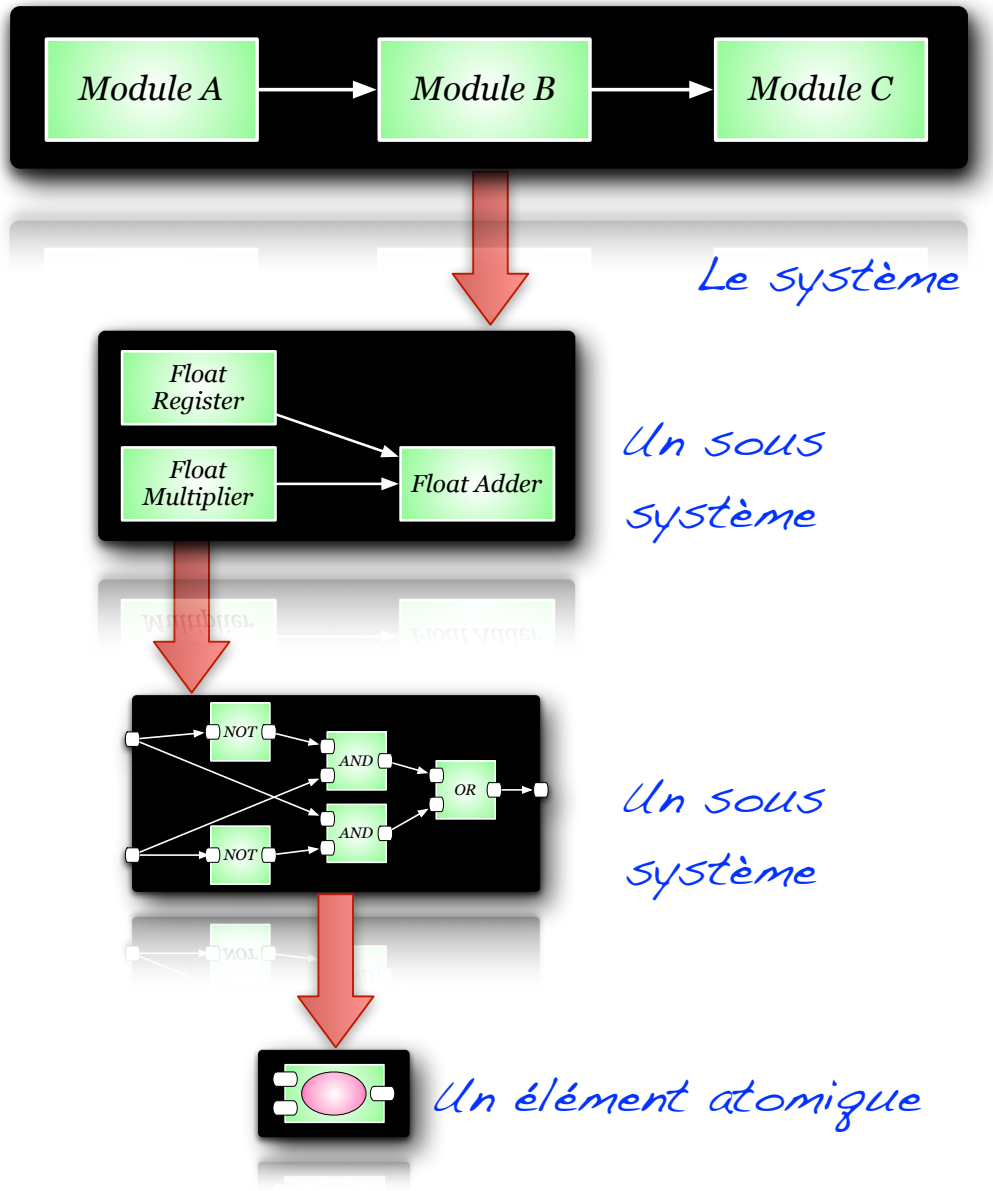
Les ports de communication appartiennet aux modules et lui servent à communiquer avec son environnement



Le canal de communication sert d'interface "phisique" afin d'assurer le transfert des informations (fils, bus, fifo, wifi, etc...)

Définition 1 : Les modules

- Un module en SystemC est être composé de ports (E/S), de canaux de communication ainsi que de processus.
- Exemples
  - Un *processeur* est un module (qui contient éventuellement d'autres sous-modules : registres, UALs, ...),
  - Les portes AND, OR, NOT,
  - Une mémoire RAM,
  - Un réseau (NoC, Internet, etc.) peut être représenté comme un module.
  - Une caméra.



Partie 1
«Les interfaces d'E/S des modules»

Définition 2 : Les ports de communication

- Les entrées-sorties des modules sont appelées des ports.
  - Les ports sont déclarés dans les classes comme des attributs publics.
- Les ports sont des éléments doublement typés,
  - Un type spécifie le sens du port de communication, entrée du module (sc_in<T>), sortie du module (sc_out<T>) ou entrée et sortie (sc_inout<T>)
  - Un type (ou paramètres) va spécifier la nature du port de communication qui va interagir avec l'extérieur (int, char*, sc_int, sc_bigunit, ma_structure, ...),
- Exemples
  - L'entrée d'horloge d'un processeur est un port d'entrée (sc_in), qu'on appellera par exemple clk. Elle sera reliée à un signal normal (un fil), donc de type booléen (bool),
    - sc_in<bool> clk;
  - Le bus d'adresse d'un processeur 32 bits nommé mon_bus est un port de sorties (sc_out). Il pourra être déclaré comme suit:
    - sc_out<sc_lv<32> > mon_bus
    - sc_out<sc_uint<32> > mon_bus

Modélisation des valeurs logiques (std_logic_vector)

- Le langage SystemC intègre des classes permettant de des types de données plus proche du matériel.
  - sc_bit
    - Cette classe permet de modéliser les bits qui peuvent prendre la valeur 1 signifiant true et 0 pour false.
  - sc_logic
    - Cette classe permet de modéliser les valeurs logiques (bits) qui nécessitent les 4 états : {0, 1, X, Z} que ne gère pas intégralement sc_bit.
  - sc_bv<N>
    - Cette classe permet de modéliser un vecteur de bits (éléments de type sc_bit) au sein d'un même objet,
  - sc_lv<N>
    - Cette classe permet de modéliser un ensemble d'éléments de type sc_lv au sein d'un même objet,

Equivalences entre les types d'E/S en VHDL et en SystemC

VHDL Port	SystemC Port	SystemC Notes
in	sc_in	
out	sc_out	Behaves like a VHDL buffer port
inout	sc_inout	
buffer	sc_inout	

VHDL	SystemC	SystemC Notes
bit	sc_bit	Use native C++ bool type instead
bit_vector	sc_bv	Faster in simulation than sc_lv
std_ulogic	sc_logic	Only support 4 types, 'X','Z','0' and '1'
std_ulogic_vector	sc_lv	Only support 4 types, 'X','Z','0' and '1'
std_logic	sc_logic_resolved	Resolution function for 'X','Z','0' and '1'
std_logic_vector	sc_signal_rv	Resolution function for 'X','Z','0' and '1'
boolean	bool	Recommended instead of sc_bit

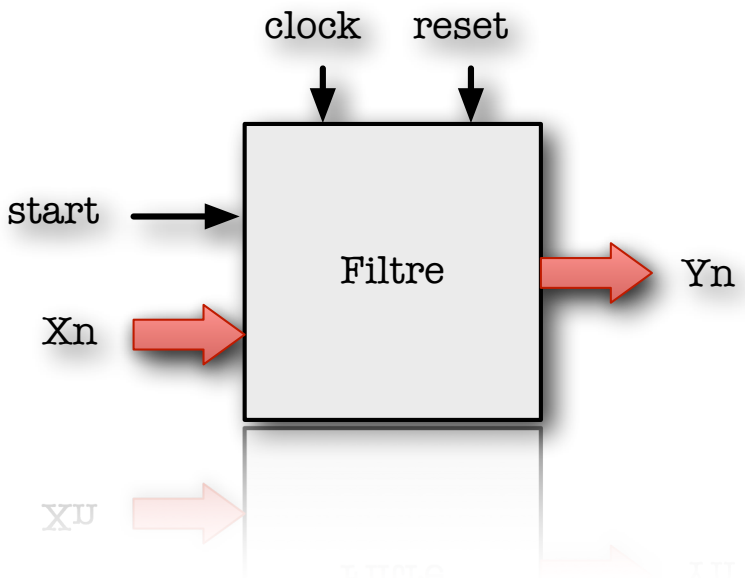
http://www.ht-lab.com/howto/vh2sc_tut/vh2sc_tut2.html

Modélisation des valeurs logiques (std_logic_vector)

```
sc_bit a, b;
bool c;
a = '1';
b = false;
a &= b;
if( c || a) {...}

sc_logic out;
bool data, enable;

if(enable)
  out = data;
 else
  out = 'Z';
```

```
sc_bv<16> data16;
sc_bv<16> data15;
sc_bv<8> x;
sc_bit read_ok;

x = "01000111";

data32.range(14,0)  = data15;
data32.range(15,15) = data15.xor_reduce();
```

Exemples de déclarations en SystemC et en VHDL



Les déclaration d'un filtre FIR en VHDL et en SystemC sont similaires au langage près.

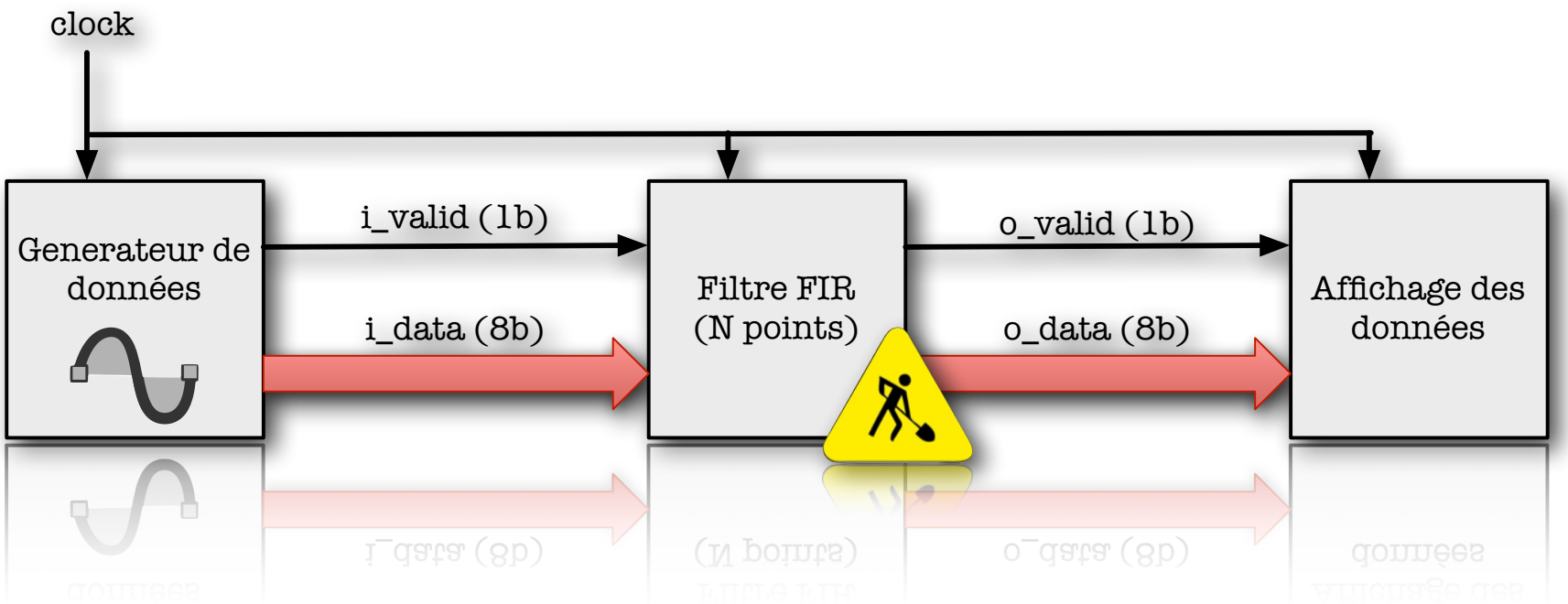
```
ENTITY Filtre is
  PORT (
     CLOCK : IN  STD_LOGIC;
     RESET : IN  STD_LOGIC;
     START : IN  STD_LOGIC;
     XN    : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
     YN    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
  );
END Filtre;
```

Déclaration en VHDL

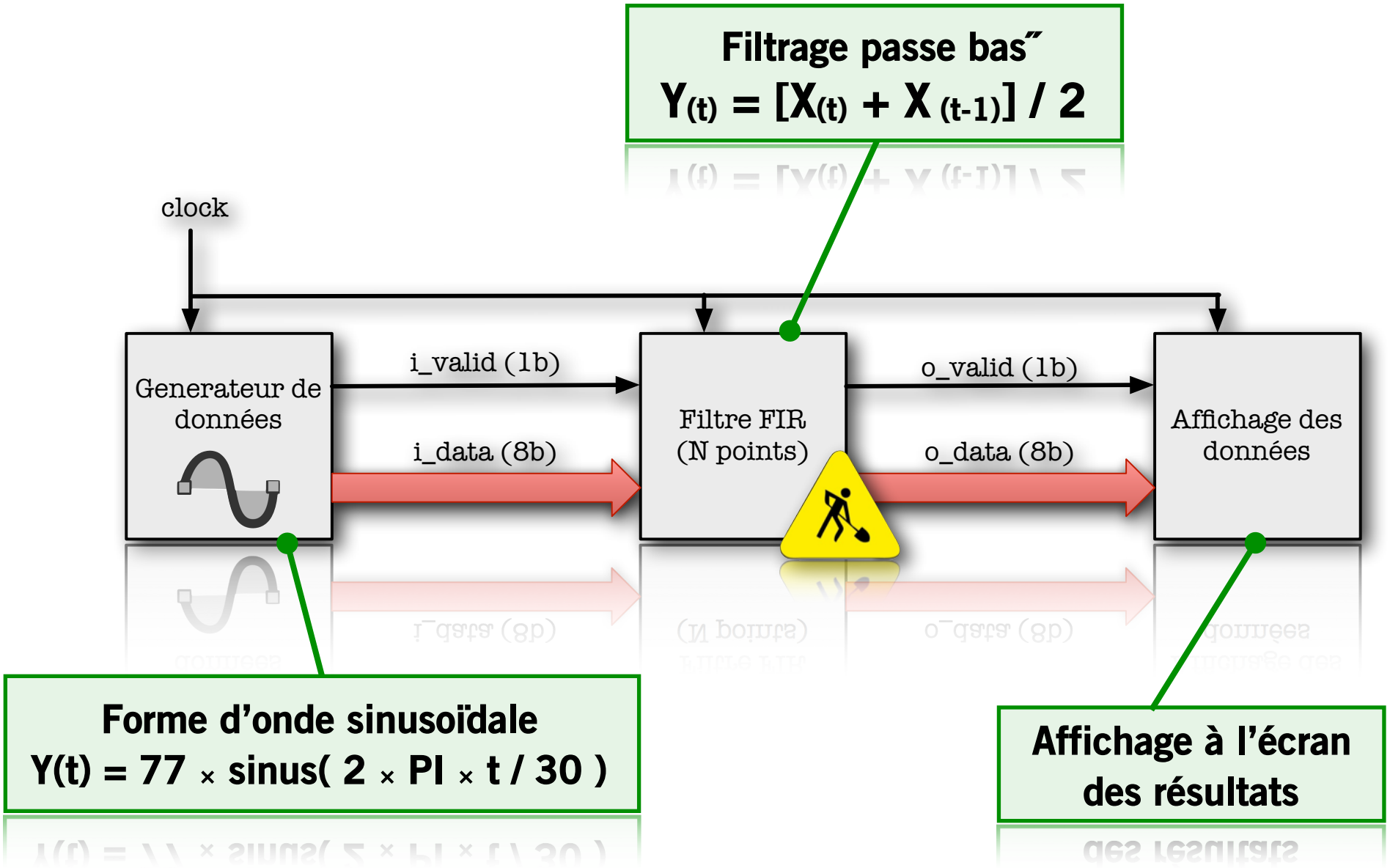
```
SC_MODULE(Filtre)
{
public:
      sc_in <bool>    clk;
      sc_in <bool>  reset;
      sc_in <bool>  start;
      sc_in < sc_lv <16> > Xn;
      sc_out< sc_lv <16> > Yn;

      SC_CTOR(Filtre){
      }
};
```

Déclaration en SystemC

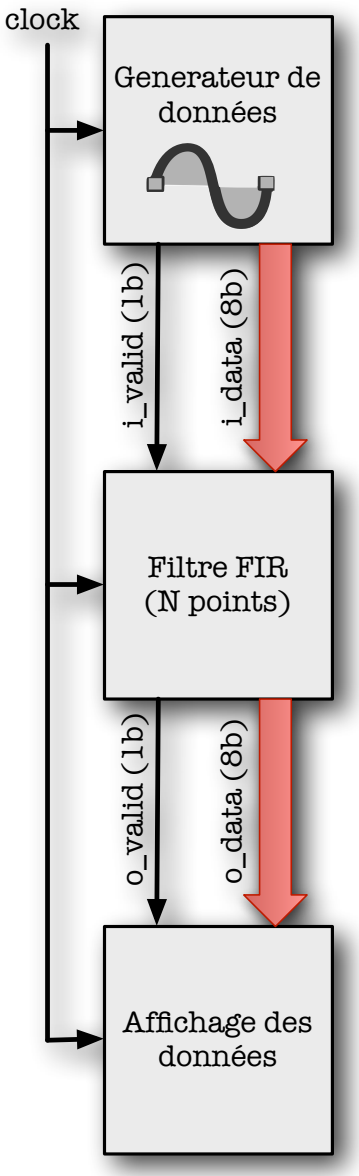
Exemple: description d'un système (VHDL versus SystemC)



Exemple: description d'un système (VHDL versus SystemC)

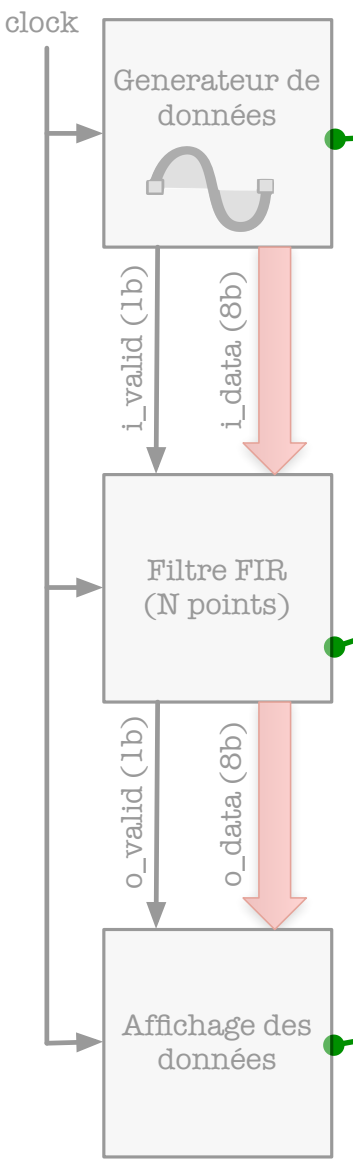


Exemple pédagogique (VHDL)



Ecrivez les entités VHDL correspondant à ce système.

Exemple pédagogique (VHDL)

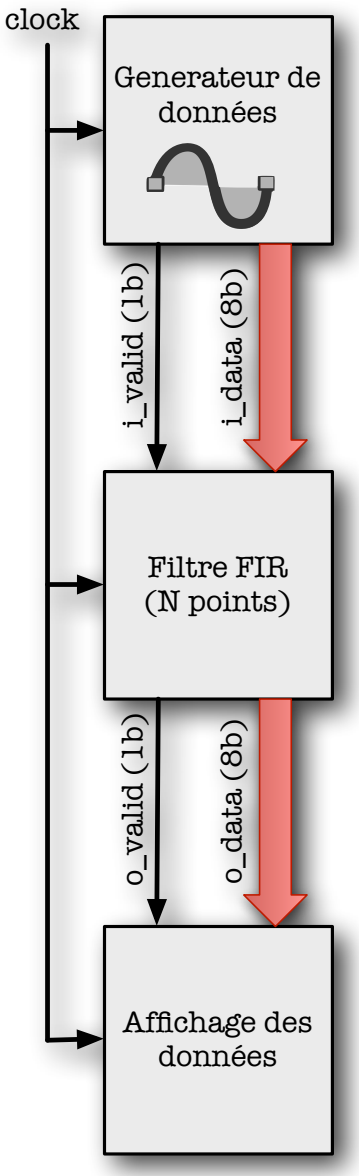



```
ENTITY Generator IS
  PORT (
     CLOCK  : IN  STD_LOGIC;
     RESET  : IN  STD_LOGIC;
     OUTPUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
     O_VALID: OUT STD_LOGIC
  );
END Generator;
```

```
ENTITY Filtre is
  PORT (
     CLOCK  : IN  STD_LOGIC;
     RESET  : IN  STD_LOGIC;
     INPUT  : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
     I_VALID: IN  STD_LOGIC;
     OUTPUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
     O_VALID: OUT STD_LOGIC
  );
END Filtre;
```

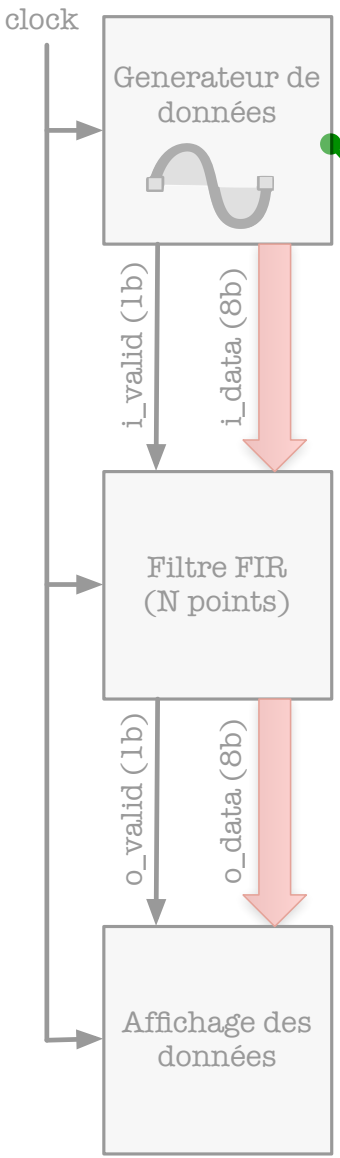
```
ENTITY Afficheur is
  PORT (
     CLOCK  : IN STD_LOGIC;
     RESET  : IN STD_LOGIC;
     INPUT  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     I_VALID: IN STD_LOGIC
  );
END Afficheur;
```

Exemple pédagogique (SystemC) - Module de filtrage



Ecrivez les modules SystemC correspondant à ce système.

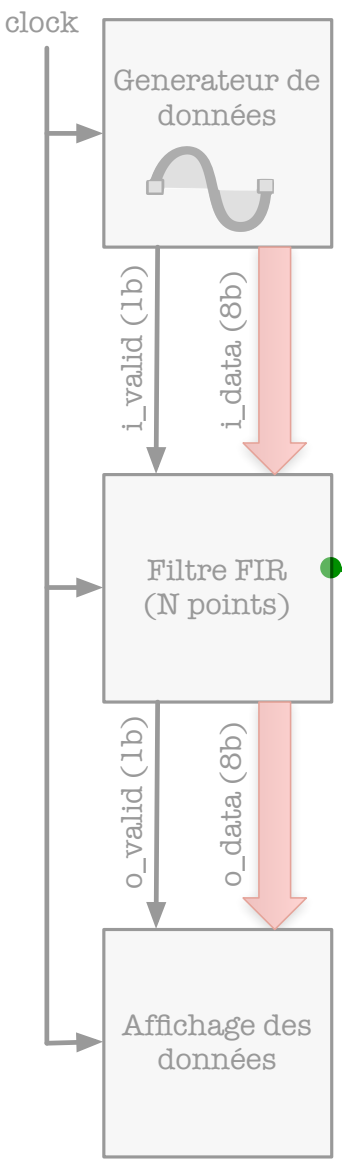
Exemple pédagogique (SystemC) - Module de filtrage




```
SC_MODULE(Gene)
{
public:
      sc_in <bool>       clk;
      sc_out<bool>  o_valid;
      sc_out< sc_lv <8> > s;

      SC_CTOR(Gene){
      }
};
```

Ecrivez les entités VHDL correspondant à ce système.

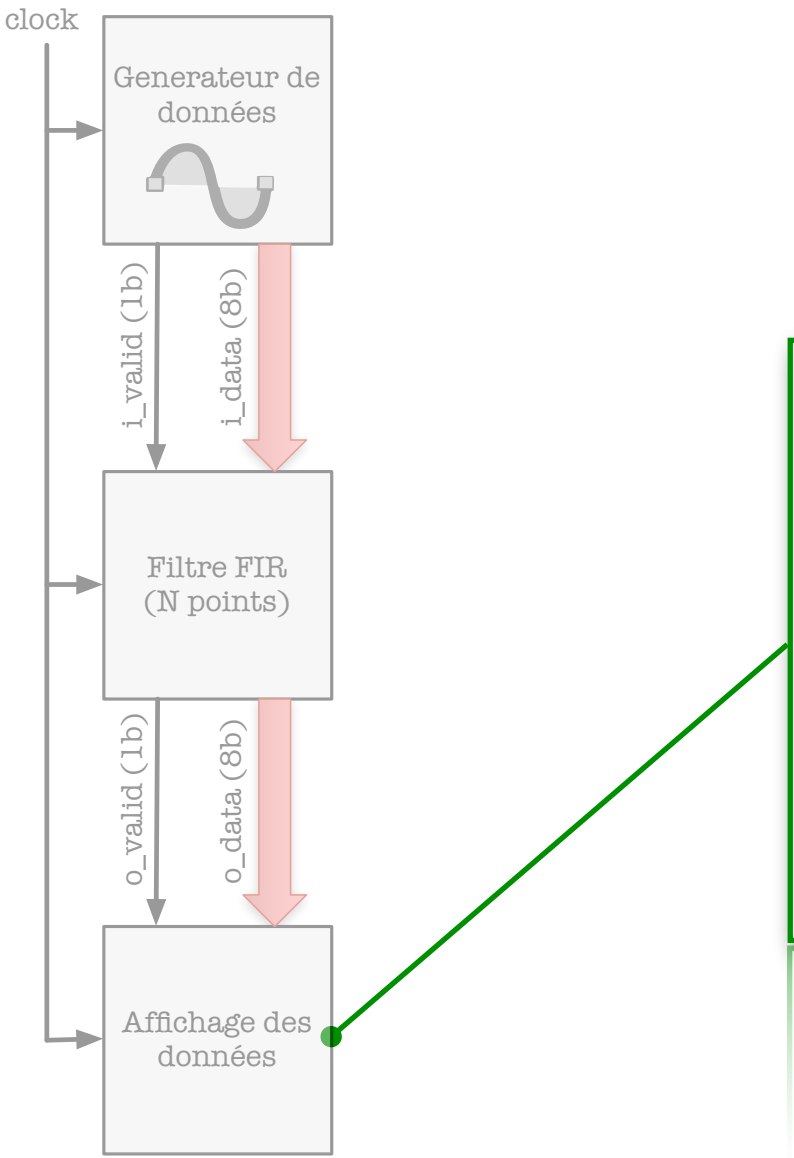
Exemple pédagogique (SystemC) - Module de génération




```
SC_MODULE(FIR_2pts)
{
public:
      sc_in <bool>       clk;
      sc_in <bool>  i_valid;
      sc_out<bool>  o_valid;
      sc_in < sc_lv <8> > e;
      sc_out< sc_lv <8> > s;

      SC_CTOR(FIR_2pts){
      }
};
```

Ecrivez les entités VHDL correspondant à ce système.

Exemple pédagogique (SystemC) - Module d'affichage




```
SC_MODULE(Terminal)
{
public:
      sc_in<bool>       clk;
      sc_in<bool>   i_valid;
      sc_in< sc_lv<8> >   e;

      SC_CTOR(Terminal){
      }
};
```

Ecrivez les entités VHDL correspondant à ce système.

Partie 1
«Modélisation du comportement»

Définition 3 : Les processus

- Les processus sont des méthodes qui décrivent toute ou partie du comportement d'un module.
- Un processus (la fonction) ne doit jamais être appelé directement,
  - Le moteur de simulation SystemC se charge d'exécuter la fonction lorsqu'un événement relatif signaux d'entrée du processus sont détectés (changement des entrées implique peut être un changement des sorties ?).
- Exemples
  - Une porte ET doit re-calculer sa sortie dès que l'une de ses entrées change. La fonction qui calcule le comportement (ET) est modélisée à l'aide d'un processus réactif sur les entrées de la porte. Lorsqu'une entrée est modifiée, le processus de calcul est exécuté automatiquement par le simulateur et la sortie mise à jour.
  - Un registre n'est mis à jour que sur front montant de l'horloge. Le processus chargé du stockage de l'entrée (et de la mise à jour de la sortie) est réactif uniquement sur l'horloge qui doit être dans sa liste de sensibilité.
  - Un composant de compression vidéo type MPEG doit être "exécuté" uniquement lorsqu'une nouvelle image est disponible et à condition qu'il ait terminé de traiter la précédente.

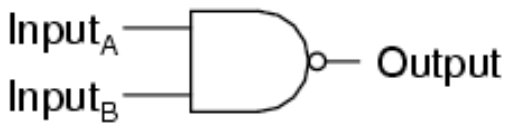
Gestion de ces différents comportements

- SystemC met à la disposition du concepteur 3 types de processus possédant des propriétés différentes,
  - SC_METHOD
    - Les processus de ce type sont privilégiés pour les composants dont le comportement est déclenché à chaque changement d'une de ses entrées. Ce type est souvent utilisé pour modéliser les composants combinatoires (asynchrones),
  - SC_THREAD
    - Les processus de ce type sont mis en oeuvre pour gérer les comportements synchrones ou la liste de sensibilité peu évoluer dans le temps (FSM dont les actions provoquant des transitions varient en fonction de l'état courant),
  - SC_CTHREAD
    - Les processus de ce type correspondent à un sous-ensemble des SC_THREAD. Ces processus évoluent uniquement sur les fronts d'horloge (sensibilité unique).
- Le choix du type de modélisation est guidé par le type de comportement à modéliser.

Définition 3 : Les processus (modélisation combinatoire)

```
#include "systemc.h"

SC_MODULE(NAND_Gate)
{
public:
     sc_in < bool > a;
     sc_in < bool > b;
    sc_out< bool > s;

     SC_CTOR(NAND_Gate)
     {
          SC_METHOD(do_nand);
          sensitive << a << b;
     }

    void do_nand();
 };
```

2-input NAND gate



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

```
#include "NAND_Gate.h"

void NAND_Gate::do_nand(){
     s = !(a && b);
}
```

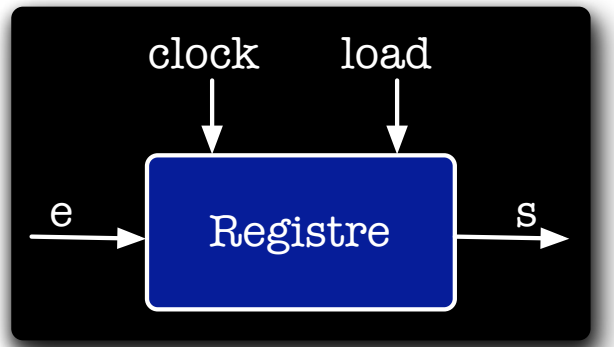
Définition 3 : Les processus (modélisation combinatoire)

```
#include "systemc.h"

SC_MODULE(NAND_Gate)
{
public:
     sc_in < bool > a;
     sc_in < bool > b;
    sc_out< bool > s;

     SC_CTOR(NAND_Gate)
     {
          SC_THREAD(do_nand);
          sensitive << a << b;
     }

    void do_nand();
 };
```

```
#include "NAND_Gate.h"

void NAND_Gate::do_nand(){
     while( true ){
          wait( );
          s = !(a && b);
     }
}
```

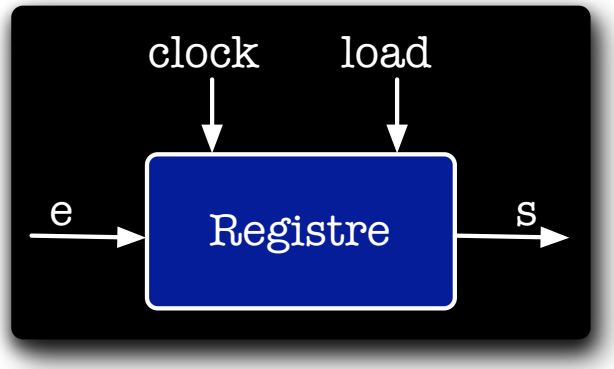
Définition 3 : Les processus (modélisation séquentielle)

```
#include "systemc.h"

SC_MODULE(Registre)
{
private:
     sc_lv<8> lastValue;

public:
     sc_in <bool> clk;
     sc_in <bool> l;
     sc_in < sc_lv<8> > e;
    sc_out< sc_lv<8> > s;

     SC_CTOR(Registre)
     {
          SC_METHOD(do_reg);
          sensitive << clk.pos();
          lastValue = (sc_uint<8>)0;
     }

    void do_reg();
 };
```




```
#include "Registre.h"

void Registre::do_reg(){
     if( l.read() == 1 ){
          s = e.read();
          lastValue = e;
     }else{
          s = lastValue;
     }
}
```

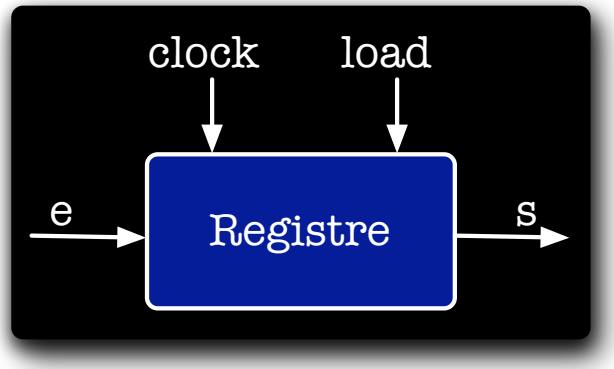
Définition 3 : Les processus (modélisation séquentielle)




```
#include "systemc.h"

SC_MODULE(Registre)
{
public:
     sc_in <bool> clk;
     sc_in <bool> l;
     sc_in < sc_lv<8> > e;
    sc_out< sc_lv<8> > s;

     SC_CTOR(Registre)
     {
          SC_THREAD(do_reg);
          sensitive << clk.pos();
     }

    void do_reg();
 };
```

```
#include "Registre.h"

void Registre::do_reg(){
     s = 0;
     while( true ){
          wait();
          if( l.read() == 1 ){
               s = e.read();
          }
     }
}
```

Définition 3 : Les processus (modélisation séquentielle)




```
#include "systemc.h"

SC_MODULE(Registre)
{
public:
     sc_in <bool> clk;
     sc_in <bool> l;
     sc_in < sc_lv<8> > e;
    sc_out< sc_lv<8> > s;

     SC_CTOR(Registre)
     {
          SC_CTHREAD(do_reg, clk.pos());
     }

    void do_reg();
 };
```

```
#include "Registre.h"

void Registre::do_reg(){
     s = 0;
     while( true ){
          wait();
          if( l.read() == 1 ){
               s = e.read();
          }
     }
}
```

Equivalences entre VHDL et SystemC

VHDL	SystemC	SystemC Notes
boolean	bool	Recommended instead of sc_bit
unsigned	sc_uint<N>	N bits unsigned integer vector (N <=64)
	sc_biguint<M>	Arbitrary length unsigned integer vector
signed	sc_int<N>	N bits signed integer vector (N <=64)
	sc_bigint<M>	Arbitrary length signed integer vector
integer	int	int size is platform dependent, use sizeof(int)
	sc_int<N>	N bits signed integer vector (N <=64)
	sc_bigint<M>	Arbitrary length signed integer vector
real	float	Simulation purpose only, Native C/C++ 32bits floating point number

http://www.ht-lab.com/howto/vh2sc_tut/vh2sc_tut2.html

Modélisation des valeurs numériques (signed/unsigned)

- Le langage SystemC intègre des classes permettant de modéliser des calculs avec une définition proche du matériel.
- Modélisation des "nombres entiers" (1 => 64 bits)
  - sc_int<T>, classe modélisant des nombres entiers positifs ou nuls de précision figée à l'instanciation de l'objet (T).
  - sc_uint<T>, classe modélisant des nombres entiers positifs ou négatifs de précision figée à l'instanciation de l'objet (T).
- Modélisation de grands nombres (<MAX_NBITS bits)
  - sc_bigint<T>, classe modélisant des nombres entiers positifs ou nuls de variant jusqu'à 512 bits. La précision souhaitée est spécifiée à l'instanciation de l'objet (T).
  - sc_biguint<T>, classe modélisant des nombres entiers positifs ou négatifs variant jusqu'à 512 bits. La précision souhaitée est spécifiée à l'instanciation de l'objet (T).

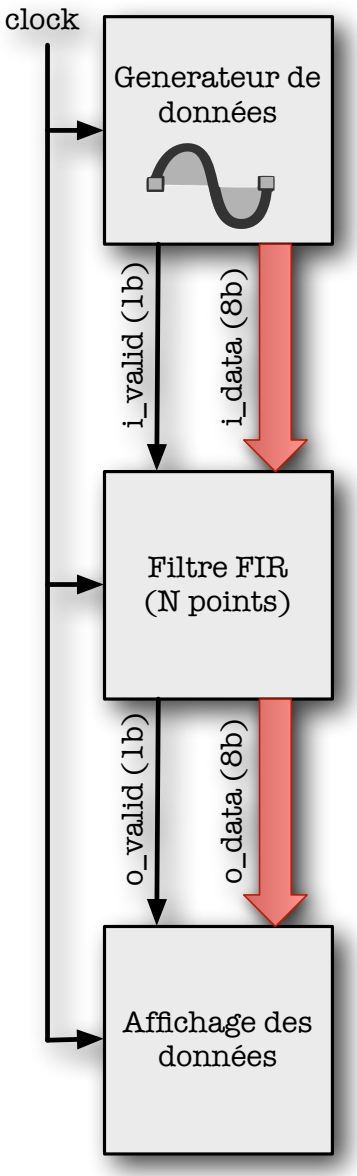
Modélisation des valeurs numériques (signed/unsigned)

- Les classes associées à la modélisation des nombres entiers proposent les méthodes suivantes :
  - Constructeurs adaptés à la création d'objets avec ou sans données à initialiser,
  - Sélection de bit à l'aide des opérateurs "[ ]"
  - Test de la valeur d'un bit contenu dans le vecteur bool test(int)
  - Forcage la valeur d'un bit contenu dans le vecteur set(int, bool)
  - Récupérer la largeur en nombre de bit de la donnée int length( )
  - Sélection d'un ensemble de bits par la méthode range(int, int)
  - Concaténation de bits avec l'opérateur ", " ou la méthode concat( )
- SystemC impose le MSB à gauche (n-1) et le LSB est à droite (0).

Manipulation des valeurs numériques (signed/unsigned)

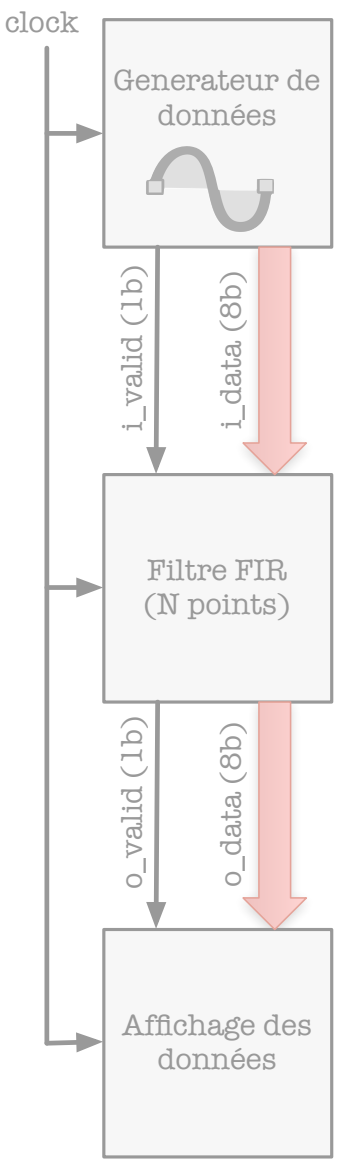
```
sc_uint <32> input, output; // entier non signé sur 32 bits
sc_uint <1>  sign;          // entier non signé sur 1  bit
sc_uint <7>  expo;          // entier non signé sur 7  bits
sc_uint <24> mant;          // entier non signé sur 24 bits

sign = input.test(32);
// sign[0] = input[32];

mant = input.range(23, 0);
expo = input.range(30, 24);

sign.set( ! sign );

expo = expo + 1;

output = (sign, expo, mant);
```

Exemple pédagogique (SystemC) - Fusion des modules



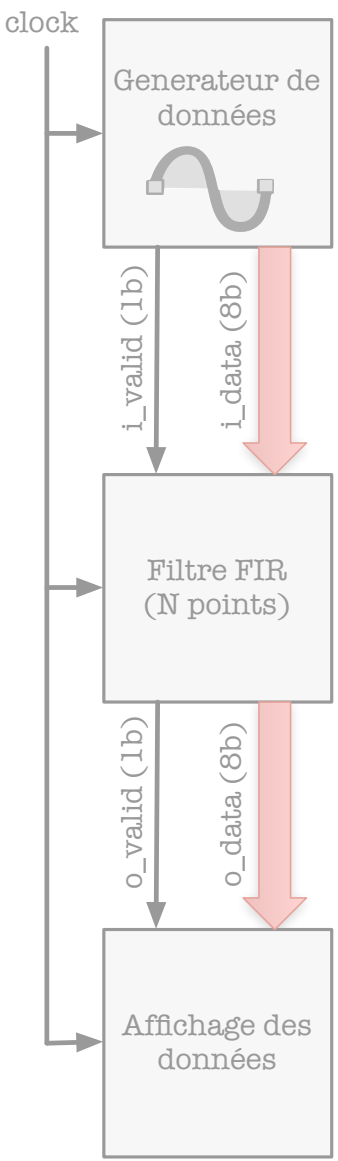
Ecrivez les architectures VHDL correspondant aux modules du système.

Exemple pédagogique (VHDL) - Fusion des modules



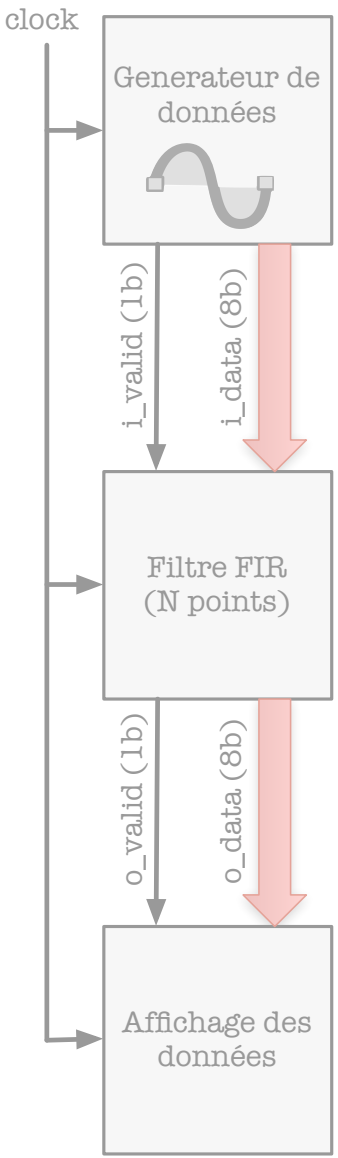

```
architecture Behavioral of Generator is
     --ROM for storing the sine values generated by MATLAB.
     type memory_type is array (0 to 29) of integer range -128 to 127;
     signal sine : memory_type :=(
          0,  16,  31,  45,  58,  67,  74,  77,  77,  74,
         67,  58,  45,  31,  16,   0, -16, -31, -45, -58,
       -67, -74, -77, -77, -74, -67, -58, -45, -31, -16);

     signal i : integer range 0 to 30:=0;
BEGIN

     PROCESS(CLOCK)
     BEGIN
          if(rising_edge(CLOCK)) then
               OUTPUT <= STD_LOGIC_VECTOR( TO_SIGNED(sine(i), 8) );
               i <= i+ 1;
               if(i = 29) then
                    i <= 0;
               end if;
               o_VALID <= '1';
          END IF;
     END PROCESS;

END Behavioral;
```

Exemple pédagogique (VHDL) - Fusion des modules




```
ARCHITECTURE Behavioral of Filtre is
     SIGNAL MEMORY : STD_LOGIC_VECTOR(7 downto 0);
BEGIN

     PROCESS(CLOCK)
          VARIABLE DATA_1 : SIGNED(7 downto 0);
          VARIABLE DATA_2 : SIGNED(7 downto 0);
          VARIABLE TEMP   : SIGNED(8 downto 0);
     BEGIN
          IF CLOCK'EVENT AND CLOCK = '1' THEN
               IF i_VALID = '1' THEN
                    DATA_1 := SIGNED( INPUT  );
                    DATA_2 := SIGNED( MEMORY );
                    TEMP   := RESIZE(DATA_1, 9) + RESIZE(DATA_2, 9);
                    OUTPUT <= STD_LOGIC_VECTOR( TEMP(8 DOWNTO 1) );
                    MEMORY <= INPUT;
                    o_VALID<= '1';
               ELSE
                    o_VALID<= '0';
               END IF;
          END IF;
     END PROCESS;

END Behavioral;
```

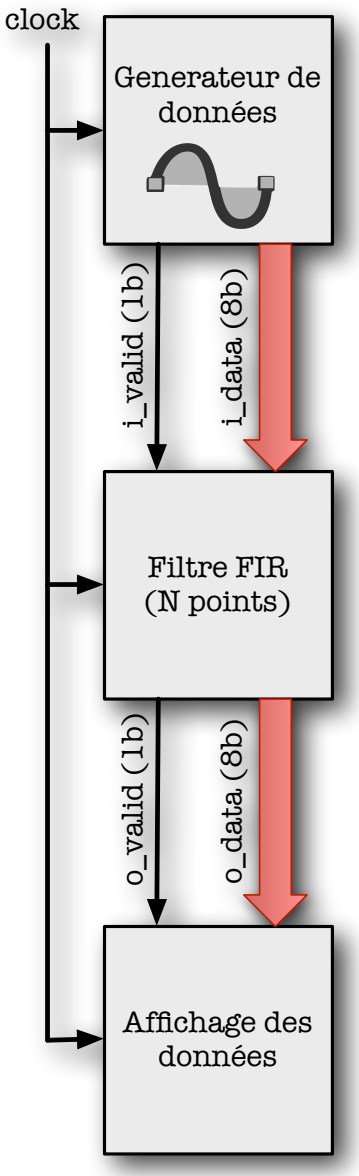
Exemple pédagogique (VHDL) - Fusion des modules




```
architecture Behavioral of Afficheur is
BEGIN

     PROCESS(CLOCK)
          VARIABLE value: INTEGER range -128 to 127;
     BEGIN
          IF CLOCK'EVENT AND CLOCK = '1' THEN
               IF i_VALID = '1' THEN
                    value := TO_INTEGER( SIGNED( INPUT  ) );
                    REPORT "SORTIE = " & INTEGER'IMAGE( value );
               END IF;
          END IF;
     END PROCESS;

END Behavioral;
```

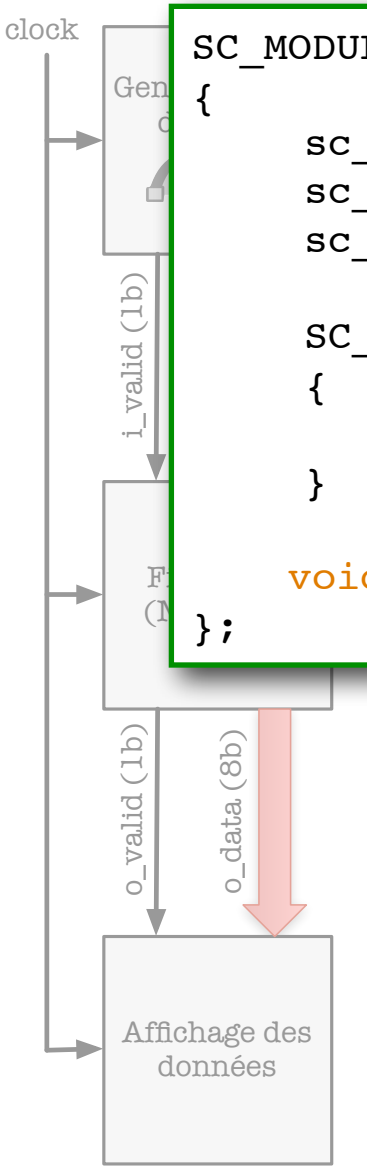
Ecrivez les processus VHDL correspondant aux modules du système.

Exemple pédagogique (SystemC) - Fusion des modules



Ecrivez les processus SystemC correspondant aux modules du système.

Exemple pédagogique (SystemC) - Fusion des modules

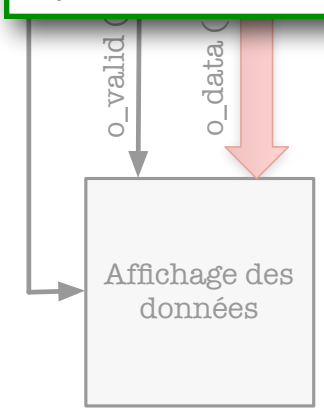



```
SC_MODULE(Gene)
{
     sc_in <bool>       clk;
     sc_out<bool>  o_valid;
     sc_out< sc_lv <8> > s;

     SC_CTOR(Gene)
     {
          SC_CTHREAD(do_gen, clk.pos());
     }

     void do_gen();
};
```

```
void Gene::do_gen()
{
     int value;
     double x = 0;
     while(1){
          value = round(127 * sin( (2 * PI * x) / 30.0) );
          s.write( value );
          o_valid.write( true );
          x += 1.0;
          wait();
          o_valid.write( false );
          wait();
     }
}
```

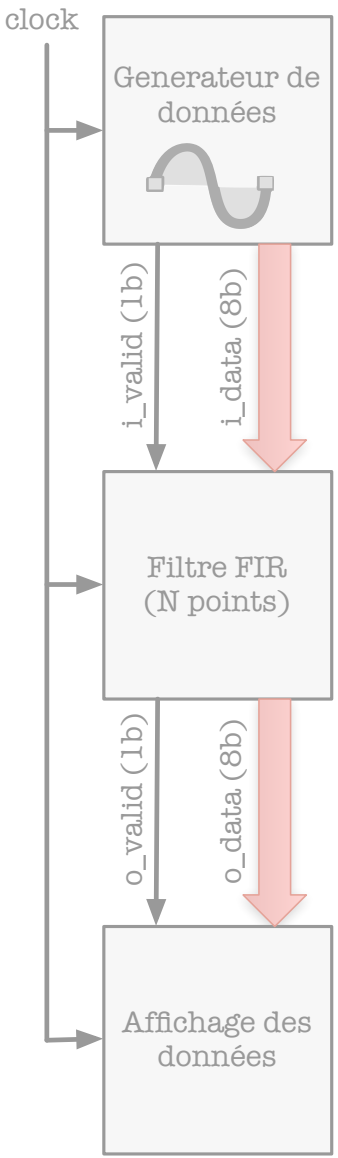
Exemple pédagogique (SystemC) - Fusion des modules

```
SC_MODULE(FIR_2pts)
{
public:
     sc_in <bool>       clk;
     sc_in <bool>  i_valid;
     sc_out<bool>  o_valid;
     sc_in < sc_lv <8> > e;
    sc_out< sc_lv <8> > s;

     SC_CTOR(FIR_2pts)
     {
          SC_CTHREAD(do_fir, clk.pos());
     }

    void do_fir();
 };
```




```
void FIR_2pts::do_fir(){
     s.write( 0 );
     sc_int <8> Xn;
     while( true ){
          wait();
          if( i_valid.read() == true ){
               sc_lv <8> IN = e.read();
               sc_int<8> in = IN;
               sc_int<8> re = (Xn + in)/2;
               sc_lv <8> OU = re;
               Xn           = in;
               s.write( OU );
               o_valid.write( true );
          }else{
               o_valid.write( false );
          }
     }
}
```

Exemple pédagogique (SystemC) - Fusion des modules




```
SC_MODULE(Terminal)
{
     sc_in<bool>       clk;
     sc_in<bool>   i_valid;
     sc_in< sc_lv<8> >   e;

     void do_print();

     SC_CTOR(Terminal)
     {
          SC_CTHREAD(do_print, clk.pos());
     }
};
```

```
#include "Terminal.h"

void Terminal::do_print()
{
     while( true ){
          wait();
          if( i_valid == true ){
               sc_lv <8> value = e.read();
               sc_int<8> conv  = value;
               cout << "Time = " << sc_time_stamp();
               cout << " / out = " << conv <<  endl;
          }
     }
}
```

Partie 1
«Liens de communication»

Définition 4 : Les canaux de communication

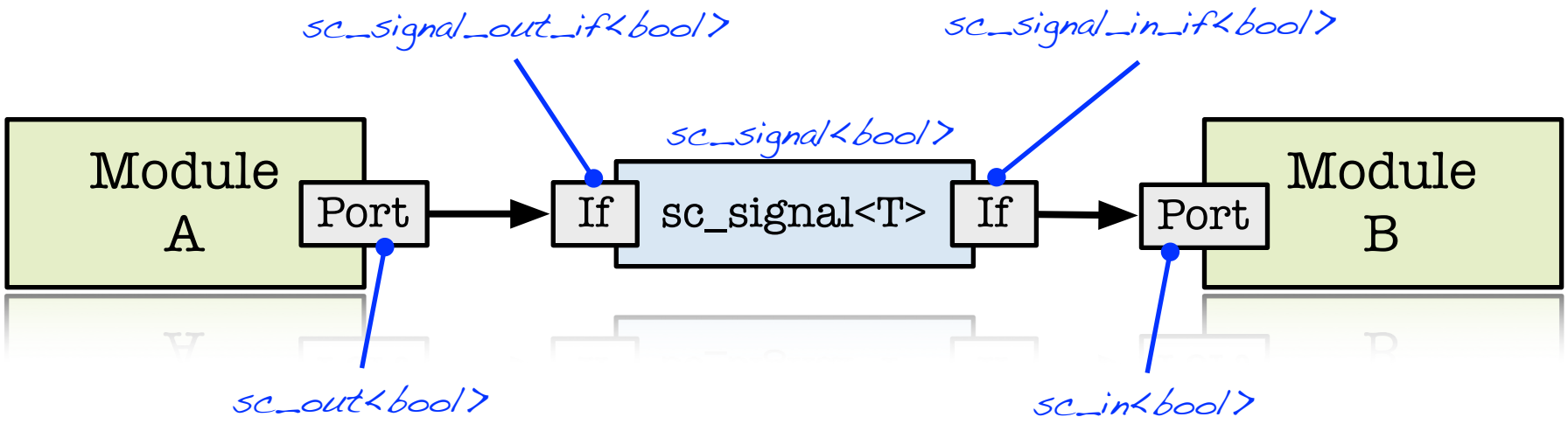
- Les canaux de communication sont les moyens mis à disposition par SystemC afin de permettre le transfert de données entre différents modules composant le système.

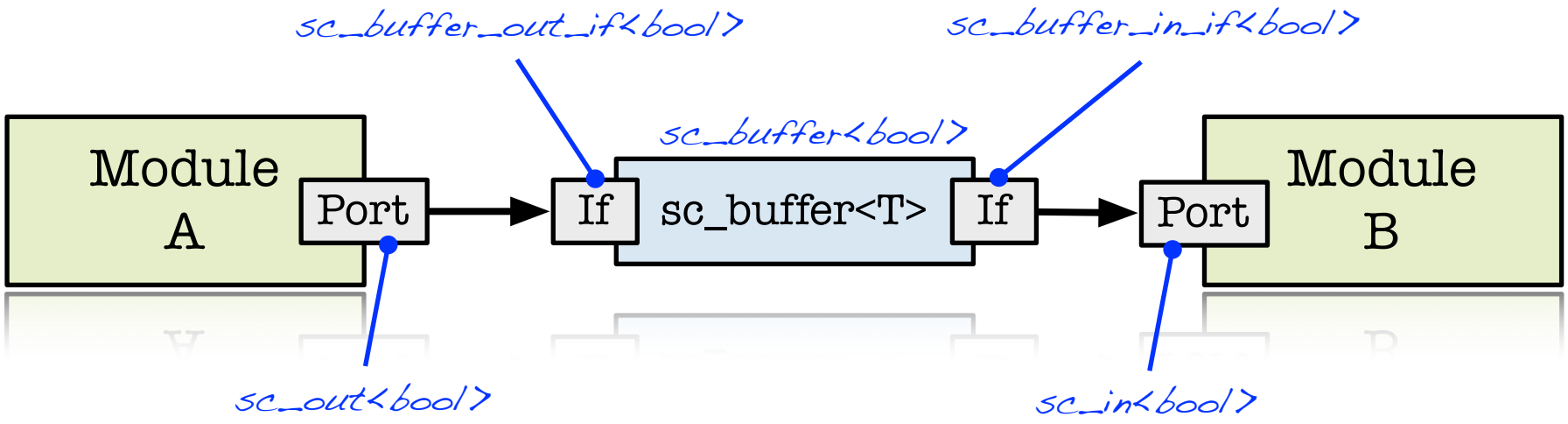
- On distincte de 2 liens de communication de niveau «physique»,
  - sc_signal<T> : les signaux sont les canaux de communication les plus simples car ils modélisent de "simples fils".
  - sc_buffer<T> : ces canaux de communication sont similaires aux sc_signal<T> à une nuance prête: à chaque écriture (write), ils produisent un événement même si les données sont identiques (ce qui n'est pas le cas de sc_signal<T>).

- Les canaux sont connectables uniquement aux ports qui sont compatibles aux interfaces de ces derniers.

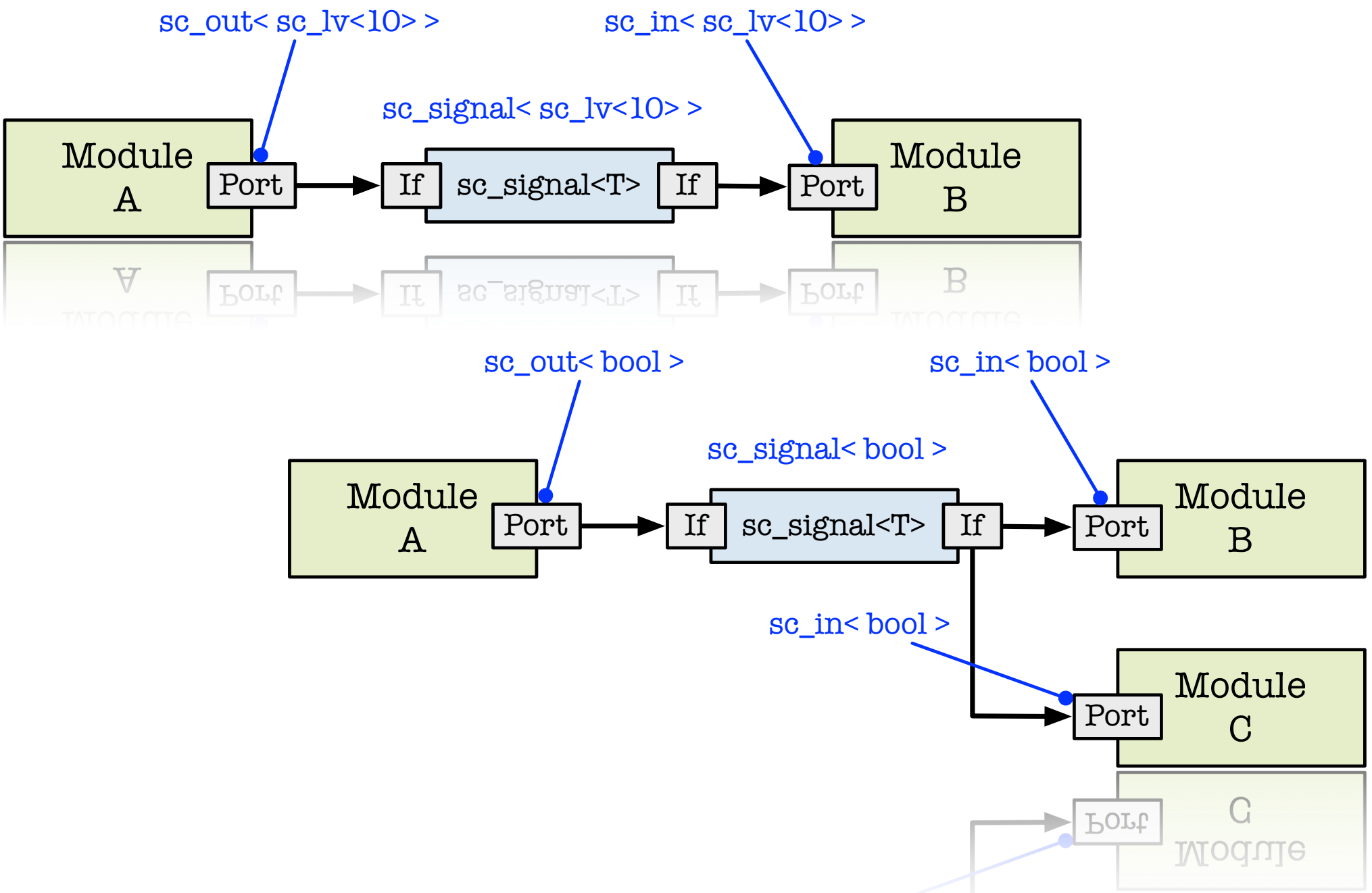
- Exemples
  - Un fil sur 1 bit (par exemple l'horloge de l'exemple ci-dessus) sera modélisé par un signal de type sc_signal<bool>.
  - Un bus de données ou d'adresse sur 32 bits sera un défini comme étant un canal de type sc_signal<sc_uint<32> > ou sc_signal<sc_lv<32> >

Cohérence entre les ports d'E/S et canaux de communication





Exemples de liens entre modules

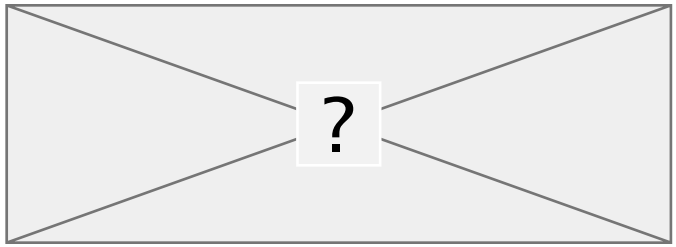


Les canaux sc_signal<T> et sc_buffer<T>

- value_changed_event( ), default_event( )
  - Renvoie une référence à un événement utilisable dans une une instruction wait().
- posedge_event(), negedge_event( )
  - Méthodes valides seulement pour les signaux binaires (bool ou sc_logic).
- const T& read( )
  - Retourne la valeur de type <T> contenue actuellement dans le sc_signal,
- write(const T& val)
  - Si valeur est != de current_value, alors stockage de la valeur programmation d'une mise à jour du signal lors du prochain delta cycle,
- posedge( ), negedge( )
  - Retourne true si le signal a subi une transition vers (X=>Y) au delta cycle précédent.

Modélisation (comportementale) d'une porte XOR

```
#include "systemc.h"

SC_MODULE(Porte_XOR_2)
{
    sc_in<bool> a, b;
    sc_out<bool> s;

    SC_CTOR(Porte_XOR_2)
    {
        SC_METHOD(do_xor);
        sensitive << a << b;
    }

    void do_xor();
};
```



On reste au niveau de l'expression du comportement.

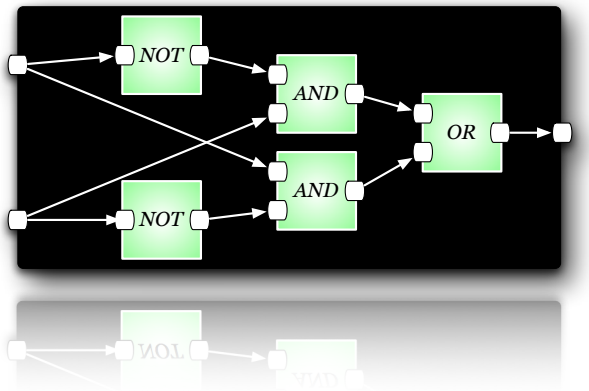
```
#include "Porte_XOR_2.h"

void Porte_XOR_2::do_xor(){
    s = (a && (!b)) || ((!a) && b);
}
```

Modélisation (structurelle) d'une porte XOR

```
SC_MODULE(Porte_XOR)
{
 private:
     Porte_NOT *not_1;
     Porte_NOT *not_2;
     Porte_AND *and_1;
     Porte_AND *and_2;
     Porte_OR  *or_1;
     sc_signal<bool> *signal_1;
     sc_signal<bool> *signal_2;
     sc_signal<bool> *signal_3;
     sc_signal<bool> *signal_4;
 public:

   sc_in<bool> a, b;
   sc_out<bool> s;

     SC_CTOR(Porte_XOR){
          // ..................
          // ..................
     }

     ~Porte_XOR(){
          // ..................
          // ..................
     }

};
```

Création des composants que l'on va interconnecter

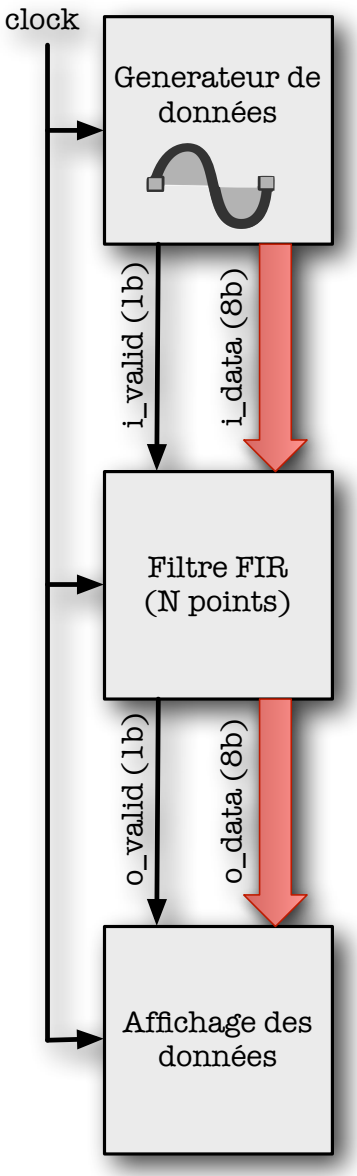
```
not_1 = new Porte_NOT("NOT_1");
not_2 = new Porte_NOT("NOT_2");
and_1 = new Porte_AND("AND_1");
and_2 = new Porte_AND("AND_2");
or_1  = new Porte_OR ("OR_1");
signal_1 = new sc_signal<bool>();
signal_2 = new sc_signal<bool>();
signal_3 = new sc_signal<bool>();
signal_4 = new sc_signal<bool>();
not_1->a( a );
not_1->s( *signal_1 );
not_2->a( b );
not_2->s( *signal_2 );
and_1->a( a );
and_1->b( *signal_2 );
and_1->s( *signal_3 );
and_2->a( *signal_1 );
and_2->b( b );
and_2->s( *signal_4 );
or_1->a( *signal_3 );
or_1->b( *signal_4 );
or_1->s( s );
```



Destruction des composants

```
delete signal_1;
delete signal_2;
delete signal_3;
delete signal_4;
delete not_1;
delete not_2;
delete and_1;
delete and_2;
delete or_1;
```

Exemple: description du système en langage VHDL



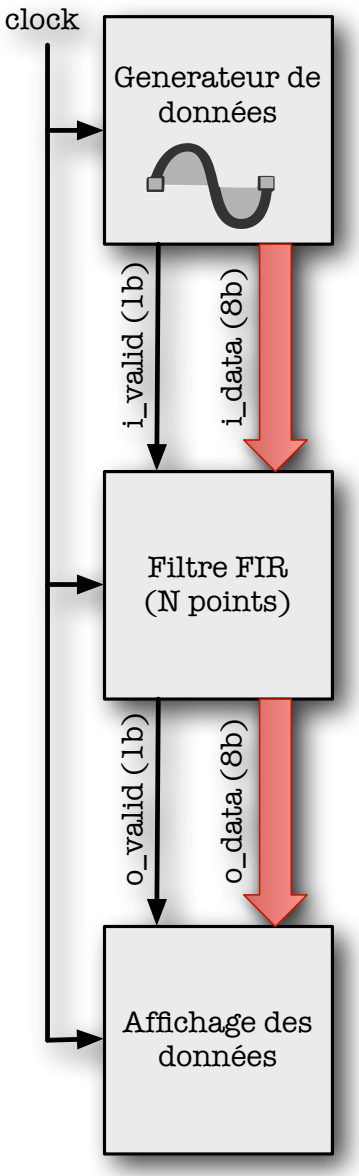
Ecrivez les processus correspondant au comportement des entités VHDL

Exemple: description du système en langage VHDL

```
ENTITY tb_system IS
END tb_system;

ARCHITECTURE behavior OF tb_system IS
   signal CLOCK    : std_logic := '0';
   signal RESET    : std_logic := '0';
   signal SIGNAL_1 : std_logic_vector(7 downto
   signal SIGNAL_2 : std_logic_vector(7 downto
   signal dVALID_1 : std_logic;
   signal dVALID_2 : std_logic;
   constant CLOCK_period : time := 10 ns;

BEGIN

   -- Clock process definitions
   CLOCK_process :process
   begin
         CLOCK <= '0';
         wait for CLOCK_period/2;
         CLOCK <= '1';
         wait for CLOCK_period/2;
   end process;

END;
```

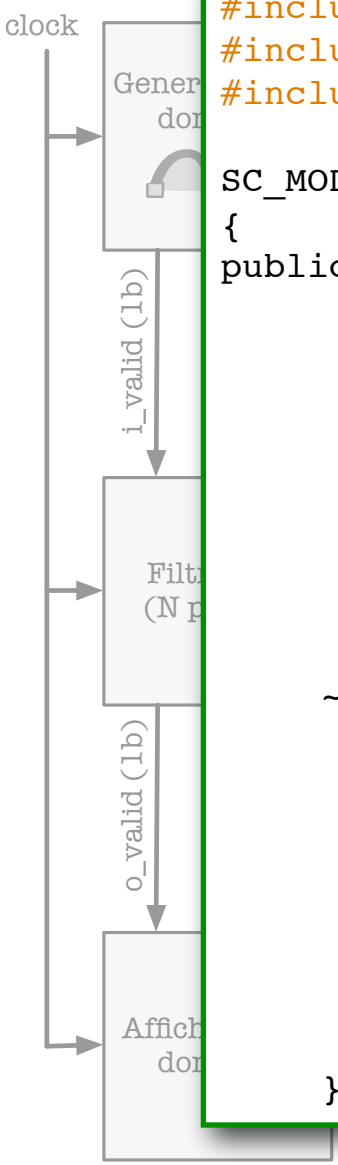
```
uu1: ENTITY work.Generator PORT MAP (
        CLOCK   => CLOCK,
        RESET   => RESET,
        o_VALID => dVALID_1,
        OUTPUT  => SIGNAL_1
     );

uu2: ENTITY work.Filtre PORT MAP (
        CLOCK   => CLOCK,
        RESET   => RESET,
        INPUT   => SIGNAL_1,
        i_VALID => dVALID_1,
        OUTPUT  => SIGNAL_2,
        o_VALID => dVALID_2
     );

uu3: ENTITY work.Afficheur PORT MAP (
        CLOCK   => CLOCK,
        RESET   => RESET,
        i_VALID => dVALID_2,
        INPUT   => SIGNAL_2
     );
```

Exemple: description du système en langage SystemC



Ecrivez les processus correspondant au comportement des modules SystemC

Exemple: description du système en langage SystemC




```
#include "../Gene/Gene.h"
#include "../modules/FIR_2pts.h"
#include "../Analyseur/Terminal.h"

SC_MODULE(top_module)
{
public:
        Gene     *gene;
        FIR_2pts *fir;
        Terminal *term;
        sc_clock                  *clk;
        sc_signal< bool >         *v1;
        sc_signal< bool >         *v2;
        sc_signal< sc_lv<8> >     *s1;
        sc_signal< sc_lv<8> >     *s2;

    ~top_module(){
        delete gene;
        delete fir;
        delete term;
        delete clk;
        delete s1;
        delete s2;
        delete v1;
        delete v2;
    }
```

```
    SC_CTOR(top_module)
    {
        gene = new Gene    ("gene");
        fir  = new FIR_2pts("filtre");
        term = new Terminal("term");
        clk  = new sc_clock("clk", 10, SC_NS);
        v1   = new sc_signal< bool >("v1");
        v2   = new sc_signal< bool >("v2");
        s1   = new sc_signal< sc_lv<8> >("s1");
        s2   = new sc_signal< sc_lv<8> >("s2");
        gene->clk(clk->signal());
        fir->clk (clk->signal());
        term->clk(clk->signal());
        gene->s(*s1); gene->o_valid(*v1);
        fir->e (*s1); fir->i_valid(*v1);
        fir->s (*s2); fir->o_valid(*v2);
        term->e(*s2); term->i_valid(*v2);
    }
};
```

Exemple: description d'un système en langage SystemC

Et maintenant on fait quoi avec tout cela ?

Modèle SystemC

simulation

Partie 1
«System simulation and data tracing»

Gestion des simulations

- Afin de vérifier la fonctionnalité du système, il faut réaliser des simulations fonctionnelles,
- La bibliothèque SystemC inclut:
  - Un simulateur de type événementiel,
  - Des commandes qui permettent d'interagir avec le simulateur (démarrer, régler la durée de simulation et suivre l'état d'avancement, etc.).

```
sc_start();                  // Simulation infinie

sc_start(200,SC_NS);         // Simulation de 200ns

// Affiche le temps courant du simulateur
cout << "Actual time : " << sc_time_stamp() << endl;
```

Exemple programme «main» lançant une simulation

```
int sc_main (int argc, char * argv []){
   Gene  g1("Data_Generator_1");
   Gene  g2("Data_Generator_2");
   Adder add("Adder_1");
   Terminal term("Terminal_1");

   sc_signal<int>  in1;
   sc_signal<int>  in2;
   sc_signal<int>  out;

   g1.s(in1); g2.s(in2);
   add.a(in1); add.b(in2); add.s(out);
   term.a(out);

   sc_start(200,SC_NS);

   return 1;
}
```

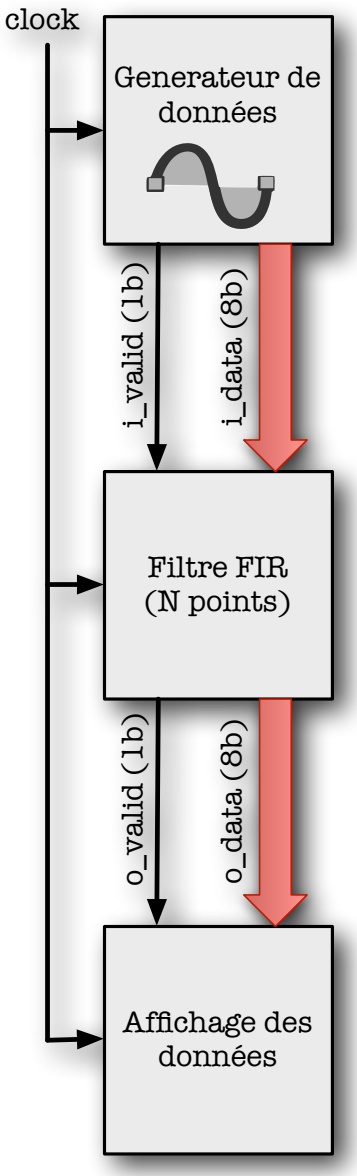
Déclaration des composants qui vont être simulés

Déclaration des canaux de communication

Liaison des ports des modules aux canaux de communication

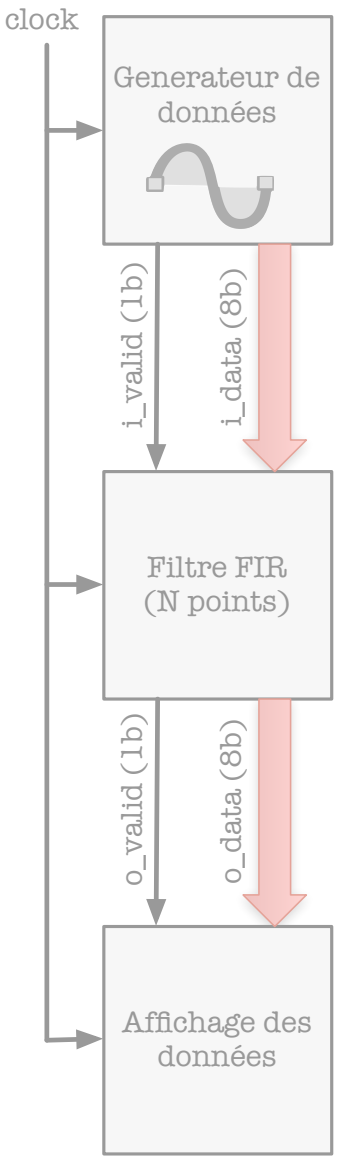
Lancement d'une simulation sur une durée de 200ns

Exemple: écriture du «programme» de simulation (sc_main)



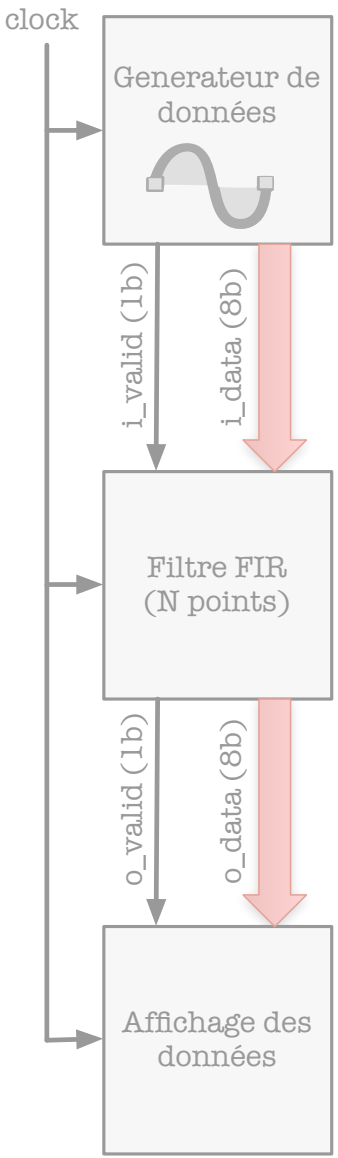
Ecrivez le programme main permettant de simuler notre modèle

Exemple: écriture du «programme» de simulation (sc_main)



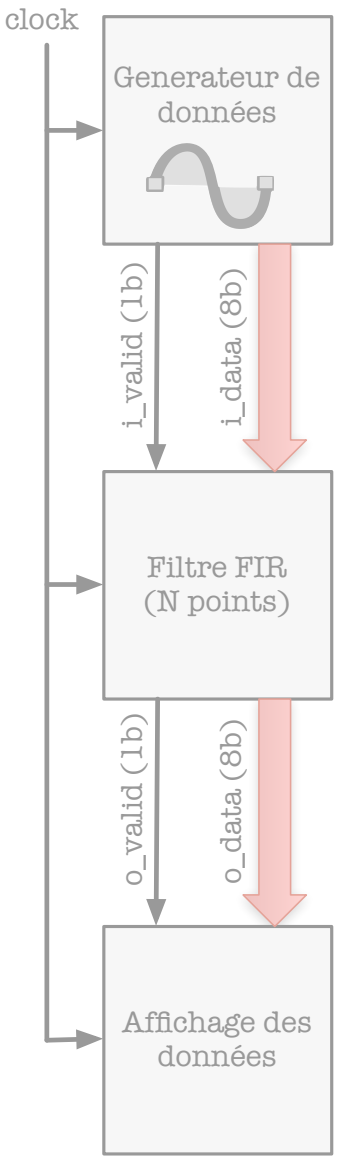

```
#include "./modules/top_module.h"
#include <iostream>

using namespace std;

int main (int argc, char * argv []){
     top_module top("top");

     sc_start(400,SC_NS);

    return 0;
}
```

Résultats observés post-simulation (terminal)




```
./bin/main
WARNING: Default time step is used for VCD tracing.
Time = 20 ns / out = 0
Time = 40 ns / out = 8
Time = 60 ns / out = 23
Time = 80 ns / out = 38
Time = 100 ns / out = 51
Time = 120 ns / out = 62
Time = 140 ns / out = 70
Time = 160 ns / out = 75
Time = 180 ns / out = 77
Time = 200 ns / out = 75
Time = 220 ns / out = 70
Time = 240 ns / out = 62
Time = 260 ns / out = 51
Time = 280 ns / out = 38
```

Résultats observés post-simulation (Xilinx ISim)




```
ISim P.40xd (signature 0x8ef4fb42)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 25 ns(1), Instance /tb_system/uu3/ : Warning: NUMERIC_STD.TO_INTEGER
at 25 ns(1): Note: SORTIE = 0 (/tb_system/uu3/).
at 35 ns(1): Note: SORTIE = 8 (/tb_system/uu3/).
at 45 ns(1): Note: SORTIE = 23 (/tb_system/uu3/).
at 55 ns(1): Note: SORTIE = 38 (/tb_system/uu3/).
at 65 ns(1): Note: SORTIE = 51 (/tb_system/uu3/).
at 75 ns(1): Note: SORTIE = 62 (/tb_system/uu3/).
at 85 ns(1): Note: SORTIE = 70 (/tb_system/uu3/).
```

Observation des signaux internes au modèle

- Il peut être nécessaire de conserver une traces de l'évolution temporelle des signaux dans les composants pour debugger ou conserver une trace du fonctionnement du système,
  - Il existe un ensemble de routines permettant de générer des chronogrammes (fichiers au format VCD),
- Pour générer un chronogramme (vcd), il faut :
  - Créer le fichier qui va contenir les données
    `sc_trace_file * ma_trace = sc_create_vcd_trace_file( "trace.vcd" );`
  - Ajouter les signaux que l'on souhaite pouvoir analyser par la suite,
    `sc_trace(ma_trace, clk,  "signal_clock");`
  - Lancer la simulation du système,
    `sc_start(200,  SC_NS);`
  - Fermer proprement le fichier contenant les signaux avant de quitter,
    `sc_close_vcd_trace_file( ma_trace );`

Exemple de mémorisation des signaux

```
sc_clock clk("TestClock", 10, SC_NS, 0.5);
sc_signal<bool> d1_d2;
sc_signal<bool> d2_d3;
sc_signal<bool> d3_d4;
sc_signal<bool> sortie;

DivFreq *d1 = new DivFreq("Diviseur_1");
DivFreq *d2 = new DivFreq("Diviseur_2");
DivFreq *d3 = new DivFreq("Diviseur_3");
DivFreq *d4 = new DivFreq("Diviseur_4");

// Interconnexion des composants ....

sc_trace_file *trace = sc_create_vcd_trace_file("My_wave_form");
sc_trace(trace, clk,    "Horloge");
sc_trace(trace, d1_d2, "Division_1");
sc_trace(trace, d2_d3, "Division_2");
sc_trace(trace, d3_d4, "Division_3");
sc_trace(trace, sortie,"Division_4");

sc_start(200, SC_NS);

sc_close_vcd_trace_file( trace );

return 0;
```

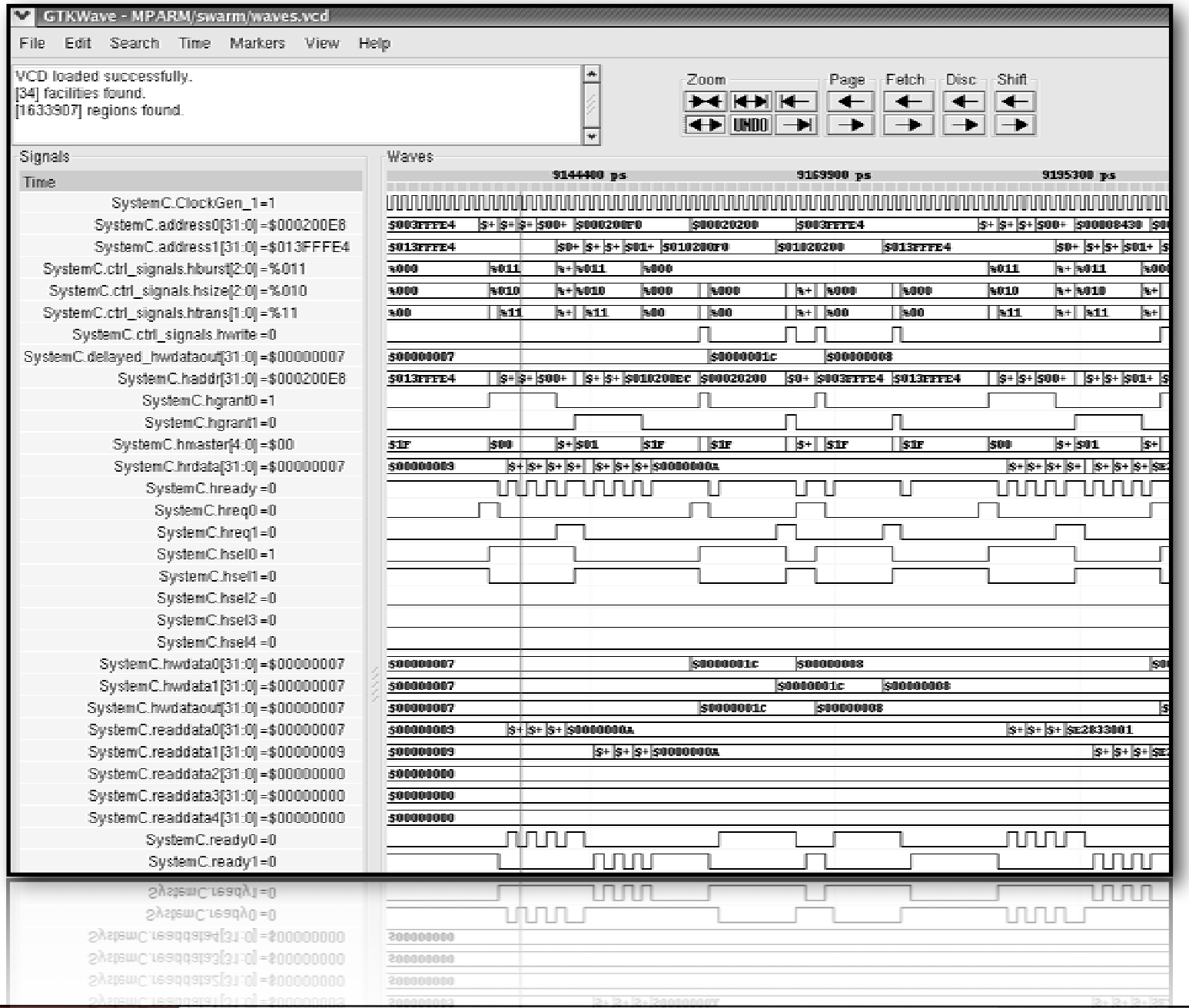
Création du fichier

Enregistrement de signaux a observer

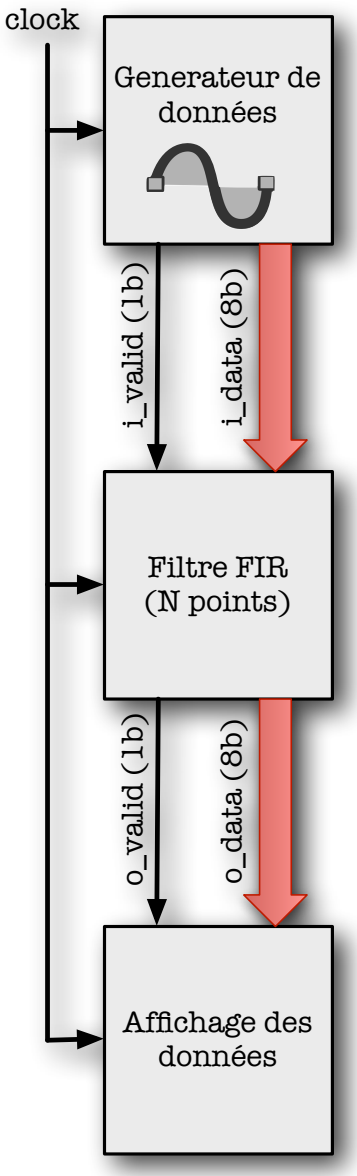
Lancement de la simulation

Fermeture du fichier

Exemple d'affichage des signaux avec Gtkwave

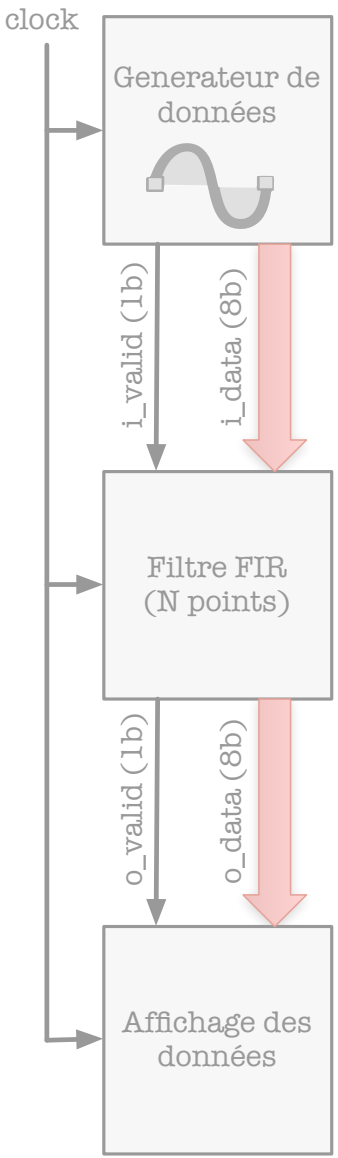


Exemple: écriture du «programme» de simulation (sc_main)



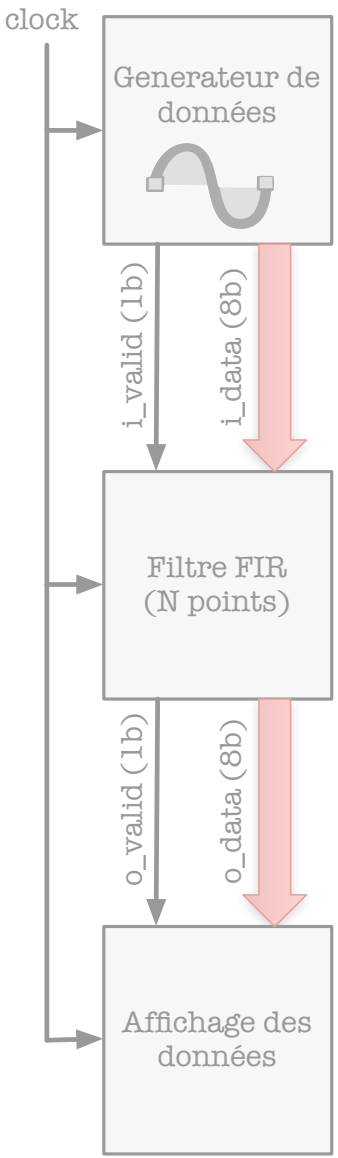
Ecrivez le programme main permettant de simuler notre modèle

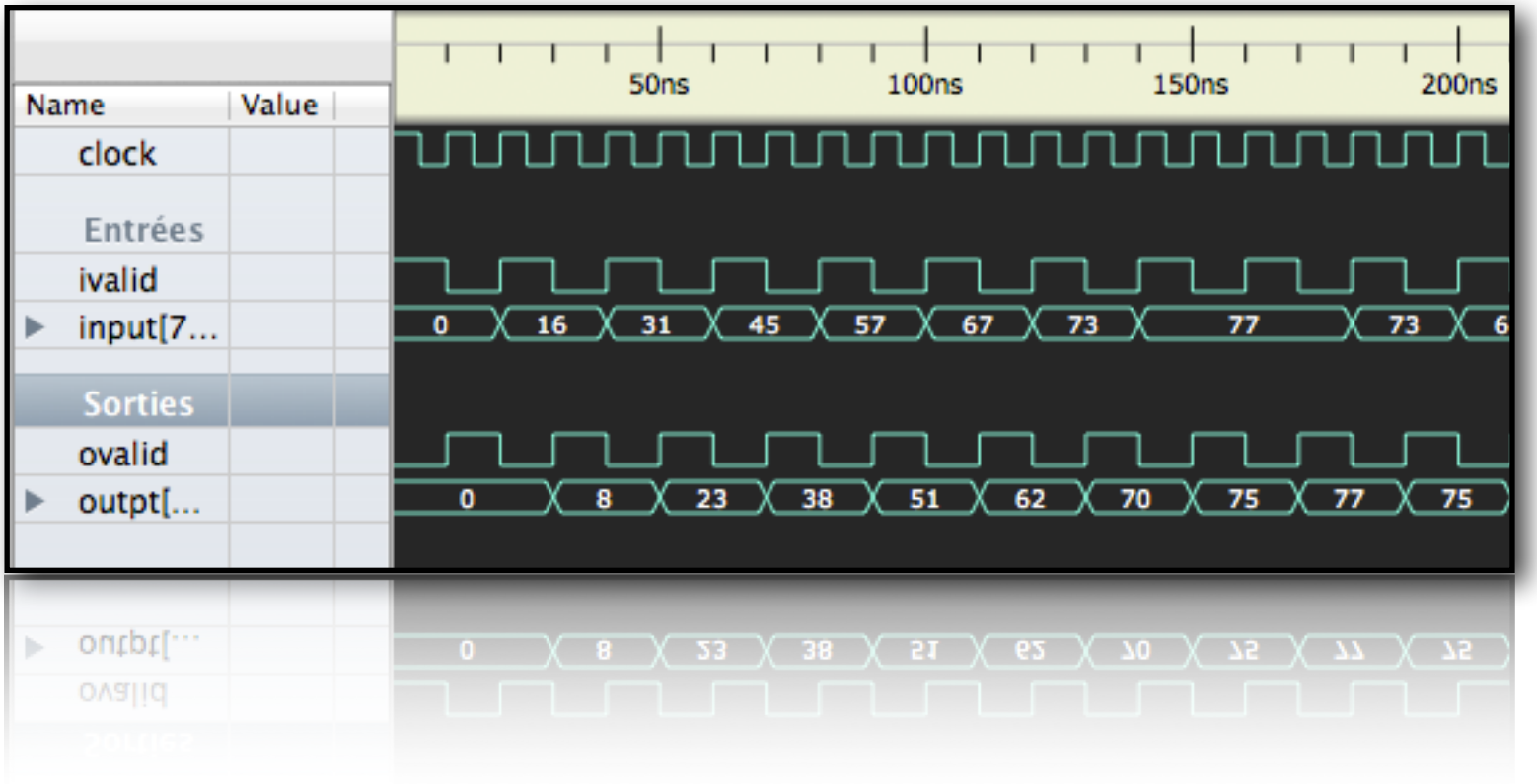
Exemple: écriture du «programme» de simulation (sc_main)



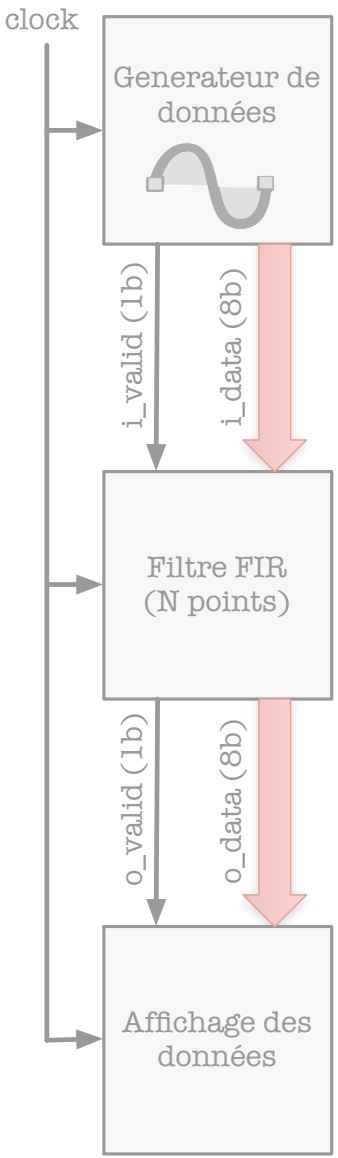

```
#include "./modules/top_module.h"
#include <iostream>

using namespace std;

int main (int argc, char * argv []){
     top_module top("top");

     sc_trace_file *trace = sc_create_vcd_trace_file( "trace" );

     sc_trace(trace, *(top.clk), "clock");
     sc_trace(trace, *(top.s1),  "input");
     sc_trace(trace, *(top.s2),  "outpt");
     sc_trace(trace, *(top.v1),  "ivalid");
     sc_trace(trace, *(top.v2),  "ovalid");

     sc_start(400,SC_NS);

     sc_close_vcd_trace_file( trace );

    return 0;
}
```

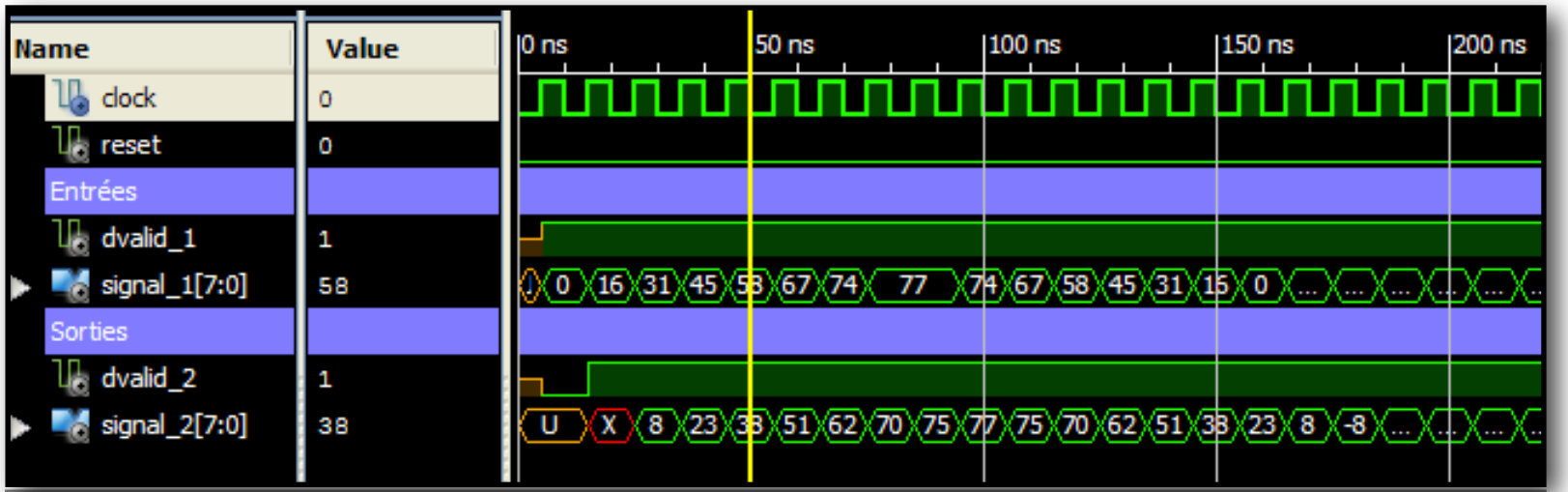
Chronogramme obtenu post-simulation (scansion)





Chronogramme obtenu post-simulation (Xilinx ISim)

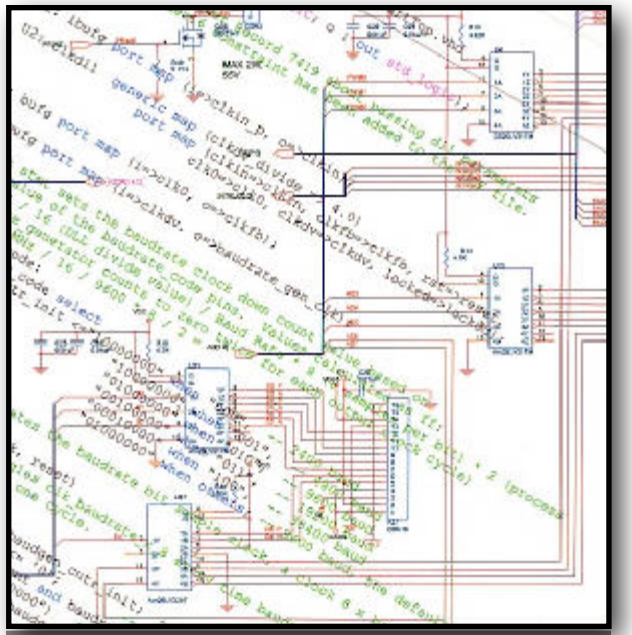




Partie 1
«Conclusion»

Conclusion de la partie introductive...

SYSTEMC™ ≈ VHDL



Modélisation matérielle à l'aide d'un langage basé sur C++

Conclusion de la partie introductive...

SYSTEMC™

Très peu d'outils compatibles

Besoin d'apprentissage d'un nouveau langage

Conclusion de la partie introductive...

Très peu d'outils compatibles

SYSTEMC™

Besoin d'apprentissage d'un nouveau langage

Quels sont les intérêts du langage SystemC ?

Partie 2
«Higher is better»

Partie 2
«Let's start at the beginning»

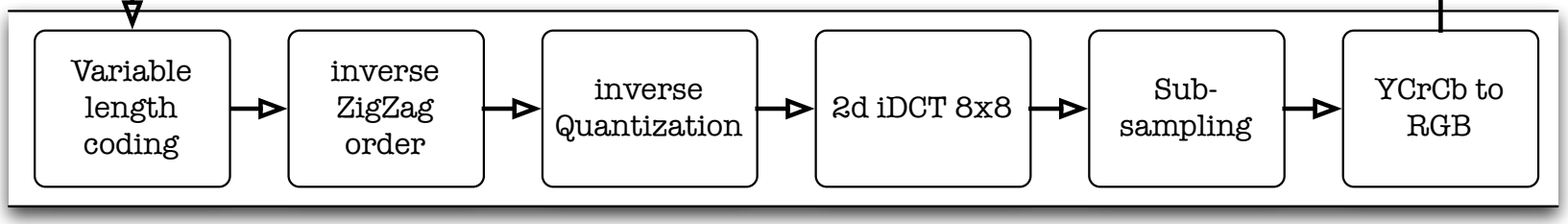
Introduction à la modélisation de «haut-niveau»

La norme de compression JPEG contient plusieurs centaines de pages !

Etape 1, s'assurer que l'on a bien lu... et compris les algorithmes mis en oeuvre...

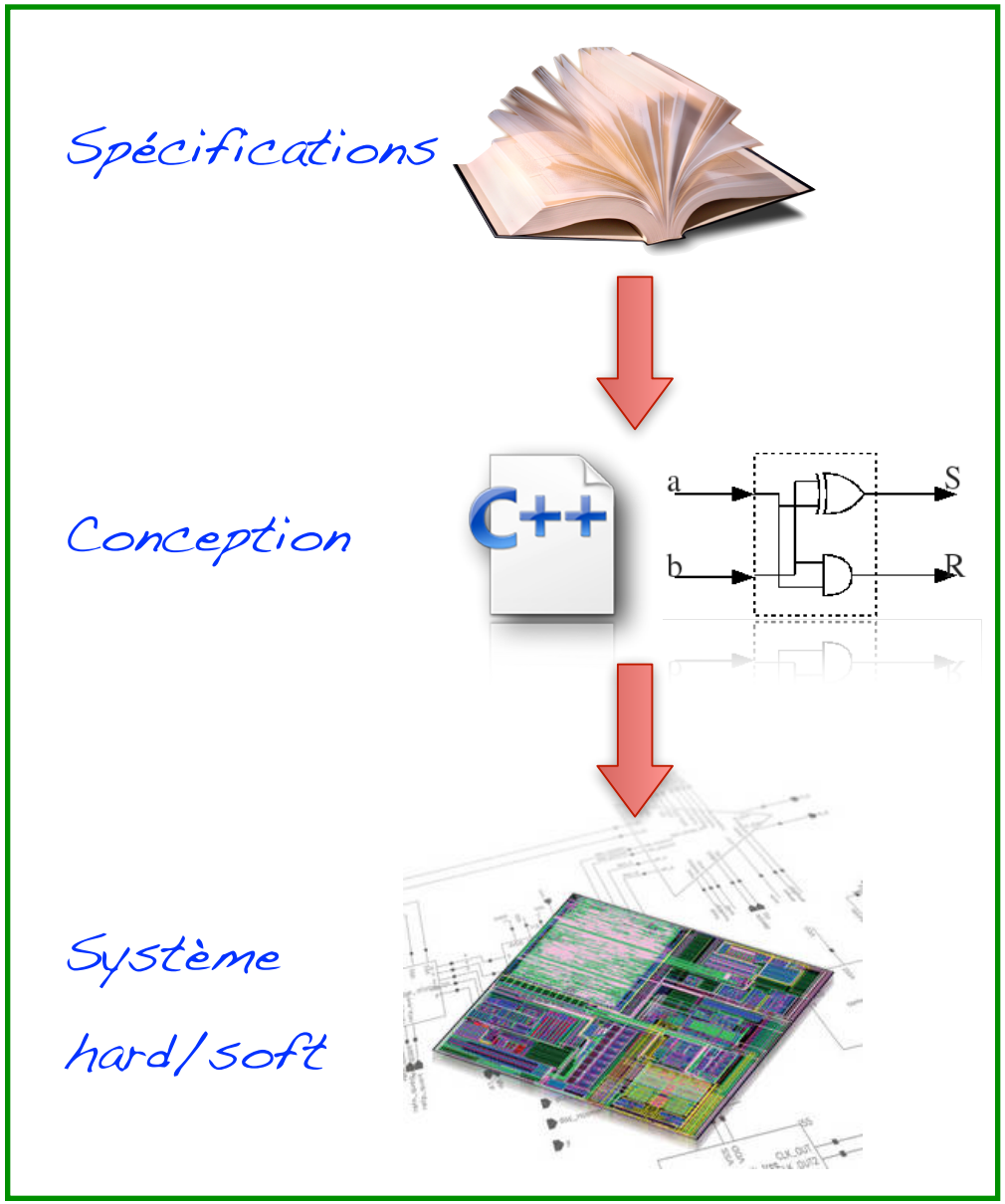
Dans le cas contraire, commencer l'implantation serait inutile !

Avant d'implanter une application il est nécessaire de la comprendre !



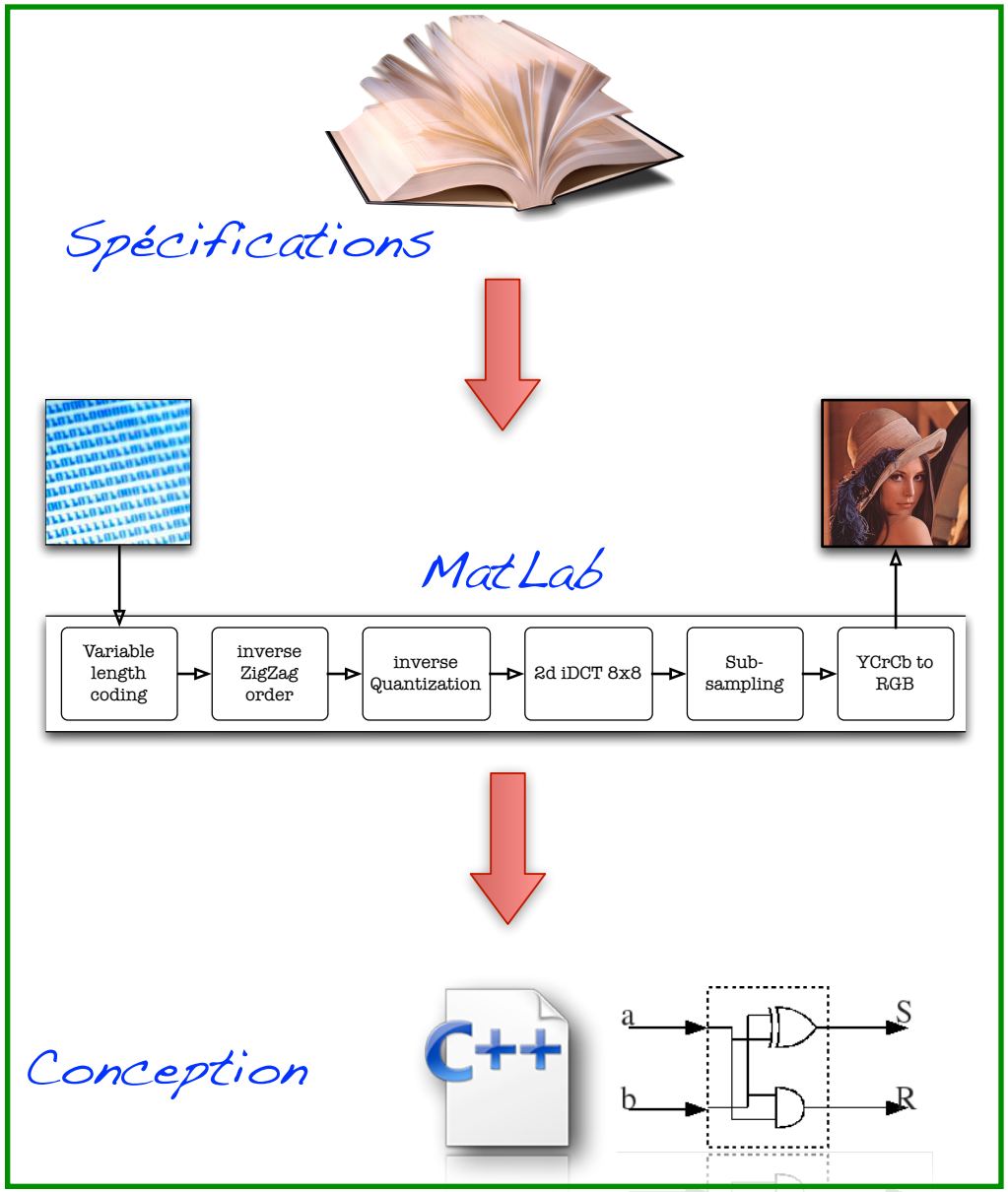
Problème des méthodologies «classiques»

- L'approche la plus «rapide» consiste à implanter directement le système à partir des spécifications,
- Très complexe à mettre en oeuvre (conception, debug),
- Pas d'étude possible des choix d'implantation,
- Approche inutilisable pour les systèmes réels.



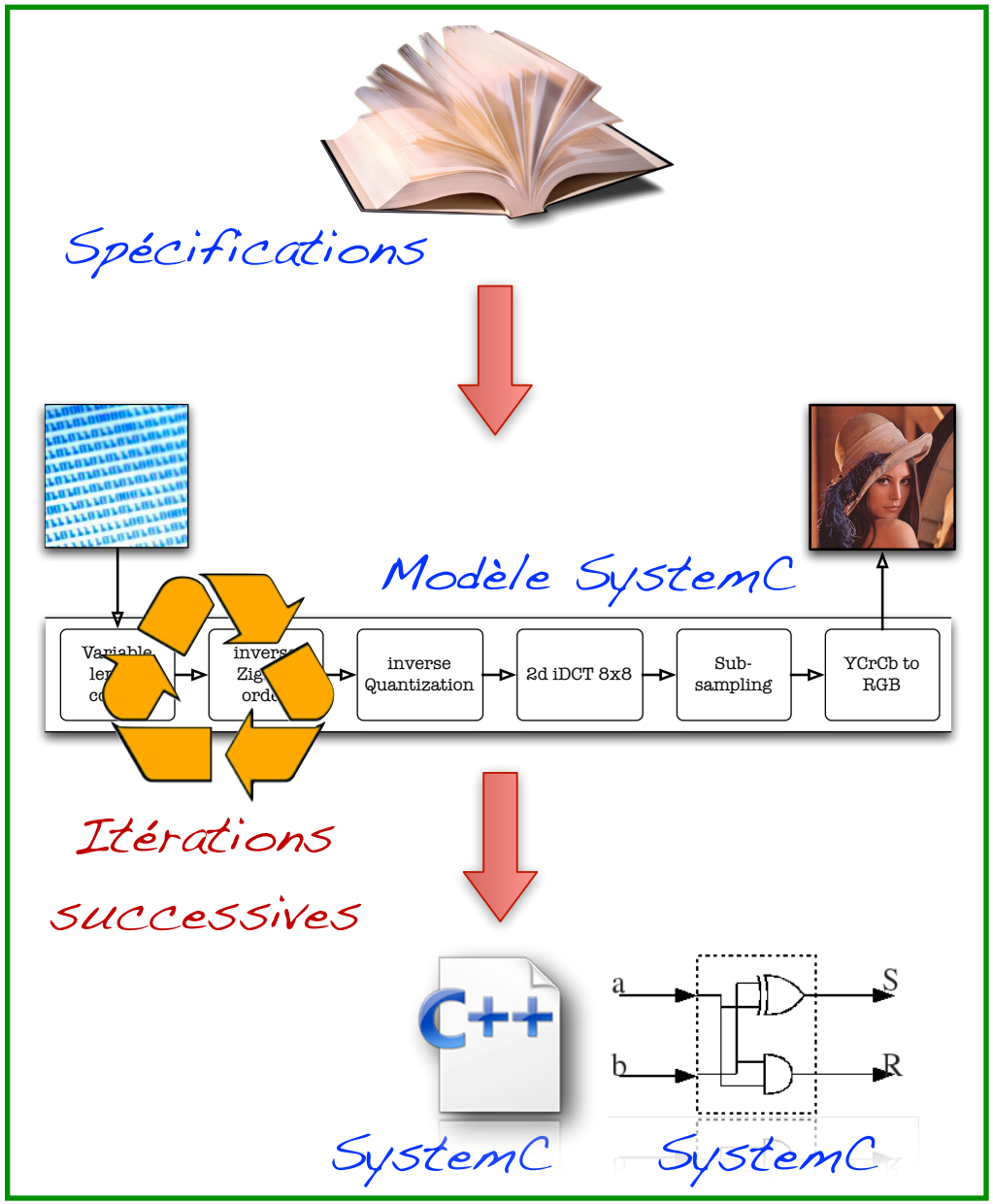
Problème des méthodologies «classiques»

- L'approche la plus courante consiste à modéliser le système à haut-niveau,
  - ➡ Validation des algorithme,
  - ➡ Modèle de référence pour la suite,
- Gap entre le modèle et l'implantation,
  - ➡ Changement de langage (VHDL),
  - ➡ Pas de notion de temps (MatLab)
  - ➡ Format des données (double => std_logic)
- Pas de possibilité de réaliser des simulations conjointes,
  - ➡ Vérifications incrémentales.

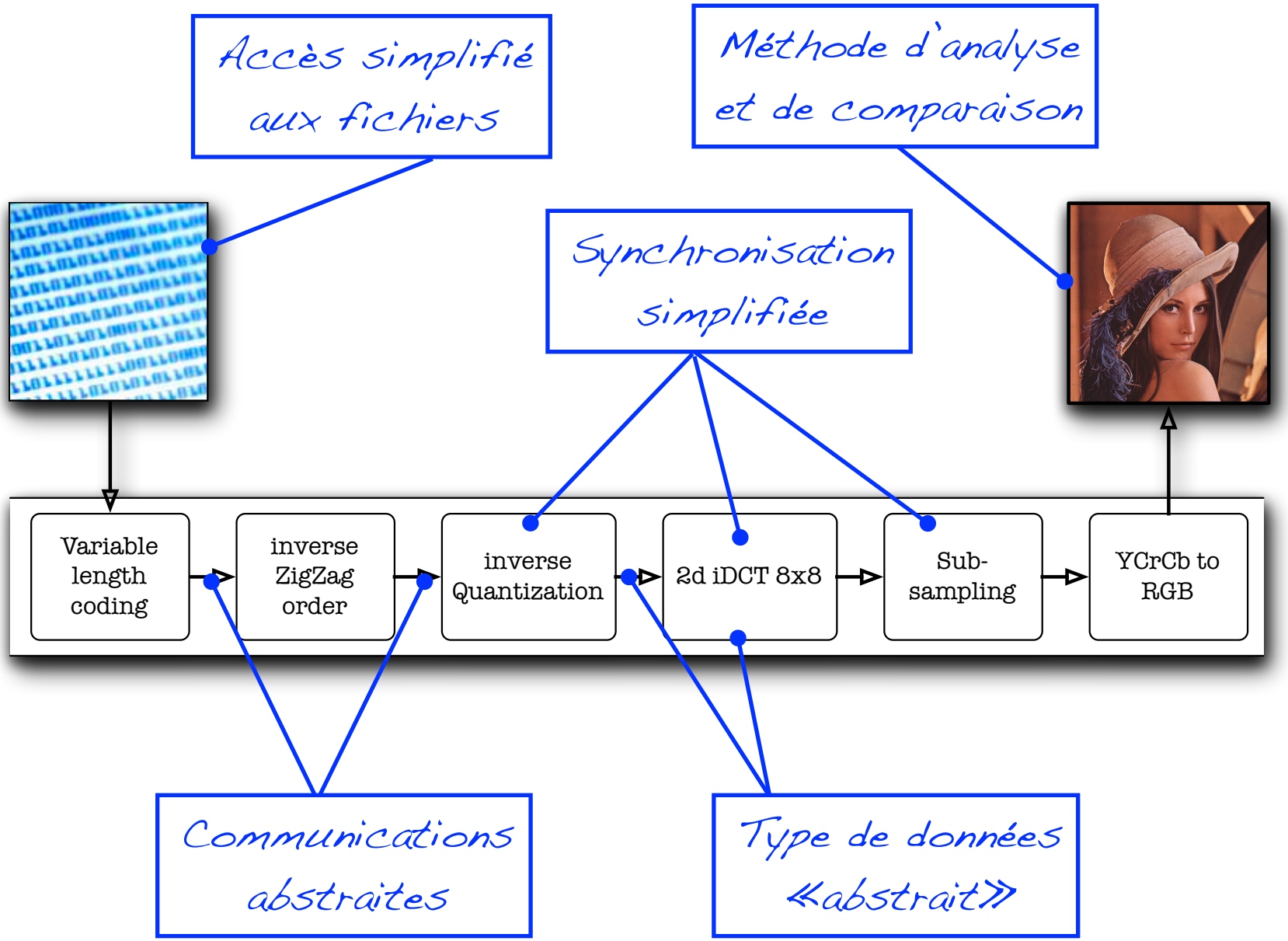


Problème des méthodologies «classiques»

- L'approche idéale consiste à n'avoir qu'un seul langage pour la modélisation et l'implantation,
  - Validation des algorithme,
  - Modèle de référence pour la suite,
  - Raffinement progressif,
- Absence de gap dans le flot
  - Simulation de modèles abstraits (Matlab like) avec des modèles de niveau RTL.
- Simulation conjointes des parties dédiées à une implantation logicielles & matérielle.



Quels sont les besoins issus d'une modélisation haut niveau ?



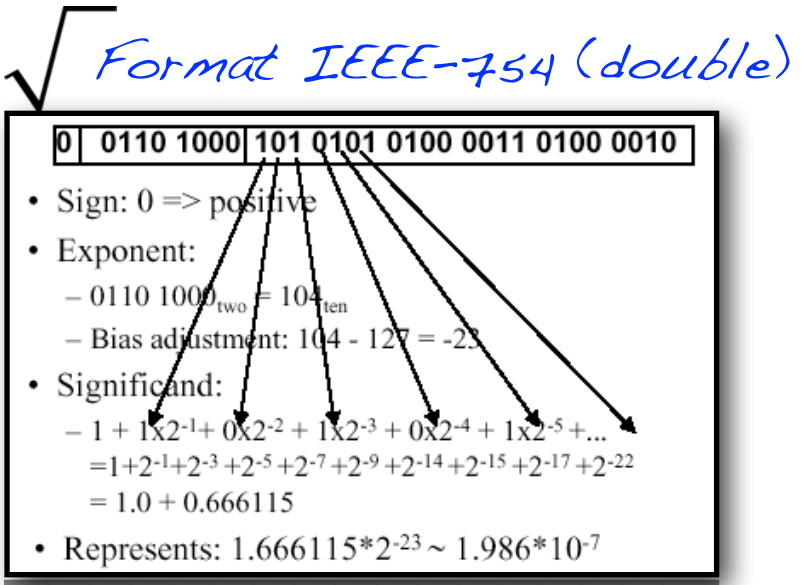
Partie 2
«les types de données abstraits»

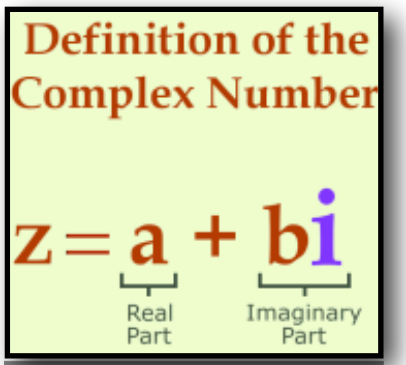
Gestion des types de données natifs

- Le langage SystemC est un sur-ensemble du langage C++, il possède donc l'ensemble des types de données natifs de ce langage,
- Le concepteur peut donc définir ses variables avec les types suivants :
  - char, unsigned char, variables codées sur 8 bits,
  - short, unsigned short, variables codées sur 16 bits,
  - int, unsigned int, variables codées sur 32 bits,
  - long, unsigned long, variables codées sur 32 bits,
  - __int64, variables codées sur 64 bits,
  - float, variables de type flottant codées sur 32 bits,
  - double, variables de type flottant codées sur 64 bits,
- Ces types de données sont directement exécutées par le processeur rendant l'exécution des simulations rapide.

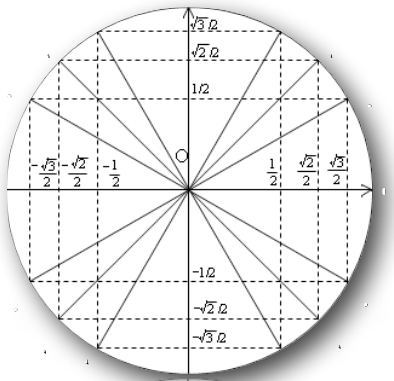
Accès aux bibliothèques mathématiques...

- Toutes les bibliothèques C/C++ sont accessibles dans les modèles.
- Exemple «math.h»
  - *Fonctions trigonométriques* (sin, cos, ...),
  - *Fonctions d'arrondi* (nombres flottants): ceil, floor, round,
  - *Fonctions diverses*: logarithmiques, exponentielles, racine carré, etc.
- Exemple «complex.h» (C++)
  - Modélisation et calculs sur des nombres complexes,
  - Modélisation: real(), imag(), arg(), norm(), polar()
  - Calculs: abs(), conj(), norm(), {+, -, *, /}, etc.





trigo



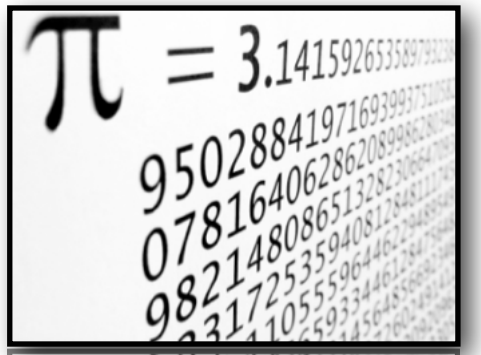
Exemples d'utilisation des bibliothèques mathématiques

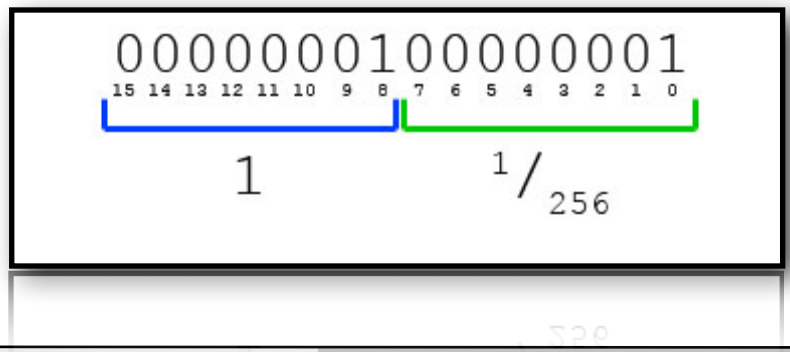
```
inline void ConversionCouleurs(int rvb[3], int ycbcr[3]){
  double y  =         0.299   * (double)rvb[0] + 0.587   * (double)rvb[1] + 0.114   * (double)rvb[2];
  double cb = 128.0 - 0.16874 * (double)rvb[0] - 0.33126 * (double)rvb[1] + 0.5     * (double)rvb[2];
  double cr = 128.0 + 0.5     * (double)rvb[0] - 0.41869 * (double)rvb[1] - 0.08131 * (double)rvb[2];
  ycbcr[0]  = (int)round(y);
  ycbcr[1]  = (int)round(cb);
  ycbcr[2]  = (int)round(cr);
}
```

```
#include "Gene.h"

#define PI (3.141592653589793)

void Gene::do_gen()
{
     int value;
     double x = 0;
     while(1){
          value = round(77 * sin( (2 * PI * x) / 30.0) );
          s.write( value );
          o_valid.write( true );
          x += 1.0;
          wait();
          o_valid.write( false );
          wait();
     }
}
```

Pourquoi utiliser des données codées en Fixed-Point ?

- Les applications du TSI sont développées à haut niveau sans prendre en considération les problématiques implantations matérielles.
- A haut niveau d'abstraction,
  - Utilisation de données et calculs flottants,
  - Pas de problème de précision (précision infinie),
- Implantation matérielle,
  - Calcul flottant = très coûteux (surface),
  - les intégrateurs travaillent sur des en virgule fixe.
- SystemC propose des classes dédiées au calcul en virgule fixe,
  - Quantifier & valider les imprécisions de calcul.





Les classes mises à disposition par SystemC

- Utilisation aisée et flexible des nombres codés en virgule fixe,
  - 4 classes différentes sont accessibles au concepteur.
- sc_fixed<T>, sc_ufixed<T>
  - Modélisent des nombres en virgule fixe possédant des paramètres de quantification et de débordement définis de manière statique (à la compilation du modèle SystemC),
  - La classe sc_fixed modélise les nombres signés et sc_ufixed les nombres non signés,
- sc_fix<T>, sc_ufix<T>
  - Modélisent des nombres en virgule fixe possédant des paramètres peuvant être changés dynamiquement (lors de l'exécution de la simulation),
  - La classe sc_fix modélise les nombres signés et la classe sc_ufix les nombres non signés,
- Afin d'accélérer les temps de simulation, il est possible de préfixer le nom des classes à l'aide de _fast,
  - Uniquement pour les objets qui possèdent une mantisse sur moins de 33 bits.

Paramètres d'utilisation de sc_fixed et sc_ufixed

- Nombre important de paramètres de configuration,
  - sc_fixed<wl, iwl, q_mode, o_mode, n_bits> object_name;
  - sc_ufixed<wl, iwl, q_mode, o_mode, n_bits> object_name;
- Liste des paramètres de configuration,
  - wl: nombre total de bit dans le mot,
  - iwl: nombre de bits alloués à la partie entière,
  - q_mode: mode de quantification
    - Spécifie le comportement lorsqu'une opération génère plus bits après la virgule qu'il n'y en a de place.
    - Valeurs = {SC_RND, SC_RND_ZERO, SC_RND_MIN_INF, SC_RND_INF, etc.}.
  - o_mode: mode de dépassement
    - Spécifie le comportement du mot lors d'une opération qui génère plus de bits avant la virgule qu'il n'y en a de disponibles.
    - Valeurs = { SC_SAT, SC_SAT_ZERO, SC_SAT_SYM, SC_WRAP, SC_WRAP_SM}.
  - n_bits: nombre de bits saturés en cas de dépassement

Exemple d'utilisation des classe de virgule fixe

```
#include <iostream>
#define SC_INCLUDE_FX
#include <systemc.h>

using namespace std;

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

    sc_fixed<8, 4, SC_RND, SC_WRAP> a(3.3333);
    sc_fixed<12, 5, SC_RND, SC_WRAP> b(5.20764);
    sc_fixed<12, 8, SC_RND, SC_WRAP> c;
    sc_fixed<12, 8, SC_RND, SC_WRAP> d;
    c = a + b;
    d = a * b;
    cout << " Valeur de a = " << a << endl;
    cout << " Valeur de b = " << b << endl;
    cout << " Valeur de c = " << c << endl;
    cout << " Valeur de d = " << d << endl;

    return 0;
}
```

Attention, définition obligatoire avant systemc.h

Après exécution

```
Valeur de a = 3.3125
Valeur de b = 5.2109375
Valeur de c = 8.5
Valeur de d = 17.25
```

Exemple d'utilisation des classe de virgule fixe (2)

```
int main (int argc, char * argv []){
     sc_ufixed < 3, 2, SC_RND, SC_SAT> pi_3   = PI;
     sc_ufixed < 4, 2, SC_RND, SC_SAT> pi_4   = PI;
     sc_ufixed < 5, 2, SC_RND, SC_SAT> pi_5   = PI;
     sc_ufixed < 6, 2, SC_RND, SC_SAT> pi_6   = PI;
     sc_ufixed < 7, 2, SC_RND, SC_SAT> pi_7   = PI;
     sc_ufixed < 8, 2, SC_RND, SC_SAT> pi_8   = PI;
     sc_ufixed <16, 2, SC_RND, SC_SAT> pi_16  = PI;
     sc_ufixed <24, 2, SC_RND, SC_SAT> pi_24  = PI;

     cout << "PI(2,1)  = " << pi_3   << endl;
     cout << "PI(2,2)  = " << pi_4   << endl;
     cout << "PI(2,3)  = " << pi_5   << endl;
     cout << "PI(2,4)  = " << pi_6   << endl;
     cout << "PI(2,5)  = " << pi_7   << endl;
     cout << "PI(2,6)  = " << pi_8   << endl;
     cout << "PI(2,14) = " << pi_16  << endl;
     cout << "PI(2,22) = " << pi_24  << endl;

     // ……..................................

}
```

```
g++ -O2 -Wall -I/Library/SystemC/sys
./bin/main
PI(2,1)  = 3
PI(2,2)  = 3.25
PI(2,3)  = 3.125
PI(2,4)  = 3.125
PI(2,5)  = 3.15625
PI(2,6)  = 3.140625
PI(2,14) = 3.1416015625
PI(2,22) = 3.14159274101257324218 75
Line 18, Column 44
```

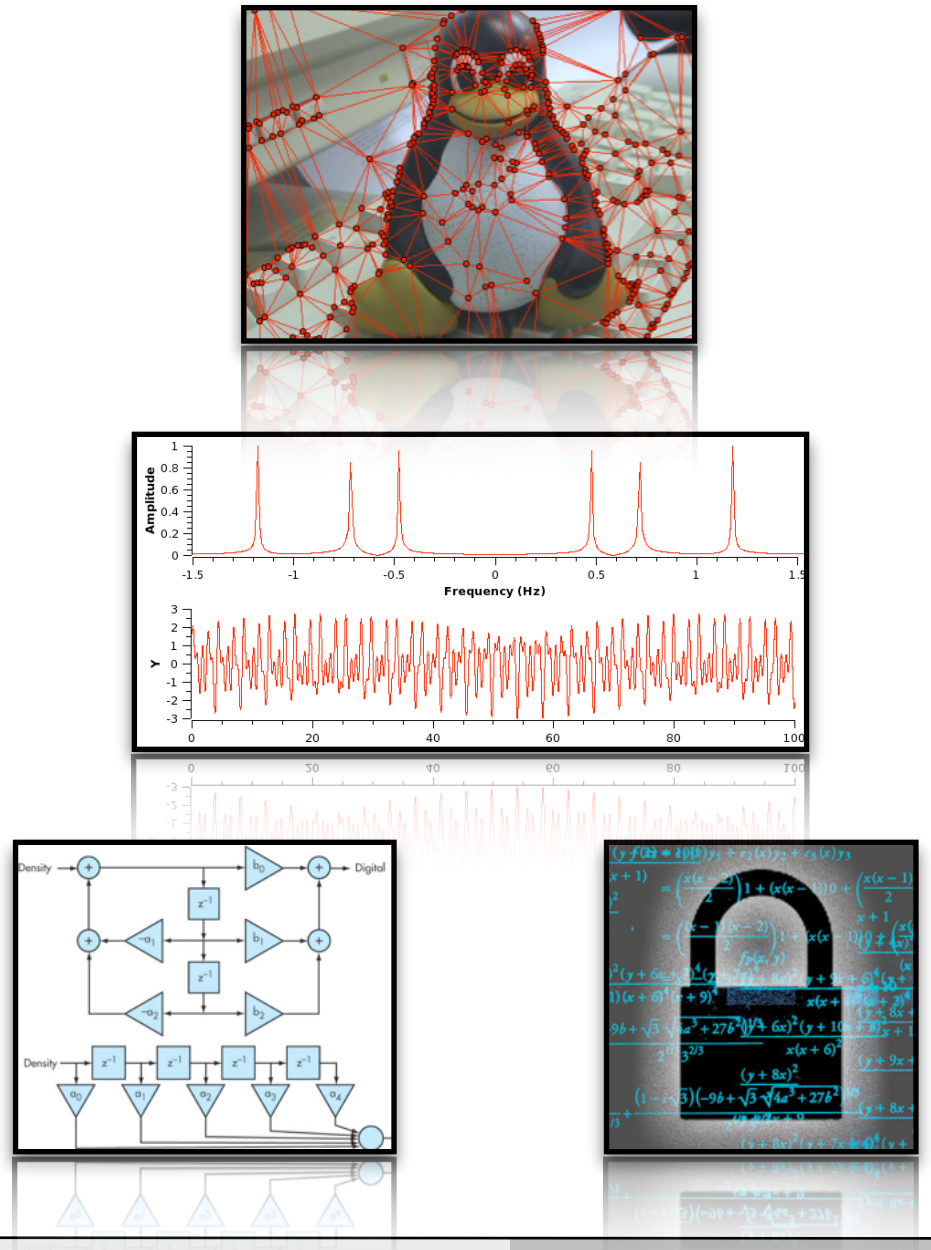
Exemple d'utilisation des classe de virgule fixe (2)

```
int main (int argc, char * argv []){
     sc_ufixed < 3, 2, SC_RND, SC_SAT> pi_3   = PI;
     sc_ufixed < 4, 2, SC_RND, SC_SAT> pi_4   = PI;
     sc_ufixed < 5, 2, SC_RND, SC_SAT> pi_5   = PI;
     sc_ufixed < 6, 2, SC_RND, SC_SAT> pi_6   = PI;
     sc_ufixed < 7, 2, SC_RND, SC_SAT> pi_7   = PI;
     sc_ufixed < 8, 2, SC_RND, SC_SAT> pi_8   = PI;
     sc_ufixed <16, 2, SC_RND, SC_SAT> pi_16  = PI;
     sc_ufixed <24, 2, SC_RND, SC_SAT> pi_24  = PI;

     // ……..................................

     int r = 10;
     cout << "PI(2, 1)*R^2  = " << (int)(pi_3*r*r)  << endl;
     cout << "PI(2, 2)*R^2  = " << (int)(pi_4*r*
     cout << "PI(2, 3)*R^2  = " << (int)(pi_5*r*
     cout << "PI(2, 4)*R^2  = " << (int)(pi_6*r*
     cout << "PI(2, 5)*R^2  = " << (int)(pi_7*r*
     cout << "PI(2, 6)*R^2  = " << (int)(pi_8*r*
     cout << "PI(2,14)*R^2  = " << (int)(pi_16*r
     cout << "PI(2,22)*R^2  = " << (int)(pi_24*r

     return 0;
}
```

```
PI(2,22) = 3.14159274101257324218 75
PI(2, 1)*R^2  = 300
PI(2, 2)*R^2  = 325
PI(2, 3)*R^2  = 312
PI(2, 4)*R^2  = 312
PI(2, 5)*R^2  = 315
PI(2, 6)*R^2  = 314
PI(2,14)*R^2  = 314
PI(2,22)*R^2  = 314
Line 22, Column 45
```

Accès aux bibliothèques...

- Utilisation de bibliothèques plus complexes:
  - *Bibliothèque OpenCV (traitement d'images)*
    - *Filtrage, transformation, analyse, accès aux fichiers, etc...*
  - Bibliothèque FFTW
    - Discrete Fourier transform (DFT) à une ou plusieurs dimensions, taille variable avec données réelles ou complexes.
  - Intel® IPP (Signal Processing)
    - Codes optimisés (FIR IIR, FFT, DCT, DFT, Convolution, Corrélation, etc.)
  - Crypto++
    - Cryptohraphie (AES, DES, RSA, fonctions de hashage, etc.),

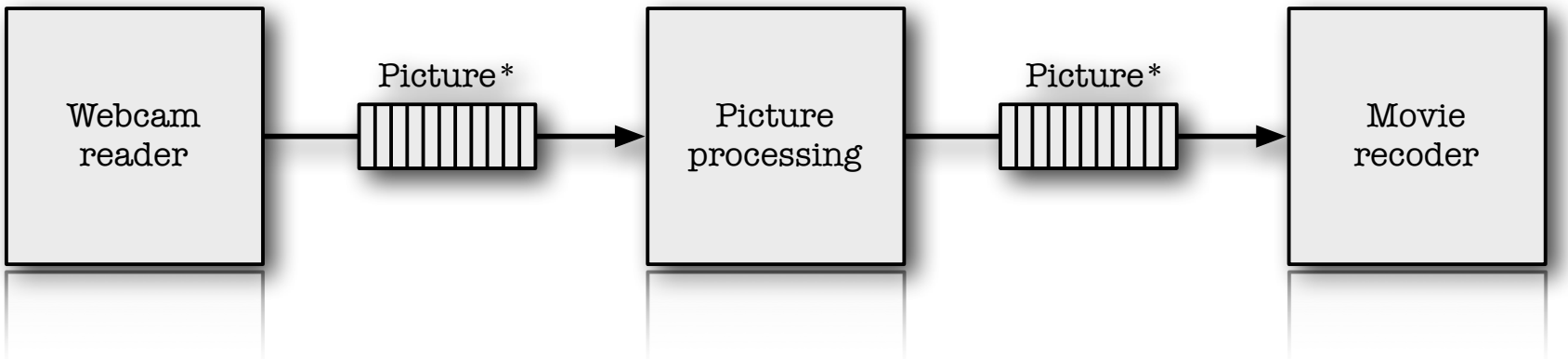


Format de données personnalisés / Code custom

- Possibilité de créer ses propres types de données et/ou classes,
  - ➡ Modélisation simplifiée des échanges de données,
    - Les modules s'échangent des types abstraits de données,
    - Pas de notion d'ordre de transfert,
- Nécessaire à haut-niveau d'abstraction.

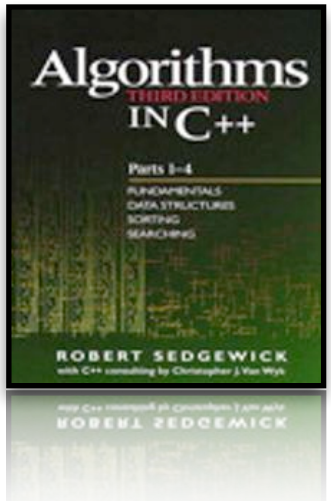
```
class Complex{
public;
      double Re;
      double Im;
};
```

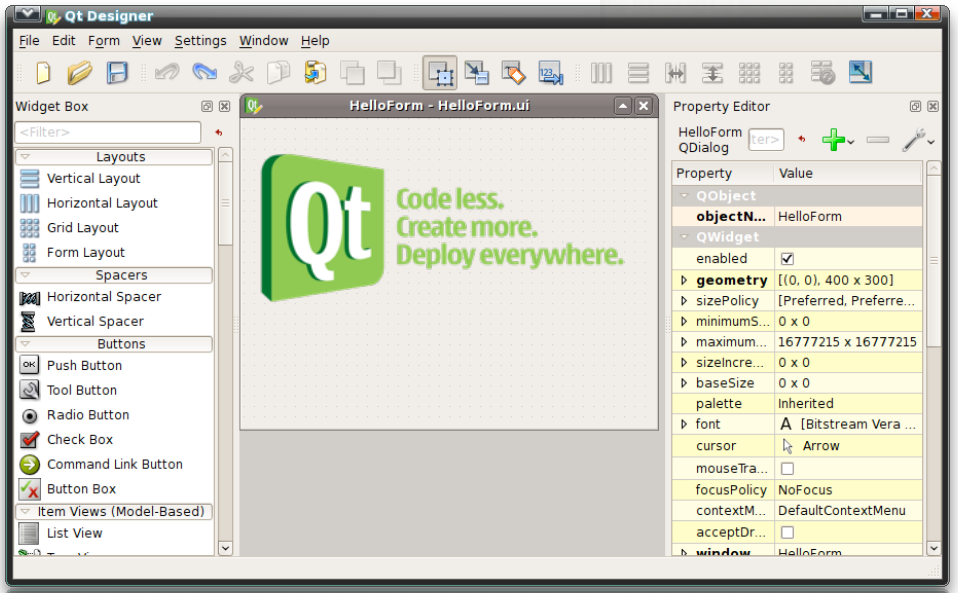
```
class Picture{
public;
      int width;
      int height;
      int *data;
};
```



Et à bien plus encore...

- Accès à toutes les classes et les structures da la STL (Standard Template Library),
  - ➡ Structures de données (vector, queue), des types de données (complex),
  - ➡ Accès au réseau (socket) au fichiers (ifstream),
  - ➡ Algorithmes de tri, de calcul, etc.
- Bibliothèques graphiques,
  - ➡ Interfaces graphiques + interaction avec l'utilisateur (QT par exemple),
- Simplifier les phases de mise au point du modèle fonctionnel.

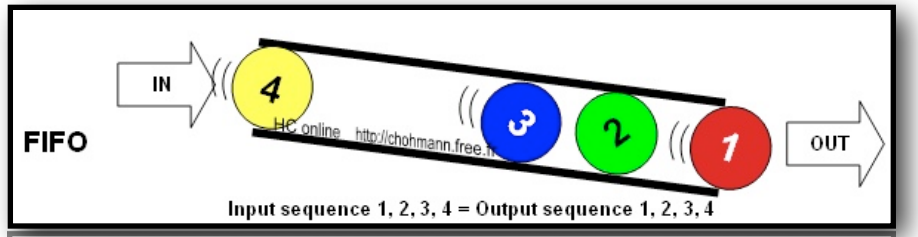


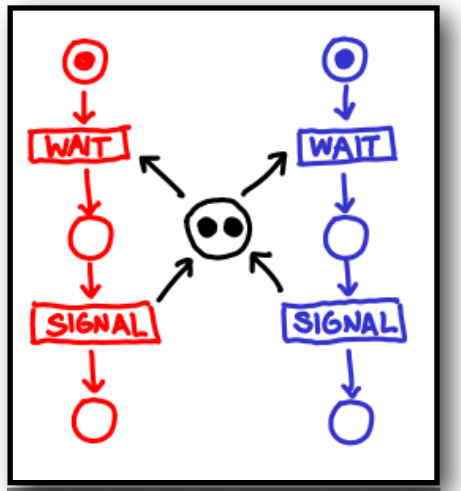


Partie 2
«les liens de communication abstraits»

Les canaux disponibles dans SystemC

- De nombreux types de canaux de communication abstraits sont disponibles,
  - sc_fifo<T> : ces canaux de communication sont abstraits (sans détail physique), ils permettent d'interconnecter les modules à l'aide de mémoires tampon génériques.
  - sc_mutex : les mutex permettent de modéliser l'exclusion mutuelle d'accès aux ressources & la synchronisation de processus,
  - sc_semaphore : les sémaphores sont similaires aux mutex, ils sont utilisés pour la synchronisation et le dénombrement.





Les canaux de type sc_mutex

- Les mutex (sc_mutex) sont des objets utilisés pour gérer très simplement l'exclusion mutuelle entre plusieurs processus.
  - Ils servent à gérer l'accès à des ressources communes, l'exclusion est binaire (un seul processus peut en obtenir l'accès à un instant "t"),
- Utilisation des mutex
  - Un sc_mutex est crée avec un nom unique.
  - Lorqu'un processus veut accéder à une ressource partagée, il doit prendre (lock) le mutex. Une fois qu'il se l'est accaparé, il est le seul à pouvoir le libérer !
  - Lorsque les opérations sont terminées le processus doit libérer (unlock) le Mutex afin de le rendre disponible pour les autres processus.
  - Si un autre processus tente de prendre un mutex alors qu'il est indisponible, alors il est mis en sommeil. Il sera reveillé lorsque le mutex sera libéré par son propriétaire.

- Les méthodes bloquantes ne peuvent être utilisées que dans les processus de type SC_THREAD.

Les canaux de type sc_mutex

- lock( )
  - Cette méthode permet à un processus de demander à récupérer l'accès à la ressource en s'accaparant le mutex. Si ce dernier est disponible, alors il lui l'est immédiatement donné. Dans le cas contraire, le processus est mis en sommeil (wait) le temps nécessaire (voir infini) au retour du mutex déjà pris par un autre processus.
- unlock( )
  - Cette méthode permet à au processus s'étant accaparé le mutex de le rendre à nouveau accessible aux autres demandeurs. La restitution du mutex est immédiate et peu entraîner le réveil d'autres processus en attente.
- bool trylock( )
  - Cette méthode est similaire à lock( ), mais n'est pas bloquante. Elle a été conçue afin de permettre l'utilisation des mutex dans les SC_METHOD et SC_CTHREAD.

- Les mutex, les sémaphores et l'ensemble des ressources gérant de l'exclusion mutuelle sont souvent source de deadlock !

Utilisation des sc_mutex - Exemple

```
classbus {
    sc_mutex bus_access;

    void write(int addr, int data) {
        bus_access.lock();
        // perform write
        bus_access.unlock();
    }
};
```

Accès en écriture à un bus (SC_THREAD)

Accès en écriture à un bus (SC_METHOD)

```
void grab_bus_method() {
    if (bus_access.trylock()) {
        /* access bus */
    }
}
```

```
SC_MODULE(Portique)
{
    sc_mutex access;
    int      compteur;

    SC_CTOR(Portique)
    {
        SC_METHOD(arrivee);
        sensitive << capteur_in;
        SC_METHOD(depart);
        sensitive << capteur_out;
    }

    void arrivee(int addr, int data) {
        access.lock();
        compteur += 1;
        access.unlock();
    }

    void depart(int addr, int data) {
        access.lock();
        compteur -= 1;
        access.unlock();
    }
};
```

Compteur de présence (visiteurs, colis, etc.)

Les canaux de type sc_semaphore

- Les Sémaphores (sc_semaphore) sont des objets utilisés pour gérer l'exclusion mutuelle tels les mutex, mais ils ne sont pas binaires (ils autorisent plus d'un accès simultané à une ressource).
  - ➡ Permettre à plusieurs processus d'accéder à une même ressource en parallèle. Le nombre limite est précisé lors de l'instanciation de l'objet (appel au constructeur).
- Utilisation des sémaphores
  - ➡ Lorqu'un processus veut accéder à une ressource partagée, il doit prendre (wait) un jeton du sémaphore. Une fois qu'il s'est accaparé un des jetons, il est le seul à pouvoir le rendre.
  - ➡ Lorsque les opérations sont terminées le processus doit rendre le jeton au Sémaphore (post) afin de le rendre disponible pour les autres processus.
  - ➡ Si un autre processus tente de prendre le Sémaphore alors que tous les jetons sont pris, alors il est mis en attente. Lorsqu'un des jetons est libéré, le processus est automatiquement réveillé.
- Les méthodes bloquantes ne peuvent être utilisées que dans les processus de type SC_THREAD (liste de sensibilité dynamique).

Les canaux de type sc_semaphore

- int wait( )
  - Cette méthode permet à un processus de demander à récupérer l'accès à la ressource en s'accaparant un jeton du sémaphore. Si ce dernier est disponible, alors il lui est immédiatement donné. La valeur du sémaphore (nombre de jetons disponibles est incrémenté de 1. Si aucun jeton est disponible (valeur 0) alors processus est mis en sommeil (wait) le temps nécessaire au retour d'un jeton.
- int try_wait( )
  - Cette méthode est similaire à lock( ), mais n'est pas bloquante. Elle a été conçue afin de permettre l'utilisation des sémaphores dans les SC_METHOD et SC_CTHREAD.
- int post( )
  - Cette méthode permet à au processus s'étant accaparé un jeton du sémaphore de le rendre à nouveau accessible aux autres demandeurs. La valeur du sémaphore (nombre de jetons disponibles est incrémenté de 1.
- int get_value( )
  - Cette méthode renvoie le nombre de jetons actuellement disponibles dans le sémaphore interrogé.

Utilisation des sc_semaphore - Exemple

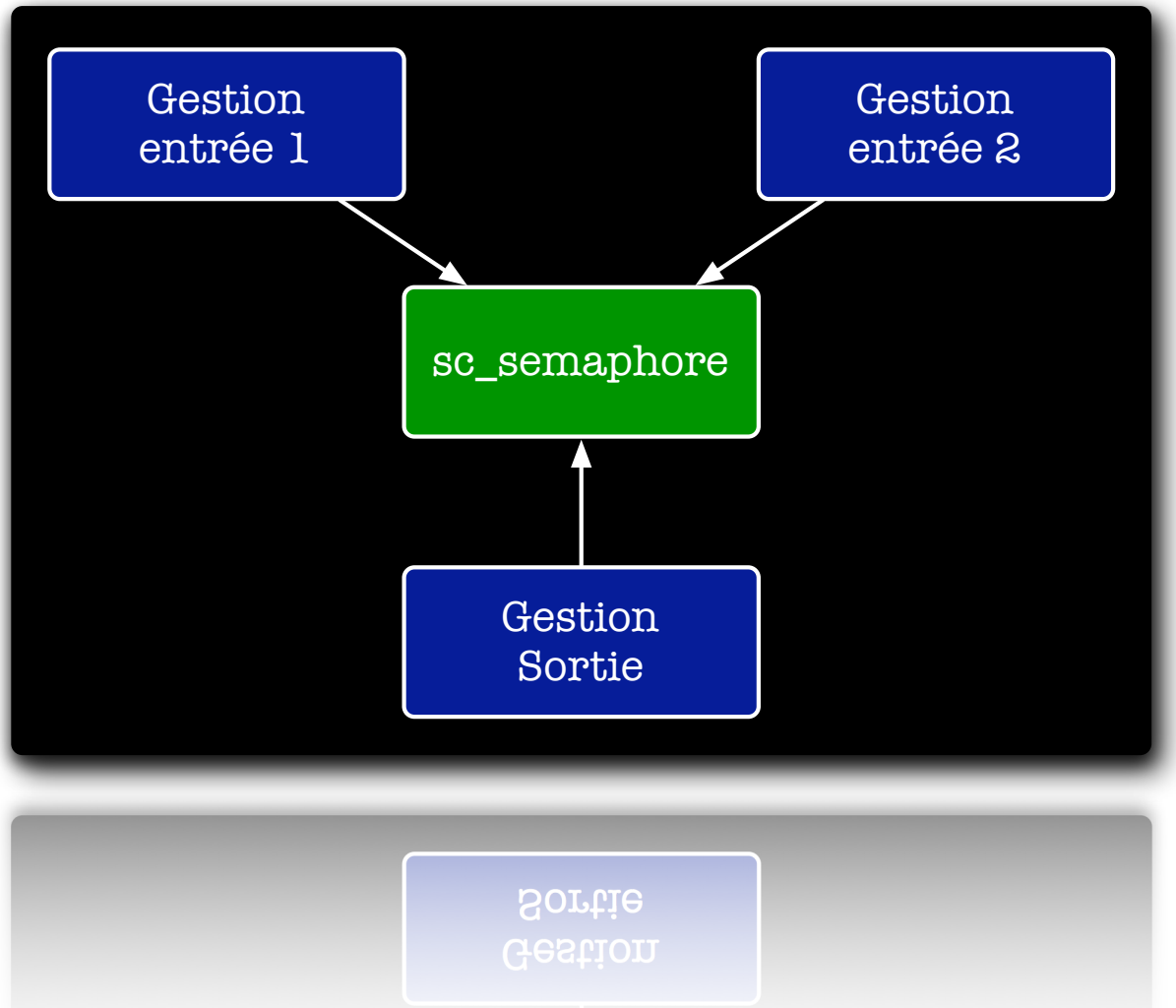
```
SC_MODULE(gas_station) {
    sc_semaphore access(12);

    void HD_Access_thread {
      while( true ) {
        // 12 Accesses max
        // to thehard drive
        access.wait();
        // Read the data ...
        access.post();
      }
    }
};
```

Le processus va demander un des jetons se trouvant dans le sémaphore afin de réguler l'accès au disque dur.

Si 12 accès sont déjà en cours alors le processus est mis en sommeil et sera réveillé lorsque son tour sera venu (gestion d'une queue)

Exemple d'application avec un sc_semaphore

Modélisation d'un circuit permettant de gérer les barrières d'entrée et de sortie d'un parking avec prise en compte du nombre des places disponibles.



Code source réalisant la gestion du parking

```
SC_MODULE(Parking)
{
 private:
   sc_semaphore *places;

 public:
   sc_in<bool>  e1, e2, s1;

     SC_CTOR(Parking)
     {
           places = new sc_semaphore(20);
           SC_THREAD(handle_IN_1);
           sensitive << e1.pos();
           SC_THREAD(handle_IN_2);
           sensitive << e2.pos();
           SC_THREAD(handle_OUT_1);
           sensitive << s1.pos();
     }

     ~Parking();

   void handle_IN_1 ();
   void handle_IN_2 ();
   void handle_OUT_1();
};
```

```
Parking::~Parking(){
      delete places;
}

void Parking::handle_IN_1 (){
      while( true ){
            wait( );          // On attend une voiture
            places->wait();   // reste t'il des places ?
            // On ouvre la barriere + tempo
            // Fermeture de la porte
      }
}

void Parking::handle_IN_2 (){
      while( true ){
            wait( );          // On attend une voiture
            places->wait();   // reste t'il des places ?
            // On ouvre la barriere + tempo
            // Fermeture de la porte
      }
}

void Parking::handle_OUT_1(){
      while( true ){
            wait( );          // On attend une voiture
            // On ouvre la barriere + tempo
            // Fermeture de la porte
            places->post();   // On augmente le nombre
            // des places disponibles dans le parking
      }
}
```

Les canaux de type sc_fifo<T>

- Les FIFOs (sc_fifo) sont utilisés afin d'éviter les problèmes de synchronisation dans les modèles à haut-niveau abstraction.
- La profondeur des fifos est bornée à la création du lien (nombre d'éléments mémorisables),
  - ➡ Une fois cette limite atteinte, les écritures seront bloquantes : le processus sera arrêté jusqu'à ce qu'une donnée soit consommée libérant ainsi un emplacement.
  - ➡ De manière analogue, les lectures dans une sc_fifo sont bloquantes lorsqu'aucune donnée n'est disponible.
- La modélisation des fifos est réalisée à un niveau comportemental avec un tableau circulaire dont le temps d'accès est instantané.
- Les canaux de type sc_fifo sont employées,
  - ➡ Afin d'ignorer les problèmes de synchronisation,
  - ➡ Pour permettre un dimensionnement des interfaces de communication (lors du processus de raffinement).

Les méthodes disponibles: classe sc_fifo<T>

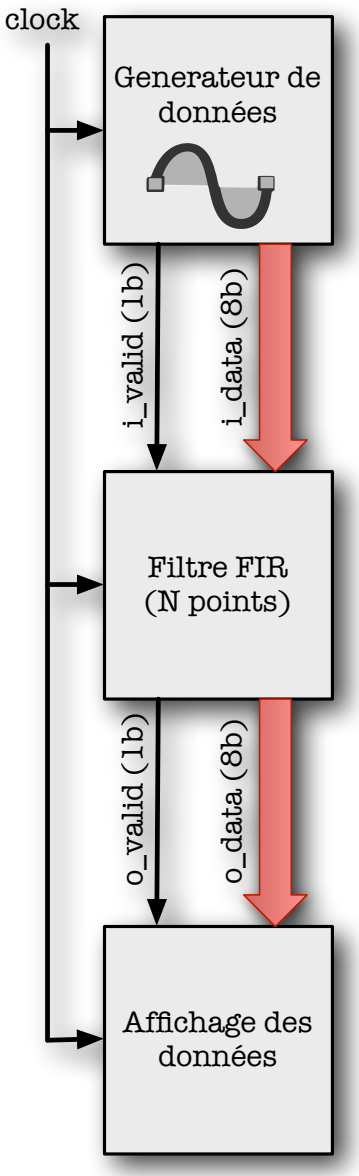
- T read( ), read(T&)
  - Ces méthodes permettent à un processus de demander à lire la plus ancienne valeur contenue dans la fifo. Si aucune donnée n'est disponible dans la fifo alors le processus est mis en sommeil. Ce dernier sera réveillé lorsqu'une donnée aura été écrite.
- bool nb_read(T&)
  - Cette méthode permet de lire une donnée dans la fifo de manière nom bloquante. La méthode retournera true si une donnée a été acquise et *false* sinon.
- int num_available( )
  - Cette méthode permet aux consommateurs de connaître le nombre de données actuellement en attente dans la fifo.
- sc_event et data_written_event( )
  - Ces méthodes renvoient une référence à un objet de type événement qui sera notifié lors d'une écriture dans la fifo. Cet objet événement peut être utilisé dans une fonction wait( ) afin par exemple d'y insérer un timeout.

Utilisation des sc_fifo<T> - Exemple

```
template <class T>
SC_MODULE(DF_Adder){
   sc_fifo_in<T> input1, input2;
   sc_fifo_out<T> output;
   sc_fifo_out<T> output2;

   void process(){
       while(1){
           output.write(input1.read() +
           input2.read());
       }
   }

   SC_CTOR(DF_Fork) {
       SC_THREAD(process);
   }
};
```

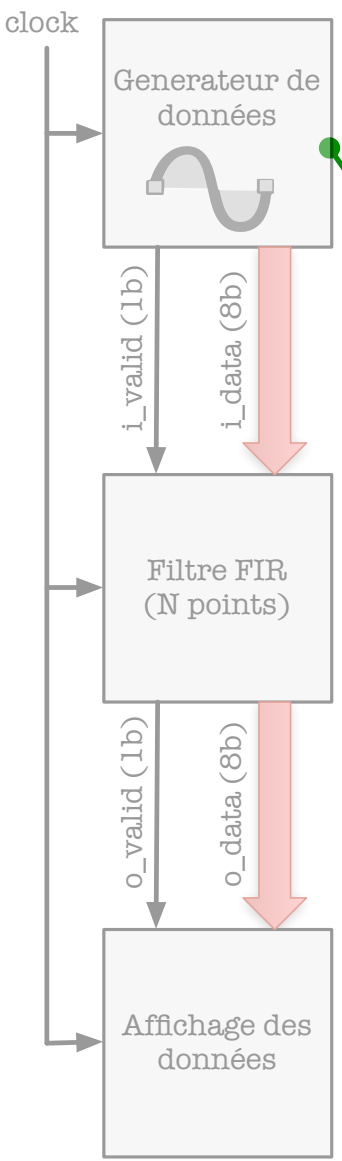
```
template <class T>
SC_MODULE(DF_Fork){
   sc_fifo_in<T> input;
   sc_fifo_out<T> output1;
   sc_fifo_out<T> output2;

   void process() {
       while(1) {
           T value = input.read();
           output1.write(value);
           output2.write(value);
       }
   }

   SC_CTOR(DF_Fork) {
       SC_THREAD(process);
   }
};
```

Exemple pédagogique (SystemC) - Module de filtrage



Modification du système => modélisation avec des communications abstraites

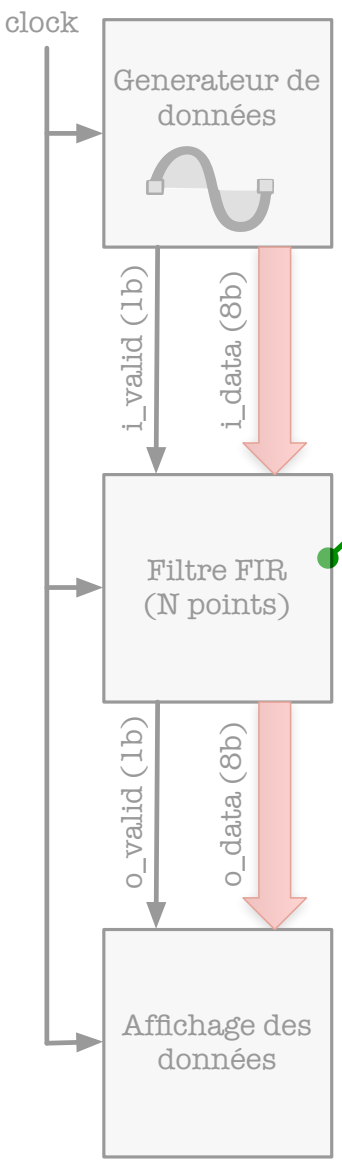
Exemple pédagogique (SystemC) - Module de génération




```
#include "systemc.h"
#include "math.h"

SC_MODULE(Gene)
{
     sc_fifo_out< int > s;

     SC_CTOR(Gene)
     {
          SC_THREAD(do_gen);
     }

    void do_gen();
};
```

```
void Gene::do_gen()
{
     int value;
     double x = 0;
     while(1){
          value = round(77 * sin( (2 * PI * x) / 30.0) );
          s.write( value );
          x += 1.0;
     }
}
```

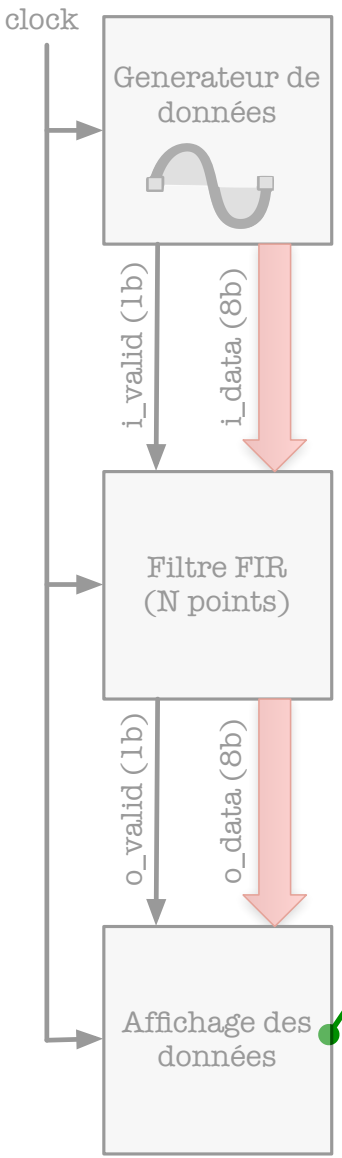
Exemple pédagogique (SystemC) - Module de filtrage




```systemc
#include "systemc.h"

SC_MODULE(FIR_2pts)
{
public:
     sc_fifo_in < int > e;
    sc_fifo_out< int > s;

     SC_CTOR(FIR_2pts)
     {
          SC_THREAD(do_fir);
     }

    void do_fir();
 };
```

```systemc
#include "FIR_2pts.h"

void FIR_2pts::do_fir(){
     int Xn;
     while( true ){
          int IN = e.read();
          s.write( ((Xn + IN)/2 ) );
          Xn = IN;
     }
}
```

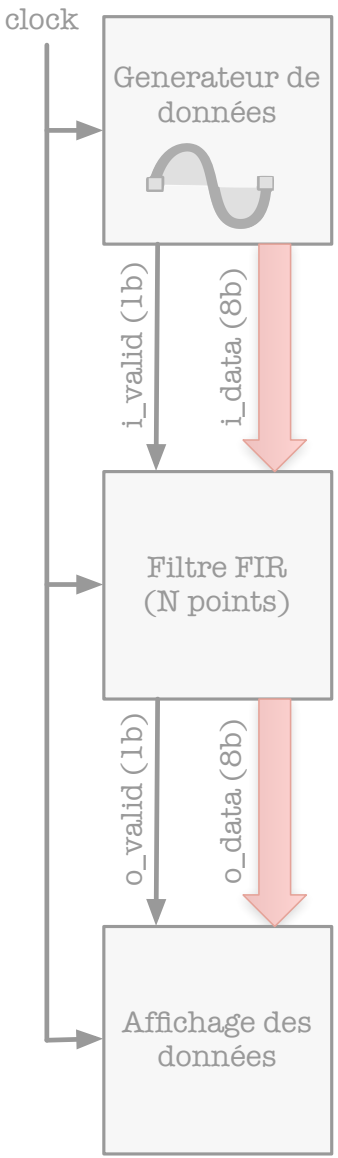
Exemple pédagogique (SystemC) - Module d'affichage




```
#include "systemc.h"

SC_MODULE(Terminal)
{
     sc_fifo_in< int >  e;

     void do_print();

     SC_CTOR(Terminal)
     {
          SC_THREAD(do_print );
     }
};
```

```
#include "Terminal.h"

void Terminal::do_print()
{
     int count = 10;
     while( true ){
          int value = e.read();
          cout << "Time = " << sc_time_stamp();
          cout << " / out = " << value <<  endl;
          if( !count-- ) sc_stop();
     }
}
```

Exemple pédagogique (SystemC) - Top module




```
SC_MODULE(top_module)
{
public:
          Gene     *gene;
          FIR_2pts *fir;
          Terminal *term;
          sc_fifo< int > *f1;
          sc_fifo< int > *f2;
public:
     SC_CTOR(top_module)
     {
          gene = new Gene    ("gene");
          fir  = new FIR_2pts("filtre");
          term = new Terminal("term");
          f1   = new sc_fifo< int >("f1", 2);
          f2   = new sc_fifo< int >("f2", 2);
          gene->s(*f1);
          fir->e (*f1);
          fir->s (*f2);
          term->e(*f2);
     }

    ~top_module(); // desctructeur
 };
```

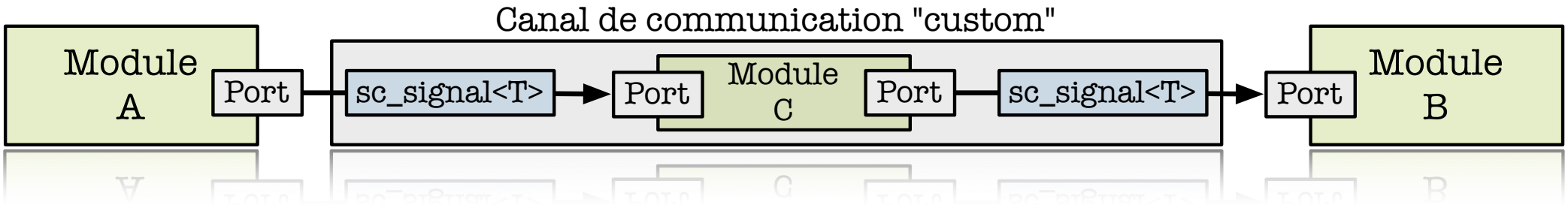
```
#include "top_module.h"

top_module::~top_module(){
     delete gene;
     delete fir;
     delete term;
     delete f1;
     delete f2;
}
```

Définition d'un nouveau canal de communication...

- En fonction du système à modéliser il peut être nécessaire de définir un nouveau canal de communication,
  - Modélisation du temps,
  - Modélisation de conflits d'accès au canal,
  - Modélisation d'arbitrage,
  - Modélisation de perte de paquets,
- Plusieurs approches possibles,
  - Créer un nouveau canal (complexe),
  - Modéliser le canal via un module,

Définition d'un nouveau canal de communication...

Création d'un canal de communication «primitif»

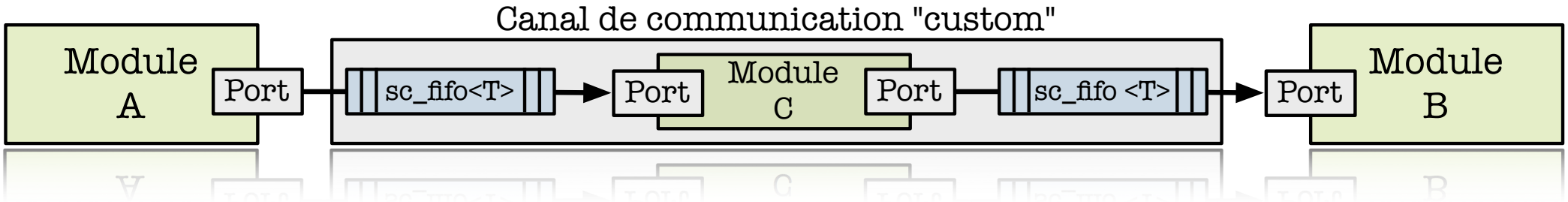


Création d'un canal de communication à base de modules

Exemple de modélisation d'un canal abstrait...

- Nous allons prendre un exemple, imaginez:
  - Des pigeons voyageurs,
  - Un chasseur (de pigeons)
- Imaginons un canal de communication «peu» fiable,
  - Certaines données peuvent disparaitre !
- Intéressant pour valider le système dans un environnement perturbé.

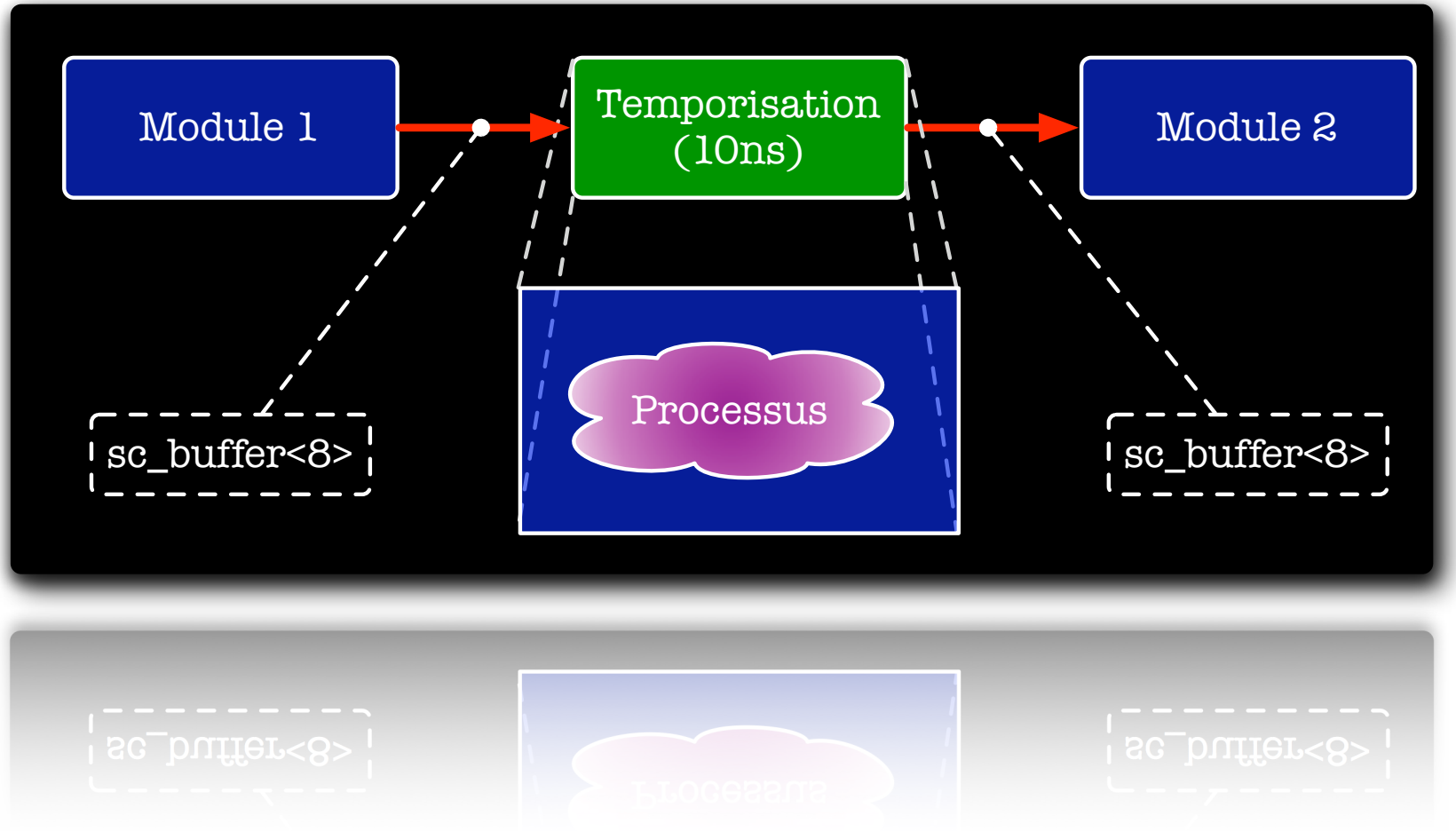
Exemple de modélisation d'un canal abstrait...




```
SC_MODULE(Pigeon)
{
public:
     sc_fifo_in <bool>  e;
     sc_fifo_out<bool>  s;

     SC_CTOR(Pigeon){
          SC_THREAD(do_pigeon);
     }

     void do_pigeon();
};
```

```
void Pigeon::do_pigeon(){
     while( true ){
          if( rand() == 13 ){
               e.read();
          }else{
               s.write( e.read() );
          }
     }
}
```

Modélisation de canaux de communication plus concrets



Nous allons dans le cadre de cet exemple définir un canal de communication qui prend en compte le temps de transfert des données (10ns) entre les différents modules afin de préciser un peu plus le comportement réel du système.

Modélisation de canaux de communication plus concrets

Le fonctionnement de ce canal de communication particulier est correct (les données sont retardées de 10ns).

Cependant des problèmes apparaissent (disparition de données) lorsque le débit des données en entrée en < 10ns ?!

```
SC_MODULE(CanalTimed)
{
   sc_out<sc_uint<8> > e;
   sc_out<sc_uint<8> > s;

   void do_tempo();

   SC_CTOR(CanalTimed)
   {
        SC_THREAD(do_tempo);
        sensitive << e;
   }
};
```

```
void CanalTimed::do_tempo()
{
   sc_uint<8> data = 0;
   while( true ){
        // Attente d'une donnee
        wait( );

        // Lecture de la donnee
        data = e.read();

        // Attente puis transfert
        wait( 10, SC_NS );
        s.write( data );
   }
}
```

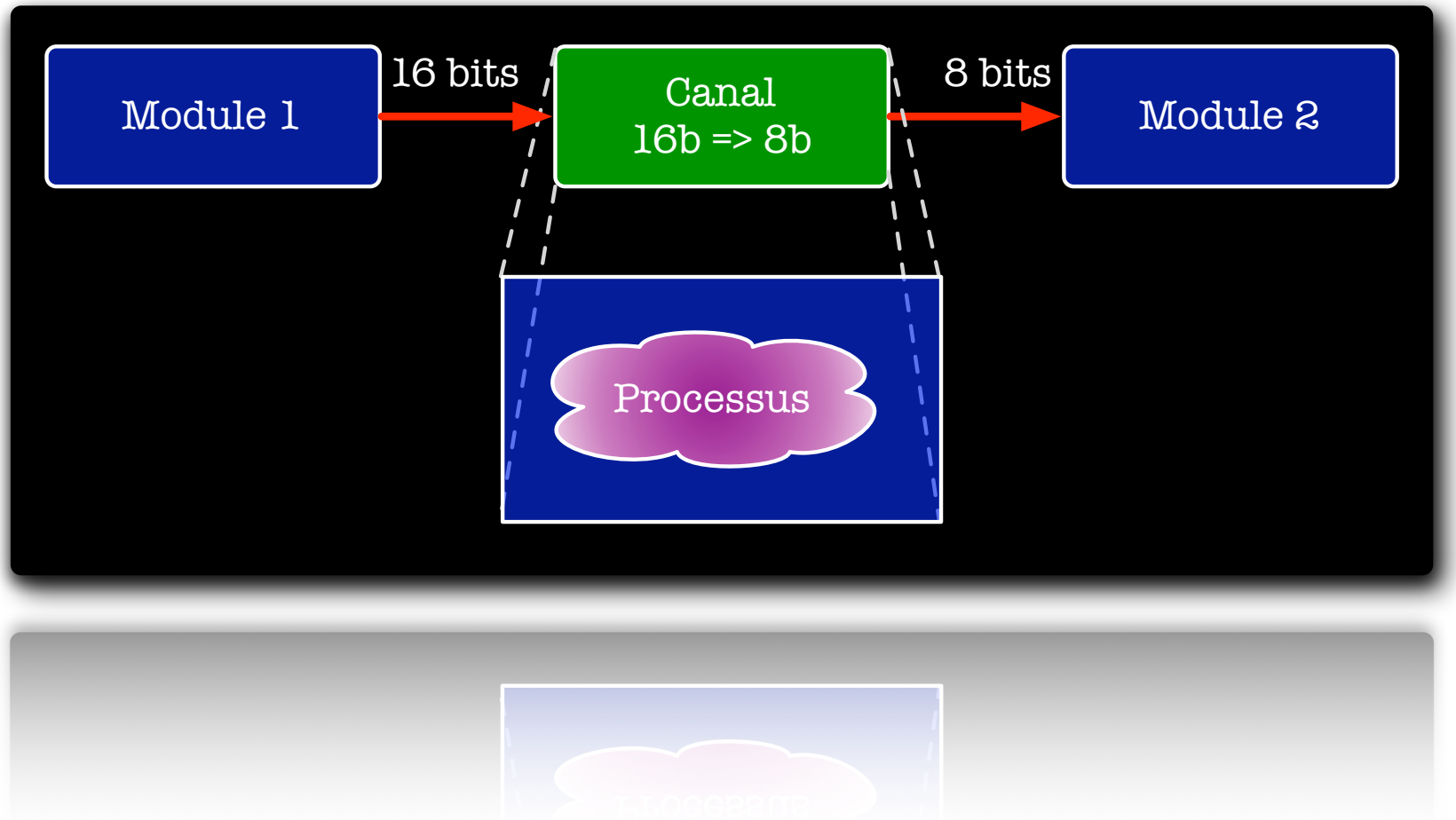
Exemple de raffinement d'un canal (sc_signal)

L'utilisation en interne de la Fifo permet de s'assurer que nous ne perdons aucun donnée dans le canal de communication.

```
SC_MODULE(CanalFifo)
{
    sc_out <sc_uint<8> > e;
    sc_out <sc_uint<8> > s;
    sc_fifo<sc_uint<8> > *fifo;

    void do_input();
    void do_output();

    SC_CTOR(CanalFifo)
    {
        fifo = new sc_fifo<sc_uint<8> >(16);
        SC_THREAD(do_input);
        sensitive << e;
        SC_THREAD(do_output);
        sensitive << e;
    }
};
```

```
void CanalFifo::do_input(){
    while( true ){
        // Attente d'une donnee
        wait( );

        // Memorisation dans la fifo
        fifo->write( e.read() );
    }
}

void CanalFifo::do_output(){
    sc_uint<8> data = 0;
    while( true ){
        // Attente d'une donnee
        data = fifo->read();

        // On attend la duree prevue
        wait(10, SC_NS);

        // On transmet la donnee
        s.write( data );
    }
}
```

Modélisation de canaux de communication plus concrets



Nous allons modéliser un canal de communication abstrait (sans connaître son implantation réelle) qui va transmettre des données reçues sur 16 bits en 2 paquets de 8 bits

Canal de communication série - asynchrone

Modélisation d'un canal asynchrone (comportement)

```
SC_MODULE(CanalAsync)
{
    sc_out<sc_uint<16> > e;
    sc_out<sc_uint<8>  > s;

    void do_gen();

    SC_CTOR(CanalAsync)
    {
        SC_THREAD(do_gen);
        sensitive << e;
    }
};
```

```
void CanalAsync::do_gen(){
    sc_uint<16> data = 0;
    while( true ){
        // On attend l'arrivee d'une donnee
        wait( );

        // On transmet lit la donnee
        data = e.read();

        // On transmet le premier octet
        wait(1, SC_NS);
        s.write( data.range(7, 0) );

        // On transmet le second octet
        wait(1, SC_NS);
        s.write( data.range(15, 8) );
    }
}
```

Canal de communication série - synchrone

Modélisation d'un canal synchrone (cycle près bit près)

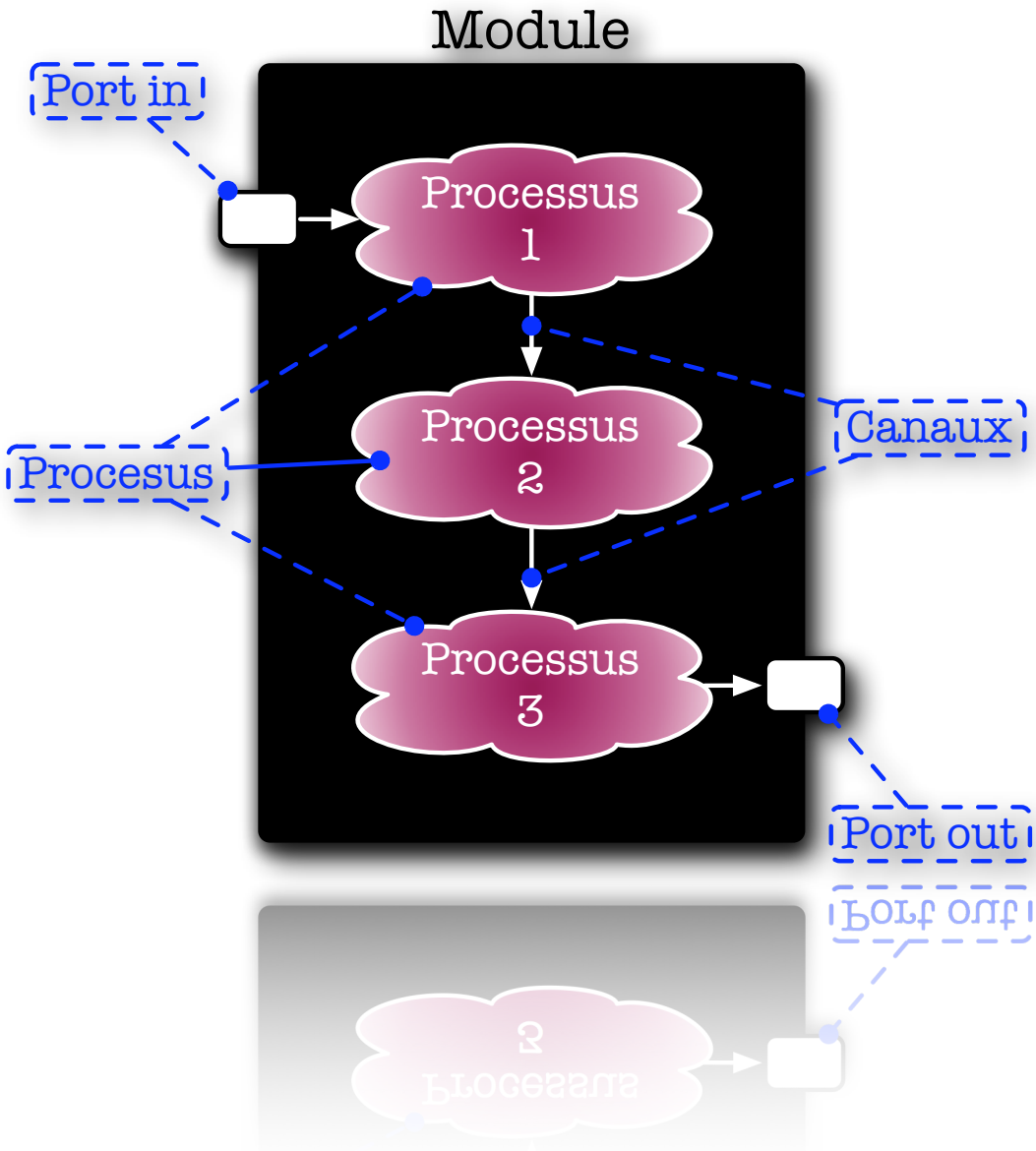
```
SC_MODULE(CanalSync)
{
   sc_in <bool> clk;
   sc_in <bool> in_valid;
   sc_out<bool> out_valid;
   sc_out<sc_uint<16> > e;
   sc_out<sc_uint<8>  > s;

   void do_gen();

   SC_CTOR(CanalSync)
   {
       SC_CTHREAD(do_gen, clk.pos());
   }
};
```

```
void CanalSync::do_gen(){
   sc_uint<16> data = 0;
   while( true ){
       // On attend l'arrivee d'une donnee
       while( in_valid.read() == false ){
           out_valid.write( false );
           wait( );
       } data = e.read();

       // On transmet le premier octet
       wait( );
       s.write( data.range(0, 7) );
       out_valid.write( true );

       // On transmet le second octet
       wait( );
       s.write( data.range(15, 8) );
   }
}
```

Partie 2
«modélisation flexible du comportement»

Introduction aux processus

- Les processus constituent le coeur des modules. Ils sont en charge de la description du comportement du composant,
- Un processus est défini comme étant une méthode,
  - A la création de l'objet, on va spécifier au simulateur quelles méthodes doivent être exécutées en cas d'apparition de stimulus externes,
- Les processus vont communiquer entre eux à l'aide de canaux de communication.



Gestion de ces différents comportements

- SystemC met à la disposition du concepteur 3 types de processus possédant des propriétés différentes,
  - SC_METHOD
    - Les processus de ce type sont privilégiés pour les composants dont le comportement est déclenché à chaque changement d'une de ses entrées. Ce type est souvent utilisé pour modéliser les composants combinatoires (asynchrones),
  - SC_THREAD
    - Les processus de ce type sont mis en oeuvre pour gérer les comportements synchrones ou la liste de sensibilité peu évoluer dans le temps (FSM dont les actions provoquant des transitions varient en fonction de l'état courant),
  - SC_CTHREAD
    - Les processus de ce type correspondent à un sous-ensemble des SC_THREAD. Ces processus évoluent uniquement sur les fronts d'horloge (sensibilité unique).
- Le choix du type de processus va être réalisé par le concepteur en fonction du comportement à modéliser,

Le fonctionnement des SC_METHOD

- Introduction
  - Les SC_METHOD sont des processus particuliers qui permettent de modéliser le comportement de composants dont la liste de sensibilité est figée (statique) lors de l'exécution de la simulation,
  - Les SC_METHOD ne sont pas des threads à part entière car ils doivent rendre la main après chaque exécution (return;) sinon cela bloque la simulation !
    - Un processus de type SC_METHOD est d'un point de vue sémantique une simple fonction (notion de processus en VHDL)
  - Les processus de classe SC_METHOD sont lancés à chacun des changements intervenus sur leur liste de sensibilité (A puis B => 2 exécutions),
- Remarques
  - Les variables locales contenues dans des méthodes de type SC_METHOD sont ré-initialisées à chaque entrée dans le processus.
  - Si des données doivent être mémorisées, il faut le faire à l'aide d'attributs dans la classe (variables globales dans ce cas de figure).

Gestion de la liste de sensibilité du processus

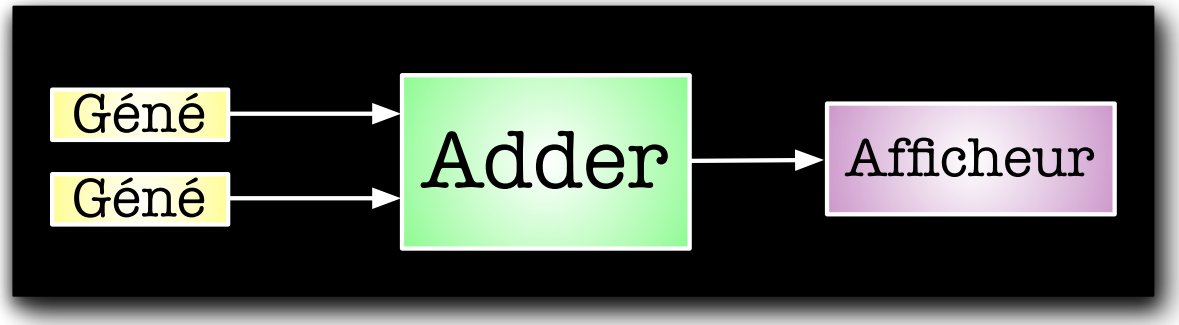
- La spécification de la liste de sensibilité se fait dans le constructeur du module (après la déclaration du processus et de son type),
  - Pour ajouter un signal dans la liste de sensibilité d'un processus, on se sert de la méthode sensitive( ). Cette méthode prend en argument un signal et s'applique à la dernière SC_METHOD enregistrée.
- Il est possible de filtrer les fronts montants ou descendants (uniquement les signaux sur 1 bit),
  - Déclarer la SC_METHOD comme sensible à tous les fronts du signal (sensitive << clock), et discriminer dans le corps du processus les fronts (if clock.posedge( ) {...}).
  - On peut aussi filtrer les fronts dans la liste de sensibilité à l'aide des constructions suivantes : sensitive << clock.pos( ) et/ou sensitive << mon_signal.neg( ), ce qui accélère les simulations (moins d'exécution des processus).
- Processus avec plusieurs sensibilités
  - sensitive << clock.pos() << clock.neg;
  - sensitive << Input_1 << Input_2;

Exemple de définition d'une SC_METHOD

```
SC_MODULE(mon_module) {

    sc_in<bool> clk;                  // Signaux arrivant en entrée de
    sc_in<bool> reset;                // la classe

   void comportement();               // Méthode contenant le comportement

    SC_CTOR(mon_module) {             // Constructueur de la classe
       SC_METHOD(ma_sc_method);       // Enregistrement du processus
       sensitive << enable;           // Insertion des signaux a gérer
       sensitive(reset.pos());        // dans la liste de sensibilité
        // ... ... ... ... ...
   }
}
```

On spécifie le type du processus ainsi que les signaux auxquels il est sensible

Module implémentant l'addition (Adder)



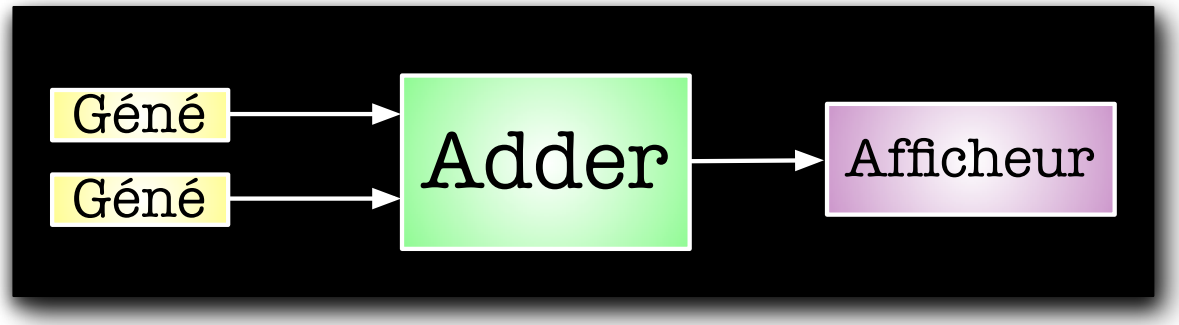
Adder.h

```
SC_MODULE(Adder)                // module (class) declaration
{
   sc_in<int> a, b;             // ports
   sc_out<int> s;

   void do_add();               // process

   SC_CTOR(Adder)               // constructor
   {
       SC_METHOD(do_add);       // register do_add to kernel
       sensitive << a << b;     // sensitivity list of do_add
   }

};
```

Adder.cpp

```
void Adder::do_add()
{
   s.write( a.read() + b.read() );
   // Identique ^ => s = a + b;
}
```

Module d'affichage des données



Terminal.h

```
SC_MODULE(Terminal)
{
   sc_in<int> a;

   void do_print();

   SC_CTOR(Terminal)
   {
       SC_METHOD(do_print);
       sensitive << a;
   }

};
```

Terminal.cpp

```
#include "Terminal.h"

void Terminal::do_print()
{
   cout << "Time = " << sc_time_stamp();
   cout << " => Valeur recue : " << a << endl;
}
```

Les processus de type SC_THREAD

Introduction

- Les SC_THREAD sont des processus particuliers implémentés sous forme de threads autonomes dans le simulateur.
- Les SC_THREAD permettent de modéliser des processus dont la liste de sensibilité peut évoluer dynamiquement durant l'exécution.
- Les SC_THREAD sont des processus développés autour d'une méthode généralement composée d'une boucle infinie qui s'endort en attente d'événements. Ces processus ne sont plus exécutés à chaque apparition d'un événement, ils sont réveillés !

Remarques

- Une fois le traitement associé à un événement effectué, le SC_THREAD est mis en sommeil (attente d'un nouvel événement) à l'aide de la méthode wait( ).
- Si un thread s'arrête en arrivant au bout du code à exécuter ou rencontre une instruction return alors il ne s'exécutera plus !
- L'état des variables internes à un processus de type SC_THREAD sont conservées en l'état lors du réveil du processus car il est juste endormi...

Mise en oeuvre des processus SC_THREAD

Création des SC_THREAD

- Le processus de création des SC_THREAD est exactement identique à celui de création d'une SC_METHOD, en remplaçant cette dernière par la macro SC_THREAD...
- La déclaration et la gestion de la liste de sensibilité est aussi identique.

Quand utiliser un SC_THREAD ?

- Quand on a besoin d'introduire des notions de temps dans les modèles (attendre non plus un événement, mais un délai relatif avant d'entreprendre une action),
- Modéliser des processus séquentiels, typiquement des machines synchrones ou asynchrones à états implicites (décrites étapes par étape).
- Modéliser des processus qui stockent localement de l'information entre deux activations successives,
- Modéliser des processus accédant à des ressources bloquantes (entrée-sorties, fichiers, Fifo, sémaphores, réseaux, ...),
- Si on a le choix, les SC_METHOD qui simulent beaucoup plus rapidement.

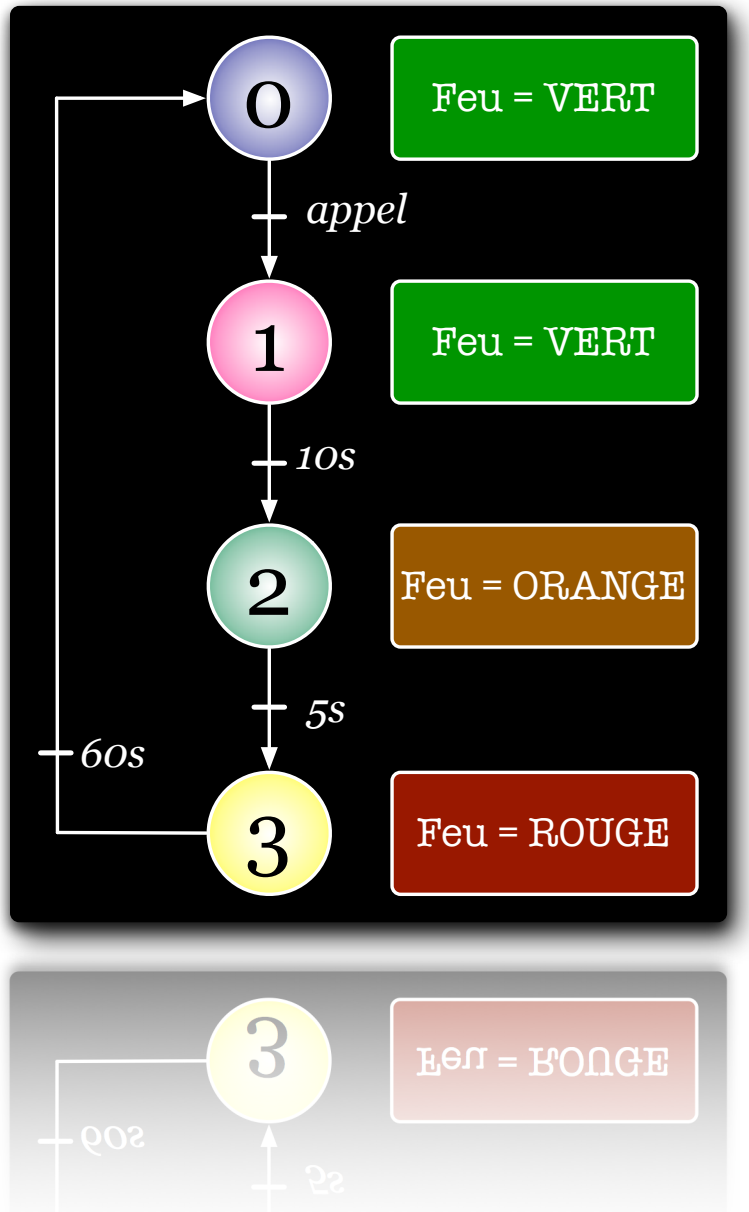
Différentes mises en attente pour un processus

Afin de simplifier l'écriture des modèles en SystemC, il est possible de spécifier la mise en attente d'événements de différentes manières

```
// Process P1 (SC_THREAD, sensitive to clock.pos())
wait();                          // wait for static sensitivity list
wait(e1);                        // wait for event
wait(e1 | e2 | e3);              // wait for first event
wait(e1 & e2 & e3);              // wait for all events
wait(200, SC_NS);                // wait for 200 ns
wait(200, SC_NS, e1 | e2); // wait, 200 ns timeout
```

Dynamic sensitivy

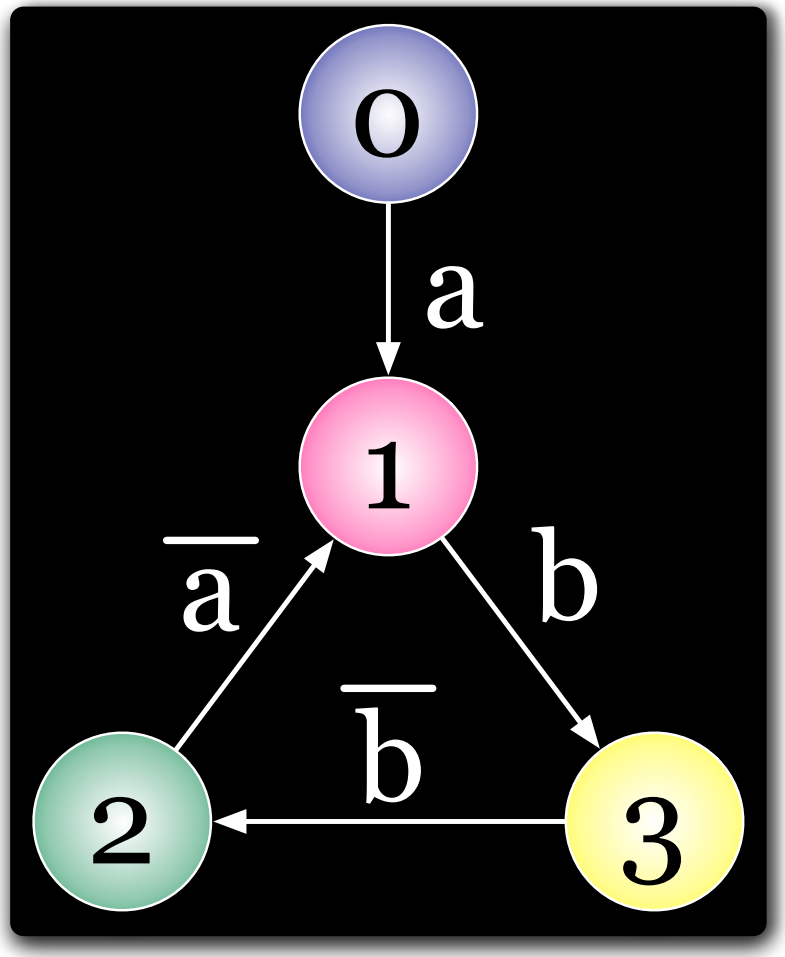
Exemple d'un automate gérant un feu rouge




```
// Ce processus est un SC_THREAD possédant
// dans sa liste de sensibilité une hologe
// de fréquence 1Hz
void Gestion_Feu::mon_thread() {
    while ( true ) {
        couleur_feu = VERT;
        wait(appel_bouton);

        // Temporisation de 10 secondes
        // puis le feu doit passer ^ l'orange
        for(int i=0; i<10; i++) wait( );
        couleur_feu = ORANGE;

        // Temporisation de 5 secondes
        // puis le feu doit passer au rouge
        wait( 5, SC_S );
        couleur_feu = ROUGE;

        // Temporisation d'une minute
        // puis on repasse au vert (initial)
        wait(60, SC_SC);
 }
}
```

Exemple d'une machine d'état (FSM)

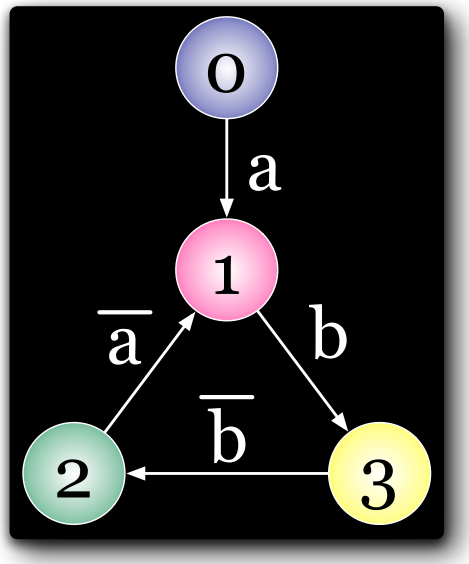
- Dans un premier temps, nous allons considérer la machine d'états présentée par le diagramme de droite,
  - Décrivez cette machine d'état en SystemC à l'aide d'un processus de type SC_THREAD,
  - Pourrait on utiliser à la place un SC_METHOD ?
  - Le fonctionnement serait il identique ?
  - Quel serait le code source équivalent ?



Exemple d'une machine d'état (FSM) - Solution (1)

Modélisation d'une FSM Asynchrone

Le comportement est modélisé à l'aide d'un processus de type SC_METHOD



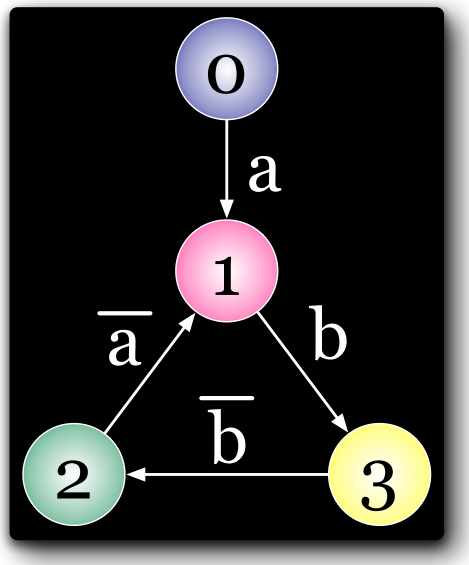

```
SC_MODULE(FSM_Async_METHOD)
{
private:
    int state;
public:
    sc_in  <bool> a;
    sc_in  <bool> b;

    void do_gen();

    SC_CTOR(FSM_Async_METHOD)
    {
        SC_METHOD(do_gen);
        sensitive << a << b;
        state = 0;
    }
};
```

```
void FSM_Async_METHOD::do_gen(){
    if( state == 0 ){     // STATE 0
        if(a.read() == 1) etat = 1;
    }else if( state == 1 ){   // STATE 1
        if(b.read() == 1) etat = 3;
    }else if( state == 2 ){   // STATE 2
        if(a.read() == 0) etat = 1;
    }else if( state == 3 ){   // STATE 3
        if(b.read() == 0) etat = 2;
    }
}
```

Exemple d'une machine d'état (FSM) - Solution (2)

Modélisation d'une FSM Asynchrone

Le comportement est modélisé à l'aide d'un processus de type SC_THREAD



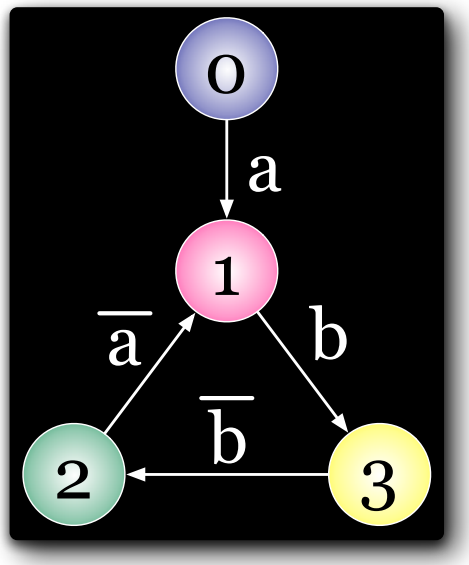

```
SC_MODULE(FSM_Async)
{
    sc_in  <bool> a;
    sc_in  <bool> b;

    void do_gen();

    SC_CTOR(FSM_Async)
    {
        SC_THREAD(do_gen);
        sensitive << a << b;
    }
};
```

```
void FSM_Async::do_gen(){
    int state = 0;
    while( true ){
        if( state == 0 ){     // STATE 0
            if(a.read() == 1) etat = 1;
            else wait( a );
        }else if( state == 1 ){   // STATE 1
            if(b.read() == 1) etat = 3;
            else wait( b );
        }else if( state == 2 ){   // STATE 2
            if(a.read() == 0) etat = 1;
            else wait( a );
        }else if( state == 3 ){   // STATE 3
            if(b.read() == 0) etat = 2;
            else wait( b );
        }
    }
}
```

Exemple d'une machine d'état (FSM) - Solution (3)

Modélisation d'une FSM Synchrone

Le comportement est modélisé à l'aide d'un processus de type SC_THREAD



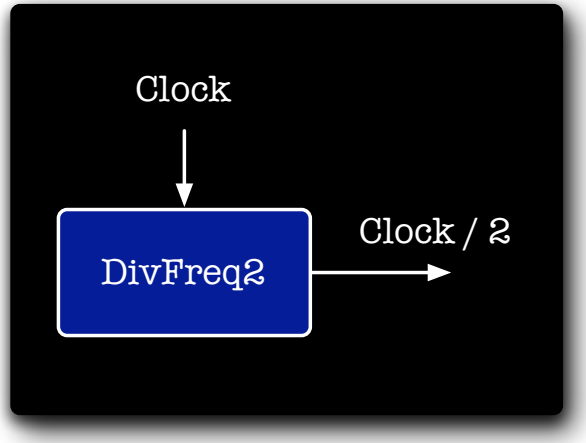

```
SC_MODULE(FSM_Sync)
{
    sc_in  <bool> clk;
    sc_in  <bool> a;
    sc_in  <bool> b;

    void do_gen();

    SC_CTOR(FSM_Sync)
    {
        SC_THREAD(do_gen);
        sensitive << clk;
    }
};
```

```
void FSM_Sync::do_gen(){
    int state = 0;
    while( true ){
        if( state == 0 ){     // STATE 0
            if(a.read() == 1) etat = 1;
            else wait( );
        }else if( state == 1 ){   // STATE 1
            if(b.read() == 1) etat = 3;
            else wait( );
        }else if( state == 2 ){   // STATE 2
            if(a.read() == 0) etat = 1;
            else wait( );
        }else if( state == 3 ){   // STATE 3
            if(b.read() == 0) etat = 2;
            else wait( );
        }
    }
}
```

Les processus SC_CTHREAD

- Introduction
  - Les processus déclarés comme étant de nature SC_CTHREAD sont proches de leurs homologues définis comme étant des SC_THREAD,
  - La nuance provient de la liste de sensibilité qui est dans ce cas figée à un unique signal qui jouera le rôle d'une horloge,
  - Il est toutefois possible de mixer les mises en sommeil (temporisation) des processus en utilisant des notions de temps (délais)
- Remarques
  - Ce type de processus est majoritairement employé dans les descriptions de bas niveau (Cycle Accurate) ou les composants sont tous synchrones en interne ainsi qu'au niveau de leurs interfaces de communication,
  - Ce sous ensemble des processus a été défini afin de simplifier la tache des outils de CAO qui génèrent automatiquement des descriptions RTL de circuit synchrones,

Exemple d'utilisation des processus SC_CTHREAD



On a introduit dans notre modèle SystemC la notion de temps de calcul et/ou propagation dans le processus de calcul.

```
SC_MODULE(DivFreq)
{
   sc_in <bool> clk;
   sc_out<bool> s;

   void do_gen();

   SC_CTOR(DivFreq)
   {
        SC_CTHREAD(do_gen, clk.pos());
   }
};
```

```
#include "DivFreq.h"

void DivFreq::do_gen()
{
    bool state = false;
    while( true ){
         wait( );
         wait(1, SC_NS);
         s.write( ! state );
    }
}
```

Impact des différentes approches sur le simulateur

- La différence majeur qui existe entre ces 3 modèles de processus que nous venons de décrire réside dans les conditions de leur déclenchement,
  - SC_METHOD : liste de sensibilité statique sur les entrées du module,
  - SC_THREAD : liste de sensibilité dynamique sur les entrées + délais,
  - SC_CTHREAD : actif uniquement sur les fronts d'une horloge définie statiquement,
- Entre chaque phase de calcul, le simulateur doit déterminer quels sont les processus à réveiller afin d'effectuer le traitement attendu,
  - Cette étape est consommatrice de temps CPU dans les systèmes complexes,
  - Réveiller un processus et l'exécuter si rien ne s'est produit est inefficace !
- Le choix du type de processus n'est pas anodin car il va impacter fortement sur les performances de simulation !
  - La majorité du temps de simulation peut être utilisée uniquement par le noyau du simulateur SystemC afin de réveiller, endormir des processus ?!

Partie 2
«Execution time modeling»

Modélisation du temps en SystemC

- Une classe nommée sc_time permet de modéliser le temps,
  - Couple (valeur, unité temporelle),
- Différentes résolutions (unités)
  - SC_SEC : seconde
  - SC_MS : milliseconde
  - SC_US : microseconde
  - SC_NS : nanoseconde
  - SC_PS : picoseconde
  - SC_FS : femtoseconde

```
sc_time periode(10, SC_NS);
sc_time delta  (1,  SC_FS);
sc_time latence;

latence = 10 SC_NS; // ERREUR !
```

La définition d'une valeur temporelle n'est possible que lors de l'instanciation d'un objet de type sc_time

Utilisation de la classe sc_time

- SystemC a surchargé un certain nombre d'opérateurs de base du langage,
- La classe sc_time supporte ainsi les opérations suivantes :
  - Affectation,
  - Addition,
  - Soustraction,
  - Multiplication,
  - Comparaisons (égalité, supérieur, etc.),
  - Affichage dans un flux standard,

```
sc_time periode(10, SC_NS);
sc_time delta  (1,  SC_FS);

sc_time latence;

sc_time borne_min;
borne_min = periode - delta;

sc_time borne_max;
borne_min = periode + delta;

sc_time delay;
if( periode > latence ){
    delay = latence;
}else{
    delay = 2 * delta;
}
```

Modélisation des horloges à l'aide de la classe sc_clock

- SystemC donne la possibilité de déclarer des horloges. C'est l'ordonnanceur qui se charge alors de la commutation des signaux. Le constructeur accepte les paramètres suivants:
  - Une chaîne de caractère précisant le nom de l'horloge,
  - La durée d'une période de l'horloge,
  - Au besoin le rapport cyclique de l'horloge s'il est différent de 50%.

```
// instancie une horloge de période 10ns
sc_clock clock1("clk100", 10, SC_NS);
sc_time t10(10, SC_NS);
sc_clock clock2("clk100", t10);

// instancie une horloge de période 15.3ns, de rapport cyclique 40%,
// démarrant au temps 45ms et dont le premier état est haut
sc_time tt(45, SC_MS);
sc_clock clock2("clk2", 15.3, SC_NS, 0.4, tt, true);
```

Utilisation du temps (sc_time) dans les processus

- La modélisation du temps permet l'expression de délais d'exécution dans les processus,
  - ➡ Etude des performances temporelles & de la synchronisation du système,
- Les spécifications temporelles sont utilisées afin d'exprimer :
  - ➡ Le temps passé à réaliser les calculs,
  - ➡ Le temps d'attente avant d'entreprendre une nouvelle action,
- Pas de réelle notion de temps dans les SC_METHOD,

```
void do_gen(){
    while( true ){
            state = ! state;
            out.write( state );
            wait(5, SC_NS);
    }
}
```

Générateur d'horloge

```
void do_FeuRouge(){
   while( true ){
        out.write( FEU_VERT );
        wait(30, SC_S);
        out.write( FEU_ORANGE );
        wait(5, SC_S);
        out.write( FEU_ROUGE );
        wait(60, SC_S);
   }
}
```

Gestion d'un feu rouge

On modélise le temps de calcul (time functionnal)

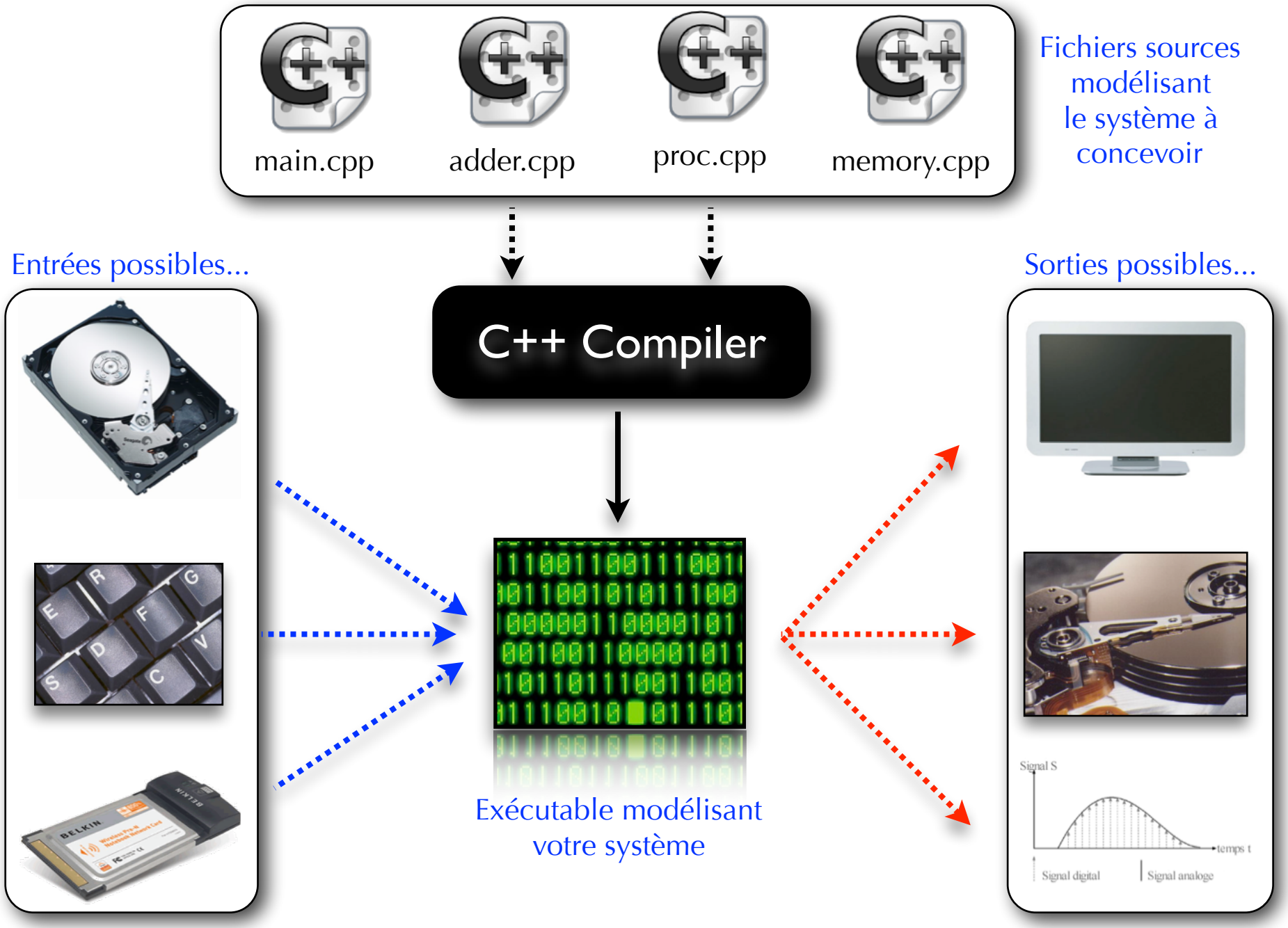
```
inline void ConversionCouleurs(int rvb[3], int ycbcr[3]){
     ycbcr[0] = (     C1 * rvb[0] + C4 * rvb[1] + C7 * rvb[2]) >> N;
     ycbcr[1] = (OF - C2 * rvb[0] - C5 * rvb[1] + C8 * rvb[2]) >> N;
     ycbcr[2] = (OF + C3 * rvb[0] - C6 * rvb[1] - C9 * rvb[2]) >> N;
}

void Conversion_TIMED::do_conversion(){
     int d[3];
     while( true ){
          d[0] = (int)e.read();
          d[1] = (int)e.read();
          d[2] = (int)e.read();

          ConversionCouleurs(d, t);
          wait(30, SC_NS);

          s.write( (unsigned char)t[0] );
          s.write( (unsigned char)t[1] );
          s.write( (unsigned char)t[2] );
     }
}
```

On modélise le temps de calcul (time functionnal)

```
#define N  12
#define C1 (sc_int<N>)( 0.299   * (1 << N))
... ... ... ... ... ... ... ...
#define OF (sc_int<N>)( 128     * (1 << N))

inline void Conversion_CYCLE_ACCURATE::do_conversion(){
    while( true ){
         int R = (int)e.read();
         int G = (int)e.read();
         int B = (int)e.read();

         int Y =  (C1 * R + C4 * G + C7 * B) >> N;
         wait(10, SC_NS);
         s.write( (unsigned char)Y );

         int Cb = (OF + C2 * R + C5 * G + C8 * B) >> N;
         wait(10, SC_NS);
         s.write( (unsigned char)Cb );

         int Cr = (OF + C3 * R + C6 * G + C9 * B) >> N;
         wait(10, SC_NS);
         s.write( (unsigned char)Cr );
    }
}
```

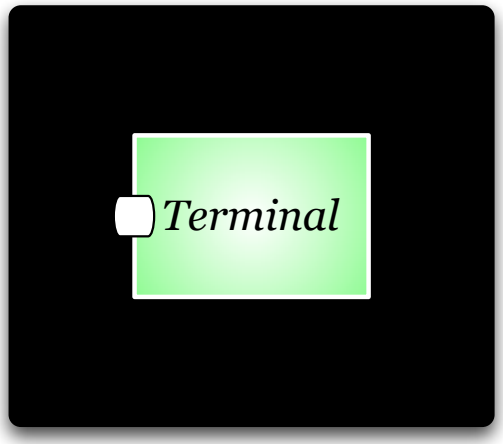
Partie 2
«analyse et comparaison des résultats»

Interaction de la simulation avec l'extérieur



Classe affichant les données à l'écran

```
SC_MODULE(Terminal)
{
  sc_in<int> a;

  void do_print();

  Terminal();

  SC_CTOR(Terminal)
  {
       SC_METHOD(do_print);
       sensitive << a;
  }

};
```




```
void Terminal::do_print()
{
   cout << "Time = " << sc_time_stamp() << endl;
   cout << " => Valeur recue : " << a << endl;
}
```

Classe stockant les données dans un fichier

```
SC_MODULE(Stockage_HD){
private:
    ofstream *sortie;

public:
    sc_in<int> a;

    SC_CTOR(Stockage_HD)
    {
        SC_METHOD(do_store);
        sensitive << a;
        sortie = new ofstream();
        sortie->open("resultats.txt");
    }

    ~Stockage_HD();

    void do_store();
};
```


```
#include "Stockage_HD.h"

Stockage_HD::~Stockage_HD(){
    sortie->close();
    delete sortie;
}

void Stockage_HD::do_store(){
    (*sortie) << a.read() << endl;
}
```

Partie 3
«SystemC usage in real life»

Partie 3
«Validation fonctionnelle»

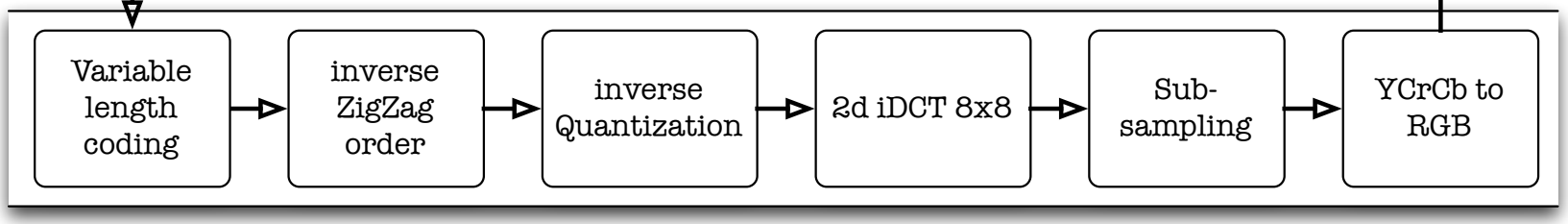
Modéliser pour valider fonctionnellement le système

La norme de compression JPEG contient plusieurs centaines de pages !

Etape 1, s'assurer que l'on a bien lu... et compris les algorithmes mis en oeuvre...

Dans le cas contraire, commencer l'implantation serait inutile !

Avant d'implanter une application il est nécessaire de la comprendre !

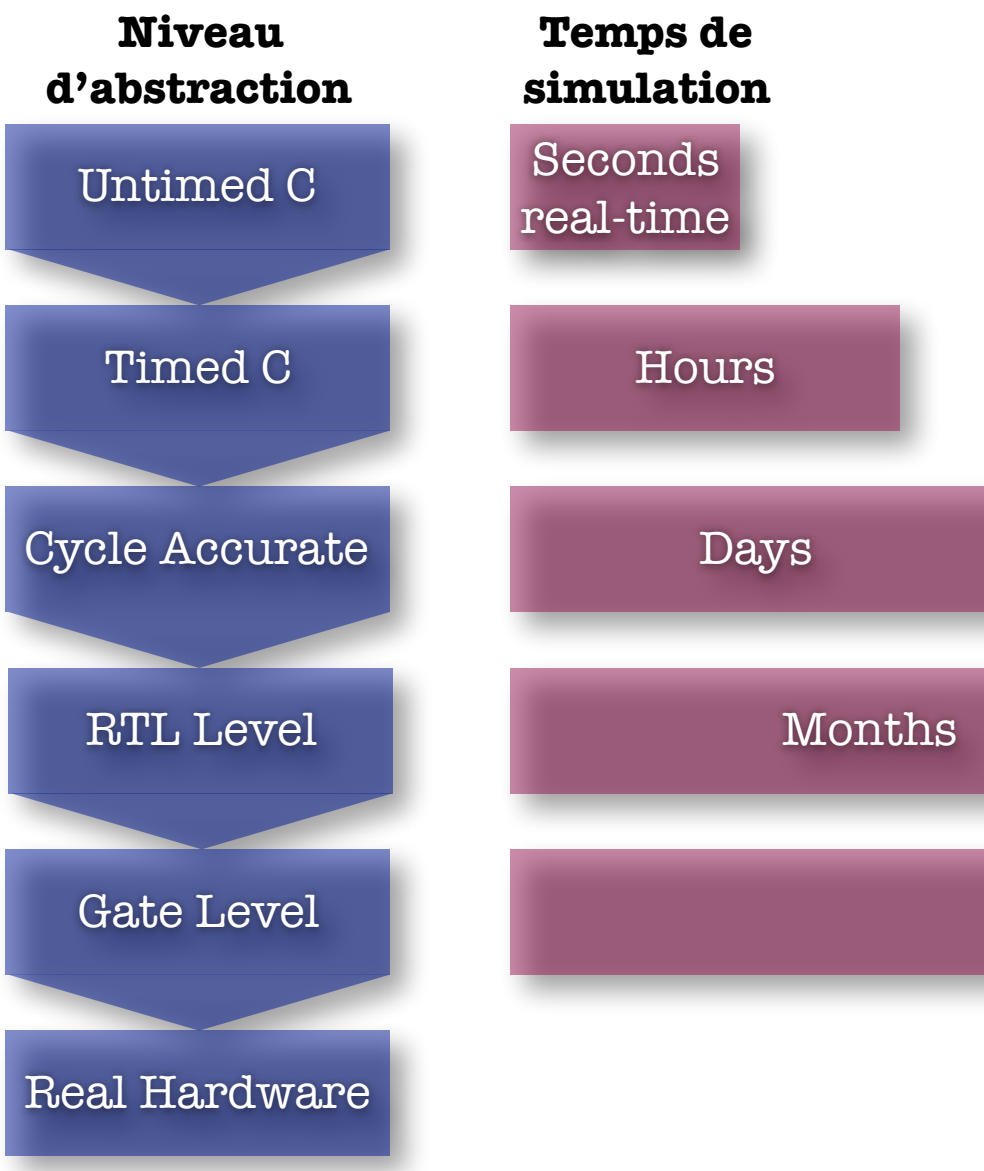


Modéliser pour valider fonctionnellement le système

- Vérifier sa compréhension des algorithmes permet d'éviter les désillusions post-implantation,
  - ➡ Comprendre ses erreurs post-implantation est chronophage,
  - ➡ Le temps... c'est de l'argent !

- Etudier les choix algorithmiques quand cela est possible,
  - ➡ Réduire la complexité calculatoire,

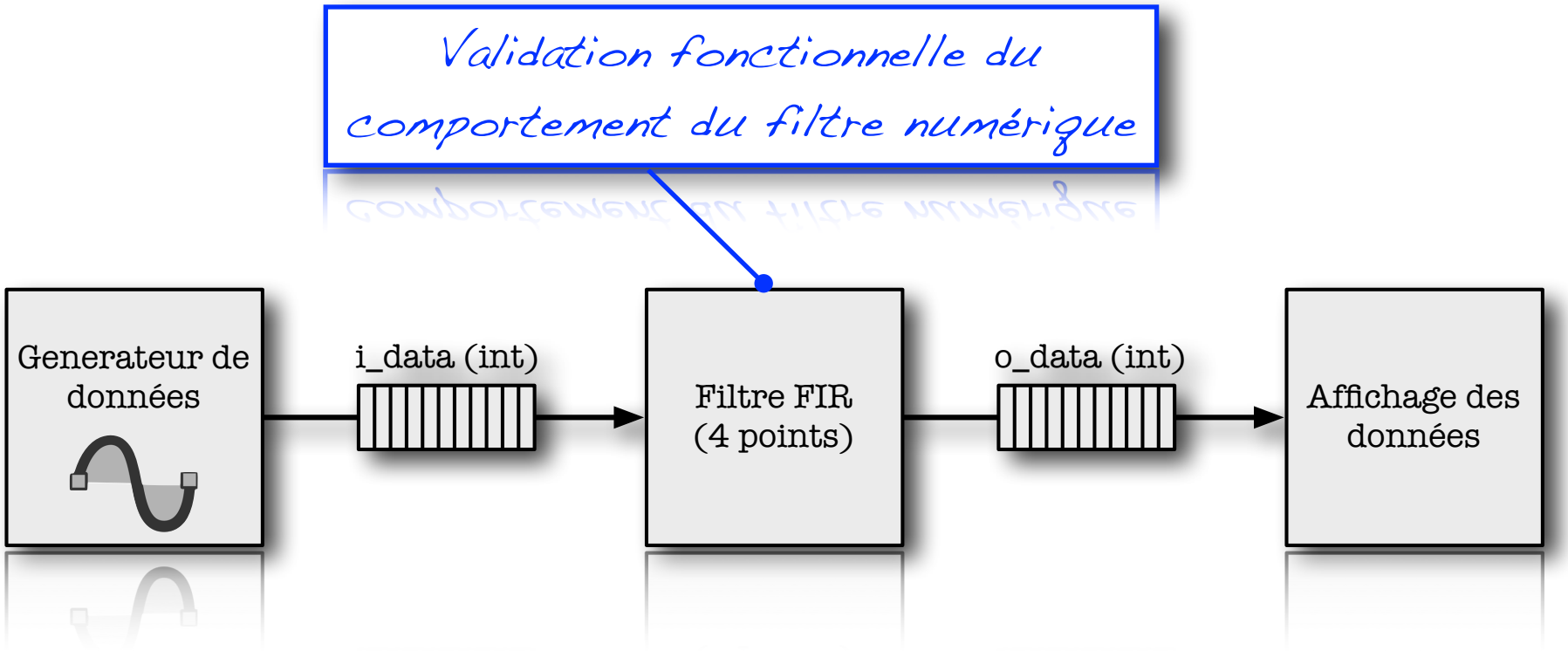
Evolution du temps de simulation des modèles

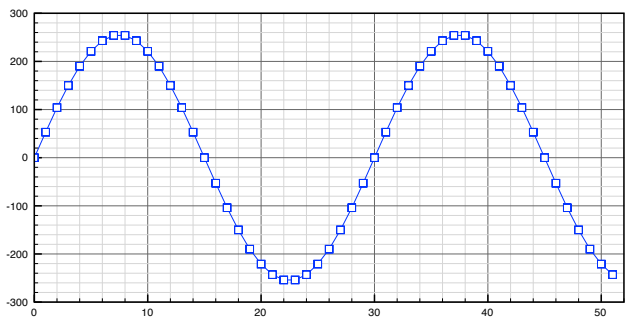


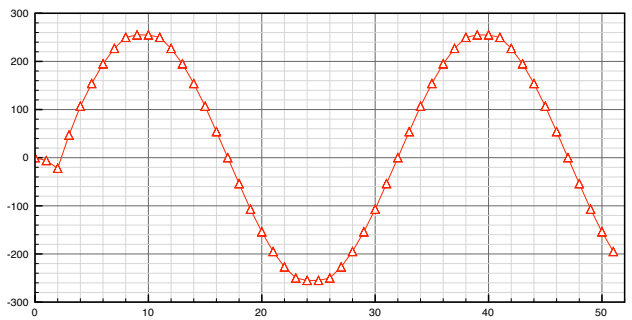
Le temps de simulation de systèmes complexes décrits à bas niveau peut prendre un temps considérable...

Il faut donc s'assurer que les problèmes ont été résolus auparavant !

Modèle SystemC élémentaire pour la validation fonctionnelle







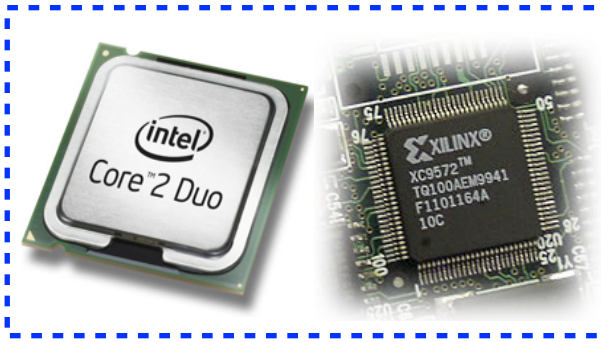
Partie 3
«Processus de raffinement»

Des spécifications à l'implantation

Spécifications de l'application

De nombreuses décisions vont être prises consciemment ou inconsciemment durant l'implantation

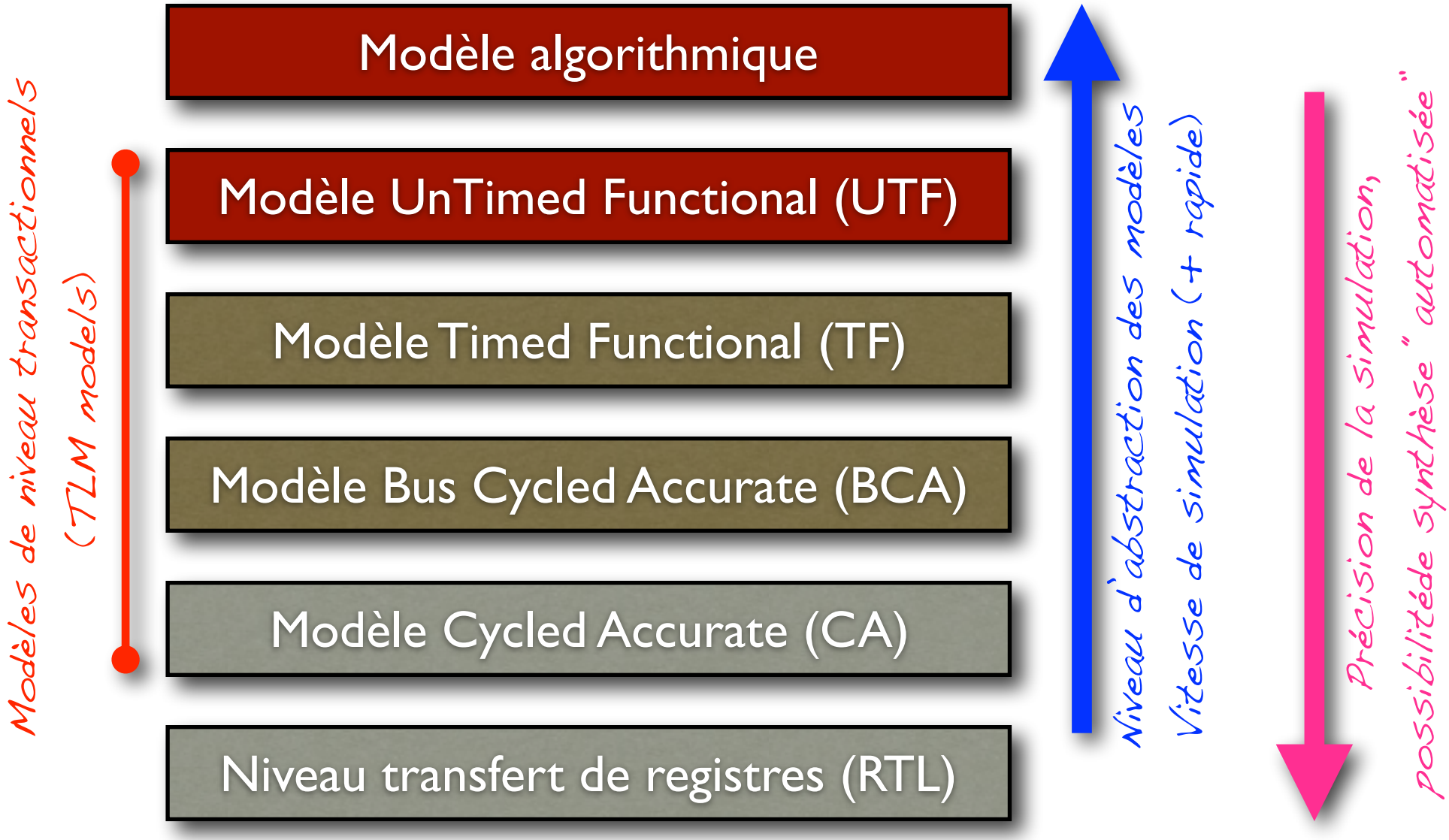
Processeur



Processeur + accélérateur

Accélérateur

Les niveaux d'abstraction et leurs spécificités



Les niveaux d'abstraction et leurs spécificités

- Modèle algorithmique
- Modèle UnTimed Functional (UTF)

Pas de notion de temps et pas de notion de transfert des données.

- Modèle Timed Functional (TF)
- Modèle Bus Cycled Accurate (BCA)

Notion de temps dans les processus et les transferts de données (modélisation gros grains)

- Modèle Cycled Accurate (CA)
- Niveau transfert de registres (RTL)

Les processus et les signaux sont au cycle près et au bit près.

Les niveaux d'abstraction et leurs spécificités

- Modèle algorithmique
- Modèle UnTimed Functional (UTF)
- Modèle Timed Functional (TF)
- Modèle Bus Cycled Accurate (BCA)
- Modèle Cycled Accurate (CA)
- Niveau transfert de registres (RTL)

Vérification fonctionnelle

Exploration algorithmique

Vérification algorithmique

Benchmarking des performances

Analyse des architectures

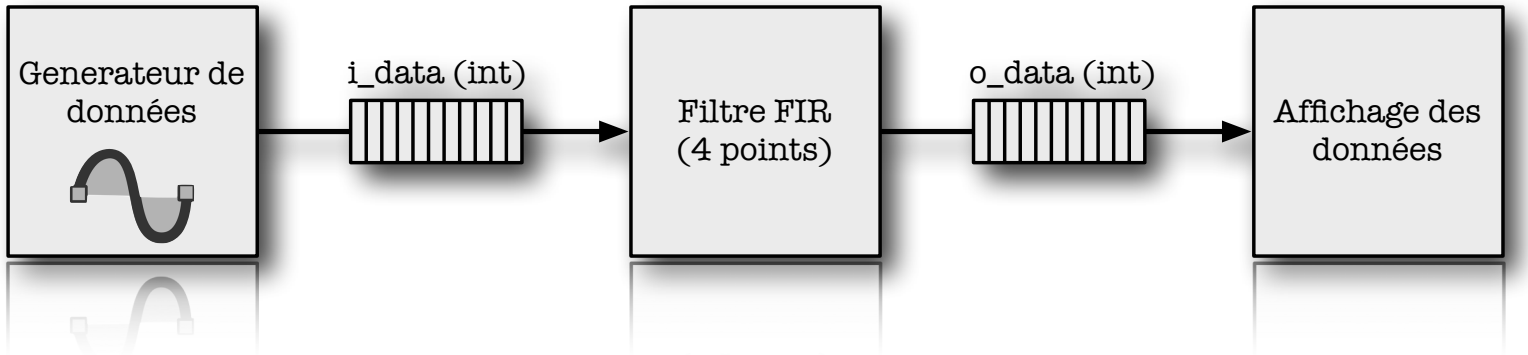
Développement des parties softs

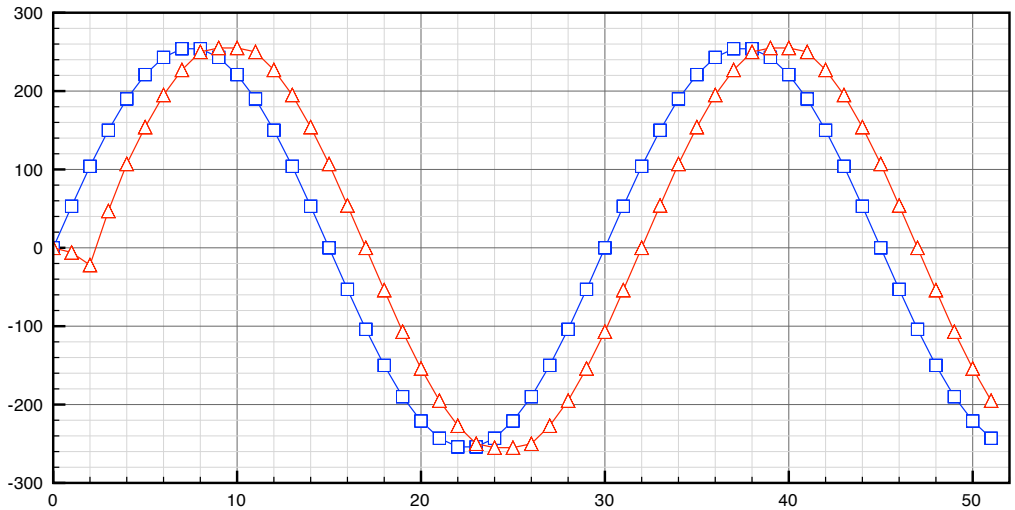
Estimation précise des perfs,

Développement des drivers

Développement des micro-architectures,

Filtre FIR 5 points de niveau algorithmique

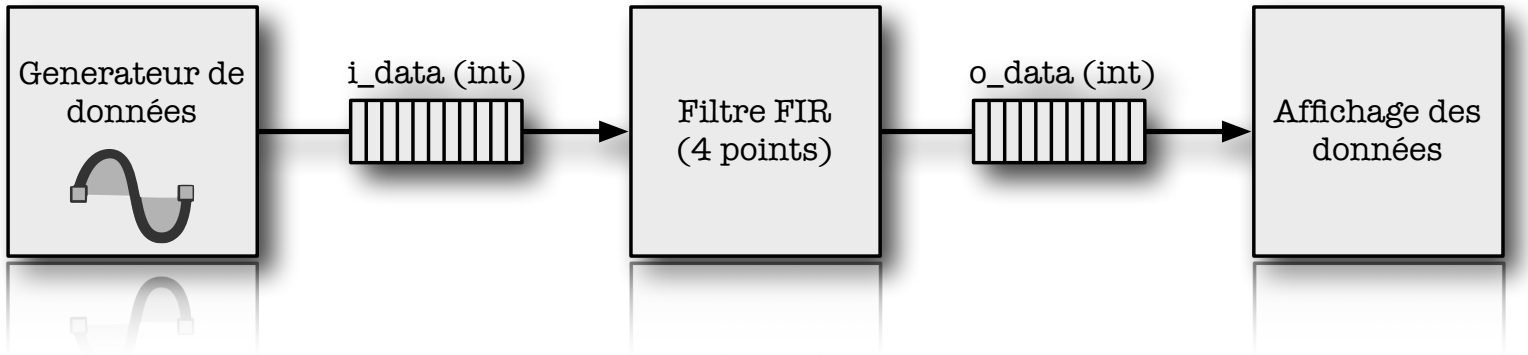




Filtre FIR 5 points de niveau algorithmique

```
int FIR(int xn){
     for(int i=0; i<4; i++){      //
          X[i+1] = X[i];          // Vieillissement
     }X[0] = xn;                  //

     double r = 0;
     for(int i=0; i<5; i++){      //
          r += X[i] * H[i];       // Calcul de la
     }                            // sortie
     return round( r );           //
}
```

Filtre FIR 5 points de niveau «Untimed-Functionnal»



Objectifs:

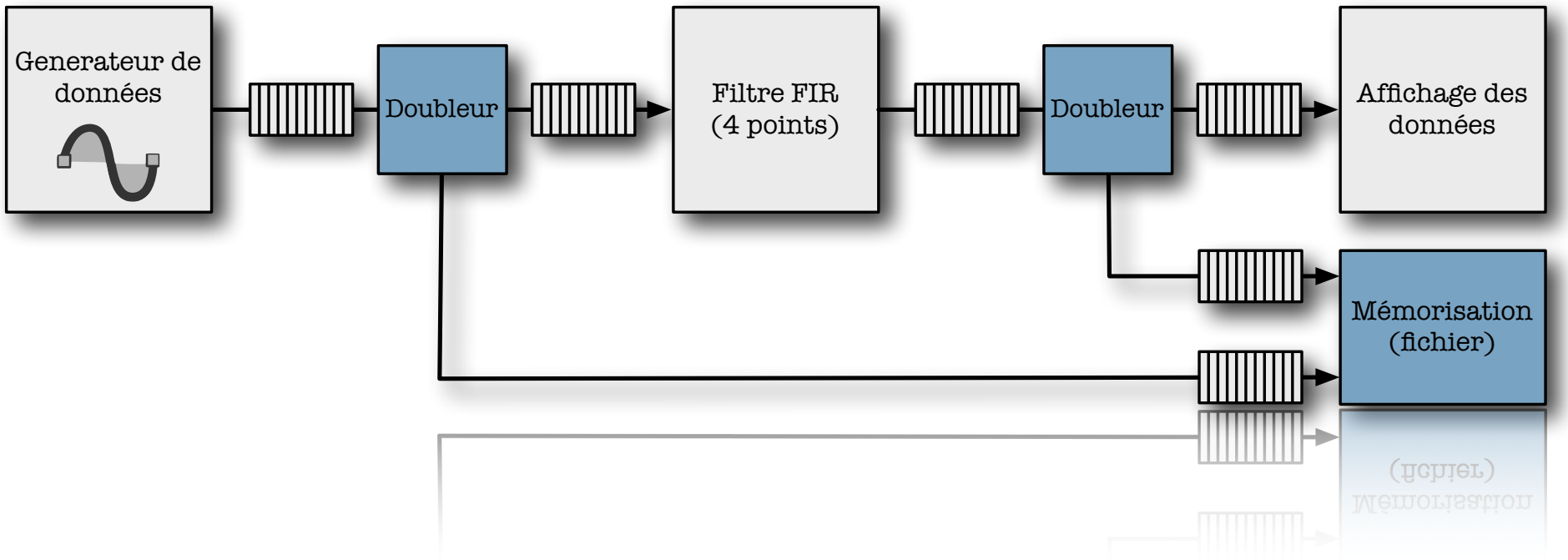
- Vérification fonctionnelle
- Exploration algorithmique

Filtre FIR 5 points de niveau «Untimed-Functionnal»

```
#include "systemc.h"

SC_MODULE(FIR_4pts)
{
public:
     sc_fifo_in <int> e;
    sc_fifo_out<int> s;

     SC_CTOR(FIR_4pts)
     {
          SC_THREAD(do_fir);
     }

    void do_fir();
 };
```

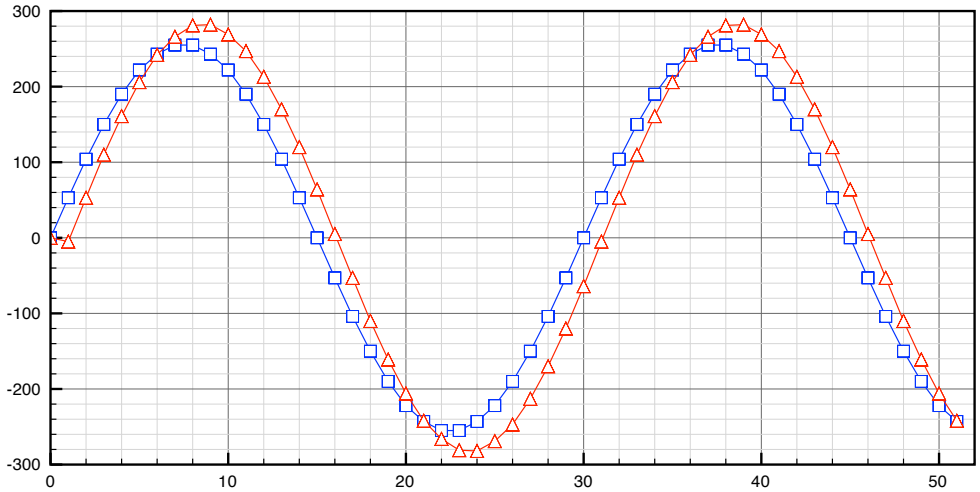
```
#include "FIR_4pts.h"

int    X[5] = {0,      0,    0,    0,    0};
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};
int FIR(int xn){
     double r = 0;
     for(int i=0; i<4; i++){
          X[i+1] = X[i];
     }X[0] = xn;
     for(int i=0; i<5; i++){
          r += X[i] * H[i];
     }
     return round( r );
}

void FIR_4pts::do_fir(){
     while( true ){
          int v = FIR( e.read() );
          s.write( v );
     }
}
```

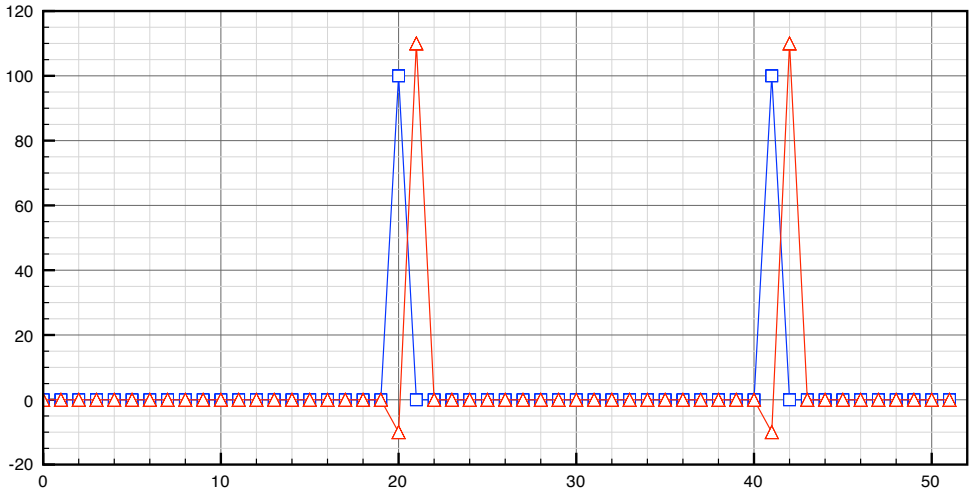
Validation du modèle «Untimed Functionnal»

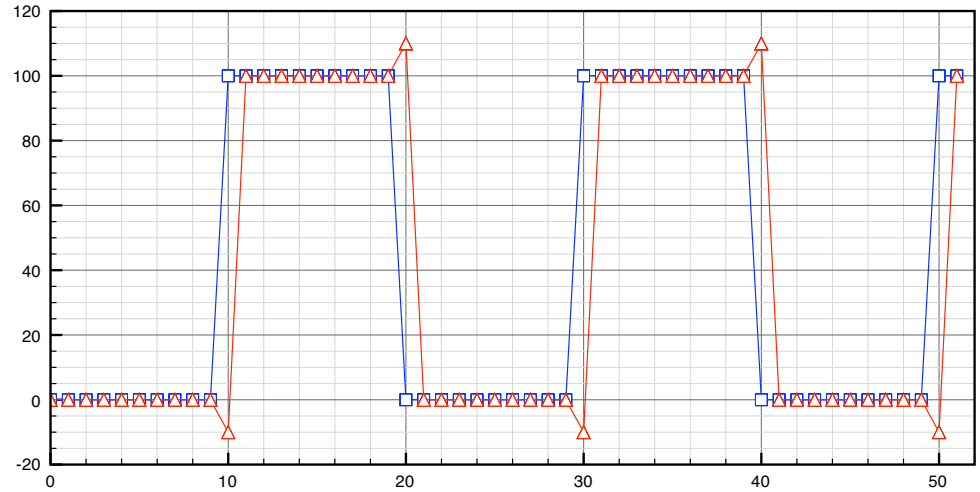


Afin de simplifier la vérification fonctionnelle, le modèle SystemC est un peu plus complexe !

Validation du modèle «Untimed Functionnal»

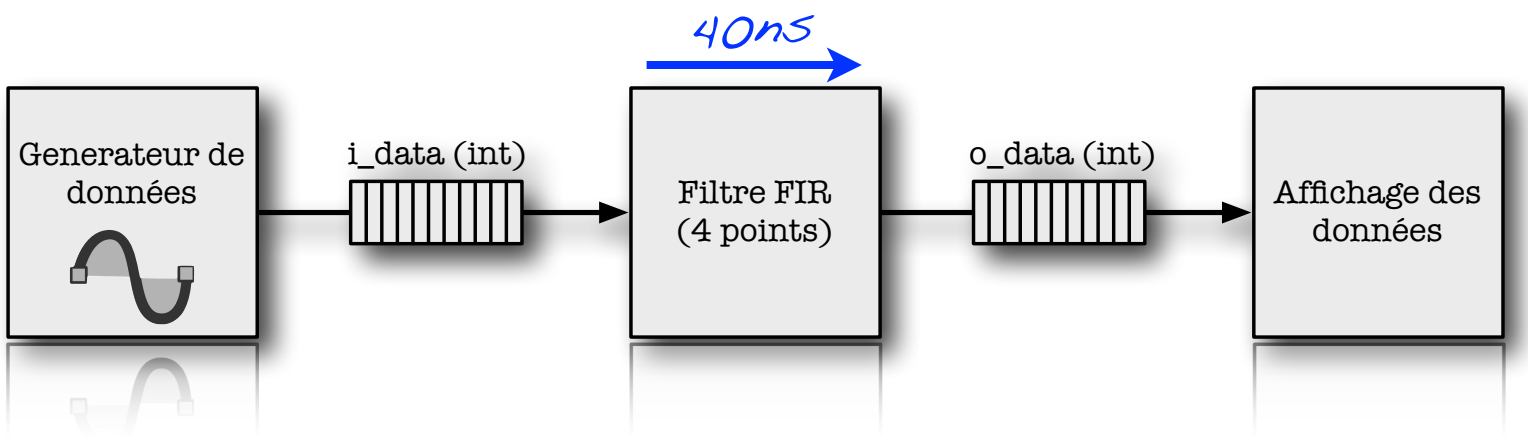






Tracé des points d'entrée et de sortie de la chaine (à l'aide du fichier de points généré).

Filtre FIR 5 points de niveau «Timed-Functionnal»



Objectifs:

- Vérification fonctionnelle
- Exploration algorithmique
- Validation gros grain des performances temporelles

Filtre FIR 5 points de niveau «Timed-Functionnal»

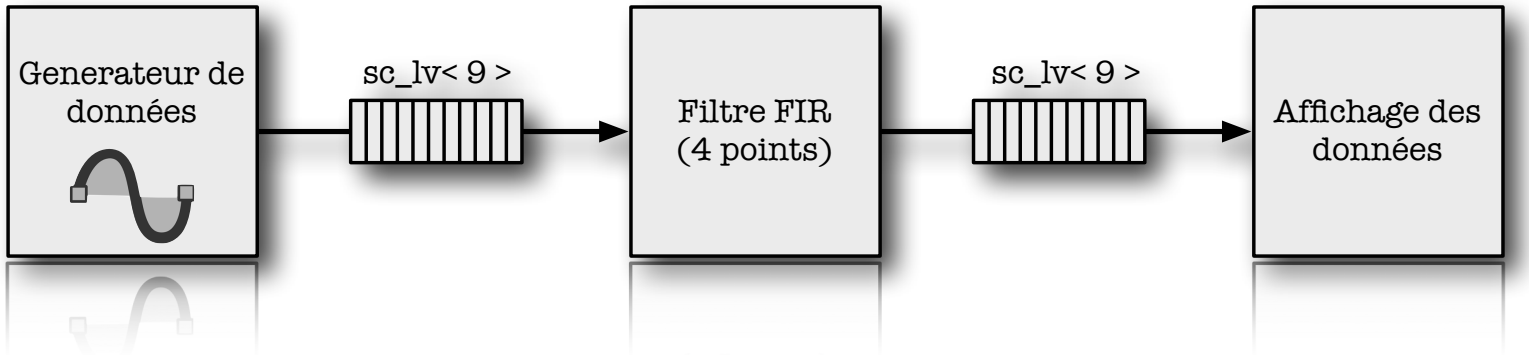
```
#include "systemc.h"

SC_MODULE(FIR_4pts)
{
public:
     sc_fifo_in <int> e;
    sc_fifo_out<int> s;

     SC_CTOR(FIR_4pts)
     {
          SC_THREAD(do_fir);
     }

    void do_fir();
 };
```

```
int    X[5] = {0,      0,    0,    0,    0};
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};
int FIR(int xn){
     double r = 0;
     for(int i=0; i<4; i++){
          X[i+1] = X[i];
     }X[0] = xn;
     for(int i=0; i<5; i++){
          r += X[i] * H[i];
     }
     return round( r );
}

void FIR_4pts::do_fir(){
     while( true ){
          int v = FIR( e.read() );
          wait(40, SC_NS);
          s.write( v );
     }
}
```

Filtre FIR 5 points de niveau «Timed-Functionnal»

```
g++ -O2 -Wall -I/Library/SystemC/systemc-2.2.0/include -m32
g++ -O2 -Wall -I/Library/SystemC/systemc-2.2.0/include -m32
./bin/main
Time = 40 ns => Valeur recue : (R=0)
Time = 80 ns => Valeur recue : (R=-5)
Time = 120 ns => Valeur recue : (R=48)
Time = 160 ns => Valeur recue : (R=99)
Time = 200 ns => Valeur recue : (R=146)
Time = 240 ns => Valeur recue : (R=187)
Time = 280 ns => Valeur recue : (R=220)
Time = 320 ns => Valeur recue : (R=242)
Time = 360 ns => Valeur recue : (R=255)
Time = 400 ns => Valeur recue : (R=256)
Time = 440 ns => Valeur recue : (R=245)
Time = 480 ns => Valeur recue : (R=225)
[Finished in 2.1s]
```

Filtre FIR 5 points de niveau «Bit-près sur les E/S»



Objectifs:

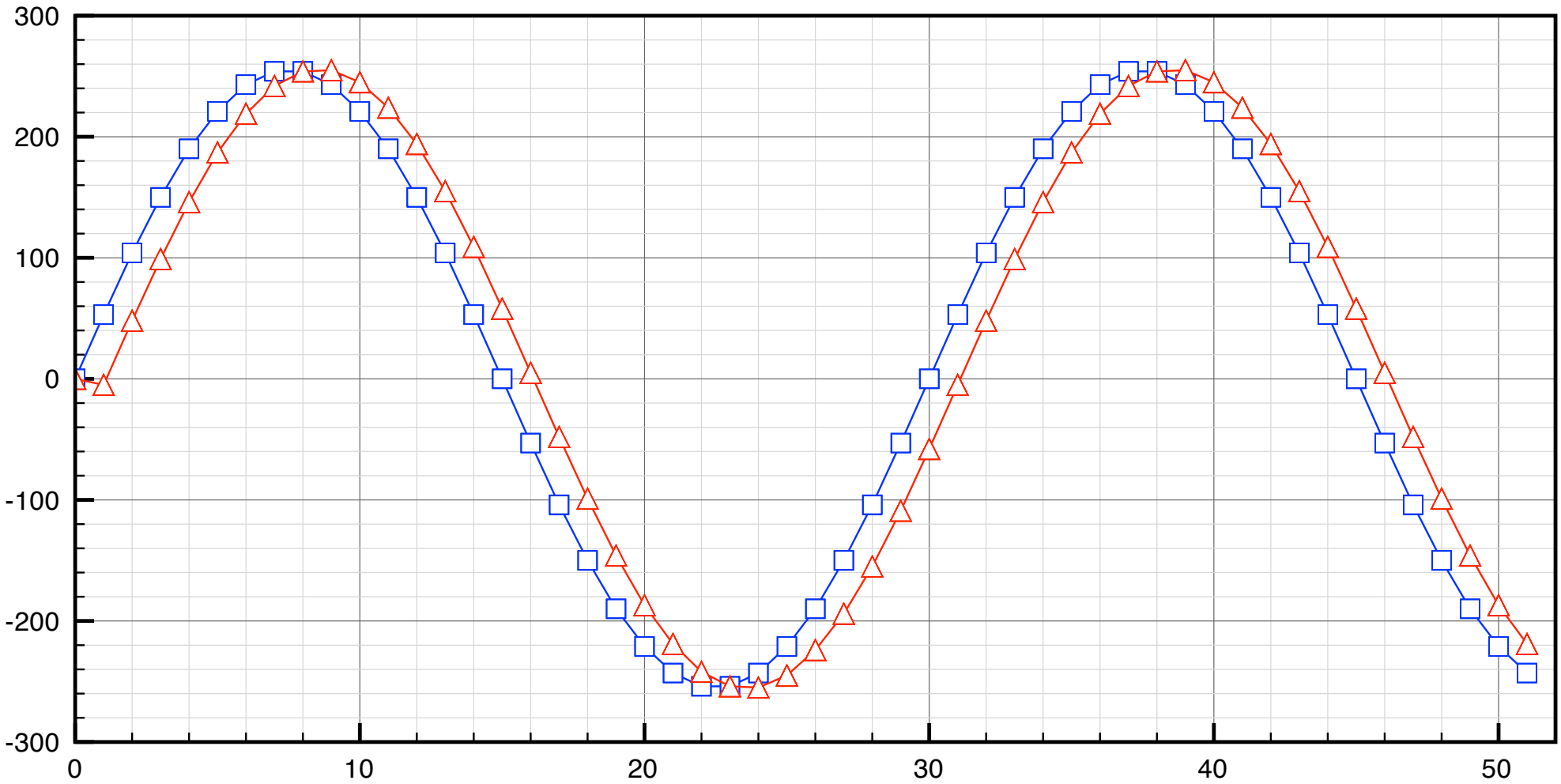
- Vérification fonctionnelle
- Spécification des E/S des modules

Filtre FIR 5 points de niveau «Bit-près sur les E/S»

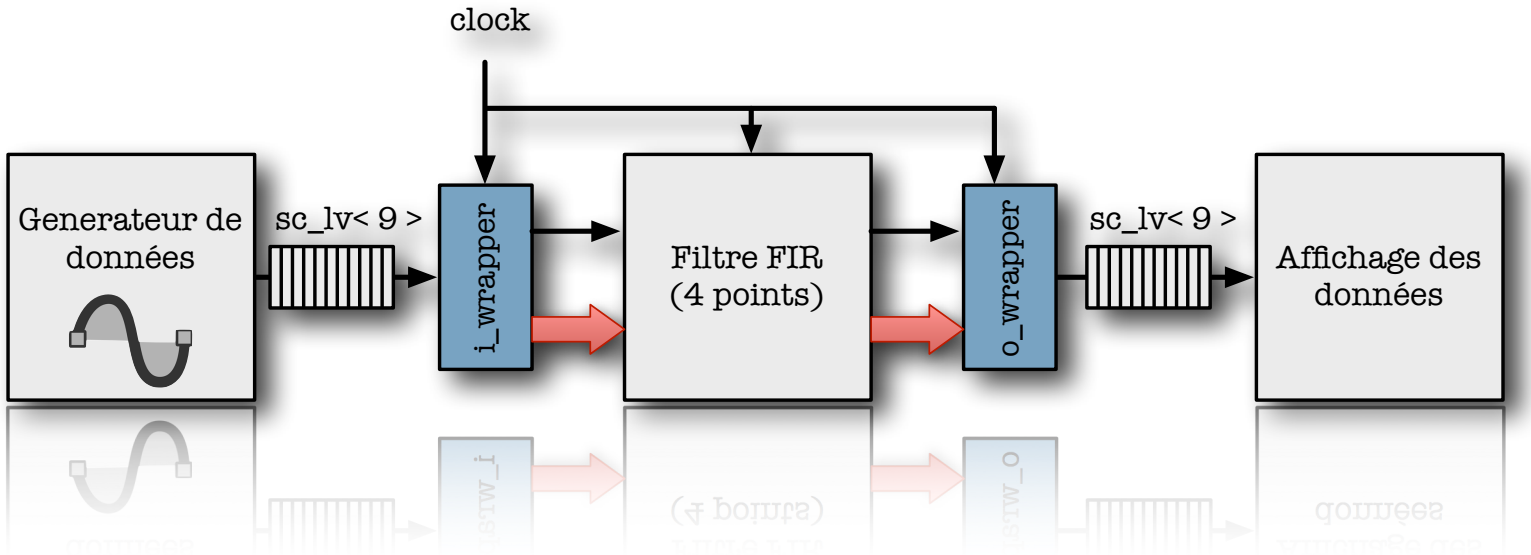
```
#include "systemc.h"

SC_MODULE(FIR_4pts)
{
public:
     sc_fifo_in < sc_lv<9> > e;
     sc_fifo_out< sc_lv<9> > s;

     SC_CTOR(FIR_4pts)
     {
          SC_THREAD(do_fir);
     }

    void do_fir();
 };
```

```
int    X[5] = {0,      0,    0,    0,    0};
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};

int FIR(int xn){
     double r = 0;
     for(int i=0; i<4; i++){
          X[i+1] = X[i];
     }X[0] = xn;
     for(int i=0; i<5; i++){
          r += X[i] * H[i];
     }
     return round( r );
}

void FIR_4pts::do_fir(){
     while( true ){
          int v = FIR( (sc_int<9>)e.read() );
          wait(40, SC_NS);
          s.write( (sc_int<9>)v );
     }
}
```

Filtre FIR 5 points de niveau «Bit-près sur les E/S»



Filtre FIR 5 points de niveau «Bus Cycle Accurate»



Objectifs:

- *Vérification fonctionnelle*
- *Spécification du comportement au cycle près des E/S*

  + permet le développement parallèle

Filtre FIR 5 points de niveau «Bus Cycle Accurate»

```
#include "systemc.h"

SC_MODULE(FIR_4pts)
{
public:
     sc_in       <bool>          clk;
     sc_in       < sc_lv<9> >    i_data;
     sc_out      < sc_lv<9> >    o_data;
     sc_in       <bool>          i_valid;
     sc_out      <bool>          o_valid;

     SC_CTOR(FIR_4pts)
     {
          SC_THREAD(do_fir);
          sensitive << clk.pos();
     }

    void do_fir();
 };
```

Filtre FIR 5 points de niveau «Bus Cycle Accurate»

```
int    X[5] = {0,      0,    0,    0,    0};
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};
int FIR(int xn){
     double r = 0;
     for(int i=0; i<4; i++){
          X[i+1] = X[i];
     }X[0] = xn;
     for(int i=0; i<5; i++){
          r += X[i] * H[i];
     }
     return round( r );
}

void FIR_4pts::do_fir(){
     while( true ){
          if( i_valid.read() == 1 ){
               int v = FIR( (sc_int<9>)i_data.read() );
               wait( ); o_valid.write( 0 ); o_data.write ( 0 );
               wait( ); o_valid.write( 0 ); o_data.write ( 0 );
               wait( ); o_valid.write( 1 ); o_data.write ( (sc_int<9>)v );
          }else{
               wait( ); o_data.write ( 0 ); o_valid.write( 0 );
          }
     }
}
```

Filtre FIR 5 points de niveau «Bus Cycle Accurate»

```
SC_MODULE(i_wrapper)
{
public:
     sc_in       <bool>        clk;
     sc_fifo_in < sc_lv<9> > e;
     sc_out      < sc_lv<9> > o_data;
     sc_out      <bool>        o_valid;

     SC_CTOR(i_wrapper)
     {
          SC_CTHREAD(do_conversion, clk.pos());
     }

    void do_conversion();
 };
```

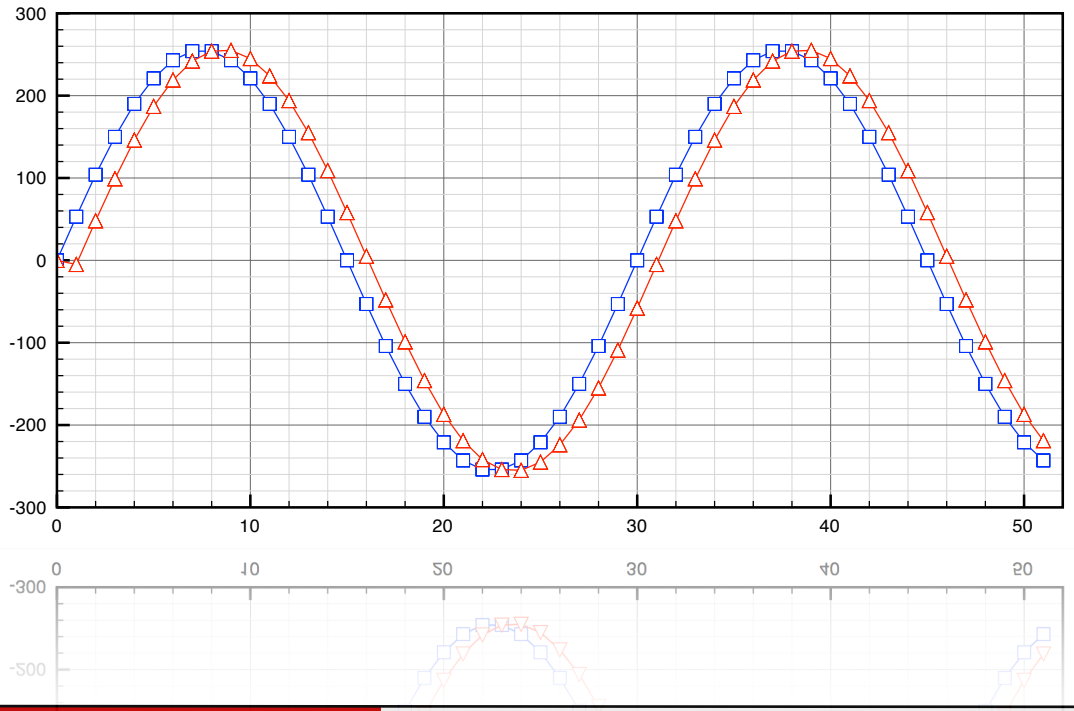
```
void i_wrapper::do_conversion(){
     while( true ){
          wait(); // ON ATTEND UN FRONT D'HORLOGE
          if( e.num_available() != 0 ){
              o_data.write ( e.read() );
              o_valid.write(   1 );
              wait();              // ON ATTEND UN CYCLE D'HORLOGE AVANT DE DEVALIDER
              o_data.write ( 0 ); // LE SIGNAL DATA_VALID. PUIS ON ATTEND ENCORE UN
              o_valid.write( 0 ); // CYCLE AVANT DE CONTINUER LE PROCESSUS.
              wait();              //
              wait();              //
          }
     }
}
```

Filtre FIR 5 points de niveau «Bus Cycle Accurate»

```
#include "systemc.h"

SC_MODULE(o_wrapper)
{
public:
     sc_in       <bool>        clk;
     sc_in       <bool>        i_valid;
     sc_in       < sc_lv<9> >  i_data;
     sc_fifo_out< sc_lv<9> >   s;

     SC_CTOR(o_wrapper)
     {
          SC_CTHREAD(do_conversion, clk.pos());
     }

    void do_conversion();
 };
```

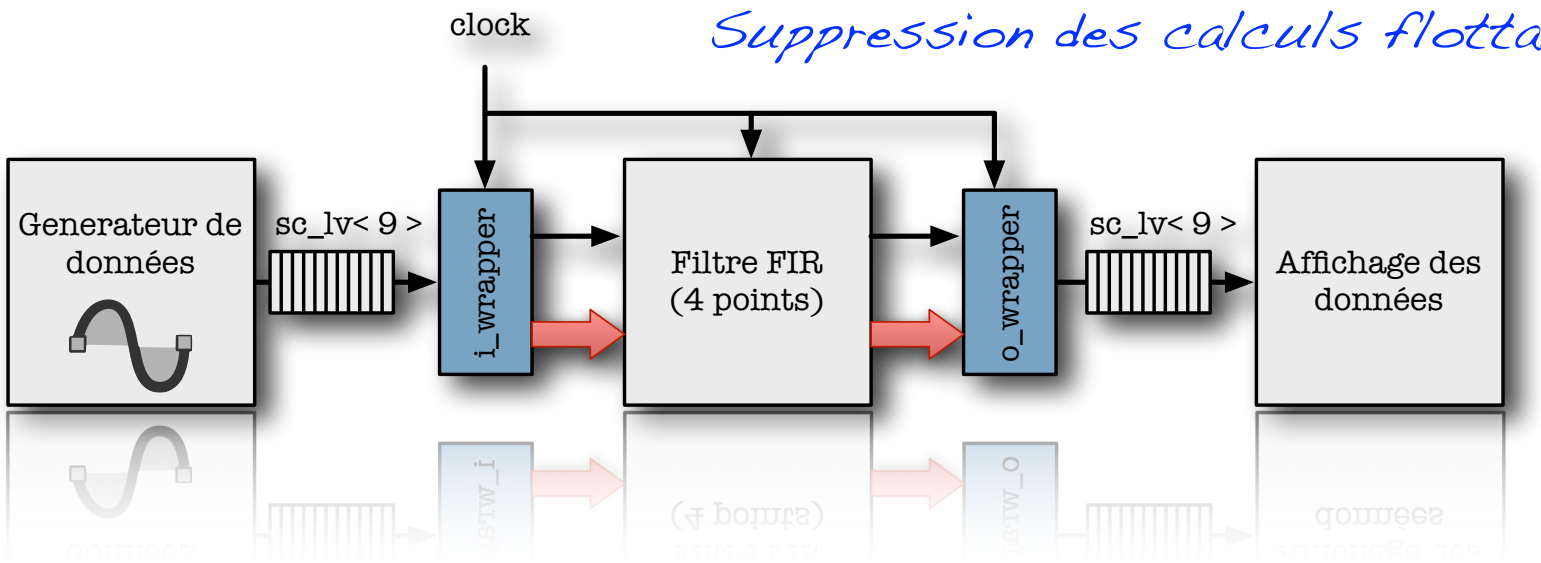
```
#include "o_wrapper.h"

void o_wrapper::do_conversion(){
     while( true ){
          wait();
          if( i_valid.read() == 1 ){
               s.write( i_data.read() );
          }
     }
}
```

Filtre FIR 5 points de niveau «Bus Cycle Accurate»



Validation des chronogrammes d'E/S ainsi que des résultats fonctionnels

Filtre FIR 5 points de niveau «Bit-près sur les calculs»



Objectifs:

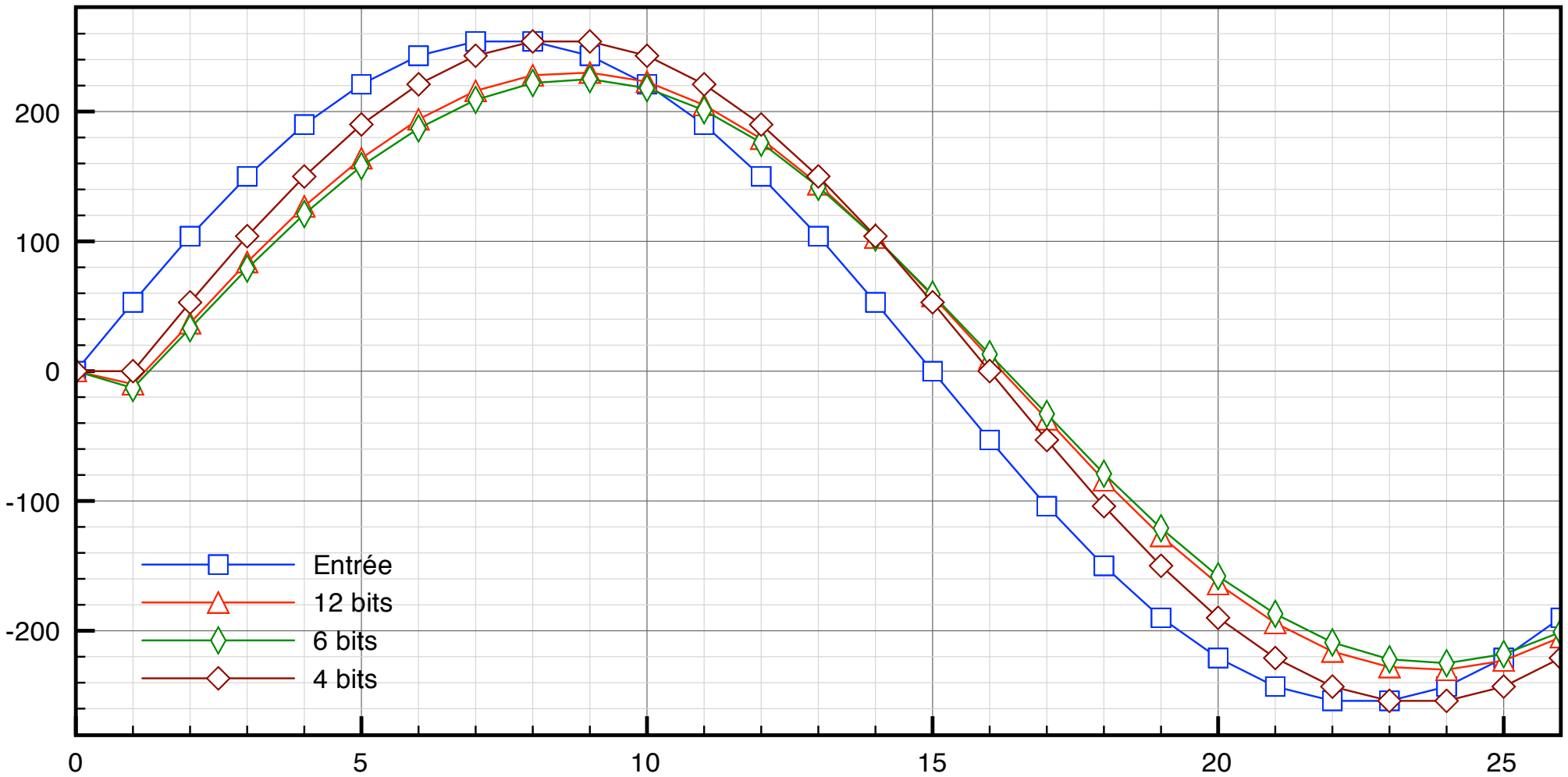
- Vérification fonctionnelle
- Spécification du format de codage
- Etude de performance (signal)

Filtre FIR 5 points de niveau «Bit-près sur les calculs»

```
#define N 6
int    X[5] = {0,      0,    0,    0,    0};
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};
sc_fixed <   N,  2, SC_RND, SC_SAT> H0 = H[0];
sc_fixed <   N,  2, SC_RND, SC_SAT> H1 = H[1];
sc_fixed <   N,  2, SC_RND, SC_SAT> H2 = H[2];
sc_fixed <   N,  2, SC_RND, SC_SAT> H3 = H[3];
sc_fixed <   N,  2, SC_RND, SC_SAT> H4 = H[4];
sc_fixed <N+10, 10, SC_RND, SC_SAT> R;

int FIR(int xn){
     for(int i=0; i<4; i++){
          X[i+1] = X[i];
     }X[0] = xn;
     R  = H0 * X[0];
     R += H1 * X[1];
     R += H2 * X[2];
     R += H3 * X[3];
     R += H4 * X[4];
     return R;
}
```

```
void FIR_4pts::do_fir(){
     while( true ){
          if( i_valid.read() == 1 ){
               int v = FIR( (sc_int<9>)i_data.read() );
               wait( ); o_valid.write( 0 ); o_data.write ( 0 );
               wait( ); o_valid.write( 0 ); o_data.write ( 0 );
               wait( ); o_valid.write( 1 ); o_data.write ( (sc_int<9>)v );
          }else{
               wait( ); o_data.write ( 0 ); o_valid.write( 0 );
          }
     }
}
```

Filtre FIR 5 points de niveau «Bit-près sur les calculs»

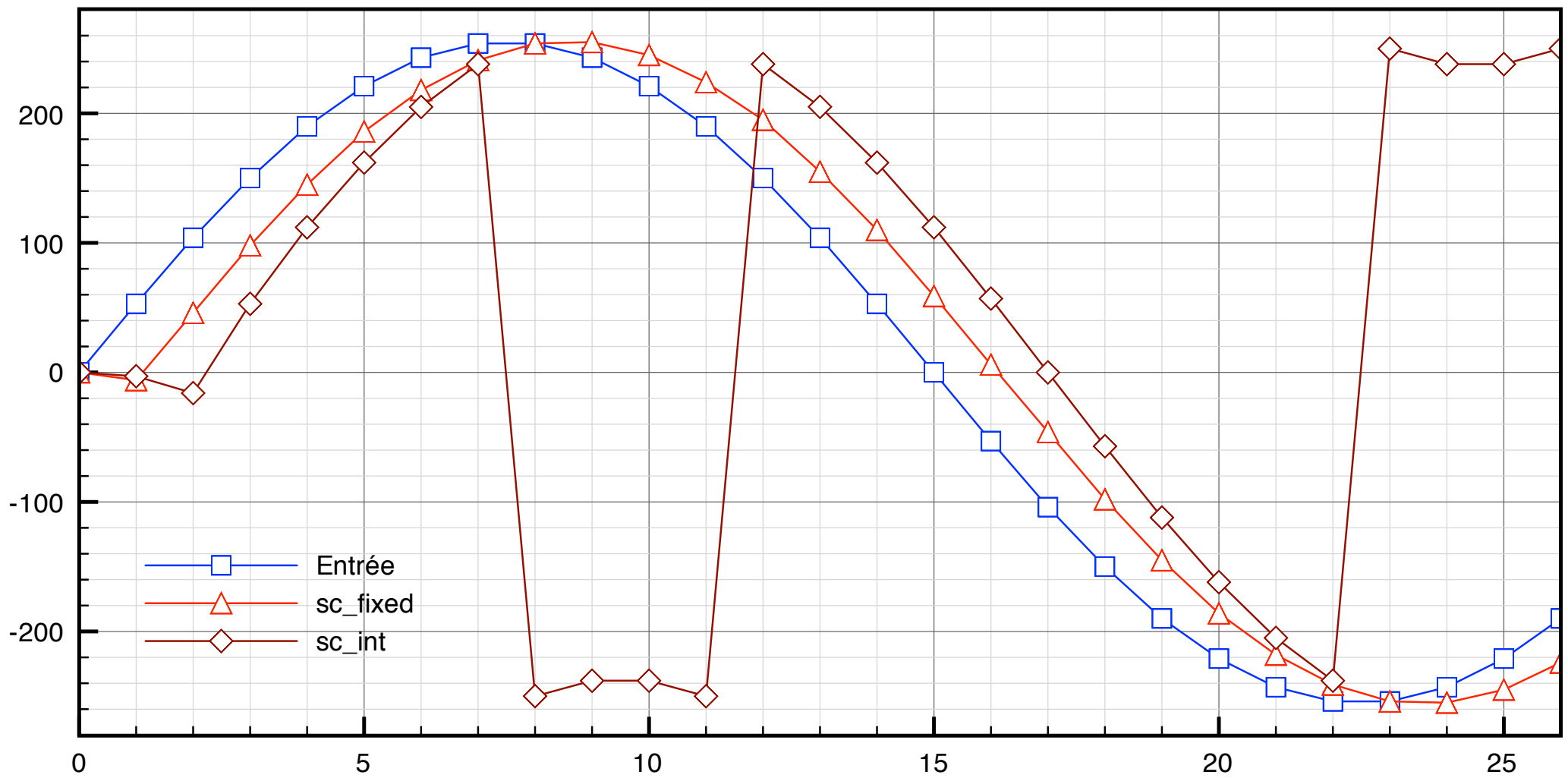


Filtre FIR 5 points de niveau «Bit-près sur les calculs»

```
#define N 6
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};
sc_int<N> H0 = (int)round((H[0] * (1 << (N-2))));
sc_int<N> H1 = (int)round((H[1] * (1 << (N-2))));
sc_int<N> H2 = (int)round((H[2] * (1 << (N-2))));
sc_int<N> H3 = (int)round((H[3] * (1 << (N-2))));
sc_int<N> H4 = (int)round((H[4] * (1 << (N-2))));
sc_int<9> X0 = 0;
sc_int<9> X1 = 0;
sc_int<9> X2 = 0;
sc_int<9> X3 = 0;
sc_int<9> X4 = 0;

int FIR(int xn){
     sc_int<16> R;
     X0 = X1; X1 = X2;
     X2 = X3; X3 = X4;
     X4 = xn;
     R  = (H0 * X0) + (H1 * X1);
     R += (H2 * X2) + (H3 * X3);
     R += (H4 * X4);
     R  =  R / (1 << (N-2));
     return R;
}
```

```
void FIR_4pts::do_fir(){
     while( true ){
          if( i_valid.read() == 1 ){
               sc_int<9> e = i_data.read();
               sc_int<9> v = FIR( e );
               wait( ); o_valid.write( 0 ); o_data.write ( 0 );
               wait( ); o_valid.write( 0 ); o_data.write ( 0 );
               wait( ); o_valid.write( 1 ); o_data.write ( (sc_int<9>)v );
          }else{
               wait( ); o_data.write ( 0 ); o_valid.write( 0 );
          }
     }
}
```

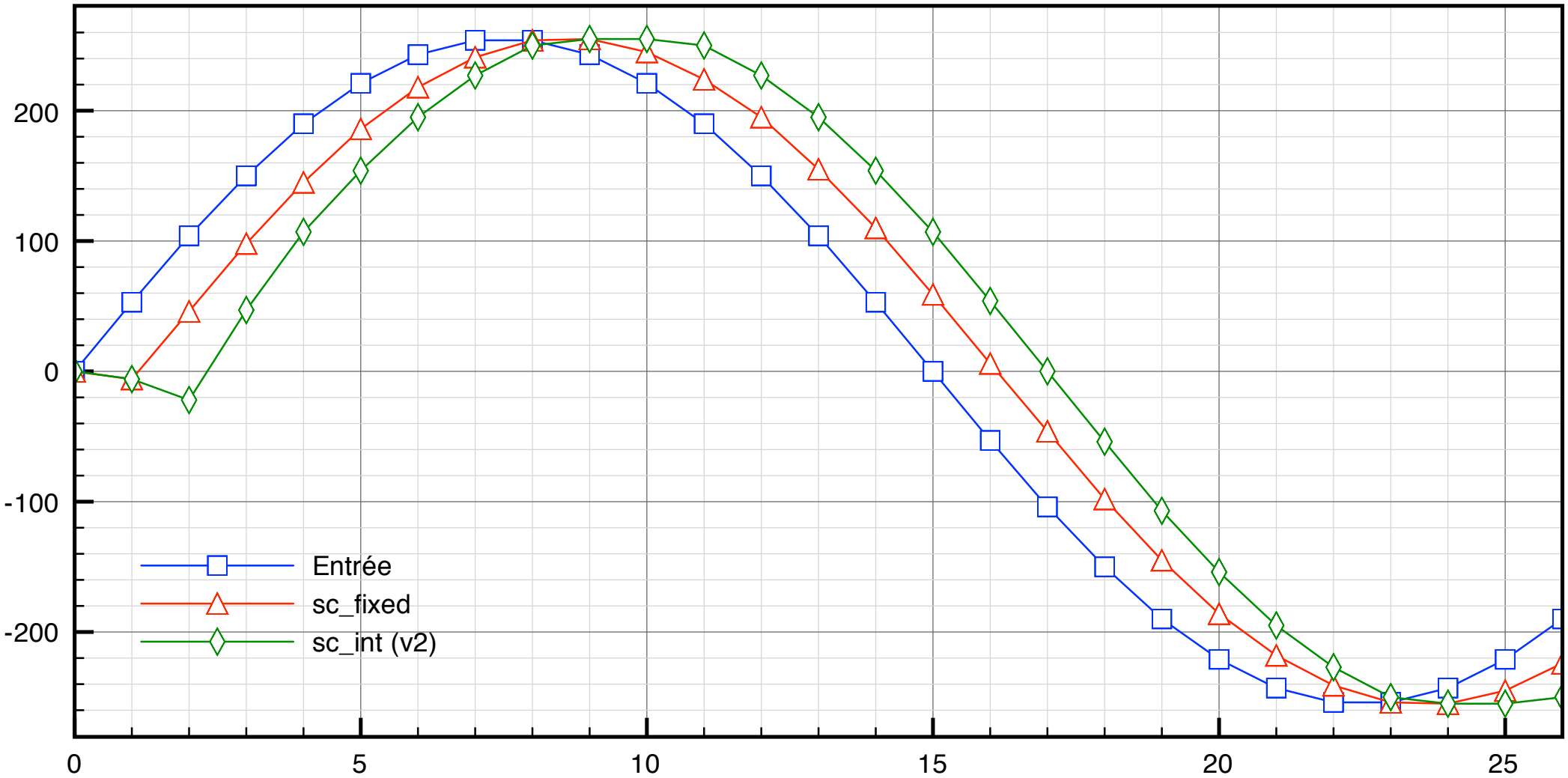
Validation fonctionnelle du modèle bit-près (comportement)



Filtre FIR 5 points de niveau «Bit-près sur les calculs»

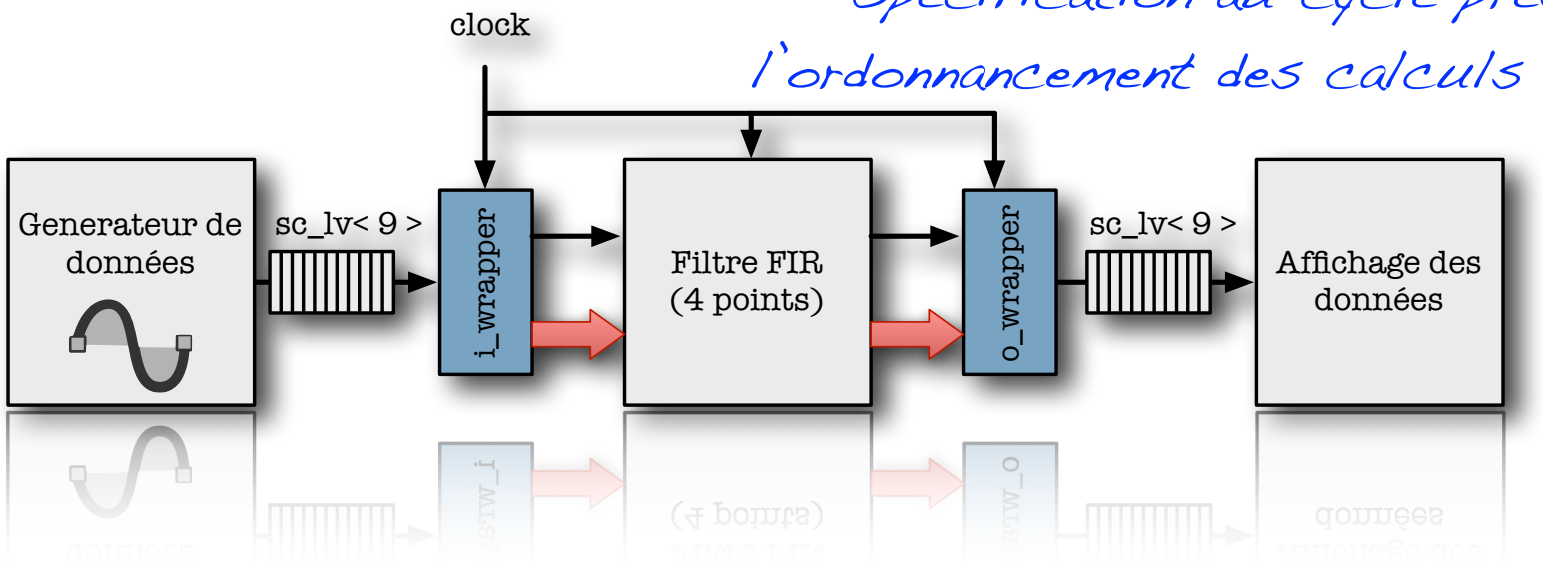
```
#define N 6
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};
sc_int<N> H0 = (int)round((H[0] * (1 << (N-2))));
sc_int<N> H1 = (int)round((H[1] * (1 << (N-2))));
sc_int<N> H2 = (int)round((H[2] * (1 << (N-2))));
sc_int<N> H3 = (int)round((H[3] * (1 << (N-2))));
sc_int<N> H4 = (int)round((H[4] * (1 << (N-2))));
sc_int<9> X0 = 0;
sc_int<9> X1 = 0;
sc_int<9> X2 = 0;
sc_int<9> X3 = 0;
sc_int<9> X4 = 0;

int FIR(int xn){
     sc_int<16> R;
     X0 = X1; X1 = X2;
     X2 = X3; X3 = X4;
     X4 = xn;
     R  = (H0 * X0) + (H1 * X1);
     R += (H2 * X2) + (H3 * X3);
     R += (H4 * X4);
     R  =  R / (1 << (N-2));
     R  = (R < -255) ? ((sc_int<16>)-255) : R;
     R  = (R >  255) ? ((sc_int<16>) 255) : R;
     return R;
}
```

Validation fonctionnelle du modèle bit-près (comportement)



Filtre FIR 5 points de niveau «Cycle Accurate»

Spécification au cycle près de l'ordonnancement des calculs internes



Objectifs:

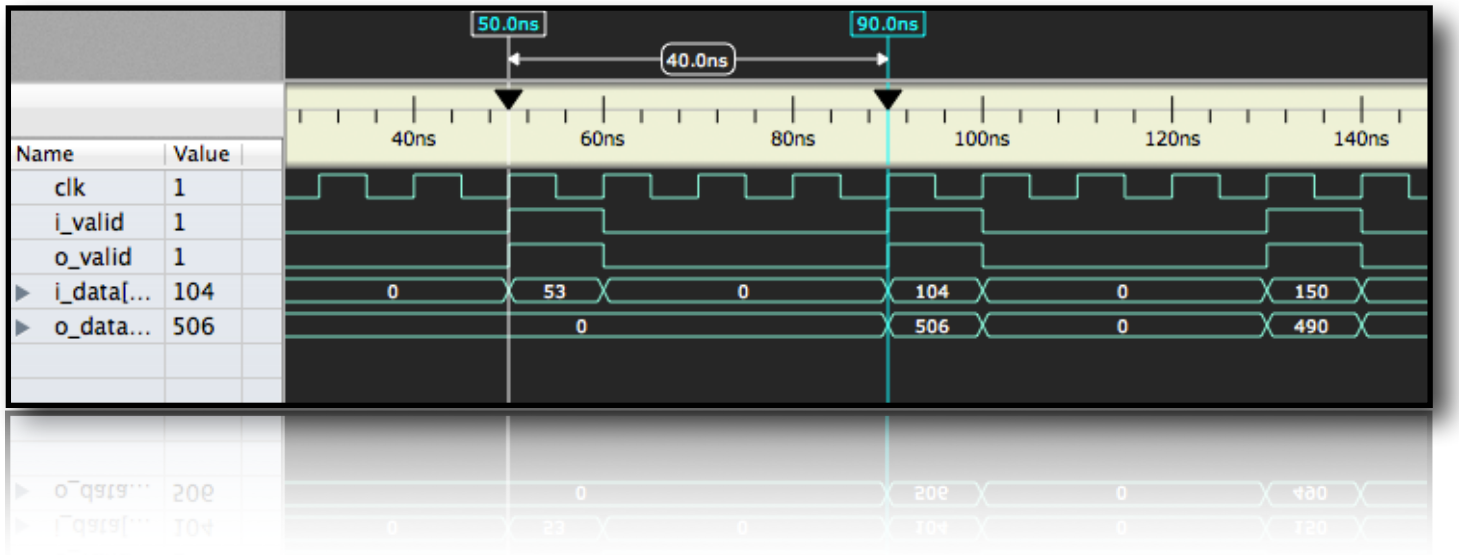
- Vérification fonctionnelle
- Spécifier la répartition des calculs (cycles)
- Niveau presque RTL (conversion VHDL)

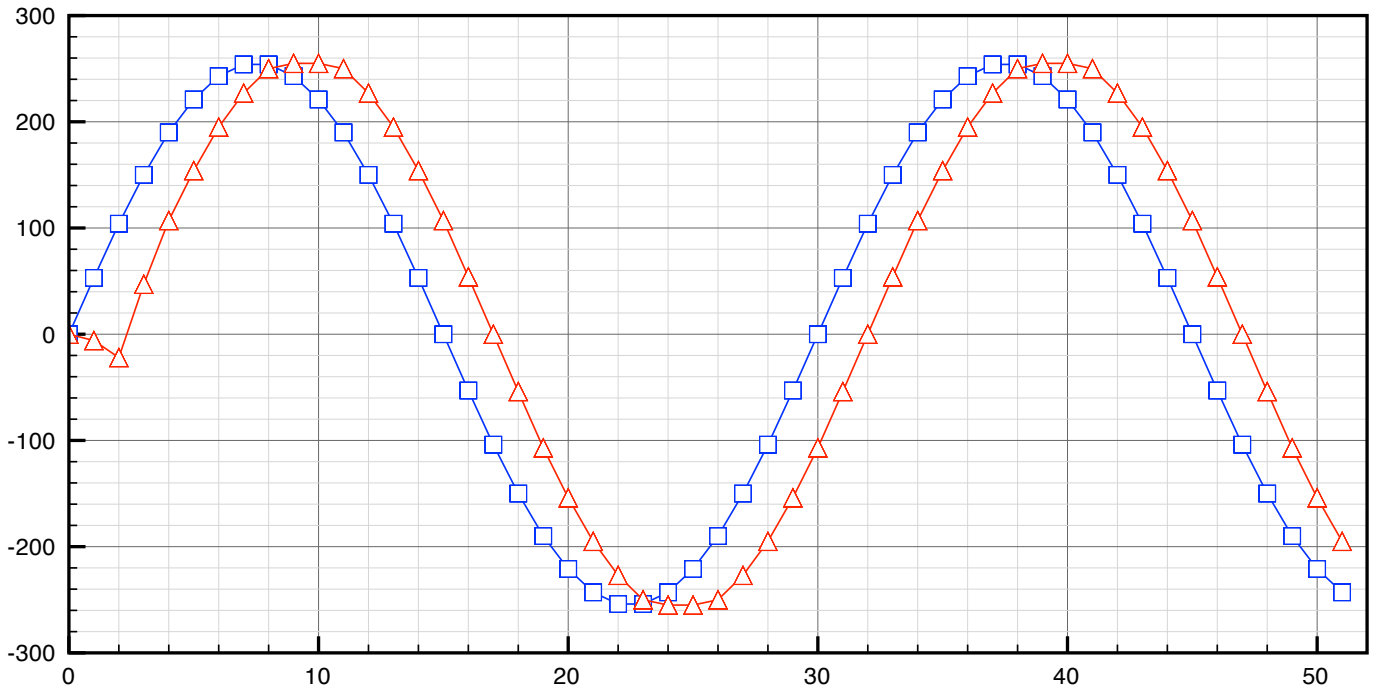
Filtre FIR 5 points de niveau «Cycle Accurate»

```
double H[5] = {-0.1, -0.2, 1.6, -0.2, -0.1};
sc_int<N> H0 = (int)round((H[0] * (1 << (N-2))));
sc_int<N> H1 = (int)round((H[1] * (1 << (N-2))));
sc_int<N> H2 = (int)round((H[2] * (1 << (N-2))));
sc_int<N> H3 = (int)round((H[3] * (1 << (N-2))));
sc_int<N> H4 = (int)round((H[4] * (1 << (N-2))));
sc_int<9> X0 = 0;
sc_int<9> X1 = 0;
sc_int<9> X2 = 0;
sc_int<9> X3 = 0;
sc_int<9> X4 = 0;
```

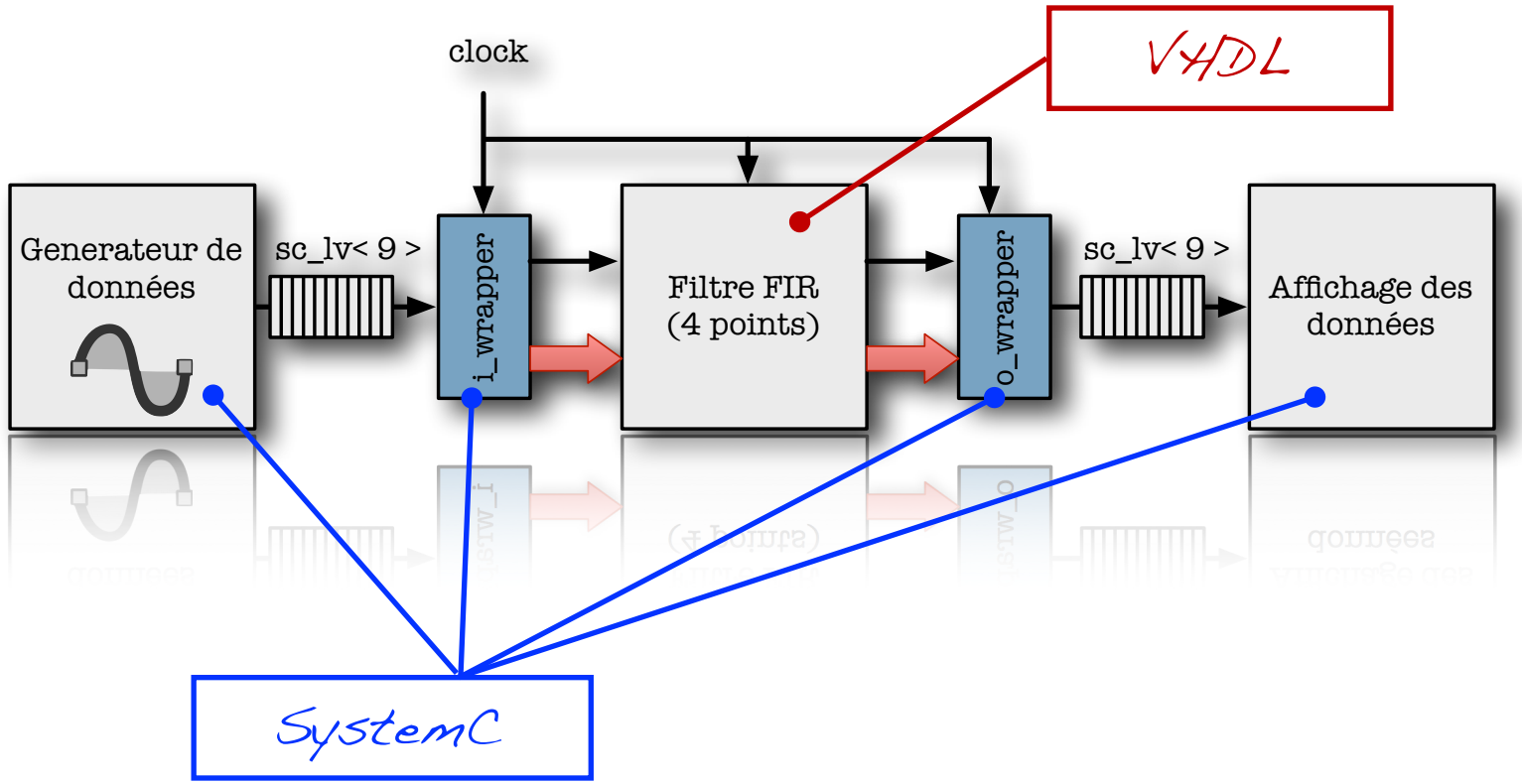
```
void FIR_4pts::do_fir(){
     sc_int<16> R;
     while( true ){
/*C1*/    if( i_valid.read() == 1 ){
/*C1*/         X4 = i_data.read();
/*C1*/        wait( );
/*C2*/        o_valid.write( 0 ); o_data.write ( 0 );
/*C2*/         R  = (H0 * X0) + (H1 * X1);
/*C2*/         R += (H2 * X2) + (H3 * X3);
/*C2*/        wait( );
/*C3*/        o_valid.write( 0 ); o_data.write ( 0 );
/*C3*/         R += (H4 * X4);
/*C3*/         R  =  R / (1 << (N-2));
/*C3*/         R  = (R < -255) ? ((sc_int<16>)-255) : R;
/*C3*/         R  = (R >  255) ? ((sc_int<16>) 255) : R;
/*C3*/        wait( );
/*C4*/        o_valid.write( 1 ); o_data.write ( R );
/*C4*/         X0 = X1; X1 = X2;
/*C4*/         X2 = X3; X3 = X4;
          }else{
               wait( ); o_data.write ( 0 ); o_valid.write( 0 );
          }
     }
}
```

Validation du comportement au niveau «Cycle Accurate»





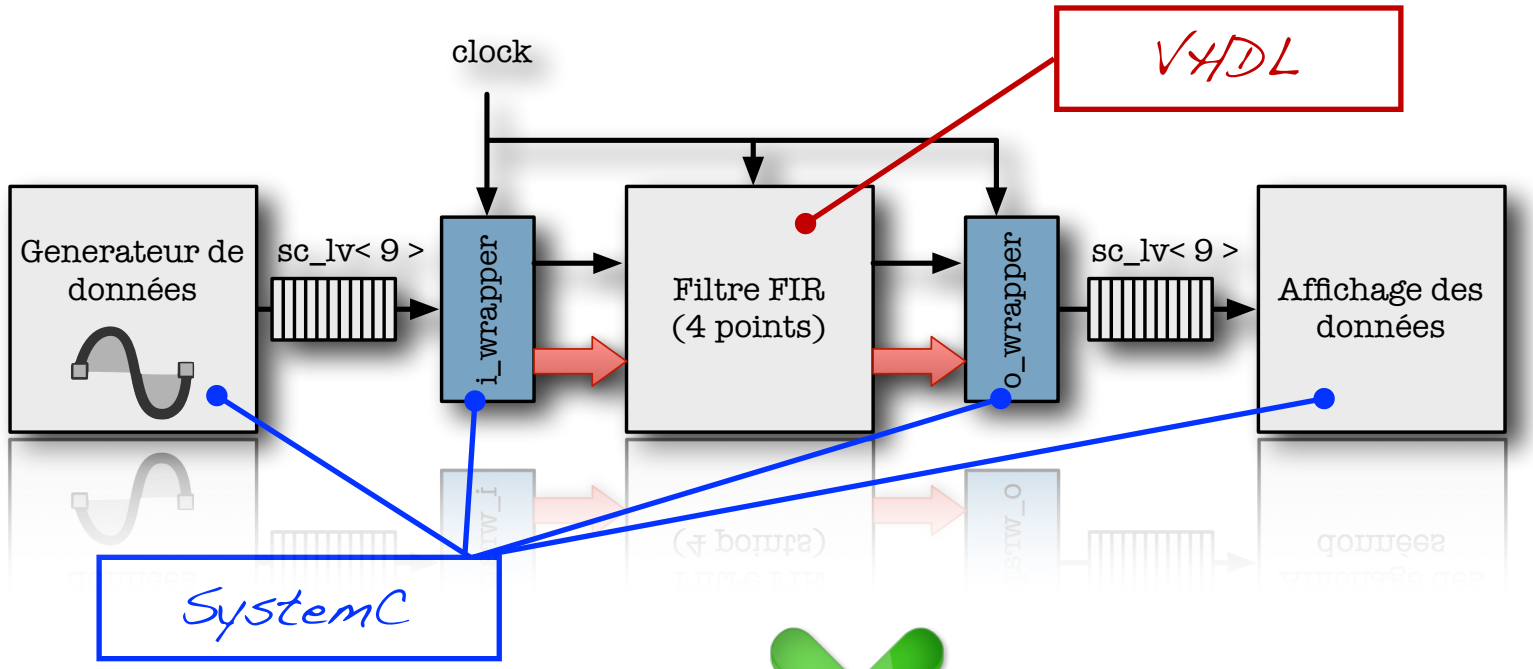
Filtre FIR 5 points décrit en VHDL de niveau RTL

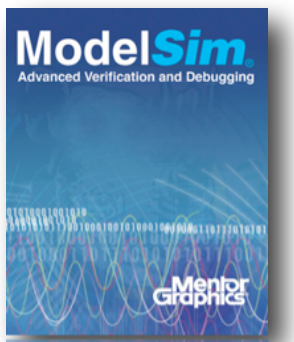


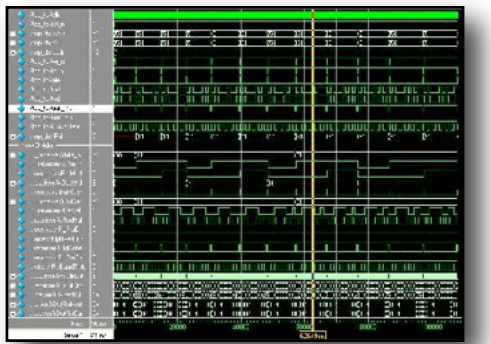
Objectifs:

- Implantation VHDL du filtre en fonction des spécification SystemC Cycle Accurate

La simulation conjointe SystemC <=> VHDL

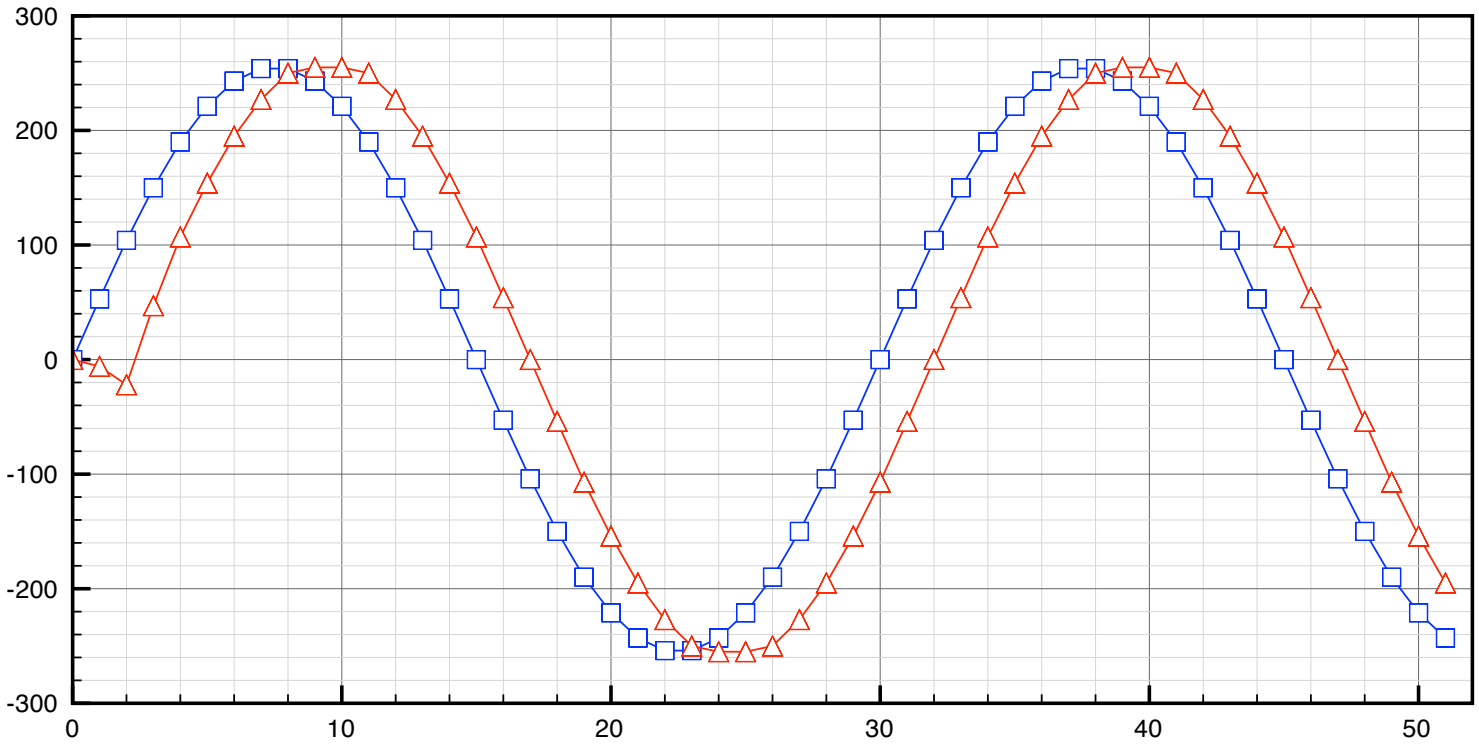






Vérification fonctionnelle de l'architecture

Validation du comportement post-simulation conjointe

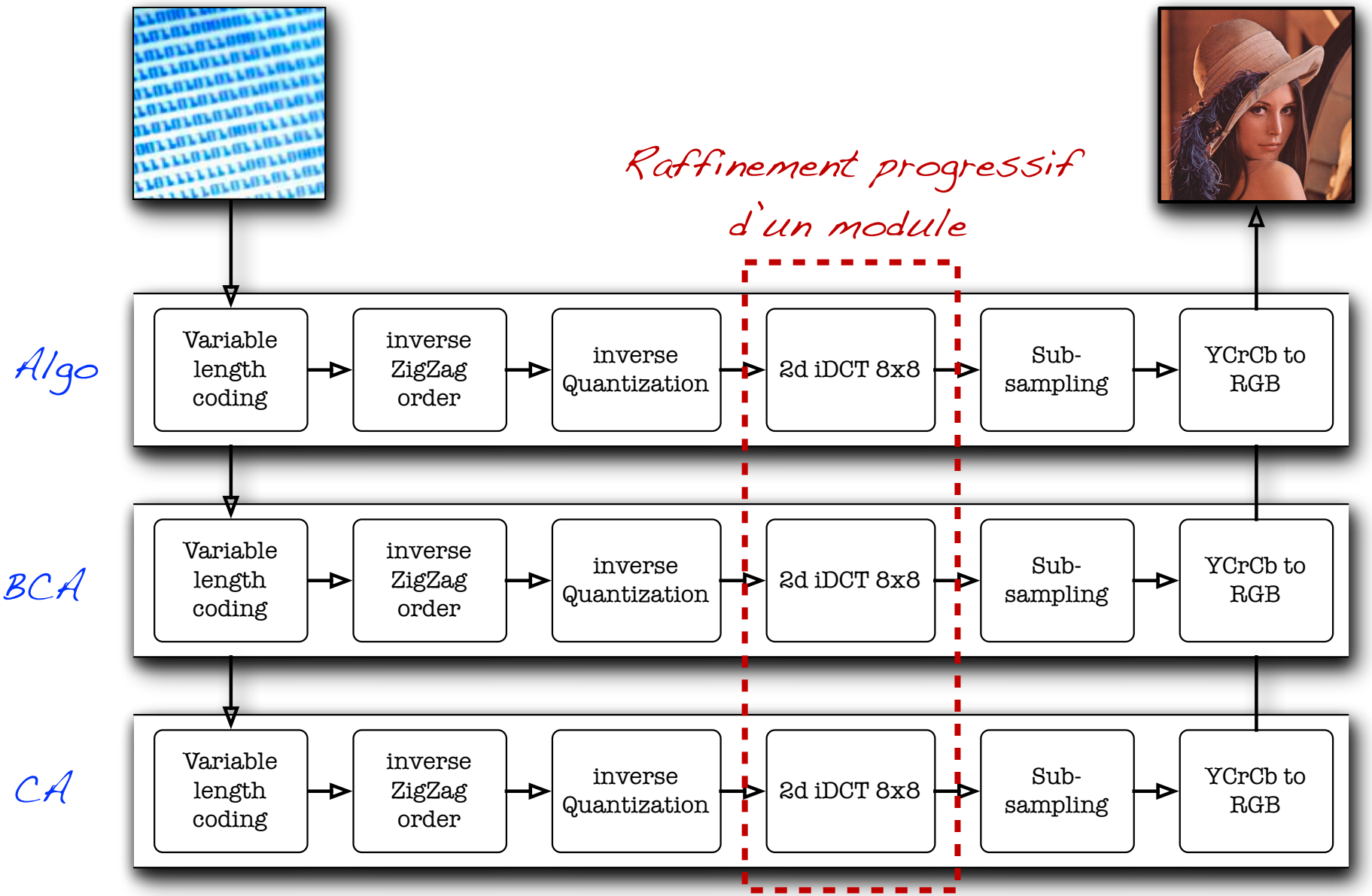


Un seul et unique modèle !

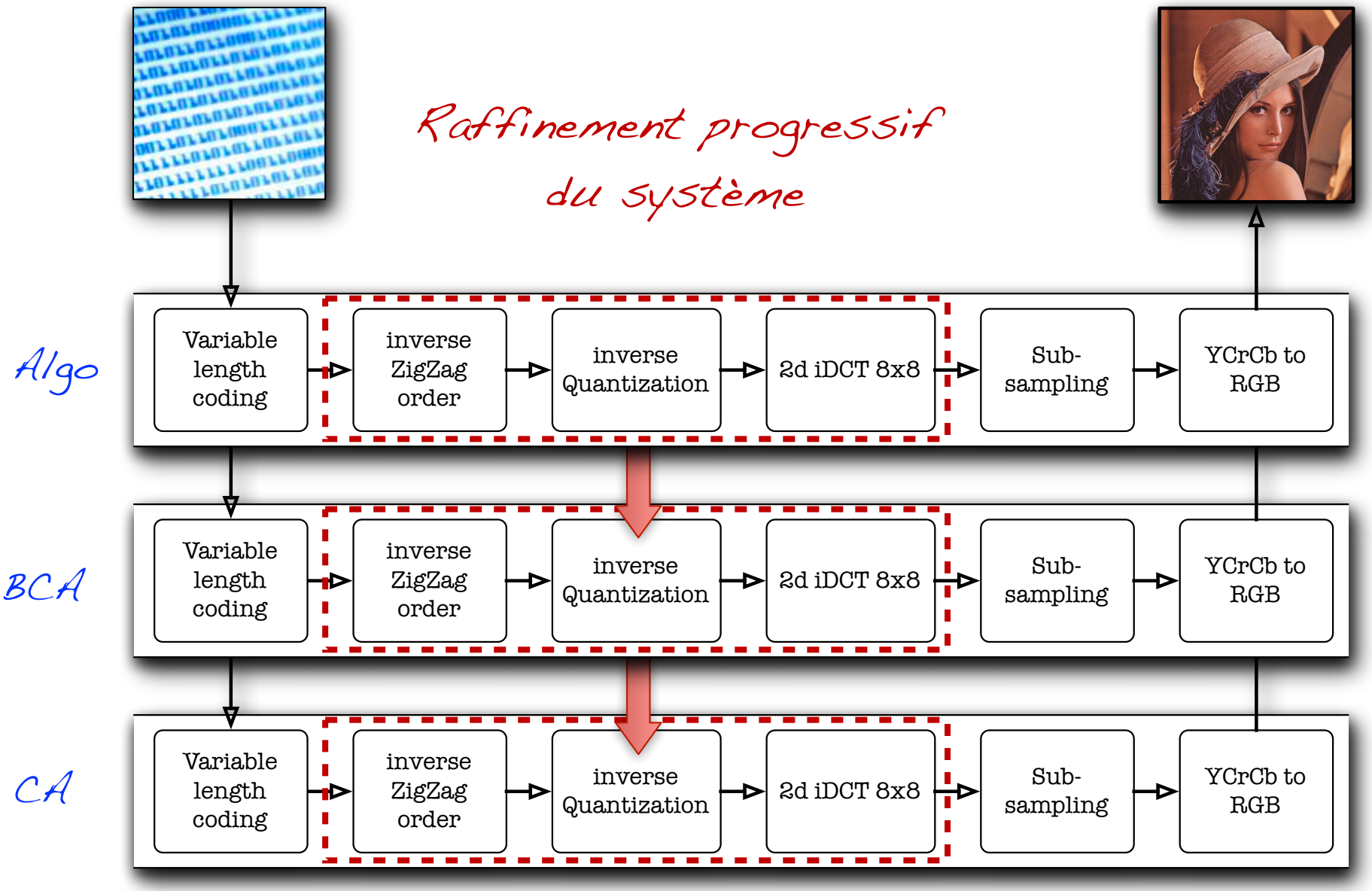
- Memes entrées, et donc mêmes sorties

- Gain en temps & en fiabilité

Généralisation de l'approche présentée (séquentielle)



Généralisation de l'approche présentée (parallèle)

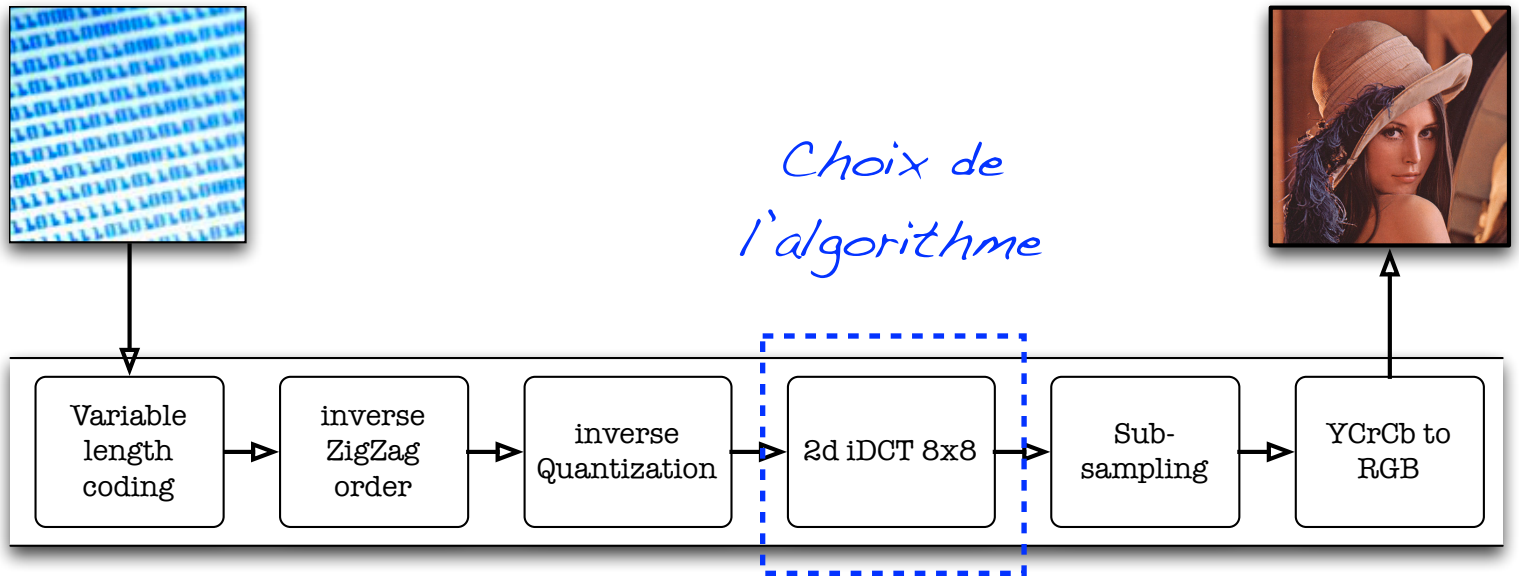


Conclusion sur l'exemple pédagogique

- L'ensemble des étapes de raffinement ne sont pas obligatoires, cela dépend grandement de votre système,
  - Certaines étapes de raffinement peuvent être réalisées en même temps (types des données sur les bus + ordre de production et de consommation des données),
  - Une fois les modèles fonctionnels établis, vous achèterez peut être une implantation particulière afin de répondre à vos besoins => vous obtiendrez une description cycle près,
  - Certains composants vont imposer des contraintes sur leurs voisins (dynamique des données, type des interfaces, débit et ordre des transferts de données, etc.),
- L'objectif n'est pas toujours d'aller vite mais de s'assurer qu'a chaque prise de décision, le système reste fonctionnel et répond aux contraintes !

Partie 3
«Etude de performance»

Etude des performances - Introduction



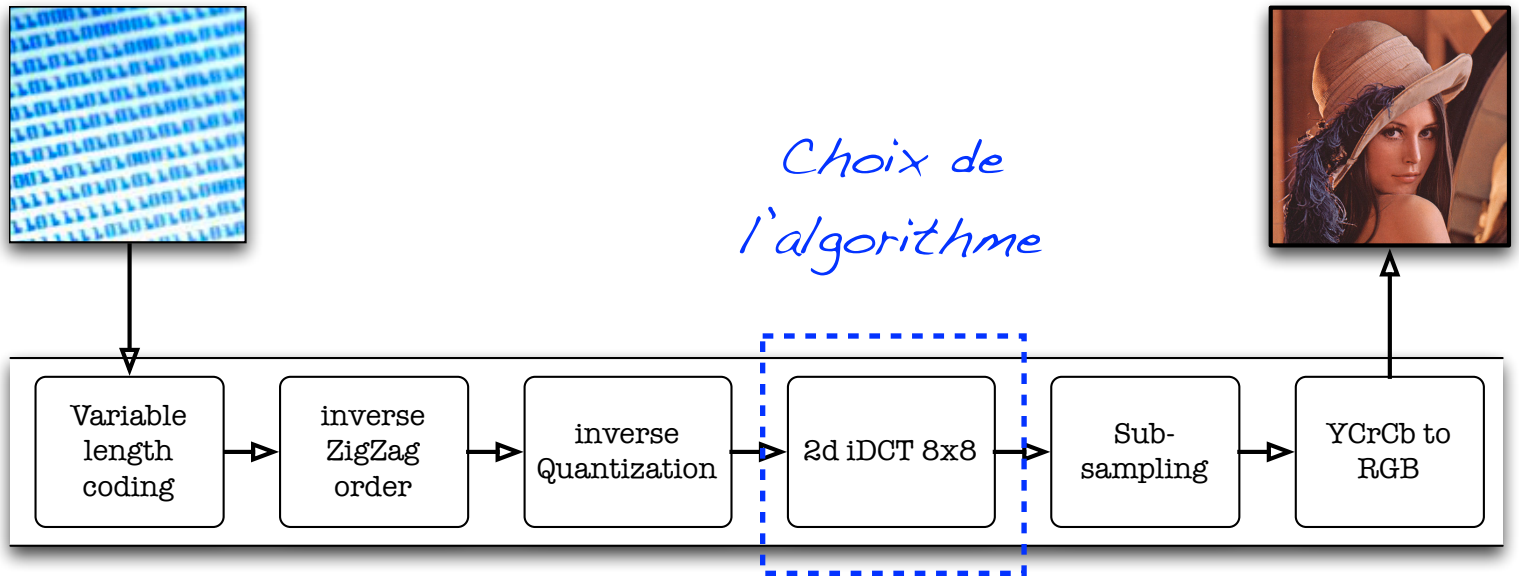
$$X_k = \frac{1}{2}x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N} n \left(k + \frac{1}{2}\right)\right]$$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{bmatrix} C_4 & C_2 & C_4 & C_6 \\ C_4 & C_6 & -C_4 & -C_2 \\ C_4 & -C_6 & -C_4 & C_2 \\ C_4 & -C_2 & C_4 & -C_6 \end{bmatrix} \cdot \begin{pmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{pmatrix} + \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & -C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \cdot \begin{pmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{pmatrix}$$

$$\begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \end{pmatrix} = \begin{bmatrix} C_4 & C_2 & C_4 & C_6 \\ C_4 & C_6 & -C_4 & -C_2 \\ C_4 & -C_6 & -C_4 & C_2 \\ C_4 & -C_2 & C_4 & -C_6 \end{bmatrix} \cdot \begin{pmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{pmatrix} - \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & -C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \cdot \begin{pmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{pmatrix}$$

http://fr.wikipedia.org/wiki/Transformée_en_cosinus_discrète

Etude des performances - Choix algorithmiques

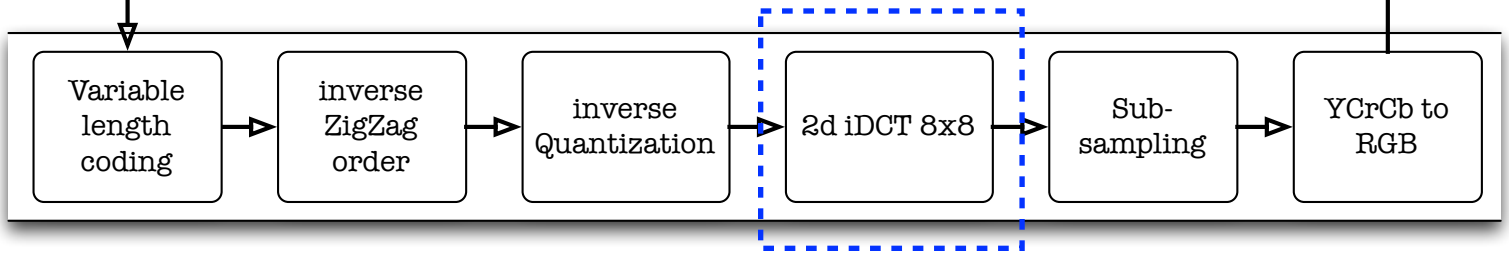


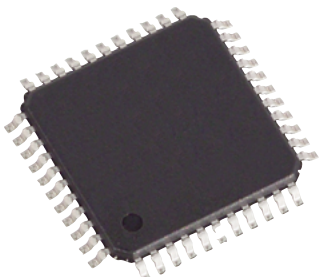
Algorithm	Multiplications	Additions
Original DCT-SQ [3]	5 (10-bit)	29 (10-bit)
Chen's DCT [4]	16 (16-bit)	28 (16-bit)
Distributed DCT [5]	0	42 (12-bit)
AI-based Chen's DCT [2]	0	132 (10-bit)
Proposed 1D AI	0	31 (10-bit)
Proposed 2D AI	0	24 (10-bit)

Multiplication-free 8x8 2D DCT architecture using algebraic integer encoding
V. Dimitrov, K. Wahid and G. Jullien

Etude des performances - Impact de la cible architecturale

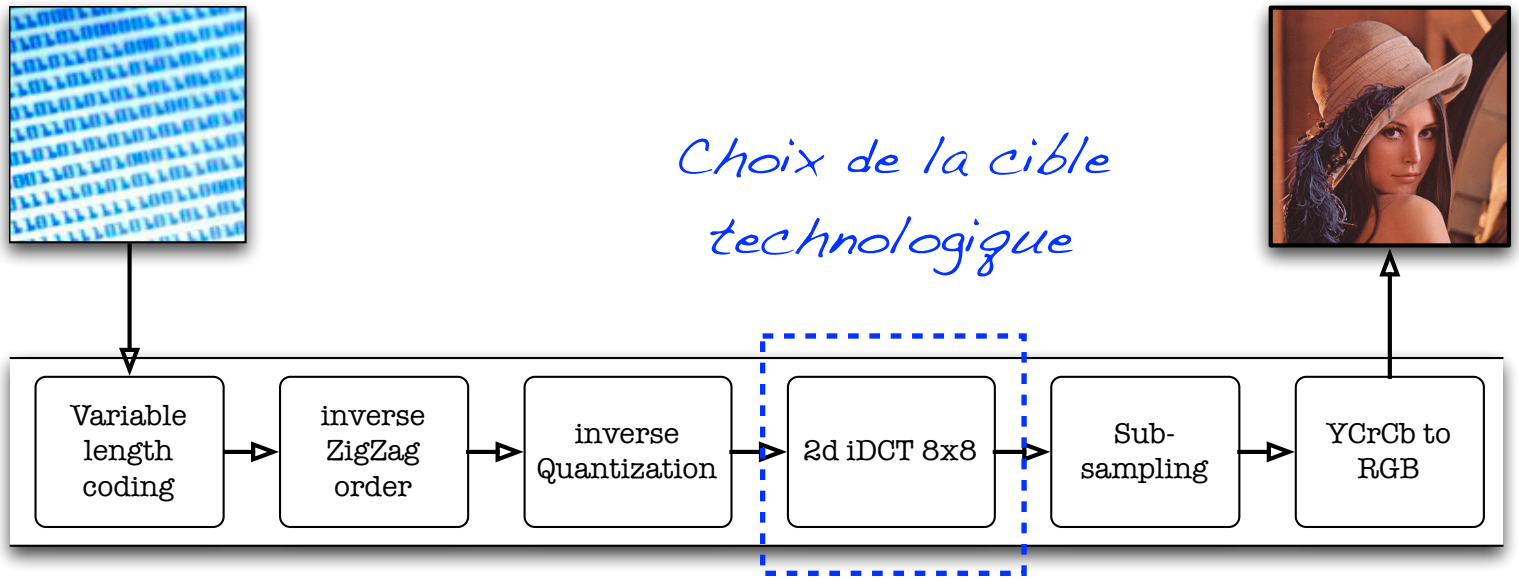
Choix de la cible technologique





Processeur *Accélérateur* *Dedicated chip*

Etude des performances - Impact de la cible architecture



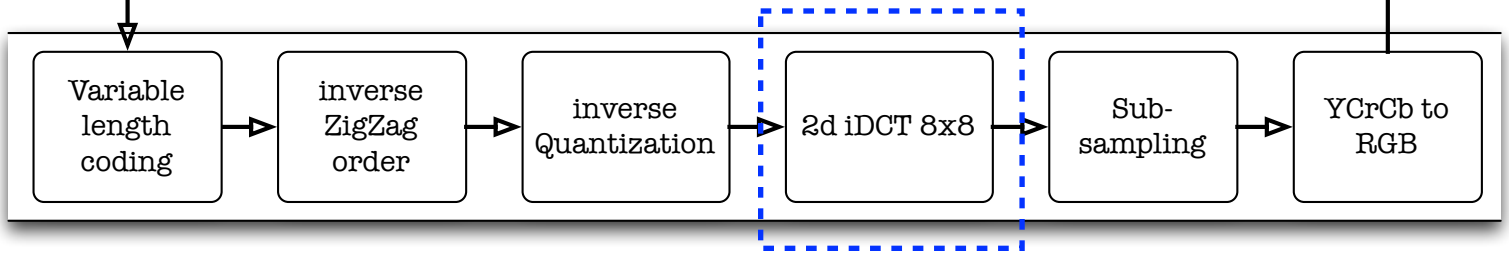
Multiplication-free 8x8 2D DCT architecture using algebraic integer encoding

Using Streaming SIMD extensions in a fast DCT algorithm for MPEG compression

Processeur

Etude des performances - Impact architectural (custom)

Choix de la cible technologique



Pipelined fast 2D DCT architecture for JPEG image compression

Processor architecture driven algorithm optimization for fast 2D-DCT

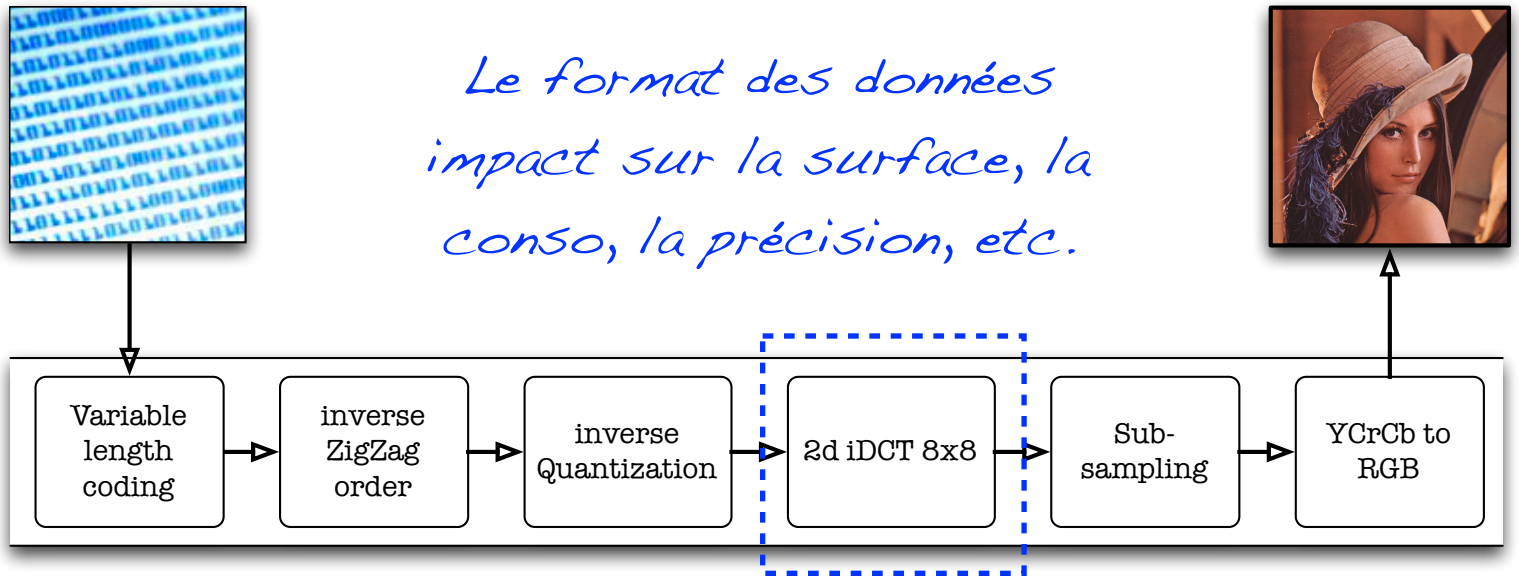
Accélérateur

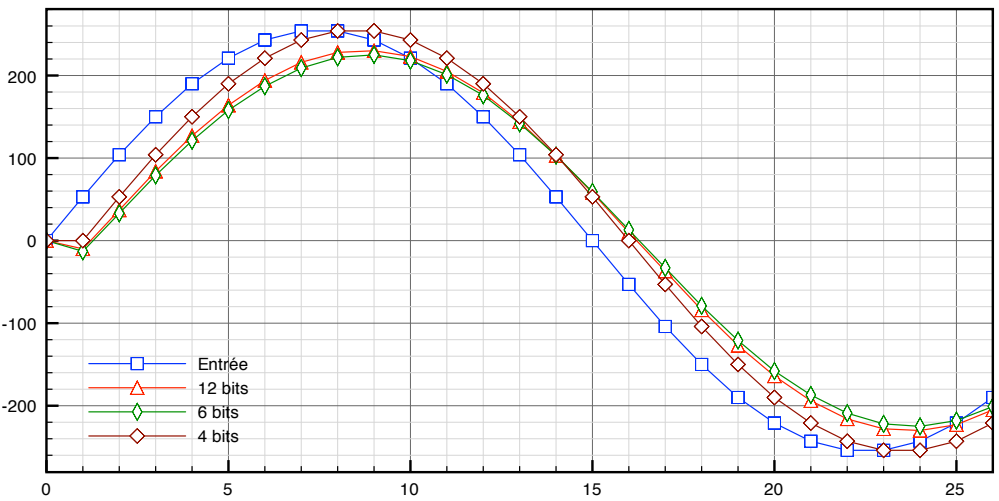
A simple processor core design for DCT/IDCT

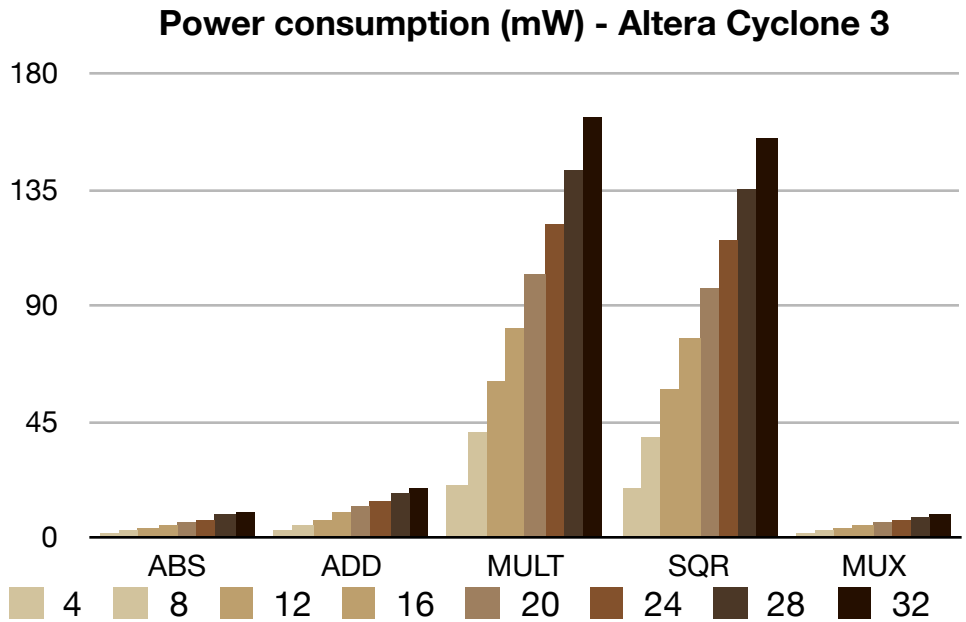
Scalable processor instruction set extension

A 100-MHz 2-D discrete cosine transform core processor

Etude des performances - Impact du format de calcul



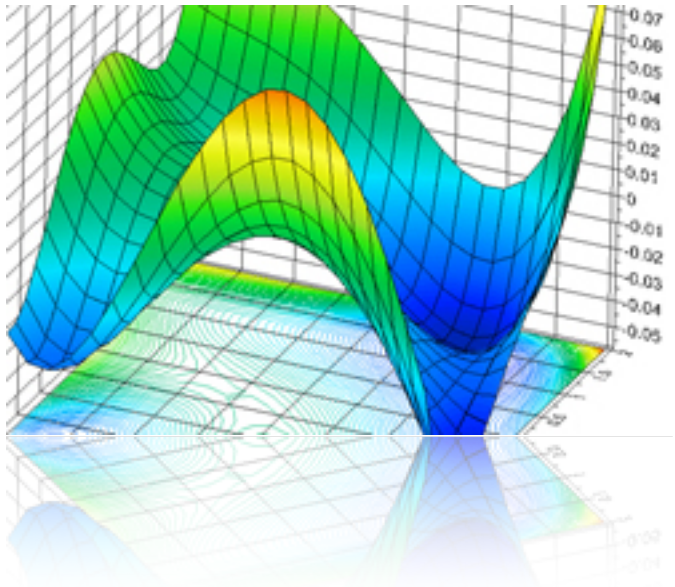




Conclusion

- L'étude de performance se fait à l'aide d'un seul modèle,
  - Implantation logicielle,
  - Implantation matérielle,
- Un seul modèle, et un seul langage permettent,
  - Gain en temps de développement,
  - Une cohérence dans l'évaluation (toujours les même tests, les mêmes résultats)
- Exploration «rapide» de diverses solutions.

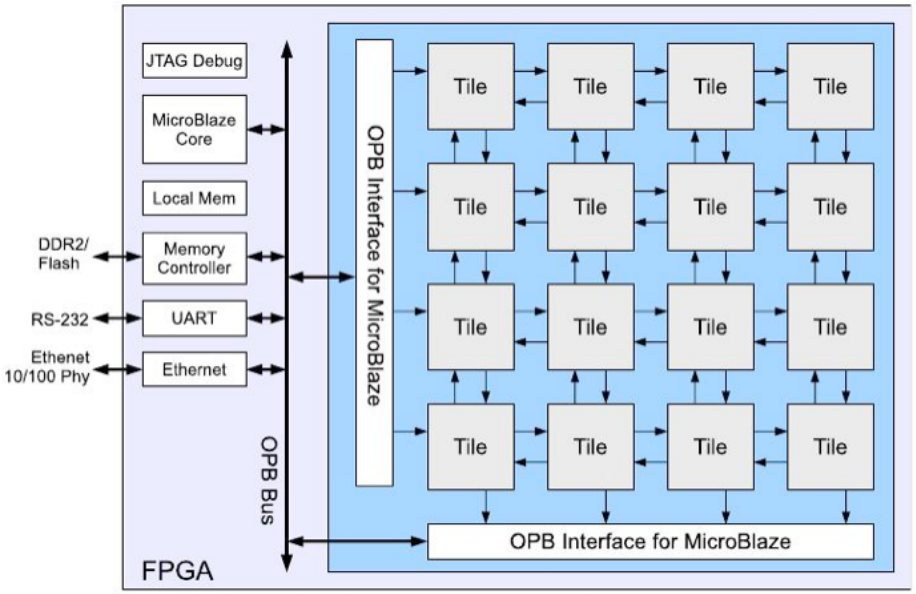
Evaluation de diverses solutions



Partie 3
«Etude transactionnelle»

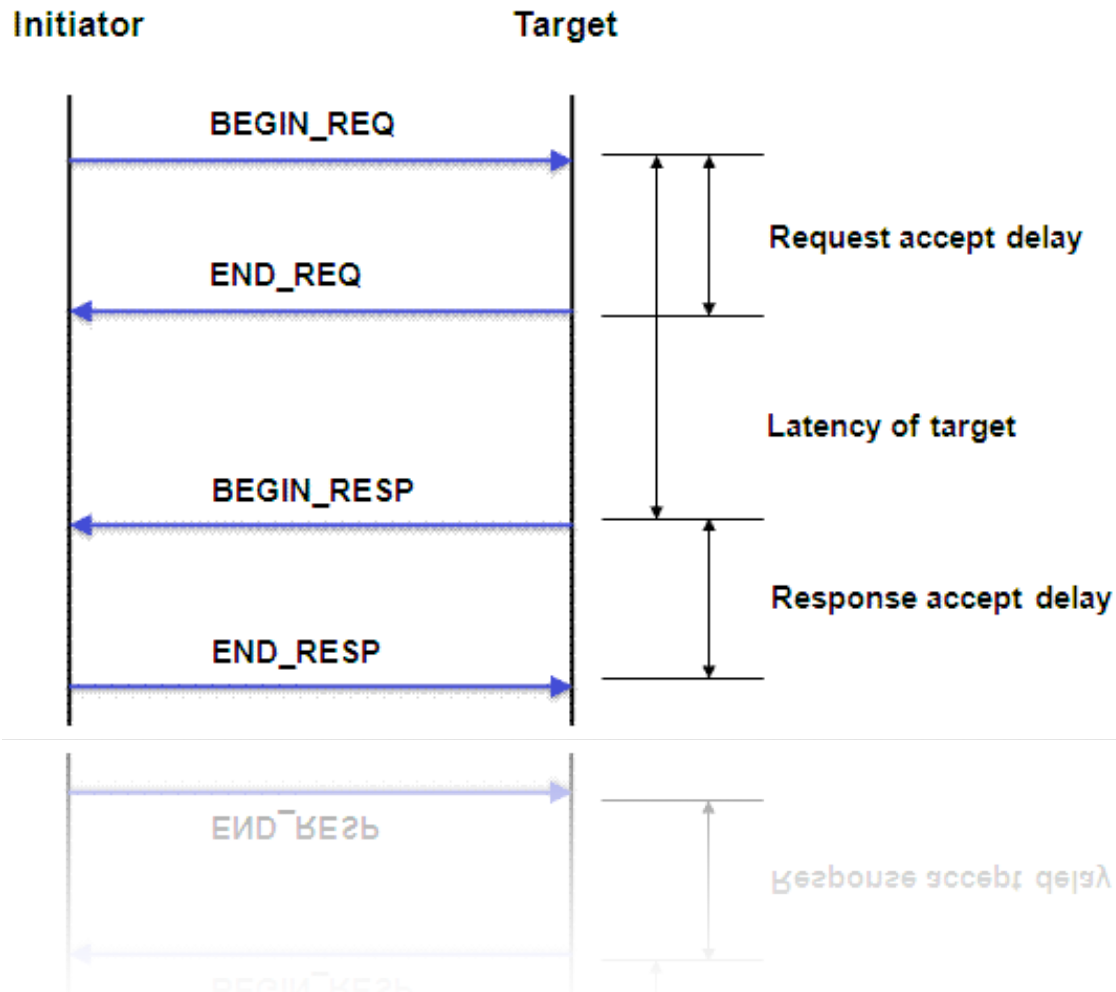
SystemC pour l'étude transactionnelle (TLM)

- L'objectif du niveau TLM n'est pas de valider les algorithmes,
  - Valider les processus de communication entre les modules,
  - Etudier l'impact des topologies sur les performances,
- Canaux de communication complexes,
  - Bus partagés (OPB, AMBA, PCIe),
  - Réseaux (NoC),
- La communications entre les modules est modélisée à l'aide d'appels de fonction.

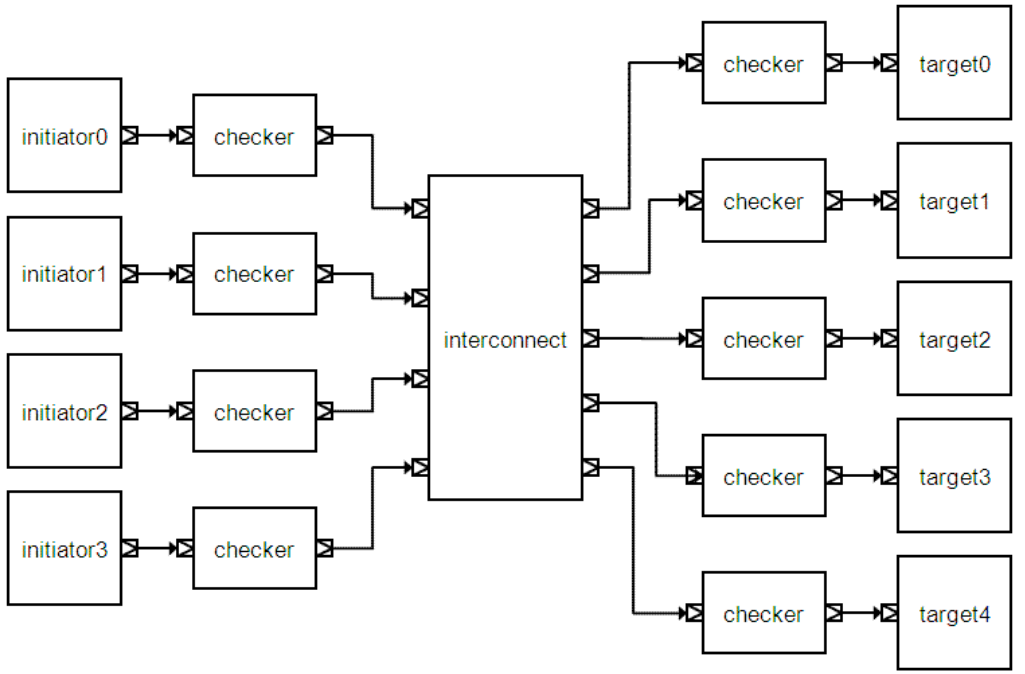


Les objectifs du niveau TLM

- Modélisation de:
  - La topologie d'interconnexion,
  - Des mécanismes de comm.,
    - Accès bloquants, non bloquants, ack...
- Identifier les problèmes potentiels,
  - Latence de communication,
  - Deadlock,
  - Saturation des canaux,
- Vitesse de simulation très rapide (>100Mips)

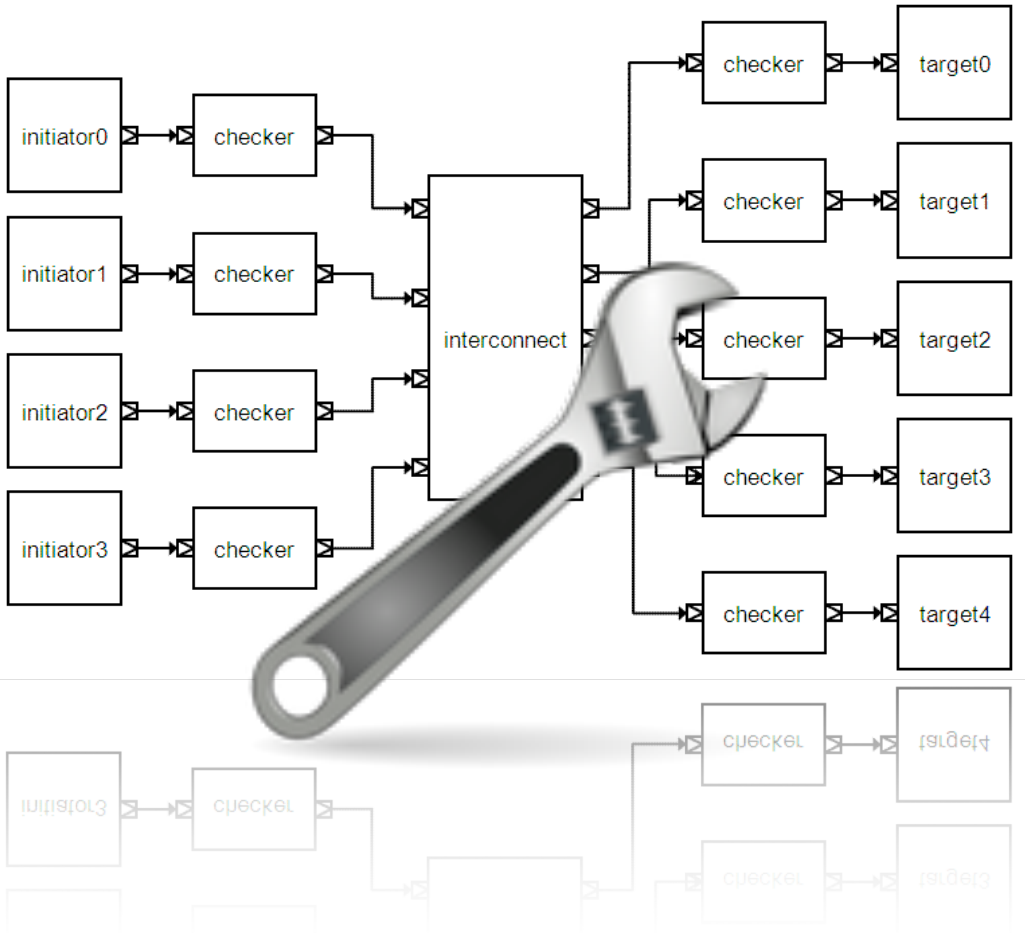


- Trois composants de base dans les modèles TLM:
  - Les «Initiators»
    - Ce sont les composants qui initient des transactions,
  - Les «Targets»
    - Ce sont les composants qui répondent aux transactions,
  - Les «Sockets»
    - Ce sont les éléments en charge du transport des transactions,
- Les transactions sont des structures de données passée de l'initiateur à la cible.



Paramètres adaptables / configurables

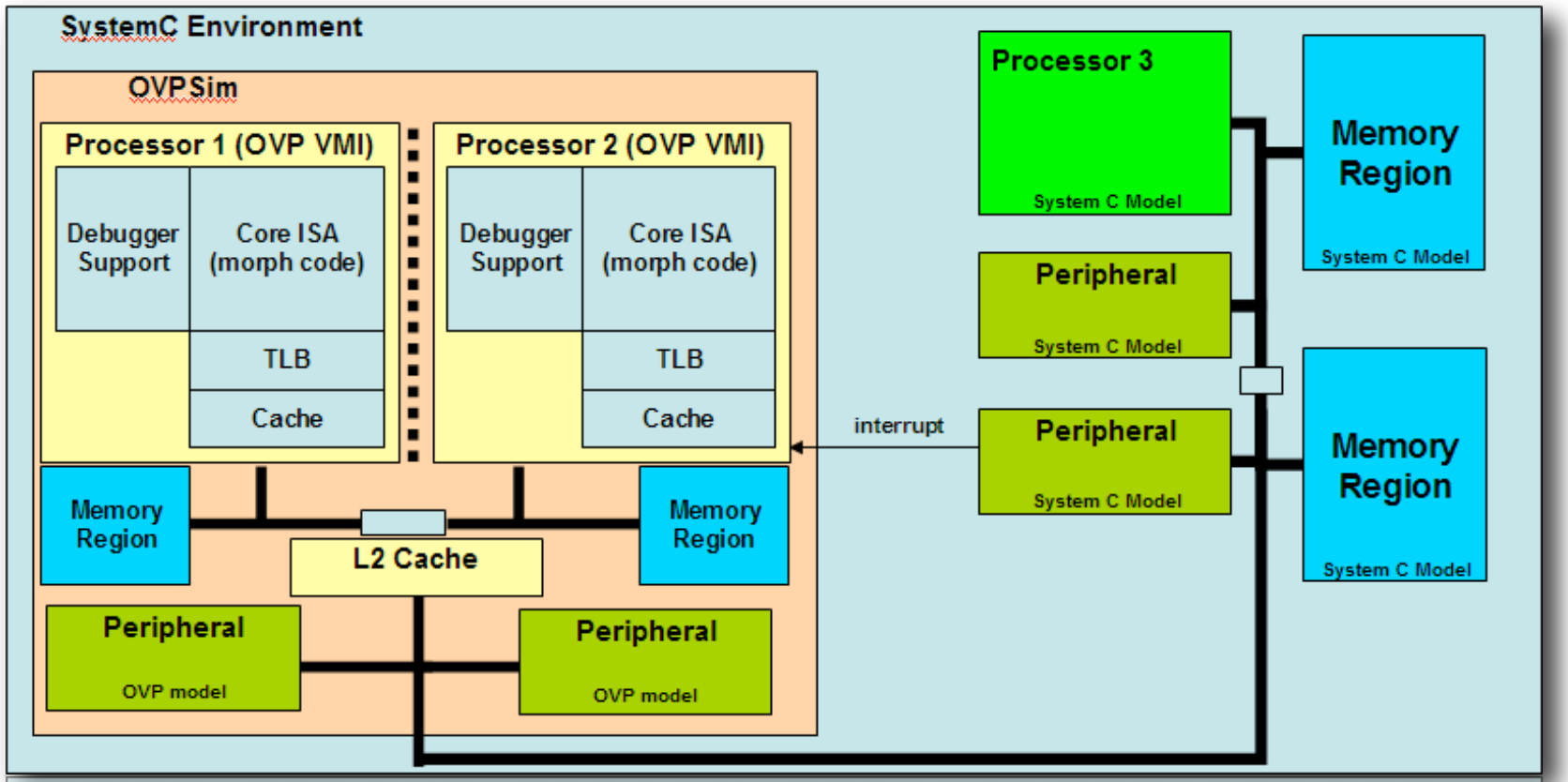
- Type des communications,
  - Bloquantes, non bloquantes,
  - Avec ou sans acknowledge,
- Caractéristiques des communications,
  - Taille des données à transmettre,
  - Type de transfert (burst),
- Description des systèmes de communication à plusieurs niveaux,
  - Untimed,
  - Bus-Cycle Accurate.



Exemple d'observation post-simulation

Partie 3
«Modélisation de plateforme matérielle»

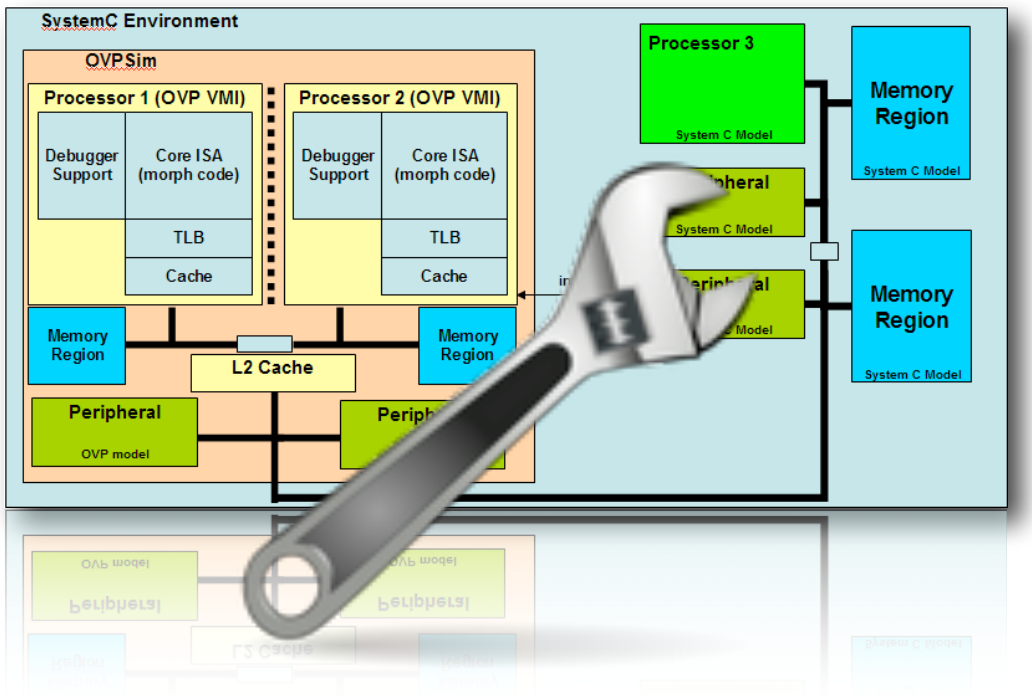
Le projet «Open Virtual Platforms»



Objectif, permettre la modélisation de plateformes (virtual plateforms) afin de valider les performances avant conception !

http://www.ovpworld.org/welcome

Le projet «Open Virtual Platforms»



Evaluer les performances de l'architecture du système:

- nombre et type de coprocesseurs,
- nombre et type de processeurs,
- topologies d'interconnexion,

Compilation logicielle

Code logiciel (benchmarks, OS, custom)

Programme exécutable

Validation / comparaison des performances des calculateurs

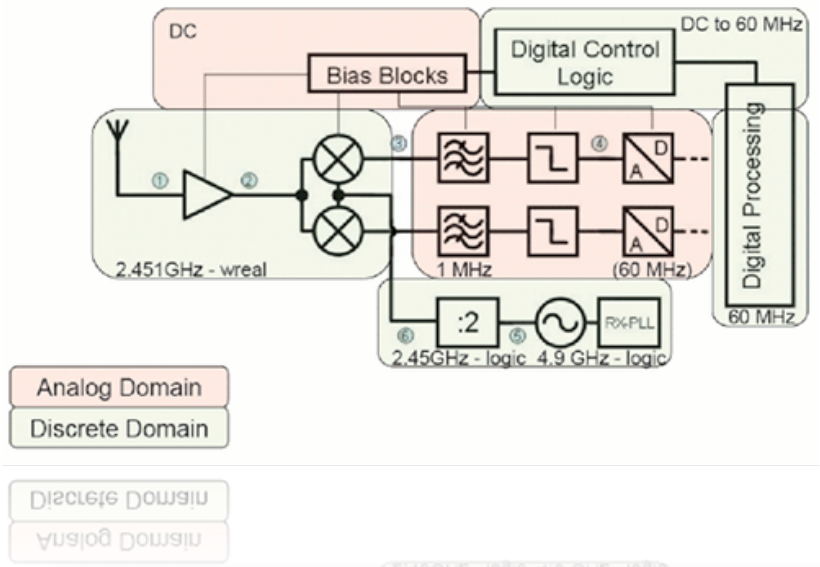
	Xilinx MicroBlaze			ARM			MIPS32		
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS
linpack	12,009,854,342	10.96s	1096	832,038,281	1.16s	718	1,450,948,024	1s	1451
Dhrystone	1,508,115,198	1.56s	969	650,077,035	0.97s	673	974,078,238	0.99s	980
Whetstone	27,108,532,649	21.4s	1267	1,185,189,267	1.54s	772	4,330,064,913	2.74s	1583
peakSpeed1	19,500,006,464	9.95s	1959	40,000,003,640	17.83s	2243	15,000,001,389	6.59s	2275
peakSpeed2	22,000,023,427	8.85s	2485	22,400,008,759	7.14s	3136	23,600,001,685	8.95s	2637
	Synopsys ARC			**Renesas v850**			**PowerPC**		
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS
linpack	4,184,162,659	5.28s	792	4,991,344,152	6.82s	732	3,163,966,107	3.84s	824
Dhrystone	1,262,082,471	1.59s	793	2,564,132,566	2.61s	982	786,068,237	1.03s	761
Whetstone	7,883,567,040	6.93s	1138	10,296,940,584	11.22s	918	6,424,865,749	5.96s	1079
peakSpeed1	39,000,001,829	15.6s	2500	28,000,003,177	10.9s	2568	20,000,002,054	10.09s	1981
peakSpeed2	22,000,002,095	6.18s	3563	22,400,007,562	7.87s	2845	22,400,002,931	8.48s	2641

All measurements on 2.83GHz Intel Core2, OVPsim 20120906

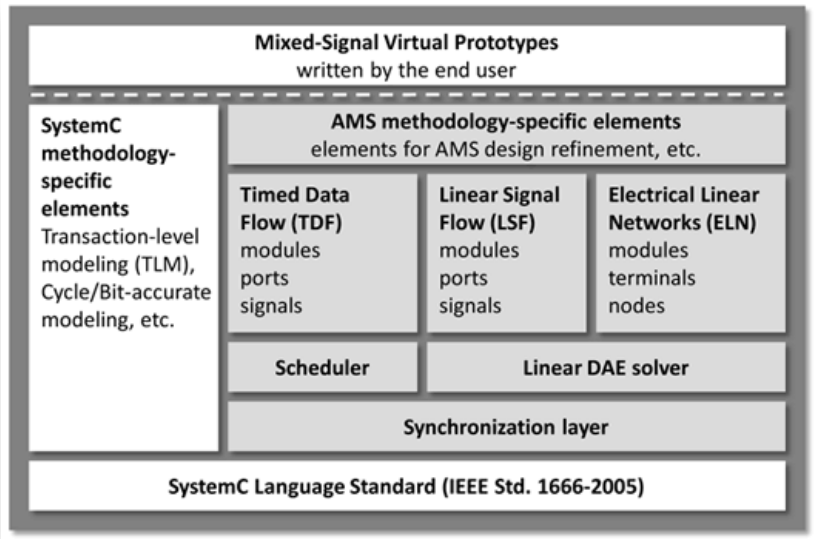
Partie 3
«Modélisation de systèmes complexes»

La modélisation de systèmes plus «complexes»

- Les systèmes réels ne sont pas constitués que d'éléments numériques...



- L'extension AMS vise à fournir des capacités de modélisation analogique au langage,
  - ➡ Un seul langage pour tout modéliser,
  - ➡ Un seul et unique simulateur,
    - Simulation conjointe facilitée...



Perspectives & Conclusion

Conclusion sur le langage SystemC

- Le langage *SystemC* offre aux concepteurs de systèmes une grande liberté dans la modélisation des systèmes à concevoir,
- Le flot de raffinement progressif à l'aide d'un seul et unique langage (jusqu'à l'implantation) permet de gagner en temps de développement,
  - ➡ Moins d'erreurs de codage dans conversions (C => VHDL),
  - ➡ Les étapes sont plus simples => moins d'erreurs de raffinement
- Il existe d'autres langages concurrents à SystemC comme SystemVerilog qui est défendu par les USA (SystemC est Européen),
- Il existe encore des lacunes dans le flots de conception d'aider le concepteurs : aide au raffinement, éditeurs performants, outils d'analyse et de synthèse logique, etc.