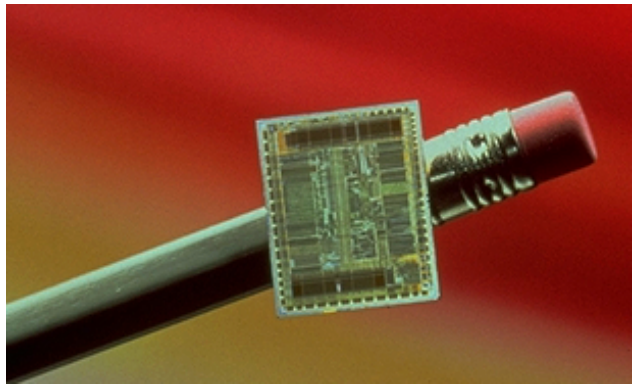


SE303 : Conception des systèmes sur puces

HLS : High Level Synthesis

Abdelmalek SI MERABET / Yves MATHIEU
abelmalek.si-merabet@telecom-paristech.fr
yves.mathieu@telecom-paristech.fr

Outline

Outline

Le rêve du concepteur

De la synthèse RTL à la synthèse HLS

- Eviter la description "au cycle près", "au registre près", "au bit près"
- On aimerait décrire :
  - Un algorithme de traitement
  - Des contraintes de vitesse (fréquence d'horloge)
  - Des contraintes de débit (nombre de traitements par seconde)
  - Des contraintes de latence (délai maximum, en nombre de cycles entre la demande d'un traitement et le résultat
- High Level Synthesis

Méthodologies HLS pour les SoC

Une foison de situations

- Choix du langage d'entrée (C, C++, SystemC, VHDL, Verilog)
- Choix du public de destination :
  - Informaticiens : faible culture des architectures matérielles ad-hoc.
  - Electroniciens : faible culture des langages de haut niveau.
- Choix de l'architecture cible :
  - Style CPU (basée sur un graphe de contrôle) associée à des opérateurs de base (multiplieurs, alus,...)
  - Style DATA-FLOW (basée sur un graphe de données) associée à des machines à états de contrôle...

Méthodologies HLS pour les SoC

Langages

- Pas de définition précise de la notion de représentation de haut niveau d'un système matériel
- Quel sous-ensemble du langage ?
- Comment spécifier des entrées/sorties
- Usage systématique de pragmas de synthèse pour exprimer l'intention du concepteur
  - Pour guider l'outil vers une solution raisonnable (parallélisme / pipeline)
  - Pour choisir des options techniques dans une technologie donnée (exemple RAM/flip-flops)
  - Pragmas différents d'un outil à l'autre.

Méthodologies HLS pour les SoC

Un temps d'apprentissage "long"

- Un abord difficile à cause du changement de paradigme.
- Une difficulté à basculer d'un outil à l'autre à cause de la différence d'approche.
- Mais un gain en temps de développement pour qui maîtrise la méthodologie
- Des exemples ridiculement simples dans les tutoriels des outils :
  - Exemple : Un débutant en HLS doit coder un filtre à réponse impulsionnelle finie...
  - RTL : Facile à coder, si on sait à l'avance le niveau de parallélisme ou pipeline voulu
  - HLS : Se prendre la tête avec les pragmas de synthèse pour obtenir le parallélisme ou pipeline voulu
  - HLS : Un schéma d'E/S figé qui ne correspond pas forcément au contexte d'utilisation voulu

Méthodologies HLS pour les SoC

Quelle question veut on résoudre ?

- Exemple : La DCT (Discrete Cosine Transform)
- Il existe des IP matérielles optimisées
- En HLS, pour tenter d'approcher leurs performances :
  - Réécrire le code C original
  - Trouver la bonne combinaison de pragmas :
    - Déroulement de boucles
    - Pipeline
    - Mémoires simple port
    - Mémoires double port
    - Registres...
  - Mettre en oeuvre l'exploration d'architectures
    - Prévu dans la plupart des outils
    - Mais insupportablement long, car fait appel à de multiples synthèses RTL

Méthodologies HLS pour les SoC

Quel problème veut-on résoudre ?

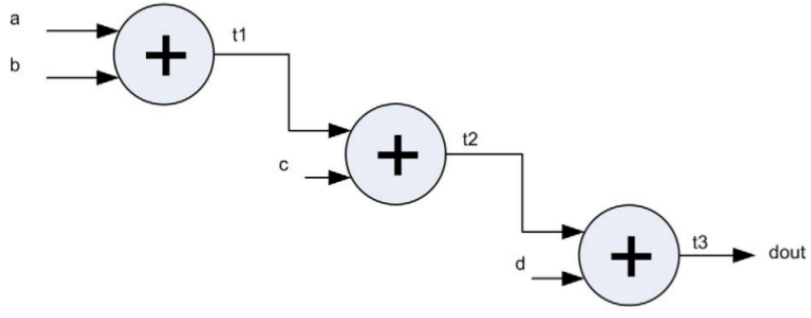
- Diminuer les temps de codage longs et fastidieux :
  - Ecriture des machines à état de contrôle.
  - Ecriture des codes de communication vers le monde extérieur (communications ad-hoc, bus normalisés)
- Le véritable problème n'est pas "Je veux coder une DCT"
- Le véritable problème est "Je veux coder une DCT..."
  - Qui communique avec un bus esclave de la norme XXX
  - Avec un bus de données de largeur YYY
  - Avec des données traitées de largeur ZZZ
  - Avec des transferts de données par paquets de TTT données

Outline

"Data Flow Graph"

Graphe des flux de données...

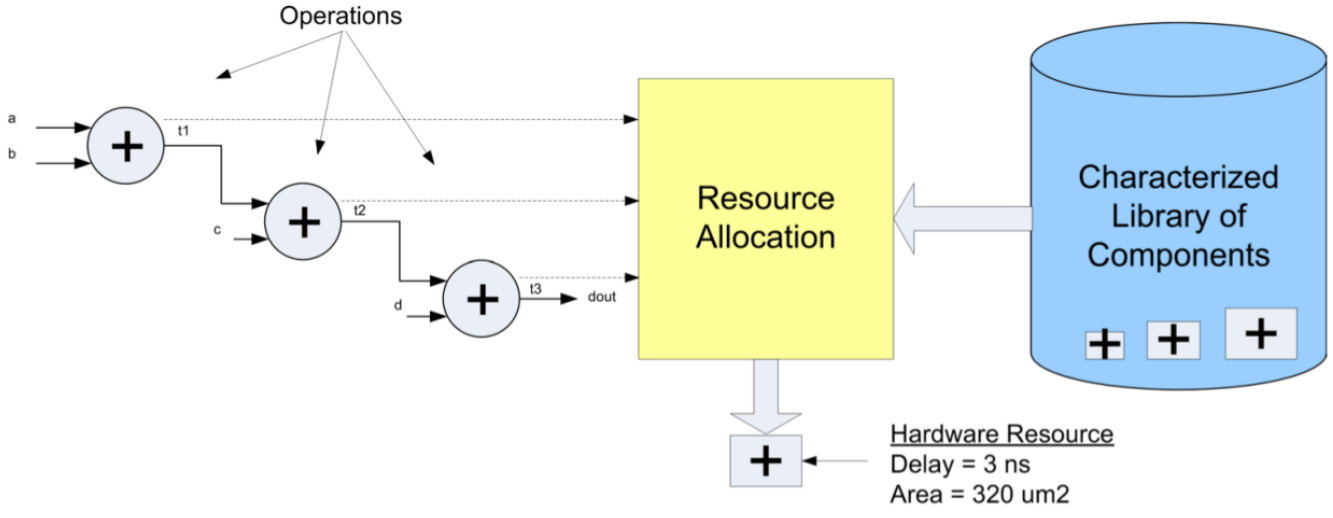
```
#include "accum.h"
void accumulate(int a,
                int b,
                int c,
                int d,
                int &dout)
{
  int t1,t2;
  t1 = a + b;
  t2 = t1 + c;
  dout = t2 + d;
}
```



0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Resource Allocation"

Allocation des ressources...

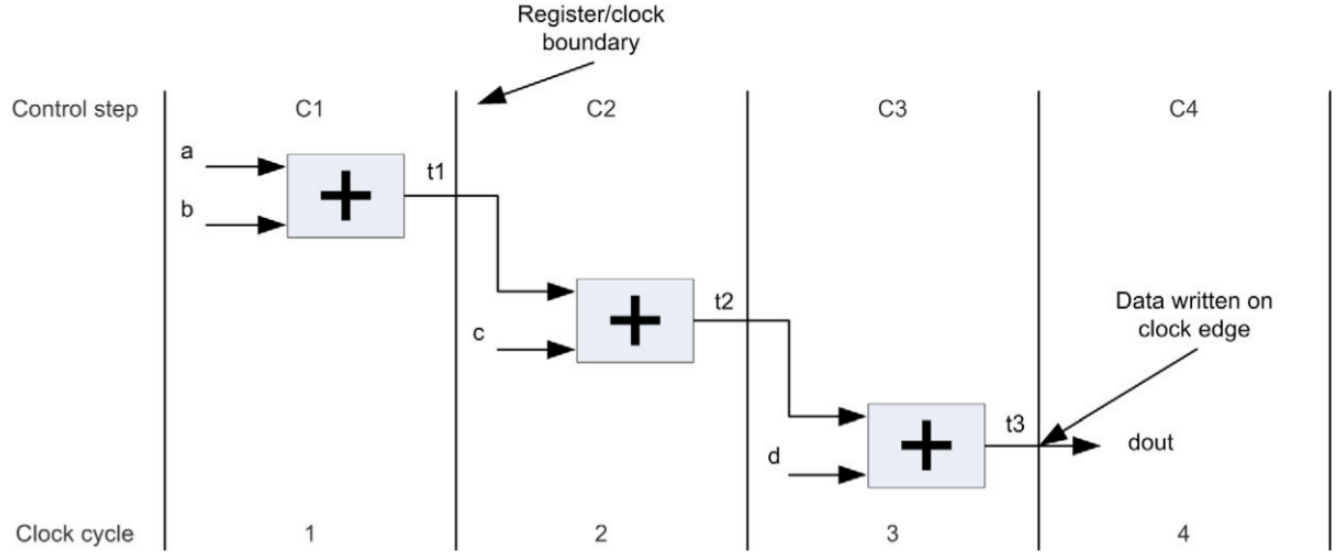


- Choix des opérateurs, dimensionnement des chemins de données
- Estimateurs de surface, estimateurs de délai
- Nécessite la connaissance des données technologiques

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Scheduling"

Ordonnancement...



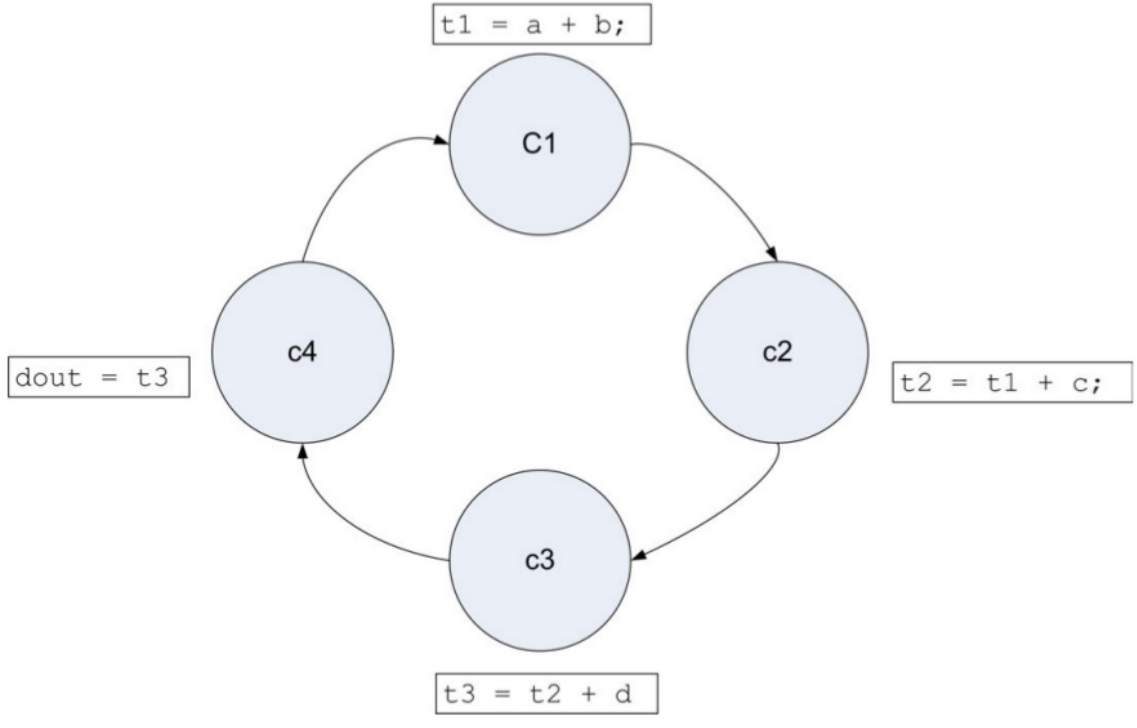
- Supposition : $T_{addition} < T_{clk}$
- Allocation de chaque opération à un cycle donné.
- Stockage des résultats intermédiaires (registre)

---

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Control steps"

Etapes de contrôle

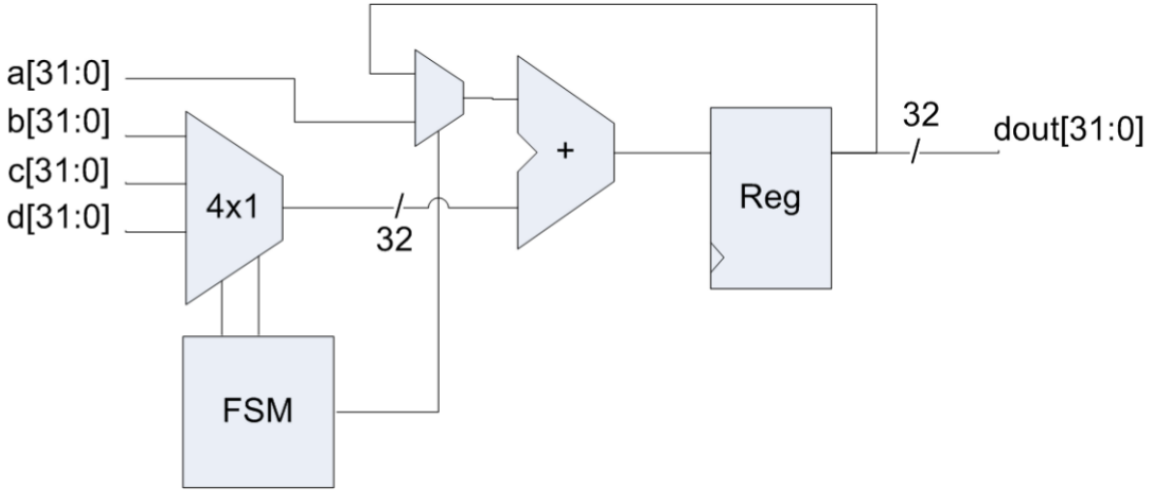


- Automate associé au chemin de données.
- Impose l'ordonnancement choisi.

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Implémentation matérielle"

Un design sans contraintes



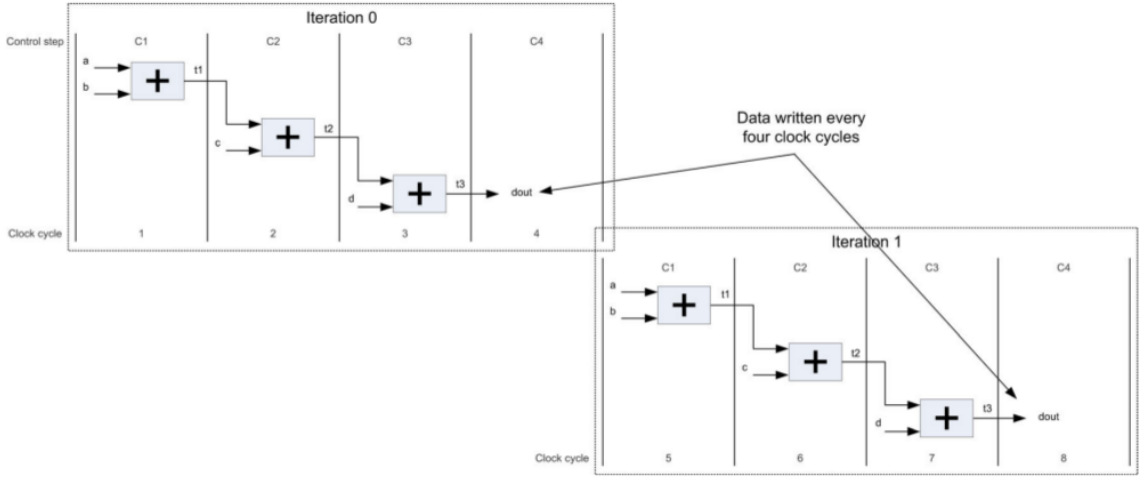
- Minimisation des ressources
- Un seul additionneur, associé à automate.

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Pipelining"

Pipeline de boucle

- "Initiation Interval" : combien de cycles d'horloge entre deux utilisations de la boucle.
- II=1 : Un nouveau calcul à chaque cycle.
- "Latency" : Combien de cycles d'horloge entre la première donnée entrante et la première donnée sortante.



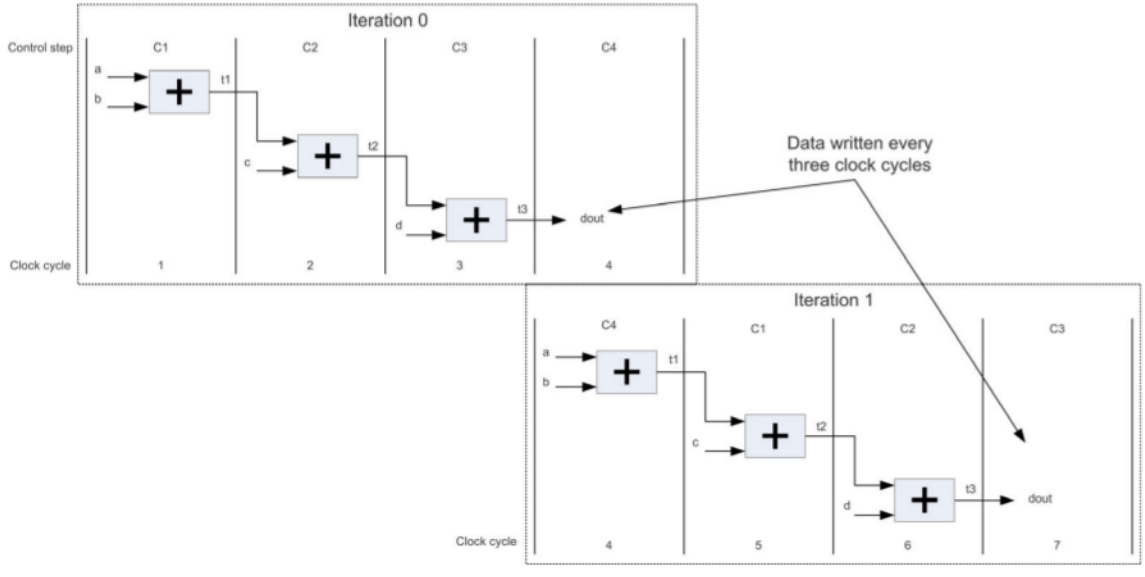
Pas de contrainte L=3, II=4

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Pipelining"

Pipeline II=3, L=3

- On contraint la synthèse (pragma, script...)
- Remarque : En C4 une sortie en même temps qu'une entrée.
- On suppose ne pas être containt pas les E/S



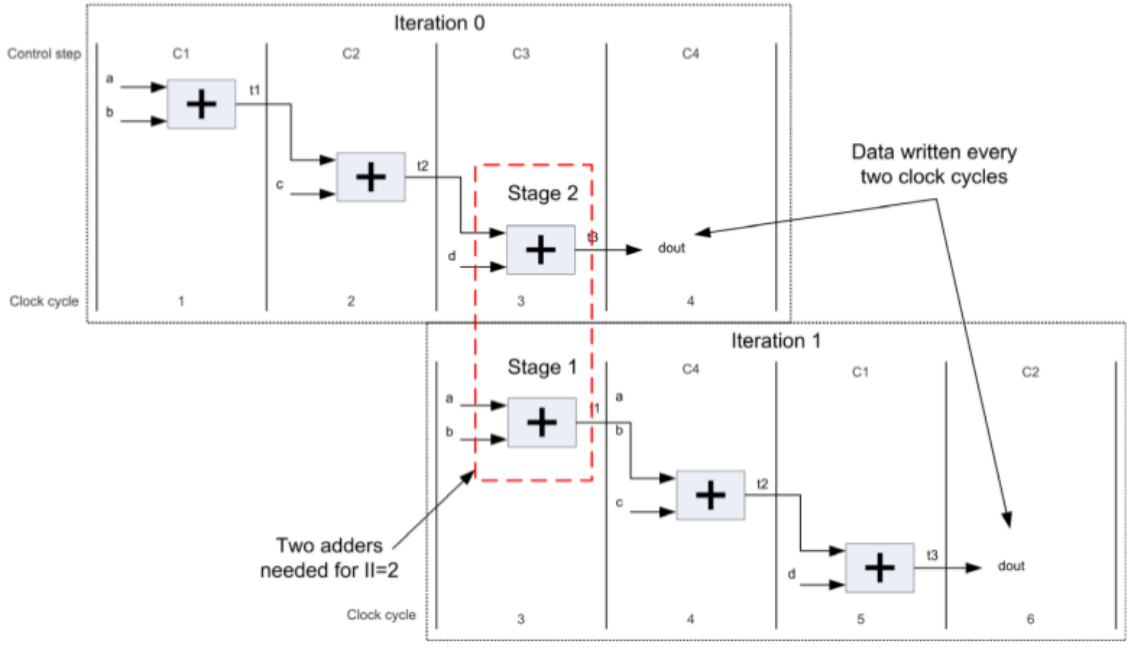
Pipeline L=3, II=3

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Pipelining"

Pipeline II=2, L=3

- On contraint la synthèse (pragma, script...)
- Il faut 2 additionneurs en parallèle.
- On suppose toujours ne pas être containt pas les E/S



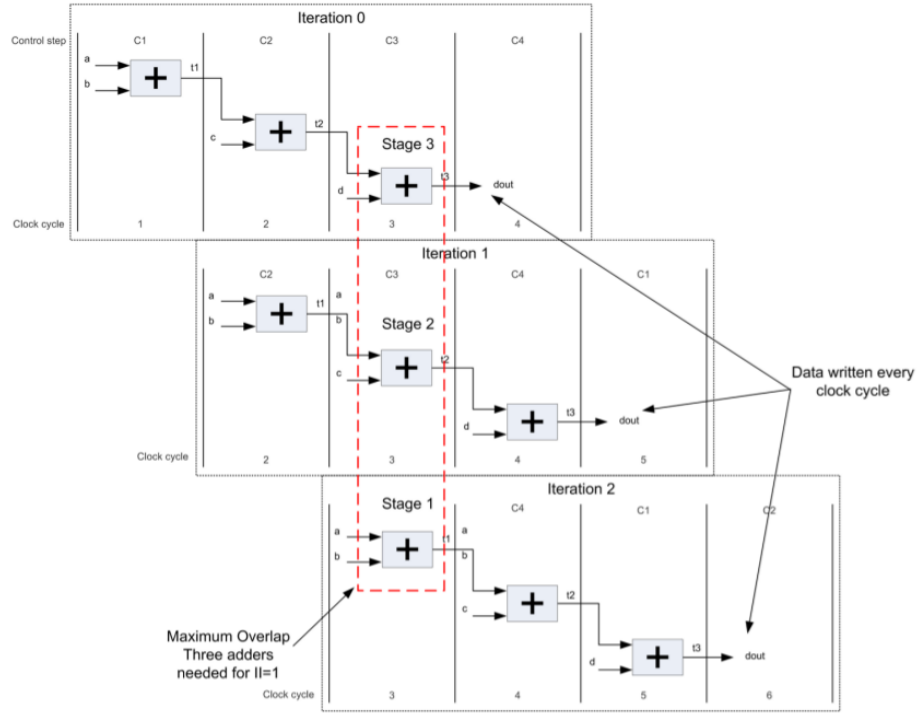
Pipeline L=3, II=2

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Pipelining"

Pipeline II=1, L=3

- Il faut 3 additionneurs en parallèle.
- Performance maximale.



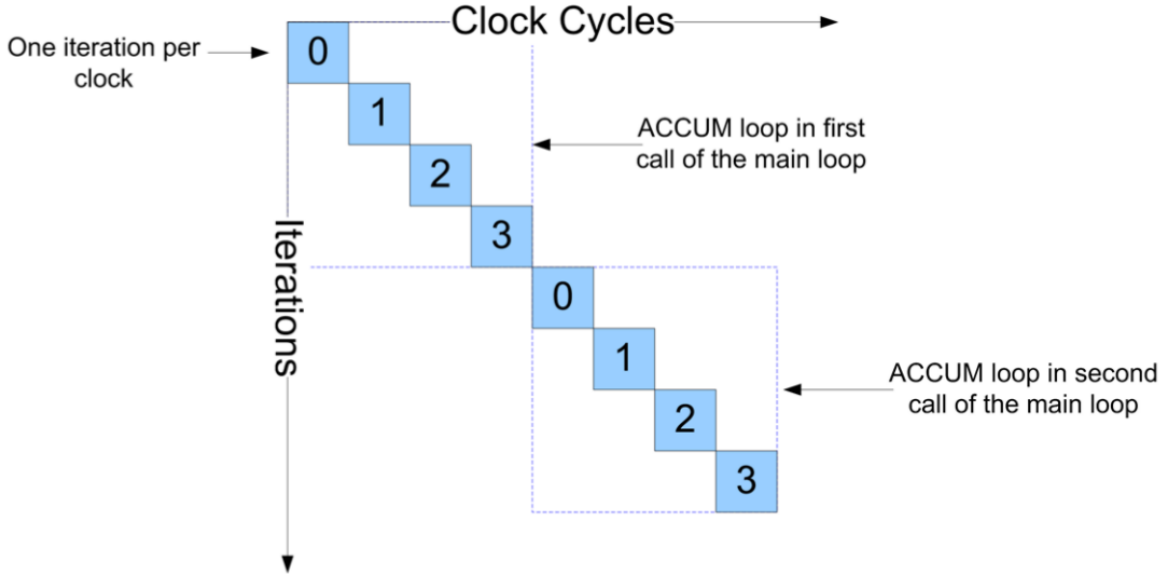
Pipeline L=3, II=1

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Unrolling"

Déroulage de boucles

- On ne joue pas sur les itérations de la boucle
- On joue sur le parallélisme dans la boucle



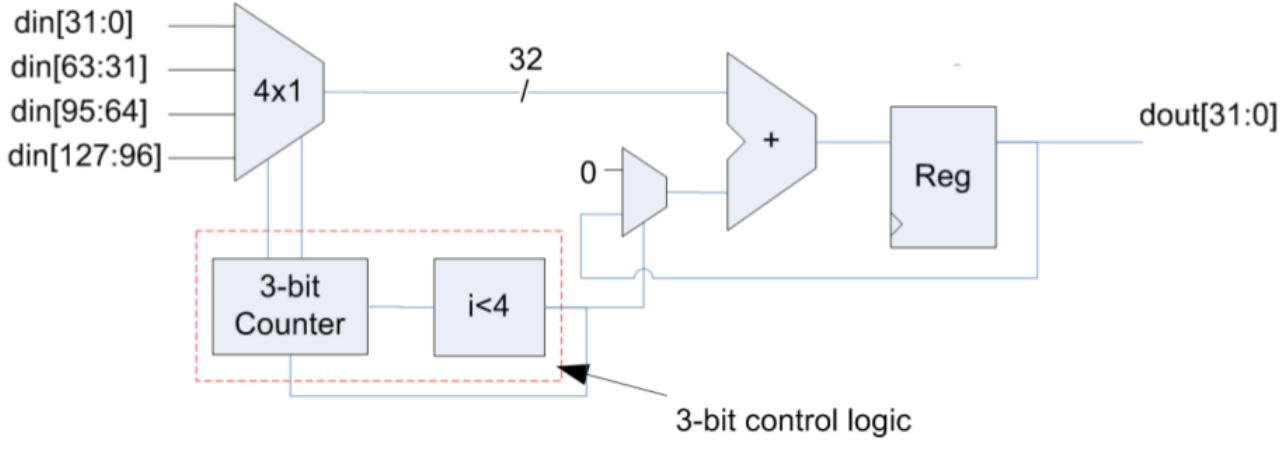
Ordonnancement initial de 2 appels successifs de la boucle.

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Unrolling"

Implémentation initiale

- Au premier cycle on charge din[31 :0]+0
- Aux 3 autres cycles on accumule



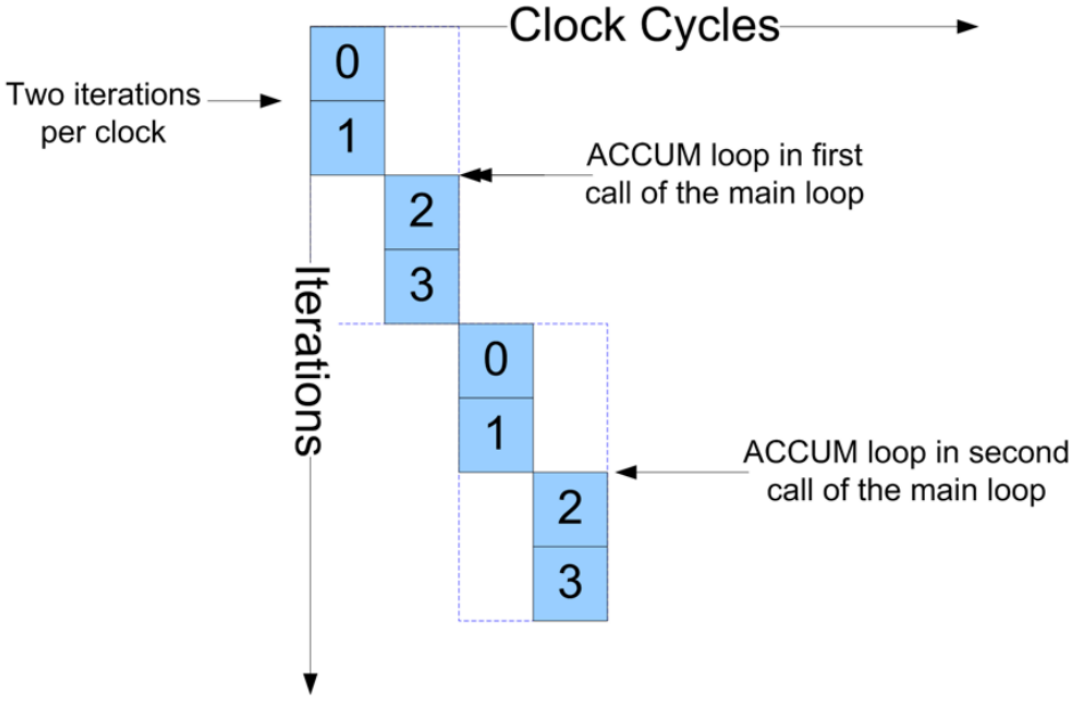
Implémentation matérielle

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Unrolling"

Déroulement partiel

- Déroulement d'un facteur 2
- Deux données traitées en parallèle dans la boucle.



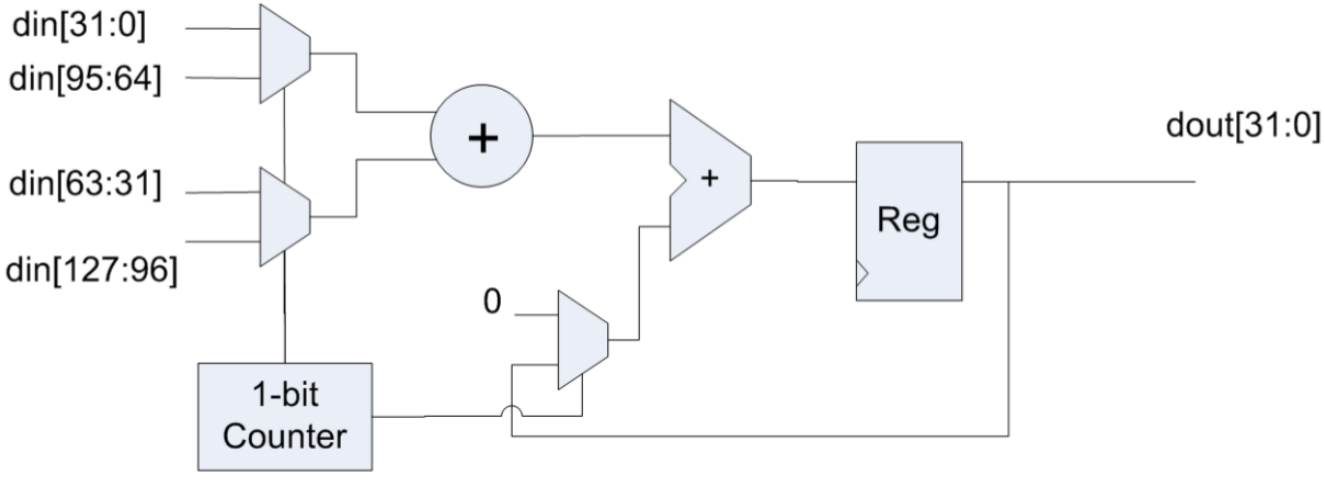
Ordonnancement pour un déroulement d'un facteur 2

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Unrolling"

Déroulement partiel

- Toujours un seul additionneur.
- Compteur plus petit.



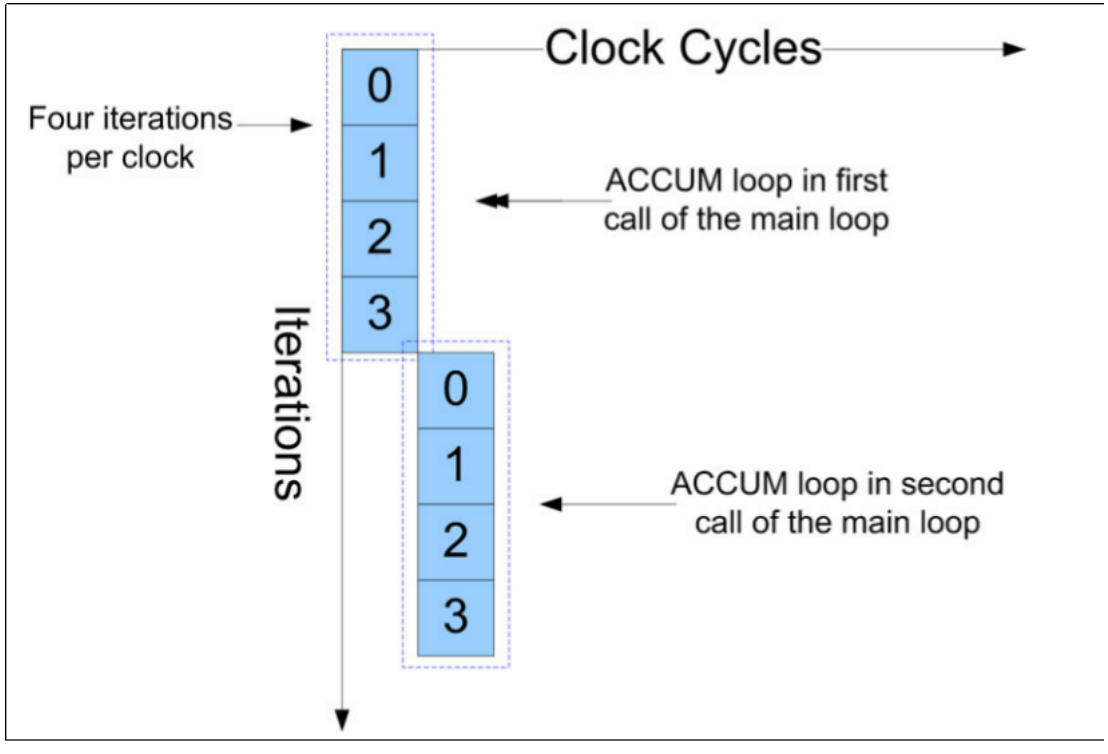
Implémentation matérielle

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Unrolling"

Déroulement total

- Déroulement d'un facteur 4
- Quatre données traitées en parallèle dans la boucle.



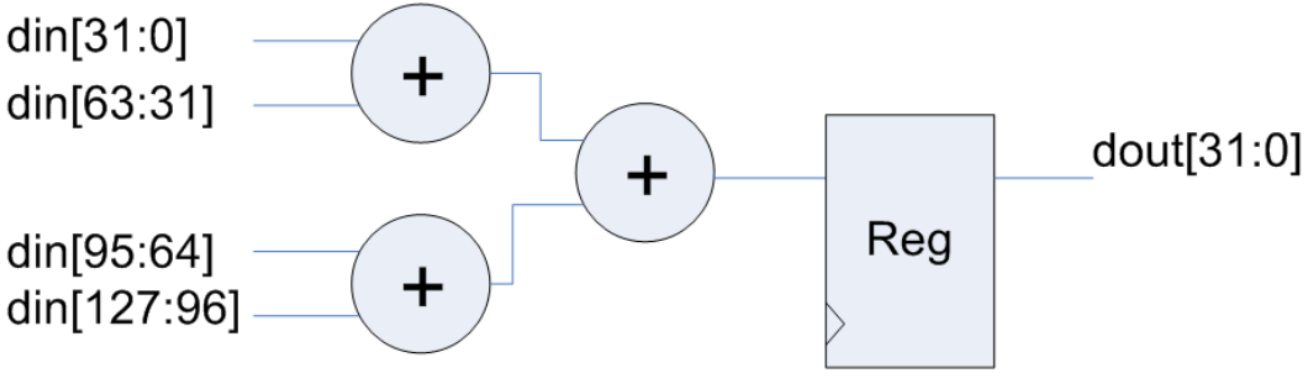
Ordonnancement pour un déroulement d'un facteur 4

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Loop Unrolling"

Déroulement total

- Trois additionneurs.
- Plus de compteur.



Implémentation matérielle

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Les boucles"

Le coeur des outils HLS

- Boucles simples à bornes statiques.
- Pipeline et Déroulement de boucle combinables.
- Création automatique des automates de gestion.
  - En RTL : codage long et fastidieux des amorçages et fin de pipeline
  - En RTL : debogage long et fastidieux
  - En RTL : On ne tente souvent qu'une seule alternative
- Bonus : Exploration automatique de différentes alternatives.
- Le résultat final optimal peut être contre-intuitif.
- STANDARD DANS TOUS LES OUTILS HLS

Outline

Langage source

Impact sur la méthodologie

- Langage C/C++ : on compile une fonction de traitement.
- Les entrées/sortie de la fonction correspondent à des ports de communication
- Il faut pouvoir préciser les protocoles utilisés
- Solution : les pragmas de synthèse.

```
void example(int A[50], int B[50]) {
//Set the HLS native interface types
#pragma HLS INTERFACE axis port=A
#pragma HLS INTERFACE axis port=B
  int i;
  for(i = 0; i < 50; i++){
     B[i] = A[i] + 5;
  }
}
```

Xilinx Vivado-HLS : Deux ports AXI4-stream séparés

Langage source

Codage en C/C++

- Regrouper les ports, préciser les adresses,...
- Tout est dans le pragma...
- Limitation aux pragmas prévus par l'outil.

```
void example(char *a, char *b, char *c) {
#pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=a bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=b bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=c bundle=BUS_A offset=0x0400
#pragma HLS INTERFACE ap_vld port=b
  *c += *a + *b;
}
```

Xilinx Vivado-HLS : Un seul port AXI4-lite pour tout le monde...

Langage source

Xilinx Vivado HLS : SystemC

- Même principe que pour C (appliqué à "sc_in" et "sc_out"
- En plus, interprétation directe de "sc_fifo"

```
SC_MODULE(sc_sequ_cthread){
    sc_fifo_out<int> dout;
    sc_fifo_in<int> din;
    ...
}

void sc_FIFO_port::Prc2() {
#pragma HLS resource core=AXI4Stream variable=din
#pragma HLS resource core=AXI4Stream variable=dout
  ...
}
```

Outline

C to Silicon

Principes

- Langage d'entrée SystemC
- On mélange niveau RTL et HLS
- Les ports d'entrée/sortie peuvent être de vrais signaux ou non...
- On peut coder au cycle près (ordonnancement manuel) mais...
- On peut monter en abstraction et laisser l'outil faire le travail...
- On peut cibler une technologie ASIC ou une technologie FPGA
- Obsolète : Stratus HLS nouvel outil, mêmes principes.

Ctos

Un SC_MODULE au niveau le plus haut

```
::::::::::::
my_top_module.h
::::::::::::
#include "systemc.h"
SC_MODULE(my_top_module) {
    sc_in<bool> clk;
    sc_out<bool> rst;
    ..
    SC_CTOR(my_top_module)
    clk("clk"),
    rst("rst")
    {
        ..
    }
    ..
    private:
        void main();
};
::::::::::::
```

```
::::::::::::
my_top_module.cpp
::::::::::::
#include "my_module.h"
..
void my_top_module::main() {
    ..
}
..
#ifdef __CTOS__
SC_MODULE_EXPORT(my_top_module);
#endif ::::::::::::
```

La macro SC_MODULE_EXPORT permet à CtoS de connaitre le top module du design.

Ctos : règles de codage

Des processus, des communications entre processus

- Des SC_METHOD pour les processus purement combinatoires.
- Des SC_CTHREAD pour les processus séquentiels synchrones (recommandé).
- Si une variable est partagée par 2 processus : utiliser un sc_signal
- Si une variable n'est utilisée que par 1 processus : ne pas utiliser un sc_signal
- Les variables globales ne sont pas supportées.

Ctos : Un SC_CTHREAD

Style de code synthétisable

```
SC_MODULE(pulser) {
 public:
  sc_in<bool> clk, rst, set_rate;
  sc_in< sc_int<16> > rate;
  sc_out< sc_int<16> > out;

  SC_CTOR(pulser) :
    clk("clk"), rst("rst"),
    set_rate("set_rate"), rate("rate"),
    out("out")
    {
      SC_CTHREAD(main, clk.pos());
      reset_signal_is(rst, true);
      }

      void main();
      int cur_rate;
};
```

```
void pulser::main()
{
  out.write(0);
  wait();
  while (!set_rate.read()) { wait(); }
  cur_rate = rate.read();
  while (1) {
    for (int i=0; i<cur_rate-1; i++) {
        wait();
    }
    out.write(1); wait(); out.write(0);
  }
}
```

Code du reset : avant le premier wait().

Reset Synchrone, Précis au niveau cycle.

Ctos : diriger le synthétiseur

Pragmas ou scripts tcl

- Les directives de synthèse peuvent être fournies :
- Dans le script de synthèse TCL :
  - Fournir des configuration générales
  - Calculer des paramètres dynamiques de configuration
  - Il faut identifier les objets sur lesquels appliquer les directives
- Par des pragmas insérés directement dans le source.
  - S'applique au code qui suit immédiatement
  - Statique

TCL :

```
constrain_latency -max 32 -name body_latency start_node_id end_node_id
```

Pragmas :

```
#pragma ctos constrain_latency -max 32 -name body_latency
{
...
}
```

Ctos : Directives de synthèse

L'exemple des RAM

- Les tableaux peuvent être transformés en mémoires (ASIC ou FPGA)
- Applicable aux variables, pas aux signaux.
- Des options d'implémentation :
  - Des tableaux de registres. (ASIC et FPGA) : Flatten array
  - L'inférence d'une RAM (FPGA) : Allocate built-in-RAM
  - L'instance d'une "blackBox" (ASIC) : allocate vendor RAM
- Attention aux options contradictoires avec le code.

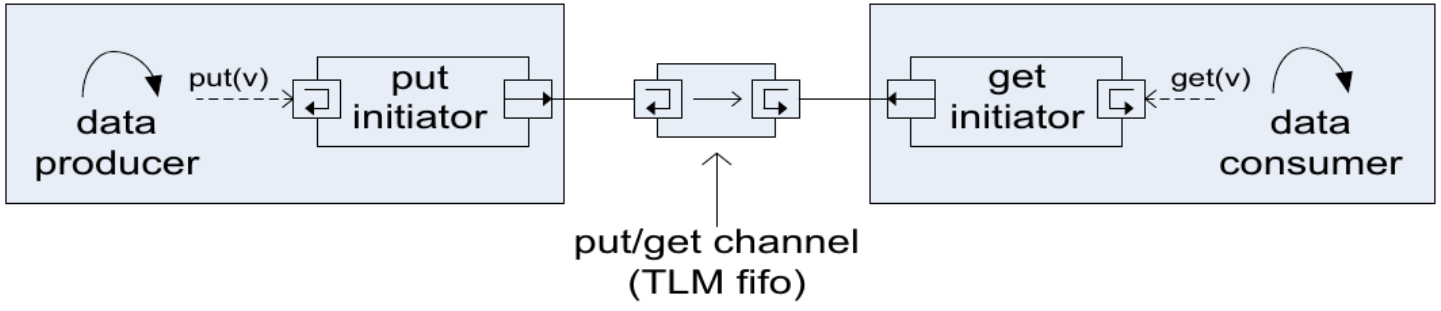
```
#pragma allocate_builtin_ram -sync_read
char mem[1024] ;
...
data_out.write(mem[add]);
wait() ;
```

- On ne peut pas ecrire et lire dans le même cycle...

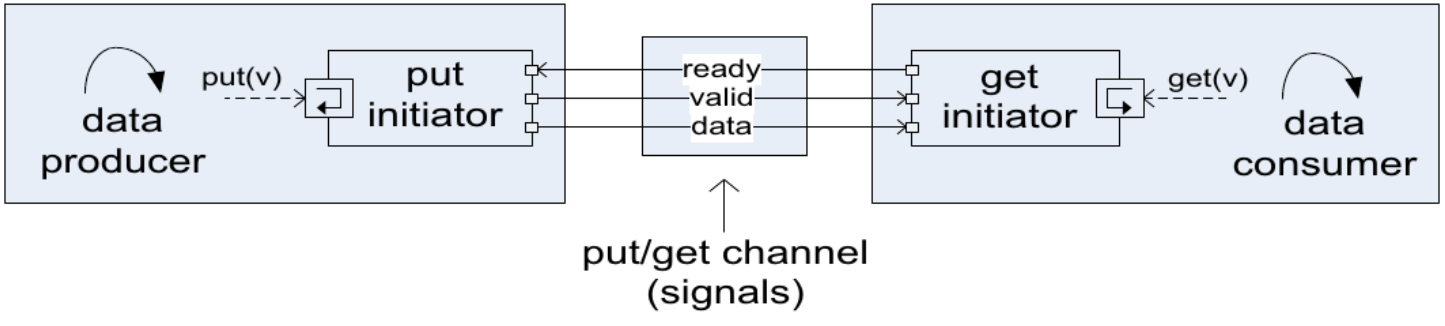
Ctos : Bibliothèques

Communications point à point : Les "Flex Channels"

- Modèle au niveau TLM

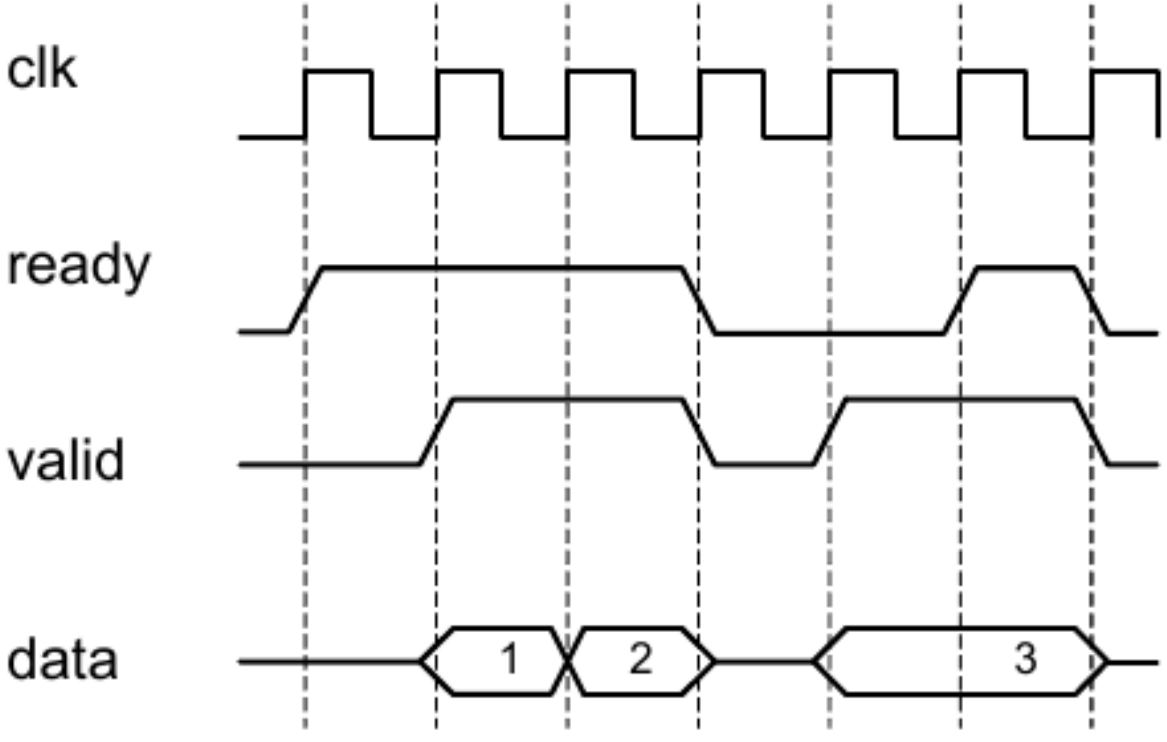


- Structure synthétisée au niveau signal



Ctos : Flex Channels

Le protocole Ready/Valid



- Débit maximum garanti (1 transfert par cycle)
- Latence exacte de 1 cycle

Ctos : Flex Channels

Terminologie des initiateurs

- Blocking initiator : Les fonctions contiennent des wait() : on attend un résultat
- Non-blocking : Les fonctions ne contiennent pas de wait() : réponse combinatoire
- May-block : Les fonctions peuvent générer une attente ou non : réponse combinatoire ou séquentielle
- Peek : On peut récupérer une donnée valide, sans la consommer.

Ctos : Blocking initiators

Un exemple de déclaration

```
SC_MODULE(DUT) {
    sc_in<bool> clk;
    sc_in<bool> nrst;
    b_get_initiator<char> din;
    b_put_initiator<char> dout;

    SC_CTOR(DUT)
      : clk("clk")
      , nrst("nrst")
      , din("din")
      , dout("dout")
    {
      SC_THREAD(process);
      sensitive << clk.pos();
      reset_signal_is(nrst,false);
      // Bind clock and reset signal to put/get channel internal logic
      din.clk_rst(clk,nrst) ;
      dout.clk_rst(clk,nrst) ;
      ...
```

Ctos : Blocking initiators

Un exemple d'utilisation

```
void process() {
    din.reset_get(); // Put/get initiators need to be reset.
    dout.reset_put();
    wait();
    while (1) {
        // Get a character (data element) ; this call will block if
        // the input channel is empty.
        char c = din.get();
        c = c - ('a' - 'A'); // Convert from lower to uppercase.;
        // We need a wait in case we did both get() and put()
        // in the current cycle (to avoid combinational loop)
        wait();
        // This call will block only if the output channel is full.;
        dout.put(c);
        //
        }
}
```

Ctos : Blocking initiators

Les déclarations et méthodes

```
//// PUT initiator ////
// Déclaration
b_put_initiator<type> name;
// Méthodes
virtual void reset_put();     // Reset the put side of the channel
virtual void put(const T &v); // Put item, waits until successfull

//// GET initiator ////
//Déclaration
b_get_initiator<type> name;
//Méthodes
virtual void reset_get(); // Reset the get side of the channel
virtual void get(T &t);   // Get item, waits until successfull
virtual T get();          // Get item, waits until successfull
```

Rappel : au moins un cycle d'attente.

Ctos : May Block initiators

Les déclarations et méthodes

```
//// PUT initiator ////
// Déclaration
put_initiator<type> name;
// Méthodes
virtual void reset_put();     // Reset the put side of the channel
virtual void put(const T &v); // Put item, waits until successfull

//// GET initiator ////
//Déclaration
get_initiator<type> name;
//Méthodes
virtual void reset_get(); // Reset the get side of the channel
virtual void get(T &t);   // Get item, waits until successfull
virtual T get();          // Get item, waits until successfull
```

Rappel : de 0 a plusieurs cycles d'attente

Ctos : Non Blocking initiators

Les déclarations et méthodes

```
//// PUT initiator ////
// Déclaration
nb_put_initiator<type> name;
// Méthodes
virtual void reset_put();  // Reset the put side of the channel
virtual bool nb_can_put() const; // Returns true id the channel is not full
virtual bool nb_put(const T &v); // Put item, return true if successfull

//// GET initiator ////
//Déclaration
nb_get_initiator<type> name;
//Méthodes
virtual void reset_get(); // Reset the get side of the channel
virtual bool nb_can_get() const; // Returns true id the channel is not empty
virtual bool nb_get(const T &v); // Get item, return true if successfull.
```

Rappel : réponse combinatoire (dans le cycle)