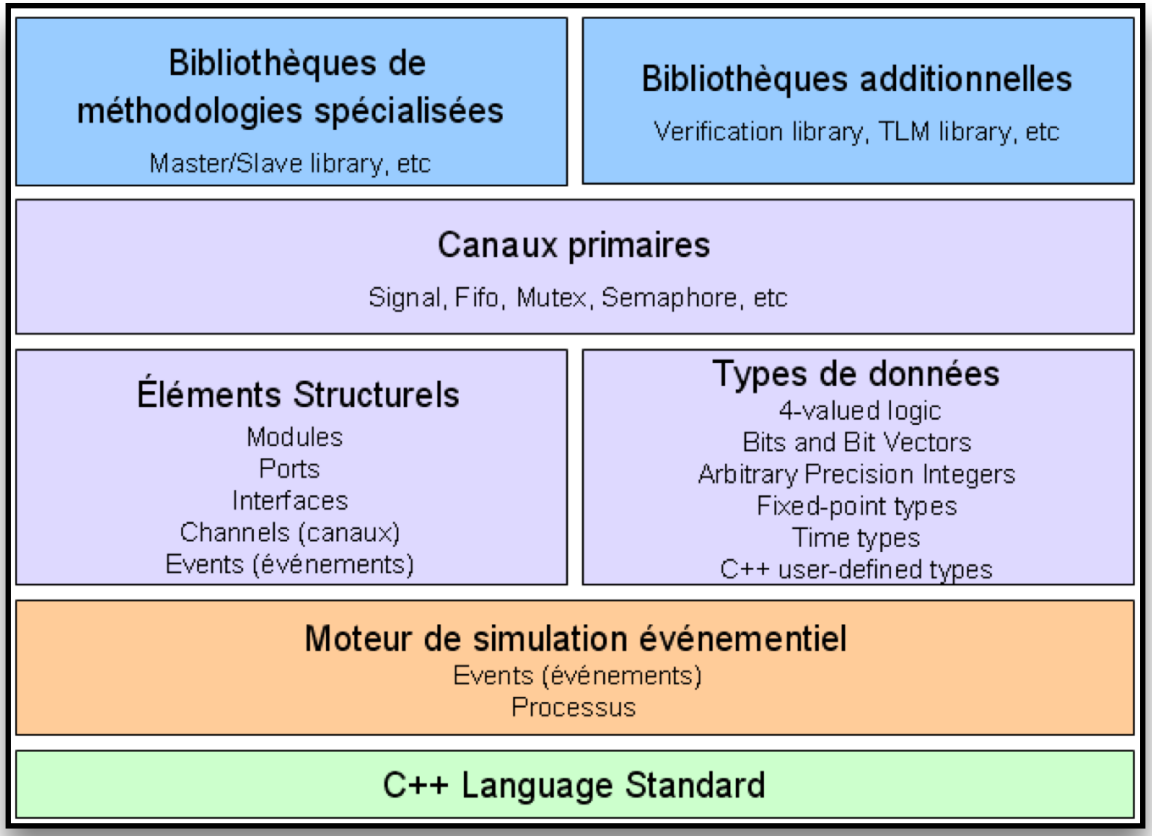
Les niveaux d'abstraction et leurs spécificités

Modèle algorithmique	Pas de notion de temps et pas de notion de transfert des données.	Vérification fonctionnelle Exploration algorithmique Vérification algorithmique
Modèle UnTimed Functional (UTF)		
Modèle Timed Functional (TF)	Notion de temps dans les processus et les transferts de données (modélisation gros grains)	Benchmarking des performances Analyse des architectures Développement des parties logicielles
Modèle Bus Cycled Accurate (BCA)		
Modèle Cycled Accurate (CA)	Les processus et les signaux sont au cycle près et au bit près.	Estimation précise des performances Développement des drivers Développement des micro-architectures
Niveau transfert de registres (RTL)		

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



SystemC program structure

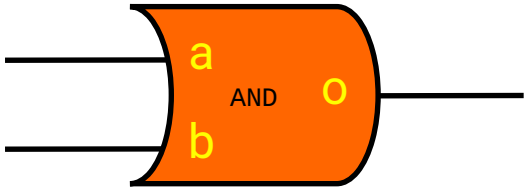
```
#include <systemc.h>
#include "and.h"
#include "or.h"
// etc..

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

  // 1: Instantiate gate objects
  ...
  // 2: Instantiate signal objects
  ...
  // 3: Connect the gates to signals
  ...

  // 4: specify which values to print

  // 5: put values on signal objects

  // 6: Start simulator run

}
```

- First a data structure is built that describes the circuit.
- This is a set of module (cell--)objects with a_ached pin objects.
- Signal objects tie the pins together.

- Then the simulation can be started.
- The simulation needs:
  - input values
  - the list of pins that is to reported.

A 2-input and-gate class in SystemC

This include file contains all systemc functions and base classes.



Instantiates the input pins a and b. They carry boolean sygnals.
This object inherits all systemC properties of a pin. how this is actually implemented is hidden from us!

All systemC classes start with sc_

This sets up a class containing a module with a functionality.

```
#include <systemc.h>

SC_MODULE(AND2)
{
  sc_in<bool> a;  // input pin a
  sc_in<bool> b;  // input pin b

  sc_out<bool> o; // output pin o

  SC_CTOR(AND2)    // the ctor
    {
      SC_METHOD(and_process);
      sensitive << a << b;
    }

  void and_process() {
    o.write( a.read() && b.read() );
  }
};
```

Similarly, a boolean output pin called o

This stuff is executed during construction of an and object

Tells the simulator which function to run to evaluate the output pin

This is run to process the input pins.

Calls read and write member functions of pins.

This is the actual and!

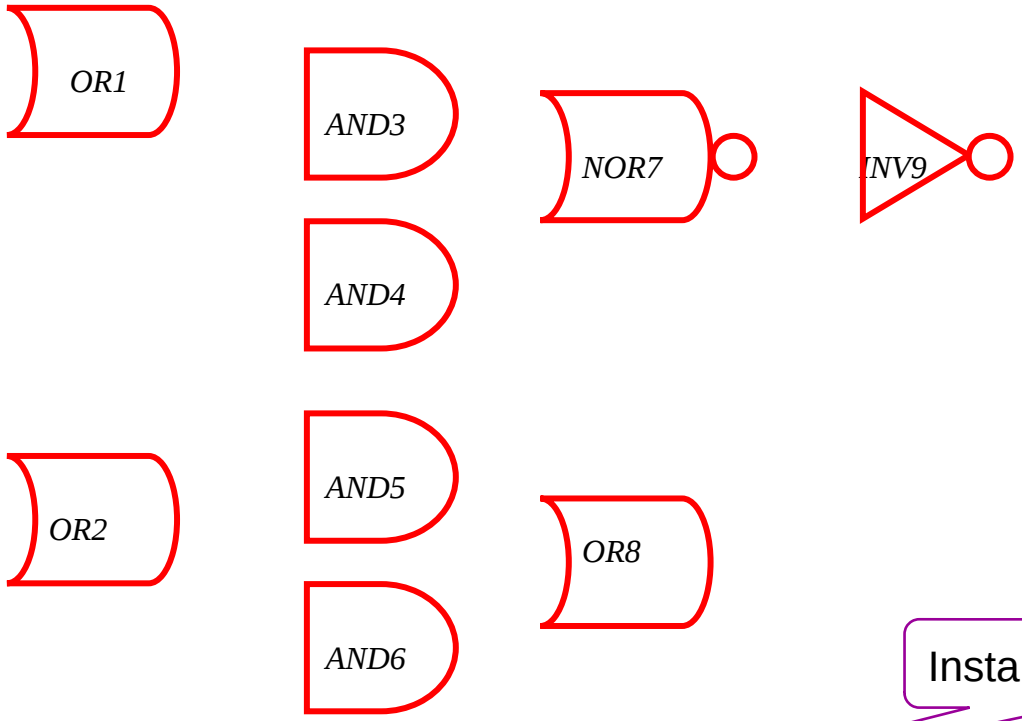
SystemC program structure

```
#include <systemc.h>
#include “and.h”
#include “or.h”
// etc..

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

  // 1: Instantiate gate objects
  …
  // 2: Instantiate signal objects
  …
  // 3: Connect the gates to signals
  …

  // 4: specify which values to print

  // 5: put values on signal objects

  // 6: Start simulator run

}
```

- First a data structure is built that describes the circuit.
- This is a set of module (cell-)objects with attached pin objects.
- Signal objects tie the pins together.

- Then the simulation can be started.
- The simulation needs:
  - input values
  - the list of pins that is to be reported.

Step 1: make the gate objects



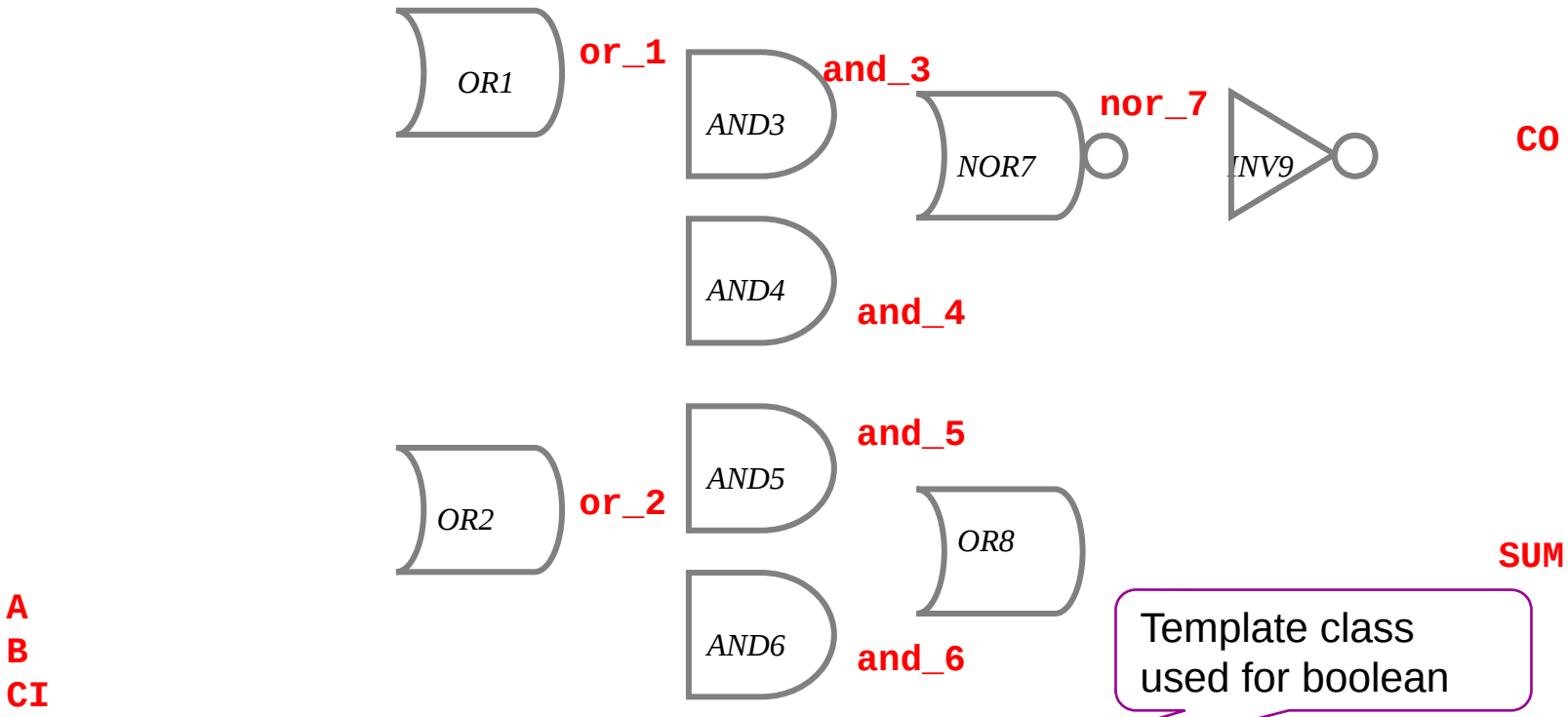
Instance name

Module type

```
// 1: instantiate the gate objects
OR2  or1("or1"), or8(“or8”);
OR3  or2(“or2”);
AND2 and3("and3"), and4("and4"), and5("and5");
AND3 and6("and6");
NOR2 nor7(“nor7");
INV  inv9(“inv9”);

// … continued next page
```

Name stored in instance

Step 2: make the signal objects

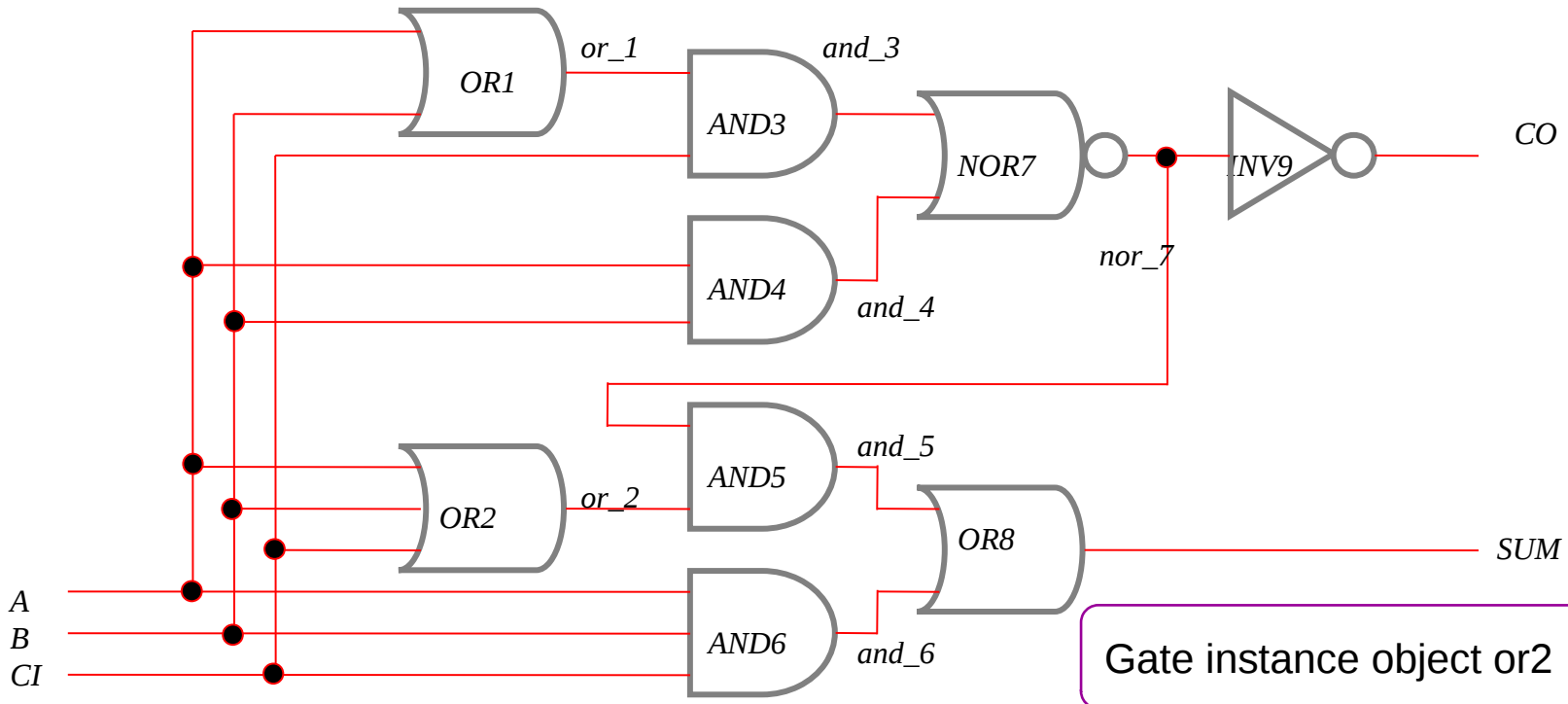


Template class used for boolean

Boolean signal

```
// … continued from previous page

// 2: instantiate the signal objects
sc_signal<bool> A, B, CI;        // input nets
sc_signal<bool> CO, SUM;         // output nets
sc_signal<bool> or_1, or_2, and_3, and_4;   // internal nets
sc_signal<bool> and_5, and_6, nor_7;        // internal nets

// … continued next page
```

Step 3: Connecting pins of gates to signals



Gate instance object or2

pin object o

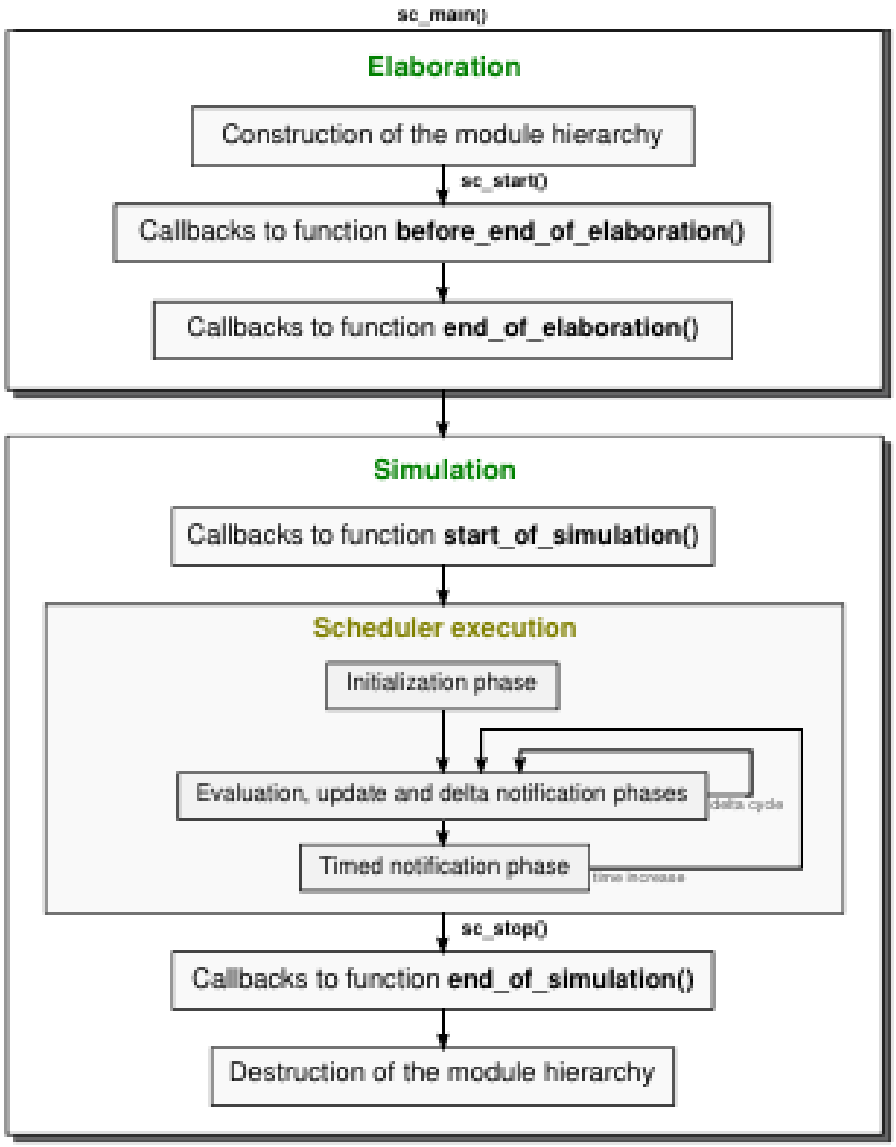
Signal net object or_2

```
// 3: Connect the gates to the signal nets
or1.a(A); or1.b(B); or1.o(or_1);
or2.a(A); or2.b(B); or2.c(CI); or2.o(or_2);
and3.a(or_1); and3.b(CI); and3.o(and_3);
and4.a(A); and4.b(B); and4.o(and_4);
and5.a(nor_7); and5.b(or_2); and5.o(and_5);
and6.a(A); and6.b(B); and6.c(CI); and6.o(and_6);
nor7.a(and_3); nor7.b(and_4); nor7.o(nor_7);
or8.a(and_5); or8.b(and_6); or8.o(SUM);
inv9.a(nor_7); inv9.o(CO);
// … continued next page
```

Simulation et traçage de signaux

```
  // .. continued from previous page
  sc_initialize();   // initialize the simulation engine

  // create the file to store simulation results
  sc_trace_file *tf = sc_create_vcd_trace_file("trace");

  // 4: specify the signals we’d like to record in the trace file
  sc_trace(tf, A, "A"); sc_trace(tf, B, "B"); sc_trace(tf, CI, “CI");
  sc_trace(tf, SUM, “SUM"); sc_trace(tf, CO, "CO");

  // 5: put values on the input signals
  A=0; B=0; CI=0;                 // initialize the input values
  sc_cycle(10);

  for( int i = 0 ; i < 8 ; i++ )  // generate all input combinations
    {
      A  = ((i & 0x1) != 0);      // value of A is the bit0 of i
      B  = ((i & 0x2) != 0);      // value of B is the bit1 of i
      CI = ((i & 0x4) != 0);      // value of CI is the bit2 of i
      sc_cycle(10);               // evaluate
    }

  sc_close_vcd_trace_file(tf);    // close file and we’re done
}
```

SystemC Elaboration and Simulation Semantics



Source : PhD thesis Liliana Andrade, HAL Id: tel-01344527, 2016

RISCV : 6 (+2) Types d’instructions

- R-TYPE : Les instructions de type R sont des instructions arithmétiques, utilisent uniquement des registres
- I-TYPE : Instructions où l’opérande 2 est de type immédiat
- B-type : Instructions de branchement conditionnel
- U-type : lui et auipc (add upper immediat to pc)
- J-type : Branchements inconditionnels
- S-TYPE : Instructions de type Store
- CSR-TYPE : Instructions CSR (control and status register)
- System-type : Instructions syst`eme (ecall/ebreak)

Risc-V : Jeu d’instructions

31 30 25	24 21 20	19 15	14 12	11 8	7	6 0	
funct7	rs2	rs1	funct3	rd		opcode	R-type
imm[11:0]		rs1	funct3	rd		opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]		opcode	S-type
imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]				rd		opcode	U-type
imm[20] imm[10:1]	imm[11]	imm[19:12]		rd		opcode	J-type

Risc-V : Jeu d'instructions

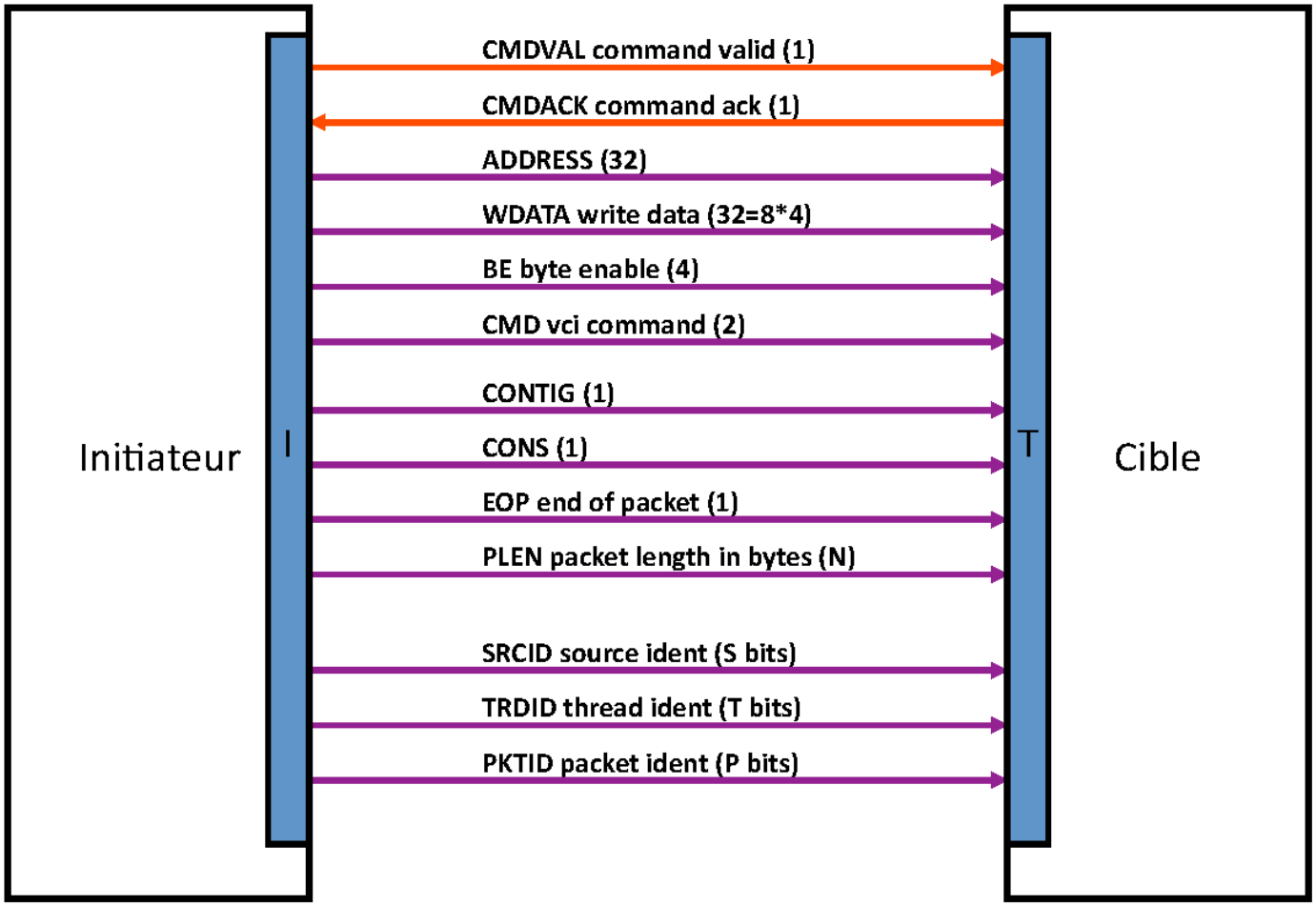
RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

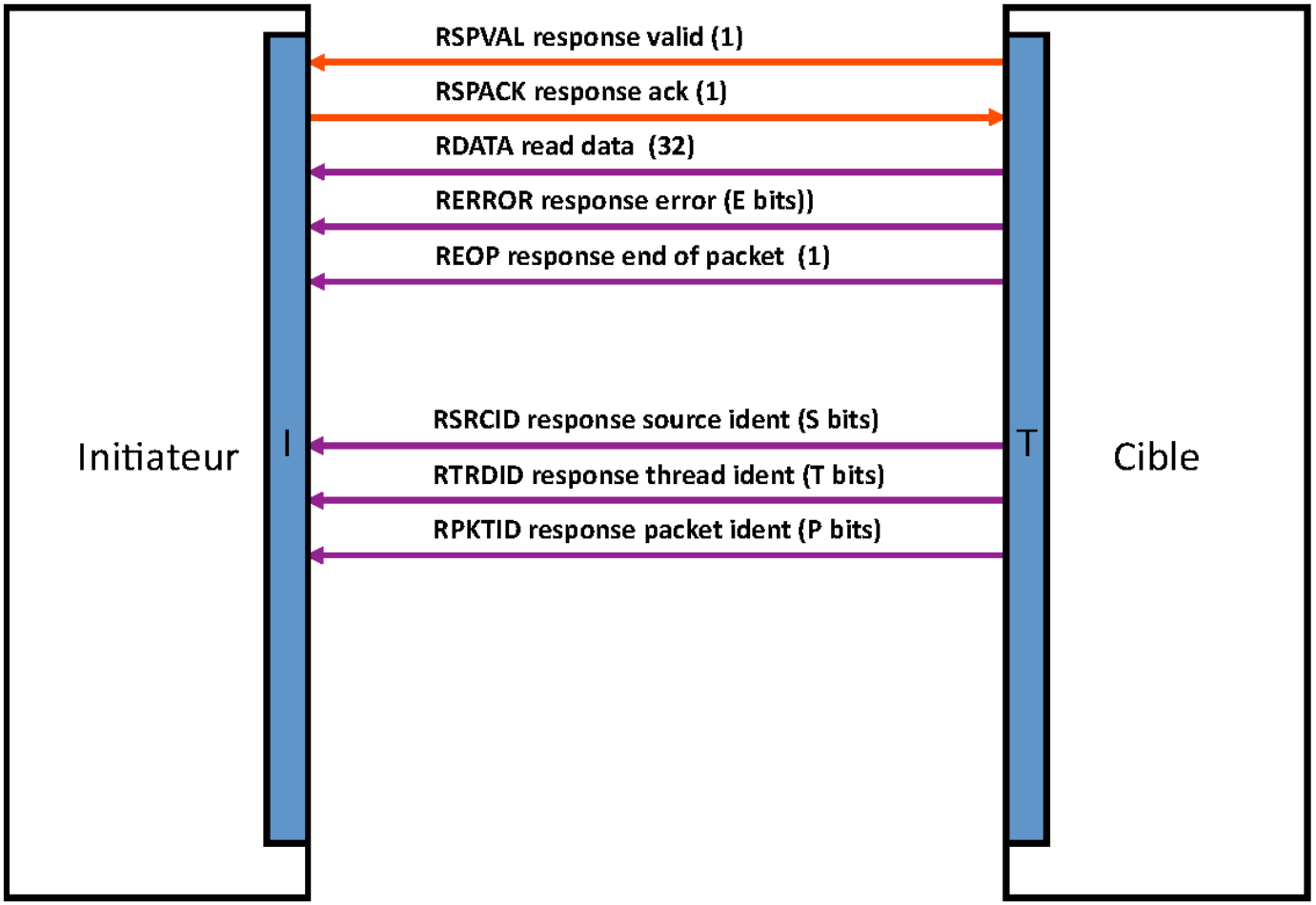
Les 2 SC_METHOD d'un composant SocLib

- **`transition(valeurs courantes des registres, entrées)`**, sensible au front montant de l'horloge calcule la prochaine valeur des registres.
- **`genMoore(valeurs courantes des registres)`**, sensible au front descendant de l'horloge, calcule les valeurs des sorties.
- **`genMealy(valeurs courantes des registres, entrées)`** (1 à N fonctions), sensible au front descendant et à certaines entrées, calcule les valeurs des sorties de Mealy.

Lien VCI, commandes



Lien VCI, réponses



Topcell Soclib pédagogique

system.cpp (1)

```
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <signal.h>

#include "shared/soclib_mapping_table.h"
#include "shared/soclib_vci_interfaces.h"
#include "soclib_vci_simpleram.h"
#include "soclib_vci_iss.h"

#define CELLSIZE        4       // Data are 4 cells(=8bits) wide=32 bits
#define ERRSIZE         1       // Error size is 1 bit
#define PLENSIZE        1       //
#define CLENSIZE        1       //
#define TRDIDSIZE       1       // TRDID unused
#define PKTIDSIZE       1       // PKTID unused too

#define ADDRSIZE     32
#define SRCIDSIZE    8
```

system.cpp (2)

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

```
   sc_clock signal_clk ("signal_clk");
   sc_signal < bool > signal_resetn ("signal_resetn");

   ADVANCED_VCI_SIGNALS <VCI_PARAM> link ("link");
```

Déclaration des signaux

```
   SOCLIB_VCI_ISS < VCI_PARAM > i0 ("i0");
   SOCLIB_VCI_SIMPLERAM < VCI_PARAM > t0 ("t0");
```

Déclaration des instances

```
   i0.CLK(signal_clk);
   i0.RESETN(signal_resetn);
   i0.VCI_INITIATOR(link);

   t0.CLK(signal_clk);
   t0.RESETN(signal_resetn);
   t0.VCI_TARGET(link);
```

Netlist

```
   sc_start(sc_core::sc_time(0, SC_NS));
   signal_resetn = false;
   sc_start(sc_core::sc_time(1, SC_NS));
   signal_resetn = true;

   sc_start();
```

Reset et lancement de la simulation

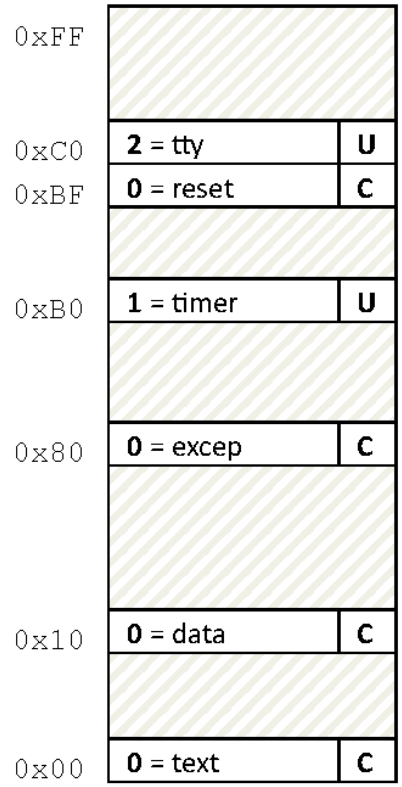
```
   return EXIT_SUCCESS;
};
```

Décodage d'adresses et masque de cachabilité

reset	0x**BF**C00000	=	**1011 1111** 11**00** 0000 0000 0000 0000 0000
text	0x**00**400000	=	**0000 0000** 01**00** 0000 0000 0000 0000 0000
excep	0x**80**000000	=	**1000 0000** 00**00** 0000 0000 0000 0000 0000
data	0x**10**000000	=	**0001 0000** 00**00** 0000 0000 0000 0000 0000
timer	0x**B0**200000	=	**1011 0000** 00**10** 0000 0000 0000 0000 0000
tty	0x**C0**200000	=	**1100 0000** 00**10** 0000 0000 0000 0000 0000
mask	0x00**3**00000	=	0000 0000 00**11** 0000 0000 0000 0000 0000

8 bits for target decoding

2 bits for cacheability



Platform address space = **Mapping table**

Exemple de fichier platform_desc

```
# -*- python -*-

import os

todo = Platform(
    'caba', 'caba_rv32_simple_top.cpp',
    uses = [
        Uses('caba:vci_xcache_wrapper',
             iss_t = 'common:gdb_iss',
             gdb_iss_t = 'common:rv32'
             ),
        Uses('caba:vci_simple_ram'),
        Uses('caba:vci_multi_tty'),
        Uses('caba:vci_simhelper'),
        Uses('caba:vci_vgmn'),
        Uses('common:elf_file_loader'),
        Uses('common:plain_file_loader'),
    ],
    cell_size = 4,
    plen_size = 8,
    addr_size = 32,
    rerror_size = 1,
    clen_size = 1,
    rflag_size = 1,
    srcid_size = 4,
    pktid_size = 4,
    trdid_size = 4,
    wrplen_size = 1
)
```

La table de segments

```
  // Mapping table

//soclib::common::Loader loader(SimParams.soft); //DG modifie

soclib::common::Loader loader("soft.bin"); // executable soft

  soclib::common::MappingTable maptabp(32, IntTab(8), IntTab(4), 0xc0000000);

  maptabp.add(Segment("boot",  0x00000000, 0x00000200, IntTab(0), true));
  maptabp.add(Segment("text",  0x60000000, 0x00100000, IntTab(0), true));
  maptabp.add(Segment("rodata",0x80000000, 0x01000000, IntTab(0), true));
  maptabp.add(Segment("fdt",   0xe0000000, 0x00001000, IntTab(0), false));

// device tree

  maptabp.add(Segment("tty",   0xd0200000, 0x00000040, IntTab(1), false));
  //maptabp.add(Segment("mem",   0x7f000000, 0x01000000, IntTab(2), false));
  maptabp.add(Segment("icu",   0xd2200000, 0x00001000, IntTab(3), false));
```

Fichier Objet ELF (1)

- Une séquence de code, données et symboles « relocable »

```
/* fun.c */
int
plus_3 ( int val )
{
    return val + 3;
}
```

```
fun.o: file format elf32-littlemips

Disassembly of section .text:

00000000 <plus_3>:
   0:  27bdfff8  addiu  sp,sp,-8
   4:  afbe0000  sw s8,0(sp)
   8:  03a0f021  move   s8,sp
   c:  afc40008  sw a0,8(s8)
  10:  8fc20008  lw v0,8(s8)
  14:  00000000  nop
  18:  24420003  addiu  v0,v0,3
  1c:  03c0e821  move   sp,s8
  20:  8fbe0000  lw s8,0(sp)
  24:  27bd0008  addiu  sp,sp,8
  28:  03e00008  jr ra
  2c:  00000000  nop
```

Relocable code
(.text section)

Fichier Objet ELF (exemple MIPS)

- Une séquence de code, données et symboles « relocable »

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000030 00  AX  0   0  4
  [ 2] .data             PROGBITS        00000000 000064 000000 00  WA  0   0  1
  [ 3] .bss              NOBITS          00000000 000064 000000 00  WA  0   0  1
  [ 4] .reginfo          MIPS_REGINFO    00000000 000064 000018 01      0   0  4
  [ 5] .pdr              PROGBITS        00000000 00007c 000020 00      0   0  4
  [ 6] .rel.pdr          REL             00000000 00036c 000008 08      9   5  4
  [ 7] .mdebug.abi32     PROGBITS        00000000 00009c 000000 00      0   0  1
  [ 8] .shstrtab         STRTAB          00000000 00009c 00004c 00      0   0  1
  [ 9] .symtab           SYMTAB          00000000 0002a0 000090 10     10   7  4
  [10] .strtab           STRTAB          00000000 000330 00003b 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Fichier Binaire ELF (exemple MIPS)

- Une séquence de code, données et symboles « non relocable »

```
/* main.c */
/* Global variable (uninitialized). */
int data0[256];

/* Global variable (initialized). */
int data1[256] = { 0 };

int
plus_3 ( int val );

int
main ( int argc, char *argv[] )
{
    /* Local variables (on stack). */
    int data2[32];
    int i;

    for ( i = 0; i < 256; ++i ) {
        data0[i] = plus_3 ( data1[i] )
    }

    return 0;
}
```

.bss section

.data section

function prototype

Main entry point (.text section)

```
...

00400154 <main>:
  400154:   27bdff58    addiu   sp,sp,-168
  400158:   afbf00a0    sw  ra,160(sp)
  40015c:   afbe009c    sw  s8,156(sp)
  400160:   afb00098    sw  s0,152(sp)
  400164:   03a0f021    move    s8,sp
  400168:   afc400a8    sw  a0,168(s8)
  40016c:   afc500ac    sw  a1,172(s8)
  400170:   afc00090    sw  zero,144(s8)

...

  4001f4:   27bd00a8    addiu   sp,sp,168
  4001f8:   03e00008    jr  ra
  4001fc:   00000000    nop

...
```

Fichier Binaire ELF (exemple MIPS)

- Une séquence de code, données et symboles « non relocable »

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
...
  [ 2] .init             PROGBITS        00400018 001018 000028 00  AX  0   0  1
  [ 3] .text             PROGBITS        00400040 001040 00026c 00  AX  0   0  4
  [ 4] .fini             PROGBITS        004002ac 0012ac 000020 00  AX  0   0  1
  [ 6] .ctors            PROGBITS        004402d0 0012d0 000008 00  WA  0   0  4
  [ 7] .dtors            PROGBITS        004402d8 0012d8 000008 00  WA  0   0  4
  [ 8] .jcr              PROGBITS        004402e0 0012e0 000004 00  WA  0   0  4
  [ 9] .data             PROGBITS        004402e4 0012e4 000008 00  WA  0   0  4
  [11] .bss              NOBITS          004402f0 0012ec 000820 00  WA  0   0  8
...

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x001018 0x00400018 0x00400018 0x002b8 0x002b8 R E 0x1000
  LOAD           0x0012d0 0x004402d0 0x004402d0 0x0001c 0x00840 RW  0x1000

 Section to Segment mapping:
  Segment Sections...
   00     .init .text .fini .eh_frame
   01     .ctors .dtors .jcr .data .bss
```

Le ldscript (exemple MIPS)

- décrit le placement des objets logiciels en mémoire

```
SECTIONS
{
   /* ... */
   . = 0x80000000;
   .excep : {
         *(.excep)
         *(.excep.*)
   }
   . = 0xbfc00000;
   .reset : {
         *(.reset)
         *(.reset.*)
   }
   . = 0x00400000;
   .text : {
      *(.text)
   }
```

```
. = 0x10000000;
.rodata : {
   *(.rodata)
   *(.rodata.*)
   . = ALIGN(4);
}
.data  : {
   *(.data)
}
.sdata : {
   *(.lit8)
   *(.lit4)
   *(.sdata)
}
_gp = .;
. = ALIGN(4);
_edata = .;
```

```
.sbss  : {
   *(.sbss)
   *(.scommon)
}
.bss   : {
   *(.bss)
   *(COMMON)
}
. = ALIGN(4);
_end = .;
_heap = .;
_stack = 0x10020000-16;
}
```

Travailler avec SoCLib Pro

```
massoc@mahler ~/opt-massoc/soclib/soclib/lib $ ls
address_decoding_table  dpp                 generic_cache  linked_access_buffer
address_masking_table   dynamic_buffer      generic_cam    loader
array_generic_fifo      elf_file_loader     generic_fifo   mapping_table
base_module             exception           generic_tlb    multi_write_buffer
circular_buffer         fb_controller       include        network_io
coff_file_loader        fd_poller           lazy_fifo      plain_file_loader
```

```
process_wrapper   vci_initiator_fsm                vci_target_fsm
pts_wrapper       vci_initiator_req                vloader
segment           vci_initiator_simple_read_req    write_buffer
tlmdt             vci_initiator_simple_req         xterm_wrapper
tlmt              vci_initiator_simple_write_req
tty_wrapper       vci_snooper
```

Travailler avec SoCLib Pro

```
massoc@mahler ~/opt-massoc/soclib $ ls
benchmarks  binary  COPYING.gpl  COPYING.lgpl  COPYING_tc.bsd  doc  README  soclib  utils
massoc@mahler ~/opt-massoc/soclib $
```

```
massoc@mahler ~/opt-massoc/soclib/soclib $ ls
communication  iss  lib  module  platform  soft
massoc@mahler ~/opt-massoc/soclib/soclib $
```

```
massoc@mahler ~/opt-massoc/soclib/soclib/iss $ ls
arm      iss   iss2_profiler   iss_boot_redirect  iss_memchecker  iss_simhelper  microblaze    mips    niosII  sparcv8  tms320c6x
gdb_iss  iss2  iss2_simhelper  ississ2            iss_profiler    lm32           microblaze10  mips32  ppc405  st231
massoc@mahler ~/opt-massoc/soclib/soclib/iss $
```

```
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component $ cd  vci_xcache_wrapper
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component/vci_xcache_wrapper $ ls
caba  tlmdt  tlmt
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component/vci_xcache_wrapper $ cd caba/
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba $ ls
doc  metadata  source
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba $ cd metadata/
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba/metadata $ ls
vci_xcache_wrapper.sd
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba/metadata $ cd ../source/
massoc@mahler ~/opt-massoc/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba/source $ ls
include  src
```

SystemC pour l’étude transactionnelle (TLM)

Les niveaux de modélisation vus jusqu’ici sont faits pour cacher certains détails :
- des portes
- la latence au-dessous du niveau cycle d’horloge

Par contre, ils sont tous “pin accurate” : connexions et registres sont visibles aux frontières structurelles, “handshake” est modélisé (exa : CABA communication VCI)

Le but sera maintenant de séparer les détails de communication entre modules de la communication entre ceux-ci

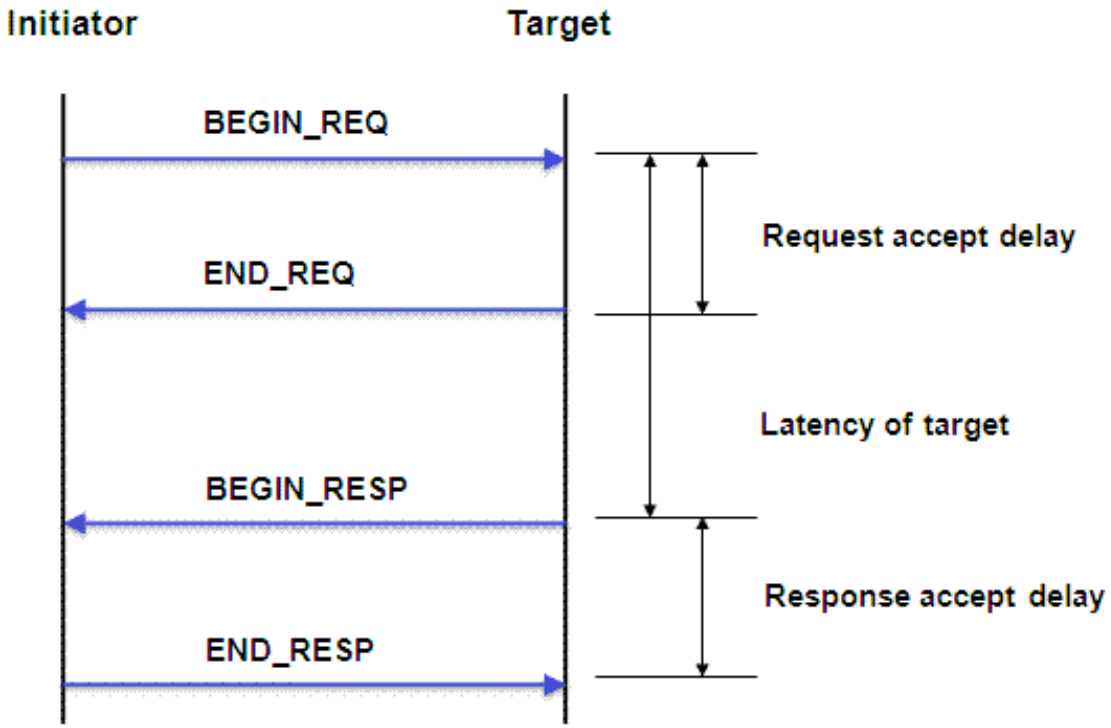
Tous les moyens de communication (FIFO, bus etc.) seront modélisés comme des canaux et présentés aux modules comme System C interface classes

Transaction = appel de fonctions appartenant à ces classes

Functionnalité non pas implémentation

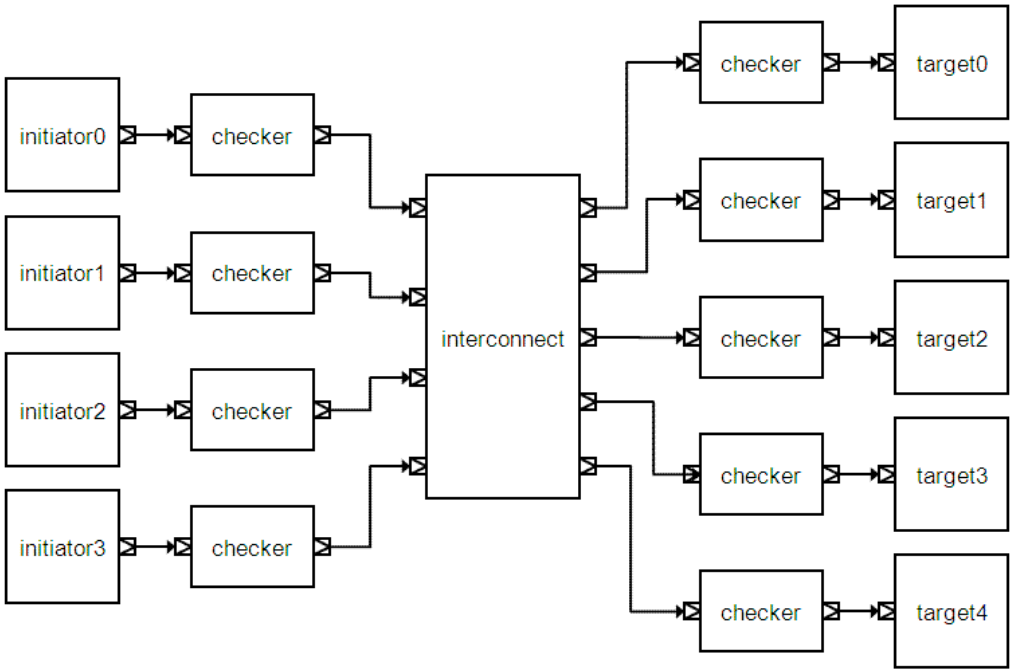
Les objectifs du niveau TLM

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



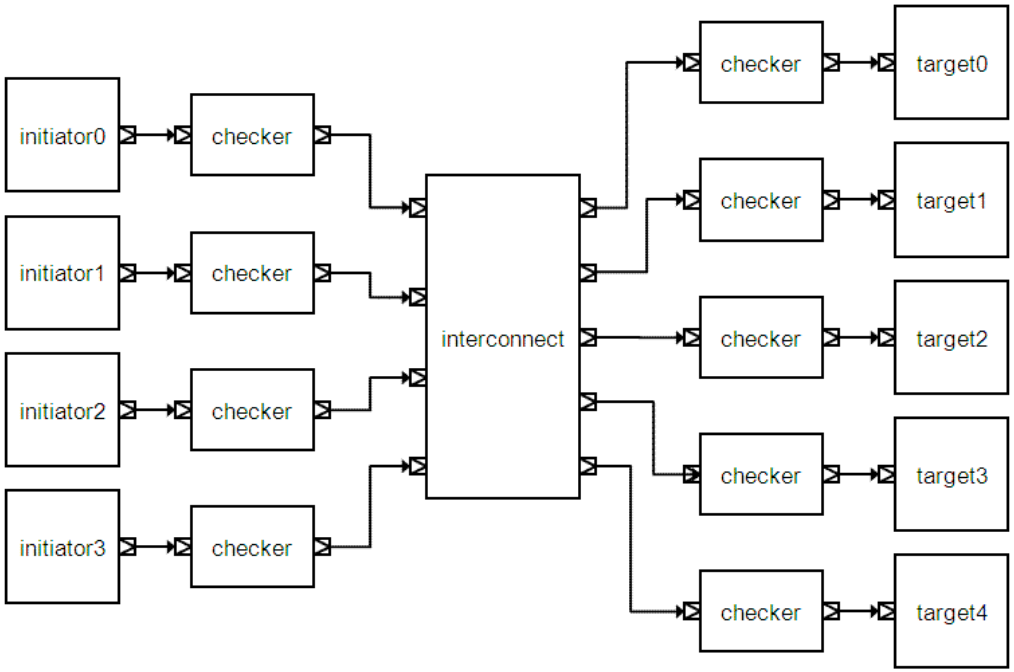
Composants de base

- 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.

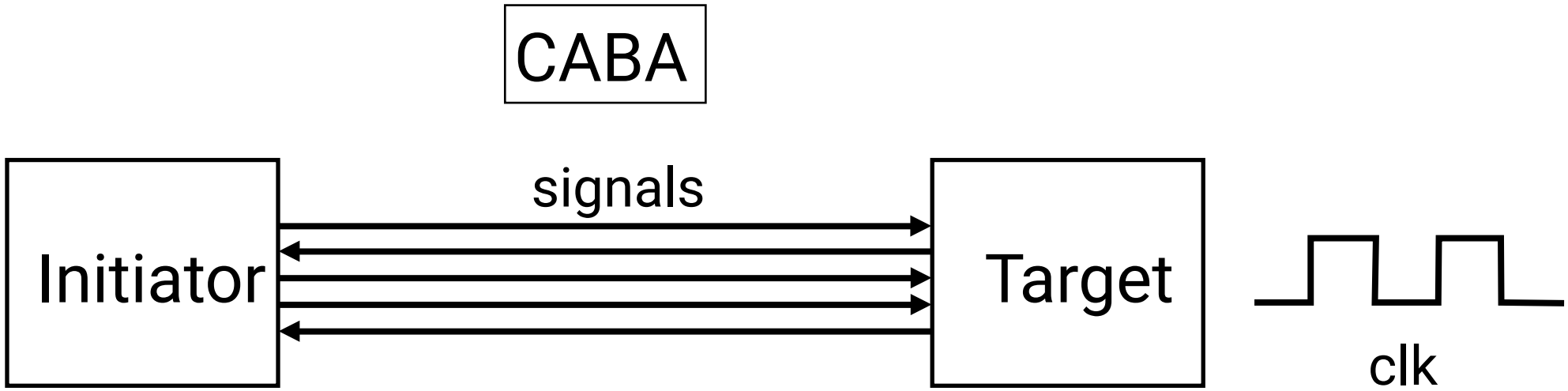


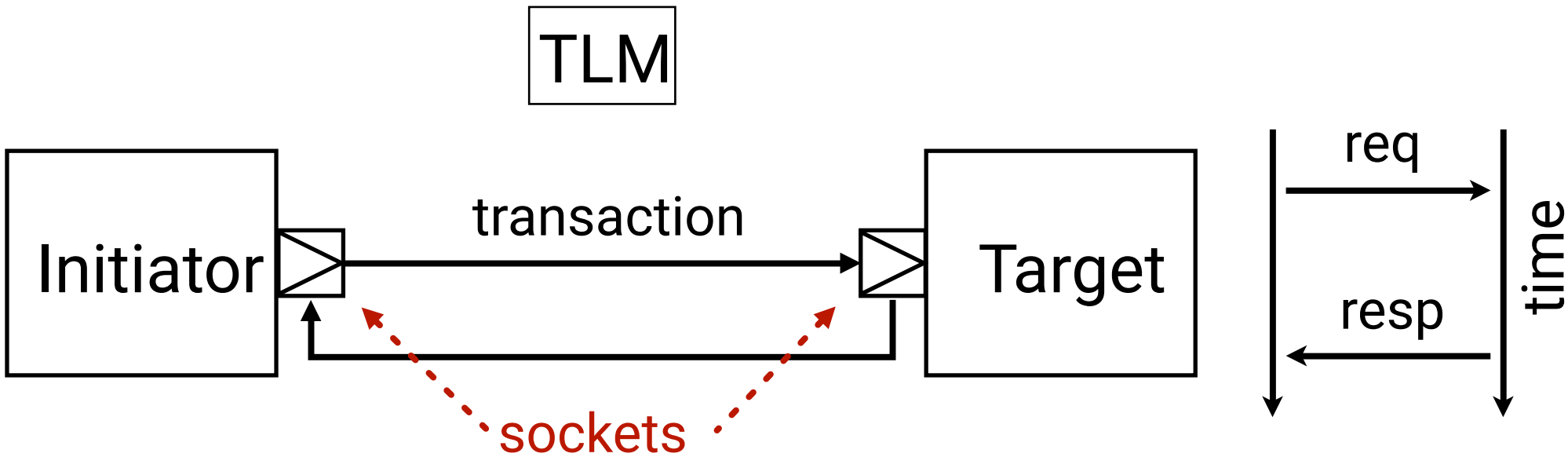
Paramètres adaptables / configurables

- 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



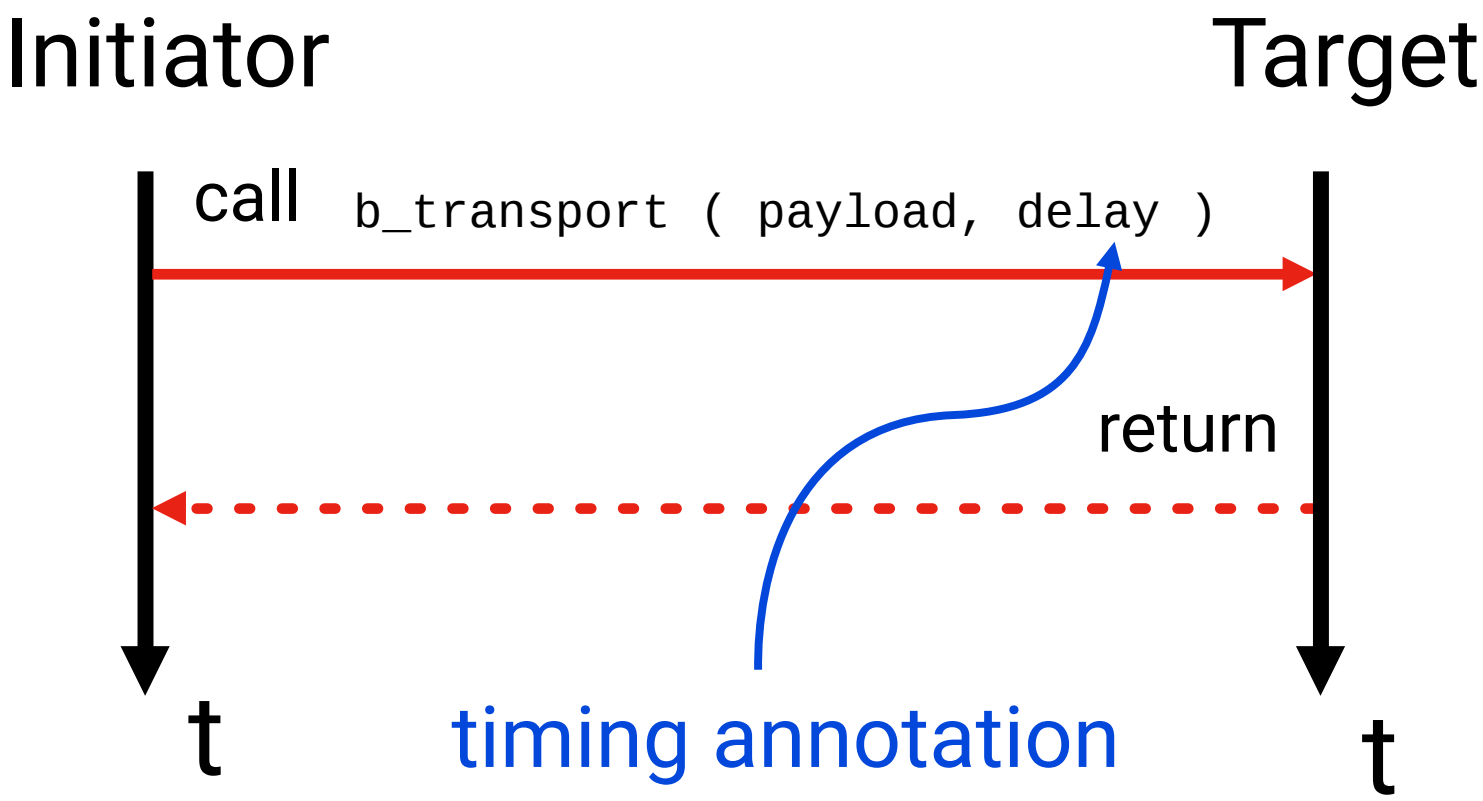
Basic Concepts





Loosely-Timed TLM

Blocking programming interface



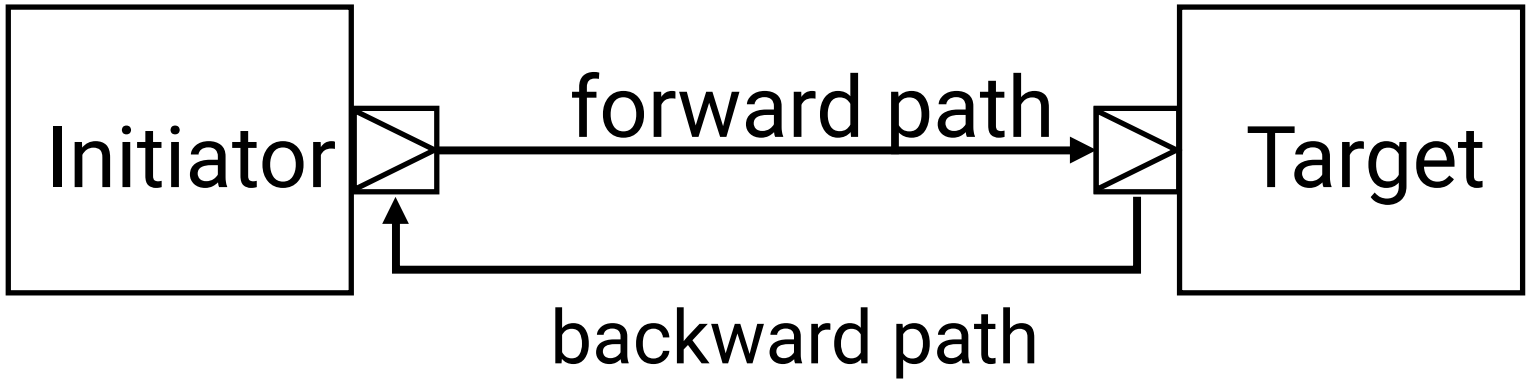
Approximate-Timed TLM

Instead of a single method call to `b_transport`, it implements paths and phases

There are two paths (i.e. two method calls):

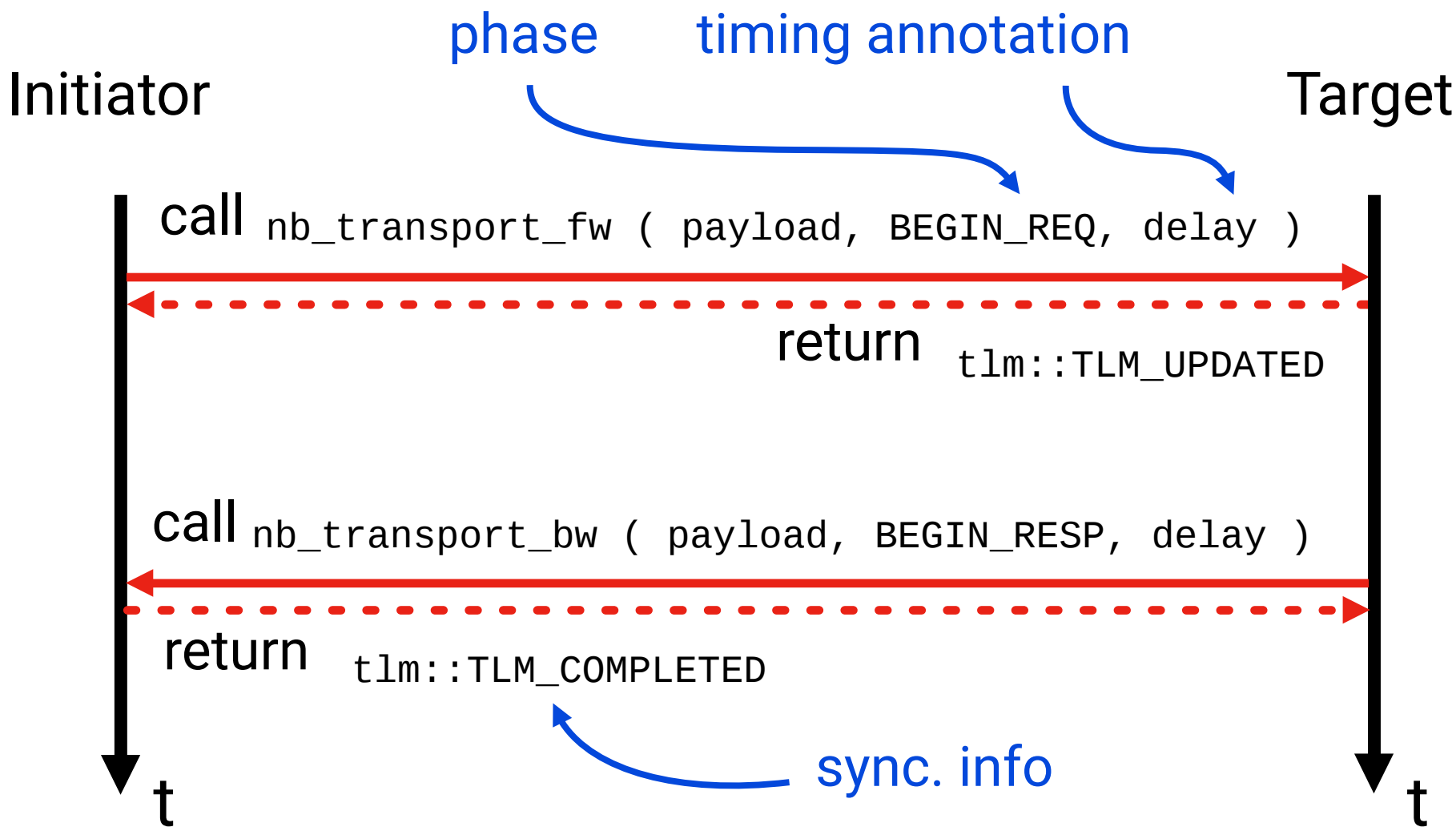
Forward path : initiator to target

Backward path : target to initiator



Approximate-Timed TLM

Non-blocking programming interface



TLM Payload API

```
class tlm::tlm_generic_payload
{
      void set_address ( sc_dt::uint64 );
      sc_dt::uint64 get_address () const;
      void set_read ();
      bool is_read () const;
      void set_write ();
      bool is_write () const;
      void set_data_ptr ( unsigned char * );
      unsigned char * get_data_ptr () const;
      void set_data_length ( unsigned int );
      unsigned int get_data_length () const;
      void set_response_status ( tlm::tlm_response_status );
      tlm::tlm_response_status get_response_status ();
/* ... */
};
```

Create an Initiator

Must derive from tlm::transport_bw_if

Bind TLM ports to the component

```
class initiator
       : public sc_core::sc_module
       , public tlm::tlm_transport_bw_if<>
 TLM interface{
public:
       tlm::tlm_initiator_socket<> socket;
       initiator ( sc_core::sc_module_name name )
{ socket.bind ( *this ); }

protected:
tlm::tlm_sync_enum nb_transport_bw ( ... ) { ... }
void invalidate_direct_mem_ptr ( ... ) { ... }
/* ... */
};
```

TLM interface

TLM interface implementation

Create a Target

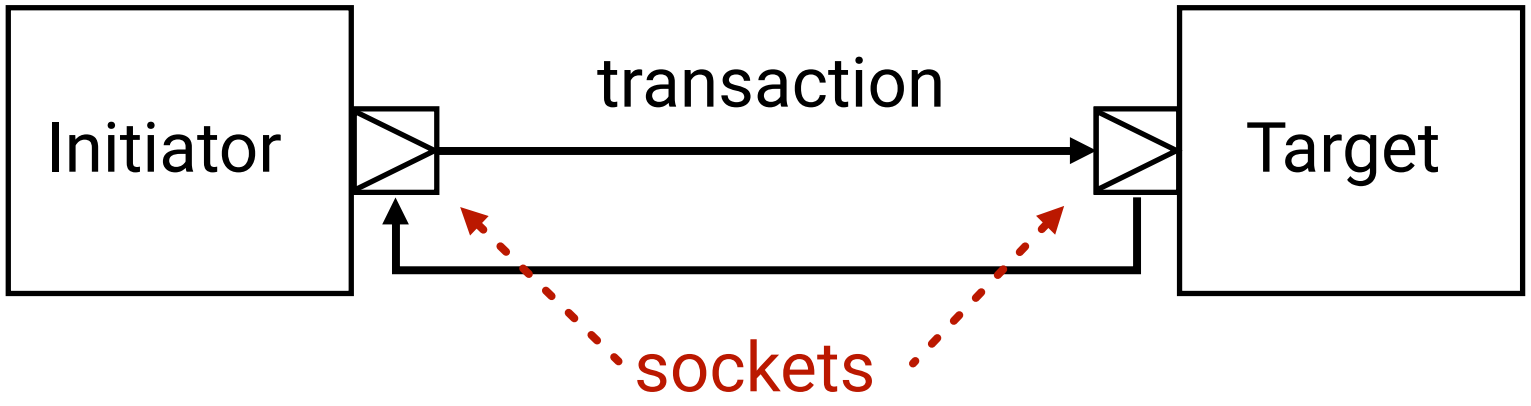
Must derive from tlm::transport_fw_if

Bind TLM ports to the component

```
class target
: public sc_core::sc_module
, public tlm::tlm_transport_fw_if<>

{
public:
tlm::tlm_target_socket<> socket;
initiator ( sc_core::sc_module_name name ) { socket.bind
(*this ); }

protected:
void b_transport ( ... ) { ... }
tlm::tlm_sync_enum nb_transport_fw ( ... ) { ... }
unsigned bool get_direct_mem_ptr int transport_dbg ( ... (
... ) {)... { ... }
 }

/* ... */
};
```

TLM interface implementation

Bind the Platform




```
int
sc_main ( int argc, char *argv[] )
{
        initiator init (``initiator'' );
Target tgt ( ``target'' );
init.socket.bind ( target.socket );
sc_start ( 100, SC_NS );
return 0;
}
```

VCI Transaction in TLM-T : Payload

Building a new VCI packet:
- create a generic payload and a soclib payload extension
- call the appropriate access functions on these two objects.

Example code for VCI read command:

```
tlm::tlm_generic_payload *payload_ptr = new tlm::tlm_generic_payload();

soclib_payload_extension *extension_ptr = new soclib_payload_extension(); ...

// set the values in tlm payload
payload_ptr->set_command(tlm::TLM_IGNORE_COMMAND);
payload_ptr->set_address(0x10000000]);
payload_ptr->set_byte_enable_ptr(byte_enable);
payload_ptr->set_byte_enable_length(nbytes);
payload_ptr->set_data_ptr(data);
payload_ptr->set_data_length(nbytes);
// set values in payload extension
extension_ptr->set_read();
extension_ptr->set_src_id(m_srcid);
extension_ptr->set_trd_id(0);
extension_ptr->set_pkt_id(pktid);
// set the extension to tlm payload payload_ptr->set_extension (extension_ptr); …
```