Techniques et outils pour la vérification de systèmes-sur-puces au niveau transactionnel

Matthieu Moy

Synchronous Languages and Reactive Systems
Laboratoire Verimag, INPG

System Platform Group
STMicroelectronics

9 Décembre 2005

Jury:	Gérard Michel	Président
	Stephen Edwards	Rapporteur
	Jean-Pierre Talpin	Rapporteur
	Florence Maraninchi	Directrice
	Laurent Maillet-Contoz	Examinateur

Techniques and Tools for the Verification of Systems-on-a-Chip at the Transaction Level

Matthieu Moy

Synchronous Languages and Reactive Systems
Verimag Laboratory, INPG

System Platform Group
STMicroelectronics

December 9th, 2005

Jury:	Gérard Michel	President
	Stephen Edwards	Reviewer
	Jean-Pierre Talpin	Reviewer
	Florence Maraninchi	Director
	Laurent Maillet-Contoz	Examiner

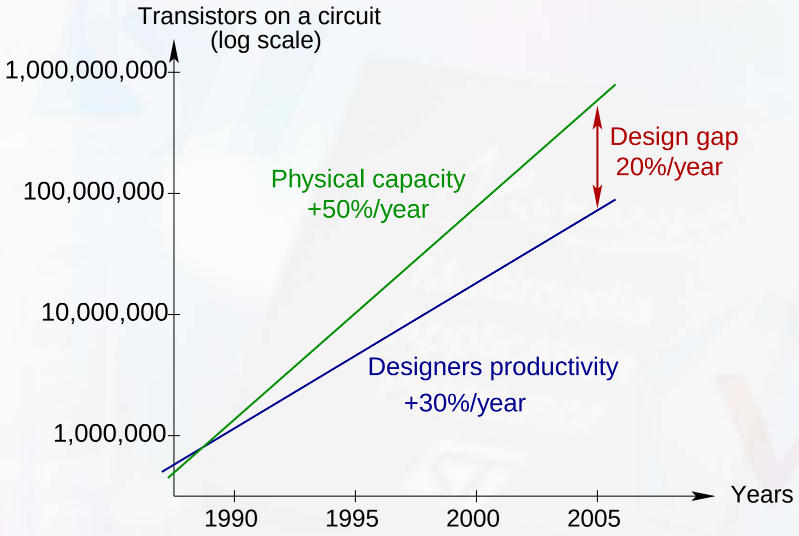
Objective of the Thesis

"Provide a connection from SystemC/TLM
to existing verification tools"

Outline

Outline

The Design Gap



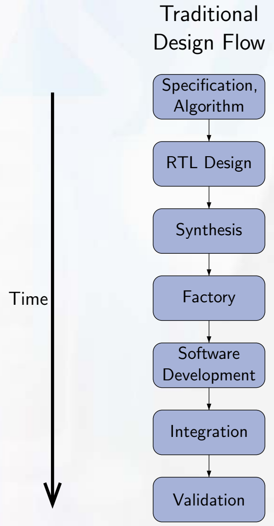
Hardware Vs Software

- Hardware
  - Fast
  - Power-efficient
- Software
  - Flexible
  - Reusable
  - Faster to write

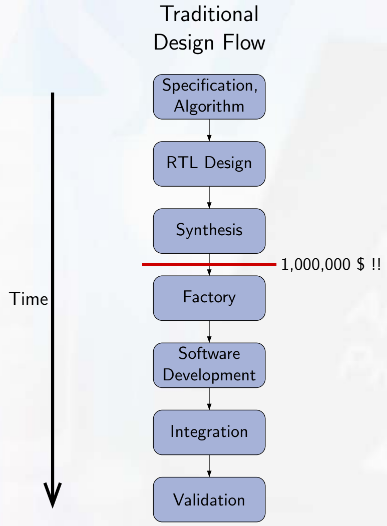
Hardware-Software Partitioning

- non-programmable ASIC: 100% Hardware
- General-purpose processors: 100% Software
- Systems-on-a-Chip (SoC): Mixture of Hardware and Software designed for each other

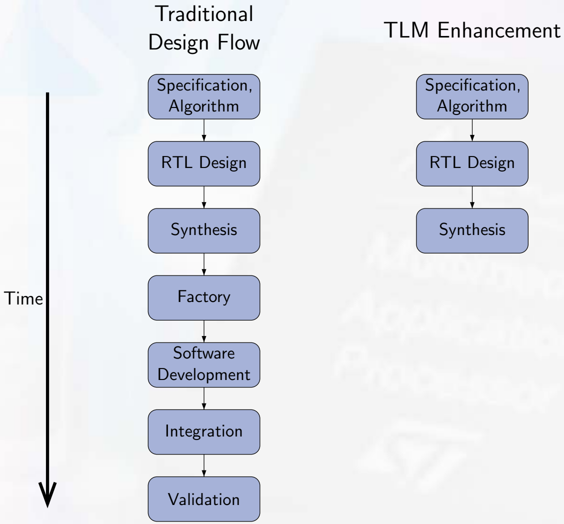
SoC Design Flow



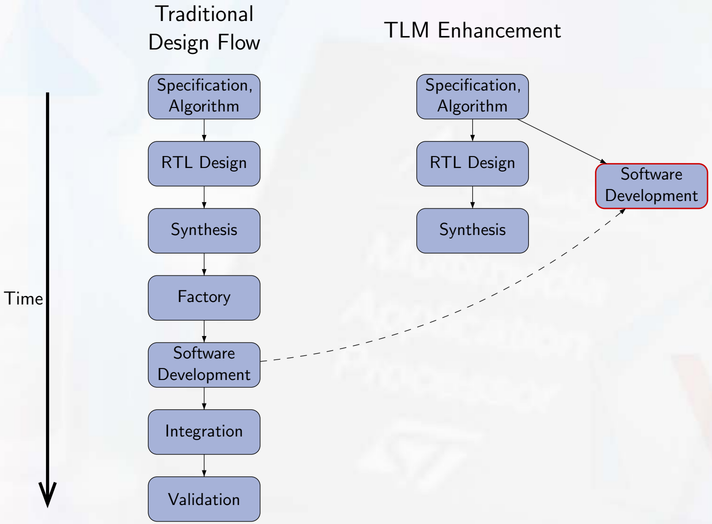
SoC Design Flow



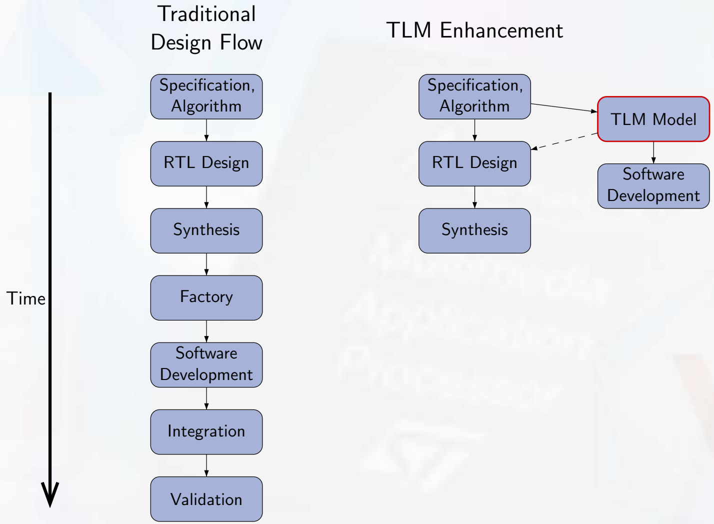
SoC Design Flow



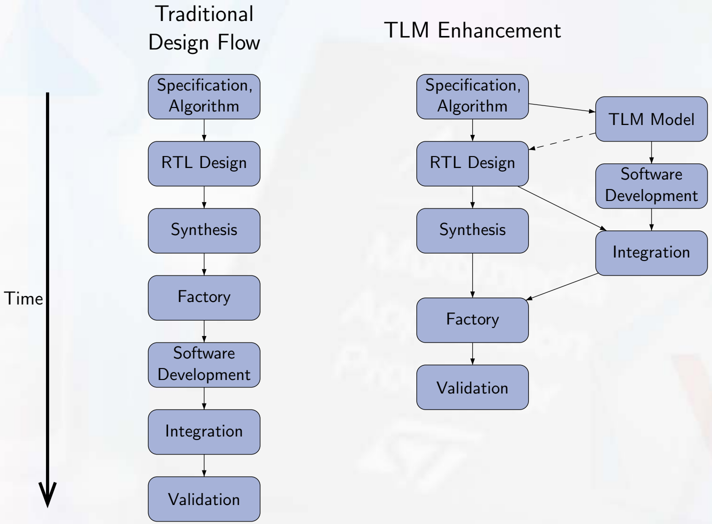
SoC Design Flow



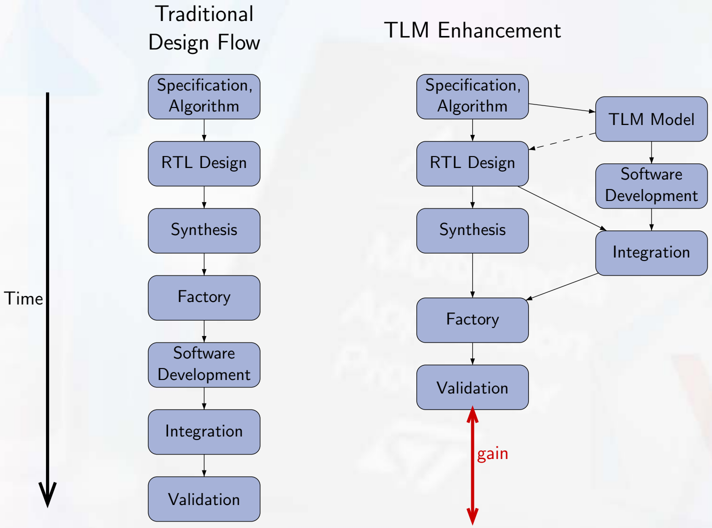
SoC Design Flow



SoC Design Flow



SoC Design Flow



The Transaction Level Model: Principles and Objectives

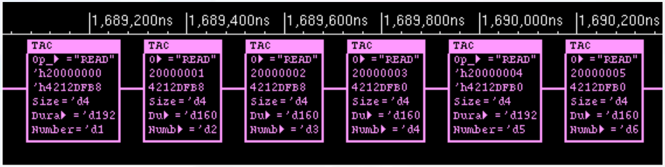
A high level of abstraction, that appear early in the design-flow

The Transaction Level Model: Principles and Objectives

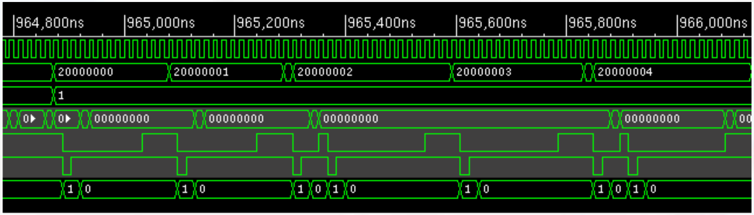
A high level of abstraction, that appear early in the design-flow

- A virtual prototype of the system, to enable
  - Early software development
  - Architecture exploration
  - Integration of components
- Abstract communication protocols and micro-architecture (remove implementation details from RTL)
  - Fast simulation ($\simeq$ 1000x faster than RTL)
  - Lightweight modeling effort ($\simeq$ 10x less than RTL)

The Transaction Level Model: Traces

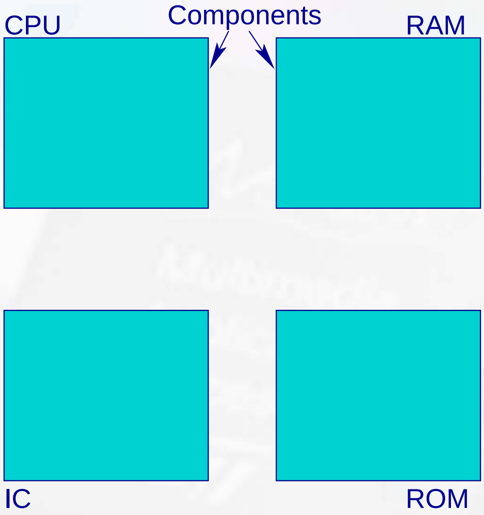

|1,689,200ns
|1,689,400ns
|1,689,600ns
|1,689,800ns
|1,690,000ns
|1,690,200ns
TAC op_▸ ="READ" 'h20000000 'h4212DFB8 Size='d4 Dura▸ ='d192 Number='d1
TAC o▸ ="READ" 20000001 4212DFB8 Size='d4 Du▸ ='d160 Numb▸ ='d2
TAC o▸ ="READ" 20000002 4212DFB8 Size='d4 Du▸ ='d160 Numb▸ ='d3
TAC o▸ ="READ" 20000003 4212DFB0 Size='d4 Du▸ ='d160 Numb▸ ='d4
TAC op_▸ ="READ" 'h20000004 'h4212DFB0 Size='d4 Dura▸ ='d192 Number='d5
TAC o▸ ="READ" 20000005 4212DFB0 Size='d4 Du▸ ='d160 Numb▸ ='d6

TLM: Transaction Level Model

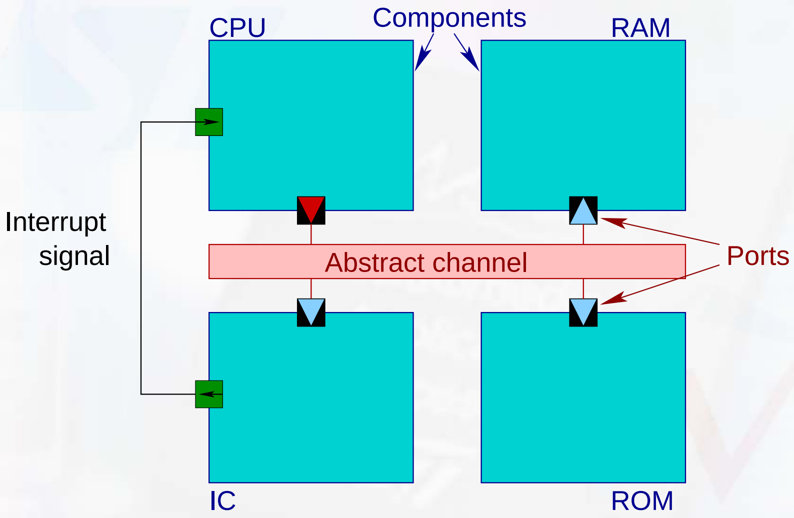


RTL: Register Transfer Level

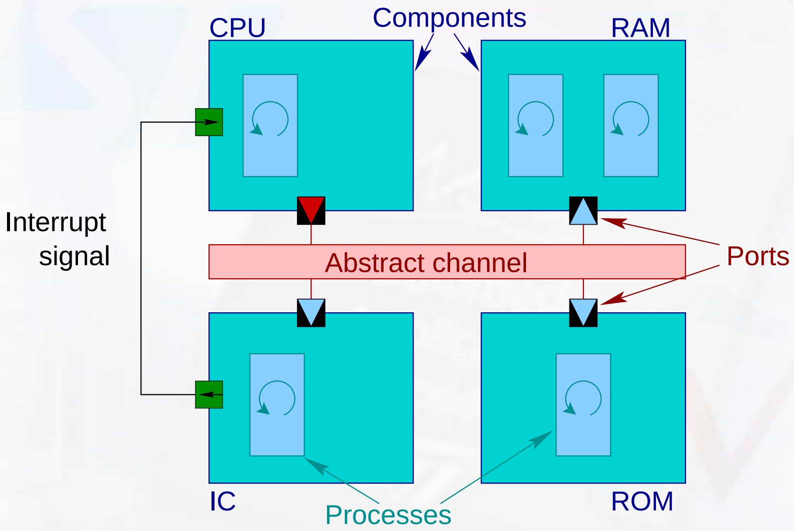
An Example TLM Model



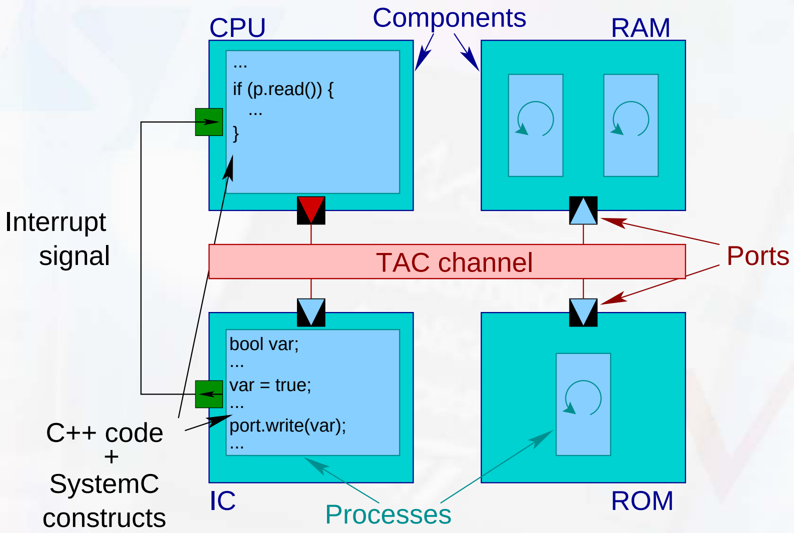
An Example TLM Model



An Example TLM Model



An Example TLM Model in SystemC



SystemC

- Useful to write Transaction Level Models

SystemC

- Useful to write Transaction Level Models
- Library for C++ ($\Rightarrow$ compilable with any C++ compiler)
- Provides
  - Some objects usable directly (`sc_signal`, `sc_event`, ...)
  - Some base classes to be implemented (`sc_module`, ...)
  - An execution kernel (including a scheduler)

Why SystemC?

- For the industry:
  - Good support for TLM and heterogeneous systems
    - Simulates fast
    - Hardware/Software
    - TLM/RTL/Gate-level
  - Many available tools
    - from CAD vendors
    - usual tools for C++ (debuggers, editors, lint, profilers, . . .)

Why SystemC?

- For the industry:
  - Good support for TLM and heterogeneous systems
    - Simulates fast
    - Hardware/Software
    - TLM/RTL/Gate-level
  - Many available tools
    - from CAD vendors
    - usual tools for C++ (debuggers, editors, lint, profilers, . . .)
- As a research objective:
  - Used in the industry
  - Many case-studies available (no need to translate them)
  - Work on a portion of the real design-flow

Transaction Level Modeling in SystemC

- SystemC provides the building blocks, but no high-level bus model
- Additional components are needed for TLM channels

Transaction Level Modeling in SystemC

- SystemC provides the building blocks, but no high-level bus model
- Additional components are needed for TLM channels
- STMicroelectronics developed several bus models
  - BASIC: an example channel
  - TAC: a TLM channel used in production
- Will hopefully be standardized

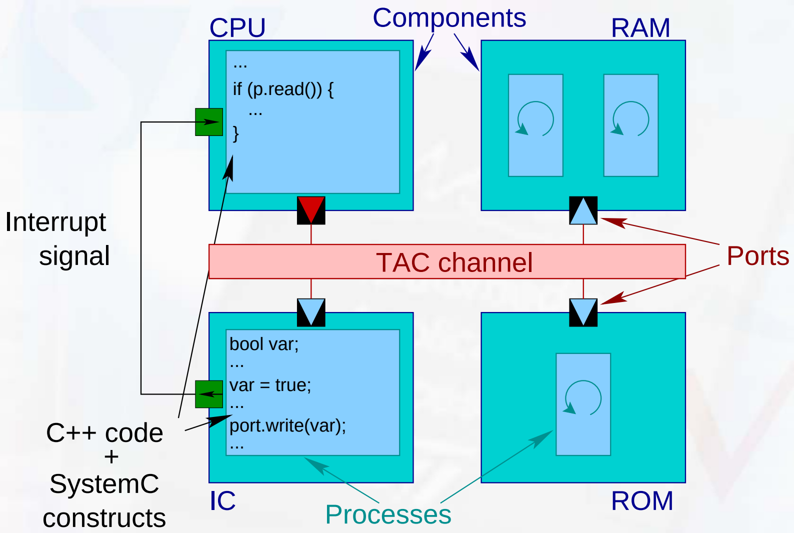
Execution of a SystemC Program

- Architecture and Behavior
  - No specific architecture description language
  - Arbitrary C++ can be used for both aspects

Execution of a SystemC Program

- Architecture and Behavior
  - No specific architecture description language
  - Arbitrary C++ can be used for both aspects
- Elaboration phase
  - Instantiate the components
  - Bind them together
- Simulation
  - Run the processes one by one
  - With a fixed architecture

An Example TLM Model in SystemC



Elaboration Phase: Build Architecture

```
int sc_main(int argc, char ** argv) {
   irq_controler  * ic = new irq_controler("IRQ");
   cpu * cpu = new cpu("CPU0");
   ram * ram = new ram("INT_RAM");
   rom * rom = new rom("ROM");
   tac_channel * channel = new tac_channel("CHANNEL");
   sc_signal<bool> sig;
   cpu->master_port.bind(channel->slave_port);
   ic->slave_port.bind(channel->master_port);
   ram->slave_port.bind(channel->master_port);
   rom->slave_port.bind(channel->master_port);
   ic->port.bind(sig);
   cpu->p.bind(sig);
   sc_start();
}
```

Elaboration Phase: Build Architecture

```
// Main function
int sc_main(int argc, char ** argv) {
   irq_controler  * ic = new irq_controler("IRQ");
   cpu * cpu = new cpu("CPU0");
   ram * ram = new ram("INT_RAM");
   rom * rom = new rom("ROM");
   tac_channel * channel = new tac_channel("CHANNEL");
   sc_signal<bool> sig;
   cpu->master_port.bind(channel->slave_port);
   ic->slave_port.bind(channel->master_port);
   ram->slave_port.bind(channel->master_port);
   rom->slave_port.bind(channel->master_port);
   ic->port.bind(sig);
   cpu->p.bind(sig);
   sc_start();
}
```

Elaboration Phase: Build Architecture

```
int sc_main(int argc, char ** argv) {
// Components Instantiation (creation of C++ objects)
   irq_controler  * ic = new irq_controler("IRQ");
   cpu * cpu = new cpu("CPU0");
   ram * ram = new ram("INT_RAM");
   rom * rom = new rom("ROM");
   tac_channel * channel = new tac_channel("CHANNEL");
   sc_signal<bool> sig;
   cpu->master_port.bind(channel->slave_port);
   ic->slave_port.bind(channel->master_port);
   ram->slave_port.bind(channel->master_port);
   rom->slave_port.bind(channel->master_port);
   ic->port.bind(sig);
   cpu->p.bind(sig);
   sc_start();
}
```

Elaboration Phase: Build Architecture

```
int sc_main(int argc, char ** argv) {
   irq_controler  * ic = new irq_controler("IRQ");
   cpu * cpu = new cpu("CPU0");
   ram * ram = new ram("INT_RAM");
   rom * rom = new rom("ROM");
   tac_channel * channel = new tac_channel("CHANNEL");
   sc_signal<bool> sig;
// Binding (link between objects)
   cpu->master_port.bind(channel->slave_port);
   ic->slave_port.bind(channel->master_port);
   ram->slave_port.bind(channel->master_port);
   rom->slave_port.bind(channel->master_port);
   ic->port.bind(sig);
   cpu->p.bind(sig);
   sc_start();
}
```

Elaboration Phase: Build Architecture

```
int sc_main(int argc, char ** argv) {
   irq_controler  * ic = new irq_controler("IRQ");
   cpu * cpu = new cpu("CPU0");
   ram * ram = new ram("INT_RAM");
   rom * rom = new rom("ROM");
   tac_channel * channel = new tac_channel("CHANNEL");
   sc_signal<bool> sig;
   cpu->master_port.bind(channel->slave_port);
   ic->slave_port.bind(channel->master_port);
   ram->slave_port.bind(channel->master_port);
   rom->slave_port.bind(channel->master_port);
   ic->port.bind(sig);
   cpu->p.bind(sig);
// Start simulation (let the kernel execute processes)
   sc_start();
}
```

Outline

Importance of TLM in the design flow

- No automatic synthesis from TLM to RTL
- A complement for RTL (not a replacement)

Importance of TLM in the design flow

- No automatic synthesis from TLM to RTL
- A complement for RTL (not a replacement)
- TLM serves as a reference model for RTL validation
- Embedded software is developed and tested partly on the TLM model

Importance of TLM in the design flow

- No automatic synthesis from TLM to RTL
- A complement for RTL (not a replacement)
- TLM serves as a reference model for RTL validation
- Embedded software is developed and tested partly on the TLM model

$\Rightarrow$ Although TLM models are not embedded in the chip, their validation is important

State of the Art

- Semantics of SystemC
  - Several papers for a semantics of RTL SystemC (very strict subset)
  - Usually do not take into account the real semantics of the scheduler
- Verification of TLM models
  - Recent research area
  - Almost nothing relevant when we started

State of the Art

- Semantics of SystemC
  - Several papers for a semantics of RTL SystemC (very strict subset)
  - Usually do not take into account the real semantics of the scheduler
- Verification of TLM models
  - Recent research area
  - Almost nothing relevant when we started
- In the meantime ...
  - Several tools for SystemC (front-ends, verification, lint, . . .)
  - Published work usually target a lower abstraction level than TLM as we use it in STMicroelectronics

Objectives of the Thesis

"Provide a connection from SystemC/TLM to existing verification tools"

Objectives of the Thesis

"Provide a connection from SystemC/TLM
to existing verification tools"

- Main difficulties
  - TLM mixes hardware and software
    $\Rightarrow$ Verification is undecidable
  - Abstractions will have to be made

Objectives of the Thesis

"Provide a connection from SystemC/TLM
to existing verification tools"

- Main difficulties
  - TLM mixes hardware and software
    $\Rightarrow$ Verification is undecidable
  - Abstractions will have to be made
- Main choices
  - Deal with real SystemC code
  - Fully automated tool-chain
  - As few abstractions as possible

Objectives of the Thesis

"Provide a connection from SystemC/TLM
to existing verification tools"

- Main difficulties
  - TLM mixes hardware and software
    $\Rightarrow$ Verification is undecidable
  - Abstractions will have to be made
- Main choices
  - Deal with real SystemC code
  - Fully automated tool-chain
  - As few abstractions as possible
- Consequences
  - We need a front-end to read the SystemC code
  - We need a semantics for SystemC
    $\Rightarrow$ Formal, Simple, Executable

Expressing properties

- Safety properties only (as opposed to liveness)

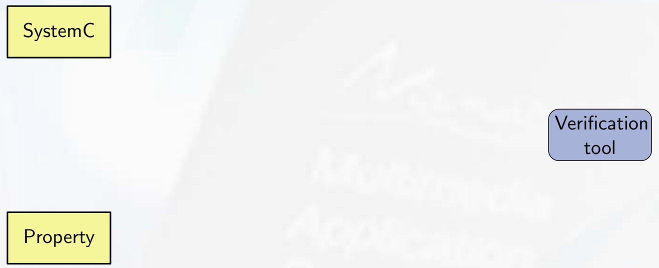
Expressing properties

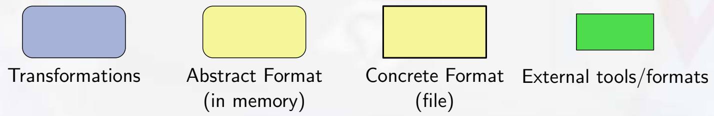
- Safety properties only (as opposed to liveness)
- No new specific language
- Express properties in C++/SystemC
  - `ASSERT(x.read() == true)`

Expressing properties

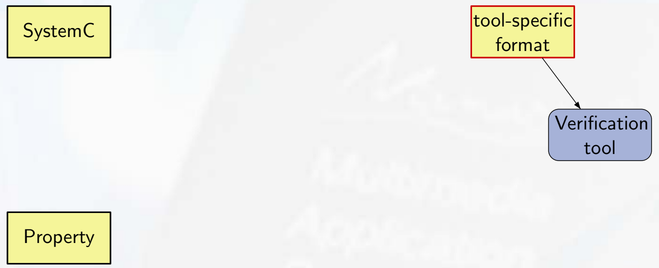
- Safety properties only (as opposed to liveness)
- No new specific language
- Express properties in C++/SystemC
  - `ASSERT(x.read() == true)`
- Use generic properties (things that you *usually* don't want)
  - Global dead-lock
  - Multiple write on a `sc_signal`
  - Process termination
  - Mutual exclusion

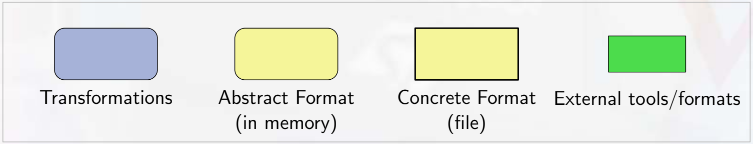
The LUSSY Tool Chain



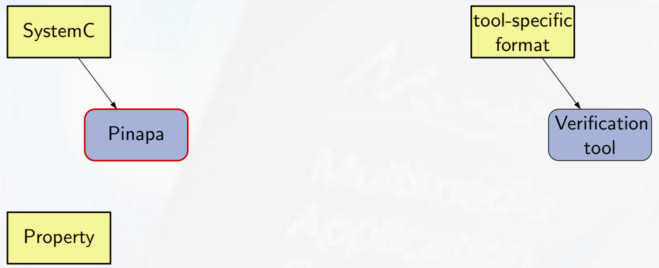


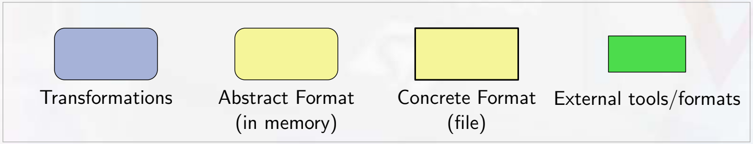
The LUSSY Tool Chain



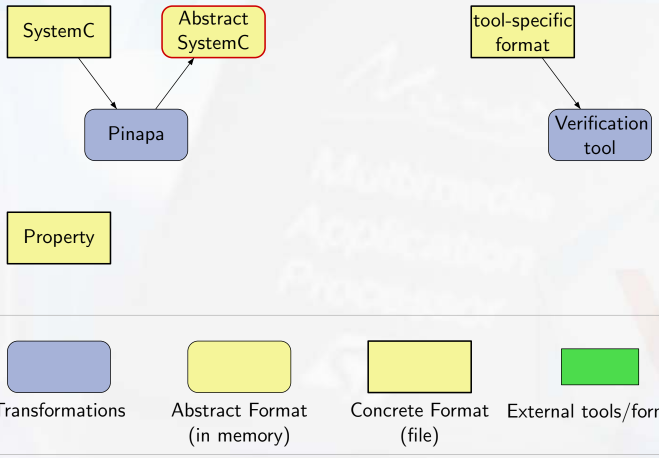


The LUSSY Tool Chain

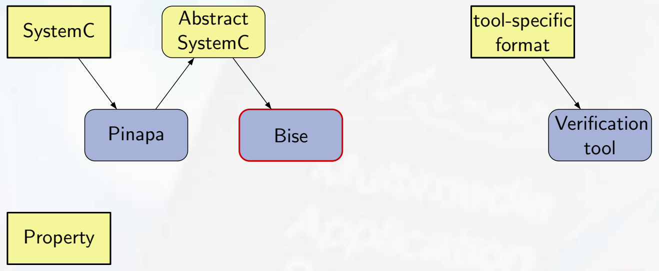


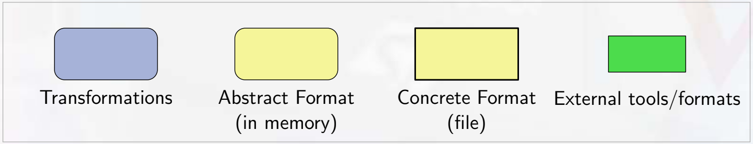


The LusSy Tool Chain

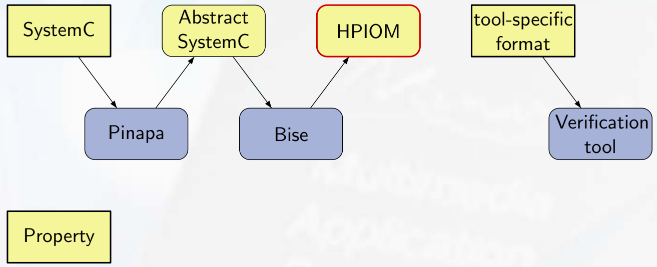


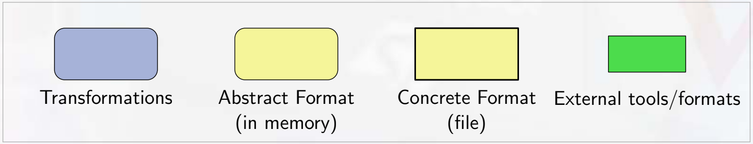
The LusSy Tool Chain



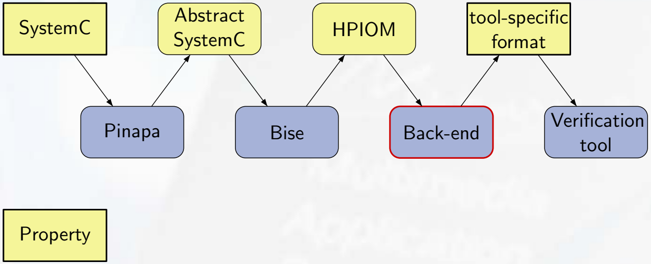


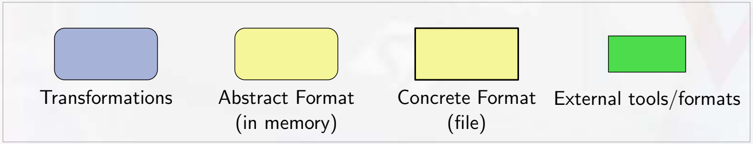
The LUSSY Tool Chain



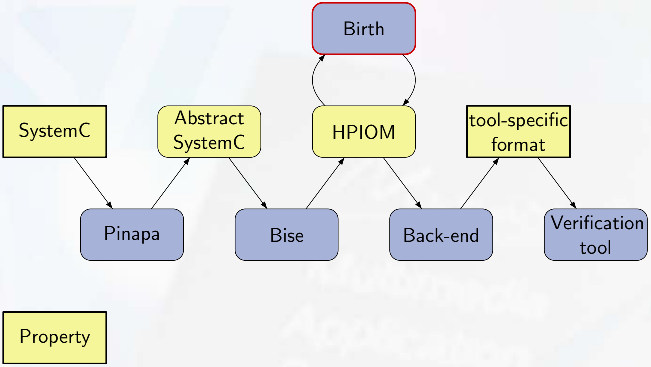


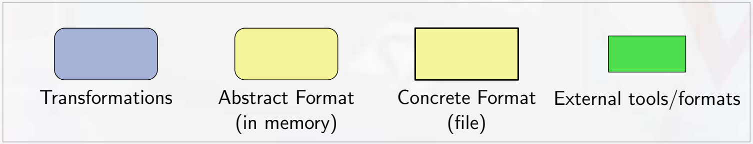
The LUSSY Tool Chain



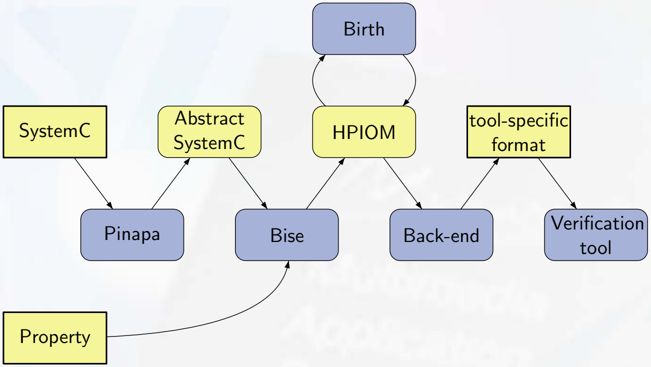


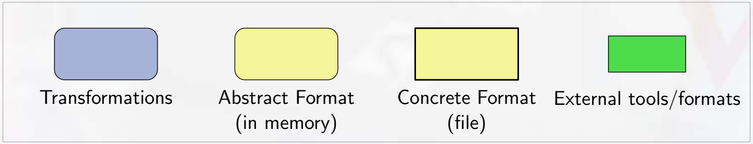
The LusSy Tool Chain



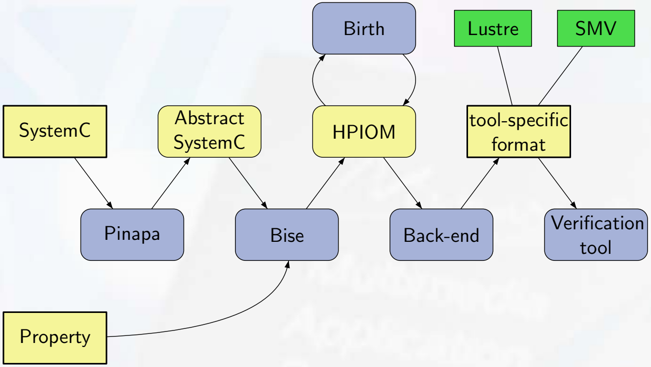


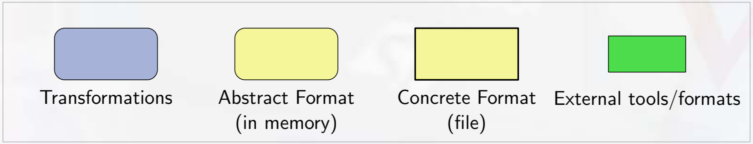
The LUSSY Tool Chain



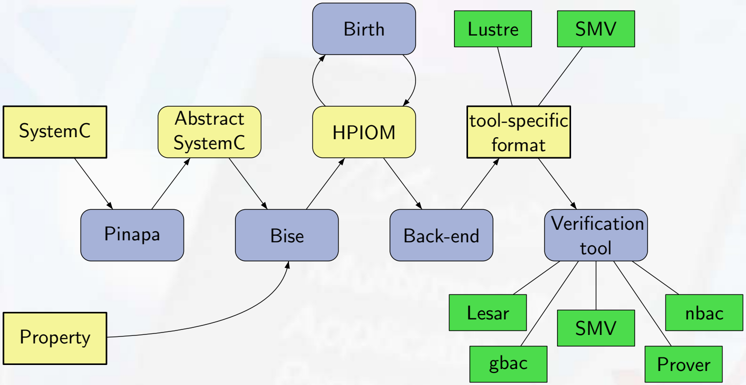


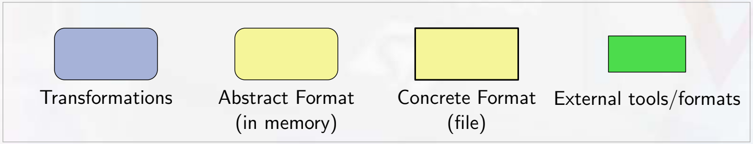
The LusSy Tool Chain



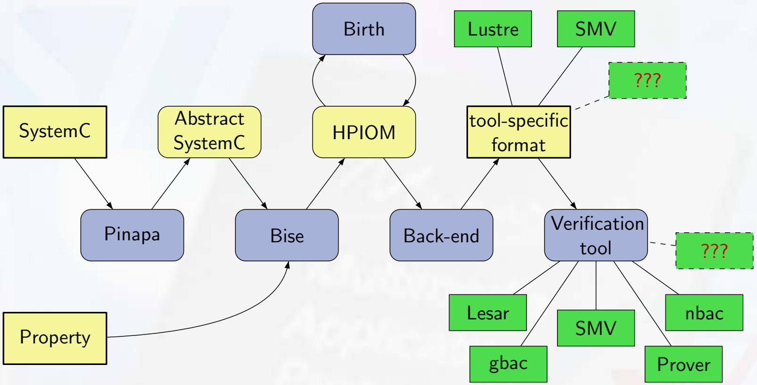


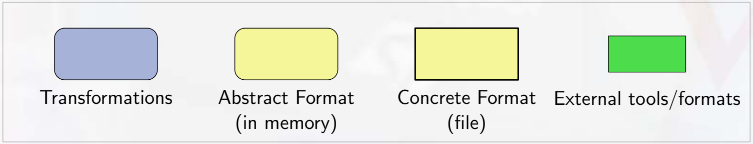
The LusSy Tool Chain



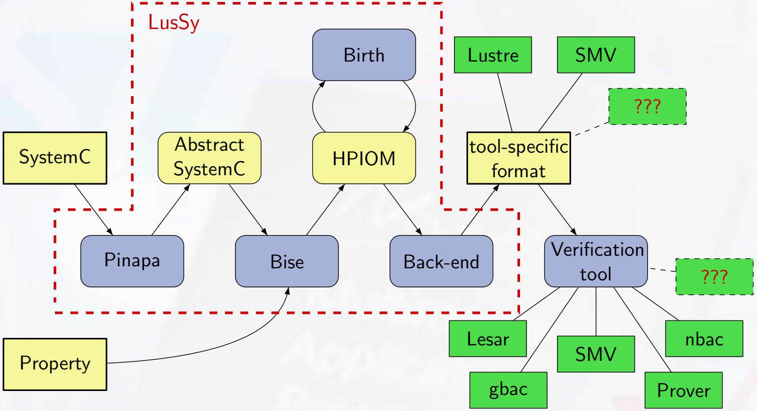


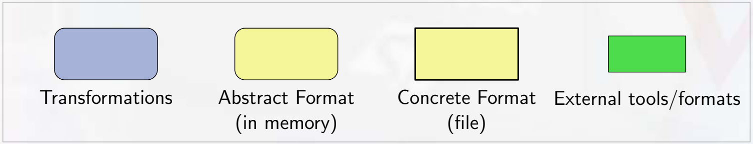
The LusSy Tool Chain





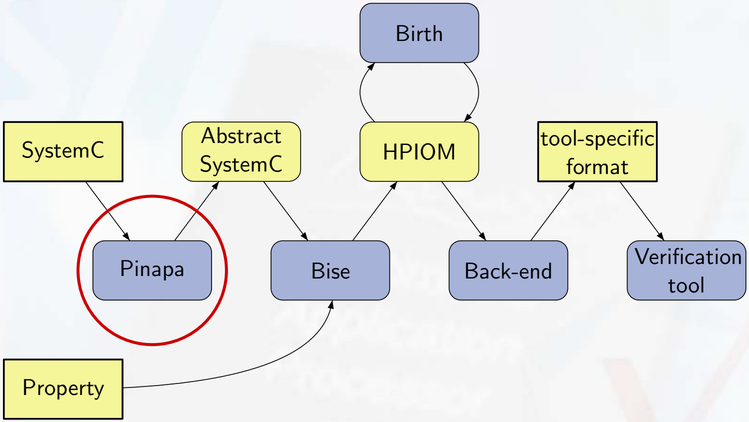
The LUSSY Tool Chain





Outline

PINAPA: Pinapa Is Not A PArser



Main Choices

There are many ways to “parse” SystemC:

- Write a grammar from scratch
  $\Rightarrow$ Needs to take all the C++ grammar into account!

Main Choices

There are many ways to “parse” SystemC:

- Write a grammar from scratch
  $\Rightarrow$ Needs to take all the C++ grammar into account!
- Use a C++ front-end, and nothing else
  $\Rightarrow$ Misses important information like architecture, built at run-time

Main Choices

There are many ways to “parse” SystemC:

- Write a grammar from scratch
  $\Rightarrow$ Needs to take all the C++ grammar into account!
- Use a C++ front-end, and nothing else
  $\Rightarrow$ Misses important information like architecture, built at run-time
- Use a C++ front-end, and recognize patterns in the elaboration phase
  $\Rightarrow$ Big limitation on the coding style of the elaboration

Main Choices

There are many ways to “parse” SystemC:

- Write a grammar from scratch
  ⇒ Needs to take all the C++ grammar into account!
- Use a C++ front-end, and nothing else
  ⇒ Misses important information like architecture, built at run-time
- Use a C++ front-end, and recognize patterns in the elaboration phase
  ⇒ Big limitation on the coding style of the elaboration
- Use a C++ front-end, and execute the elaboration phase to get the architecture
  ⇒ Much less limitation, lot of code reuse

Main Choices

There are many ways to “parse” SystemC:

- Write a grammar from scratch
  $\Rightarrow$ Needs to take all the C++ grammar into account!
  ParSyC, SystemPerl, sc2v, KaSCPar
- Use a C++ front-end, and nothing else
  $\Rightarrow$ Misses important information like architecture, built at run-time
- Use a C++ front-end, and recognize patterns in the elaboration phase
  $\Rightarrow$ Big limitation on the coding style of the elaboration
  SystemCXML, CoCentric
- Use a C++ front-end, and execute the elaboration phase to get the architecture
  $\Rightarrow$ Much less limitation, lot of code reuse
  Approach chosen for PINAPA

Static Vs Dynamic Information in SystemC Programs

Information in SystemC	C++ compiler	Pinapa
Static		
Dynamic		

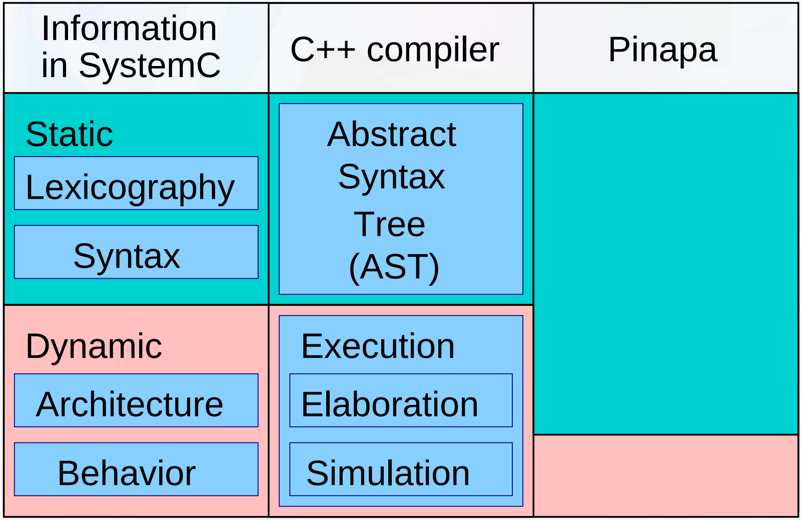
Static Vs Dynamic Information in SystemC Programs

Information in SystemC	C++ compiler	Pinapa
Static: Lexicography, Syntax		
Dynamic: Architecture, Behavior		

Static Vs Dynamic Information in SystemC Programs

Information in SystemC	C++ compiler	Pinapa
Static Lexicography Syntax	Abstract Syntax Tree (AST)	
Dynamic Architecture Behavior	Execution Elaboration Simulation	

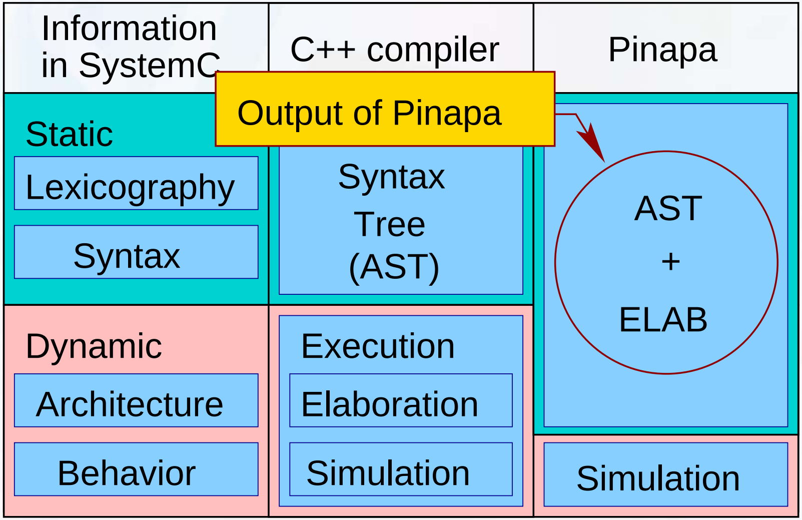
Static Vs Dynamic Information in SystemC Programs



Static Vs Dynamic Information in SystemC Programs

Information in SystemC	C++ compiler	Pinapa
Static	Abstract Syntax Tree (AST)	AST + ELAB
Lexicography		
Syntax		
Dynamic	Execution	
Architecture	Elaboration	
Behavior	Simulation	Simulation

Static Vs Dynamic Information in SystemC Programs



PINAPA: Key Ideas

- Syntax extraction (AST)
  - Easy if you have a C++ parser
  - $\Rightarrow$ Let’s reuse GCC
- Architecture extraction (ELAB)
  - Architecture is built at run-time
  - $\Rightarrow$ Let’s execute the elaboration of the program

PINAPA: Key Ideas

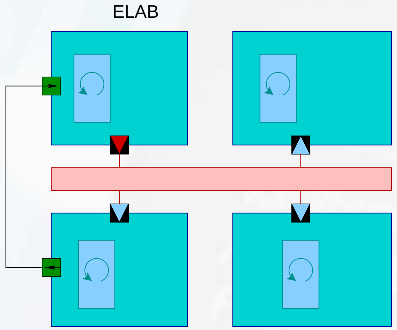
- Syntax extraction (AST)
  - Easy if you have a C++ parser
  - $\Rightarrow$ Let's reuse GCC
- Architecture extraction (ELAB)
  - Architecture is built at run-time
  - $\Rightarrow$ Let's execute the elaboration of the program

Then, what's difficult??

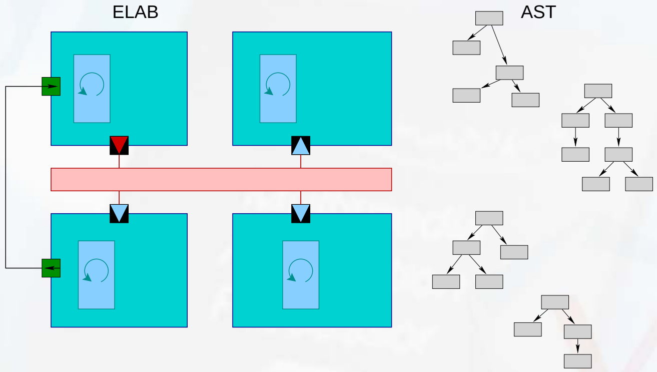
PINAPA: Key Ideas

- Syntax extraction (AST)
  - Easy if you have a C++ parser
  - $\Rightarrow$ Let's reuse GCC
- Architecture extraction (ELAB)
  - Architecture is built at run-time
  - $\Rightarrow$ Let's execute the elaboration of the program
- We have to link the syntax and architecture information

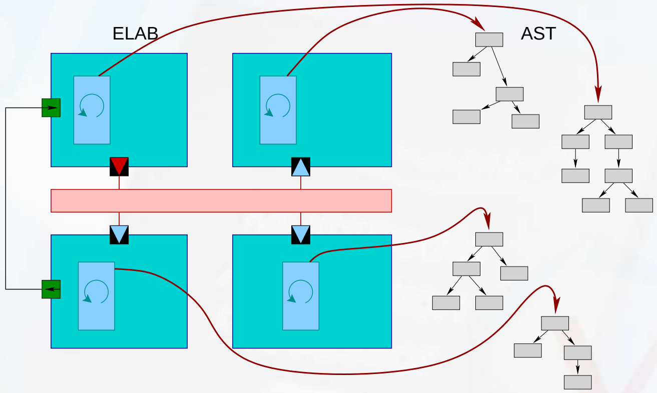
Link Between ELAB and AST



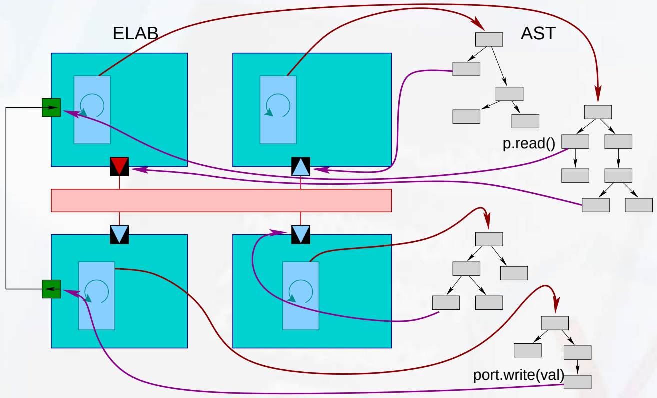
Link Between ELAB and AST



Link Between ELAB and AST



Link Between ELAB and AST

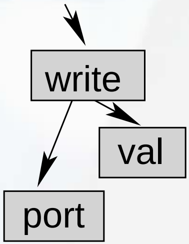


Link Between ELAB and AST: an Example

- `port.write(val);`

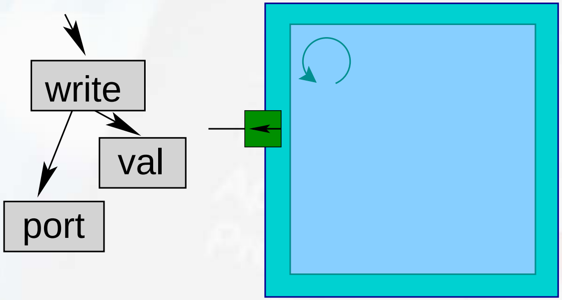
Link Between ELAB and AST: an Example

- `port.write(val);`



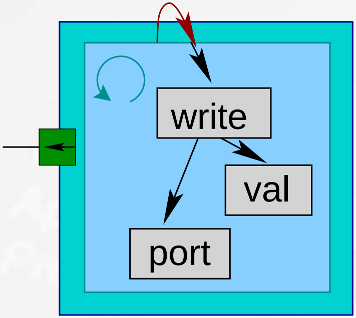
Link Between ELAB and AST: an Example

- `port.write(val);`



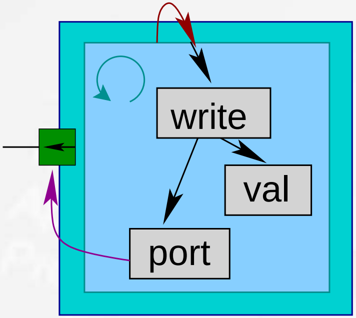
Link Between ELAB and AST: an Example

- `port.write(val);`



Link Between ELAB and AST: an Example

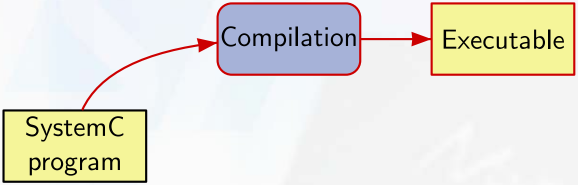
- `port.write(val);`



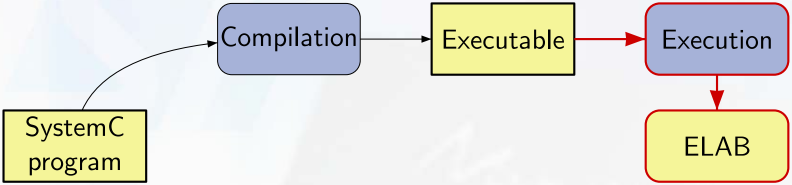
PINAPA: steps of execution

SystemC program

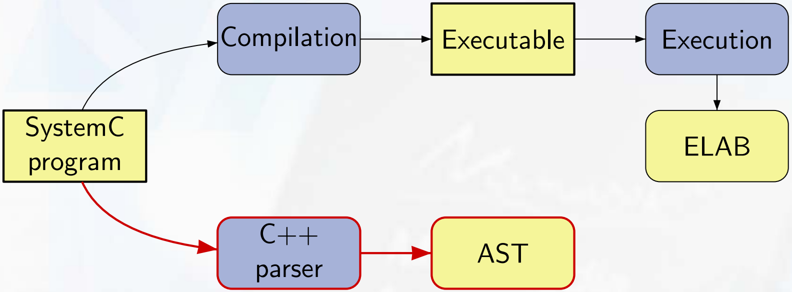
PINAPA: steps of execution



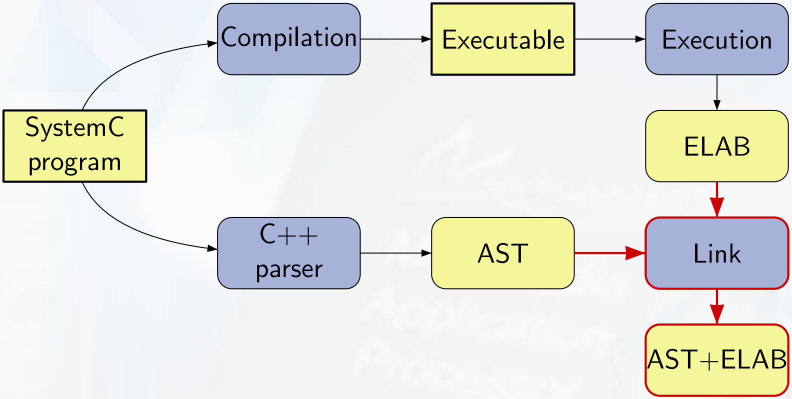
PINAPA: steps of execution



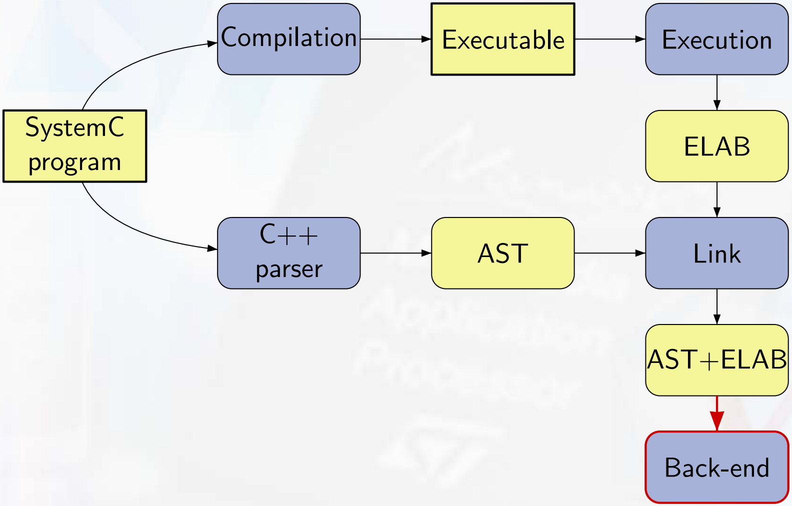
PINAPA: steps of execution



PINAPA: steps of execution



PINAPA: steps of execution



Limitations of the Approach

- No limitation regarding the code of the elaboration

Limitations of the Approach

- No limitation regarding the code of the elaboration
- AST and ELAB built correctly in any case, only link may be problematic.
  - Dynamic objects (pointers, reference) can hardly be specified with static information
  - Templates makes the task harder

Limitations of the Approach

- No limitation regarding the code of the elaboration
- AST and ELAB built correctly in any case, only link may be problematic.
  - Dynamic objects (pointers, reference) can hardly be specified with static information
  - Templates makes the task harder
- Much less limitations than other tools

Conclusion about PINAPA

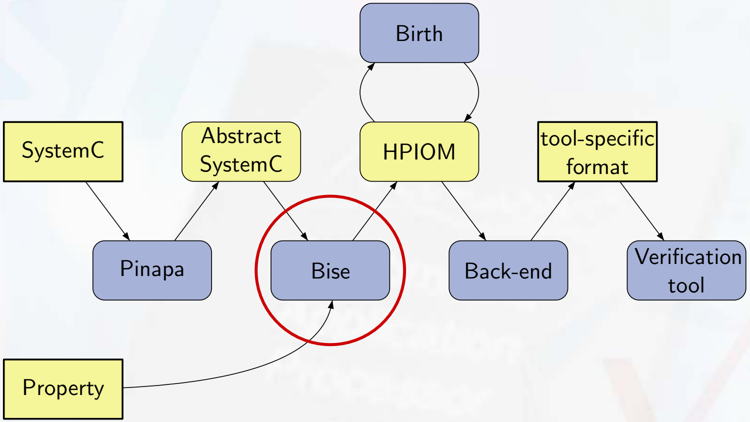
- Our approach allowed us to write a SystemC front-end
  - With very few limitations
  - Managing the TAC and BASIC channels
  - With a minimal effort ($<$ 4000 lines of C++)

Conclusion about PINAPA

- Our approach allowed us to write a SystemC front-end
  - With very few limitations
  - Managing the TAC and BASIC channels
  - With a minimal effort ($<$ 4000 lines of C++)
- PINAPA is Open Source!
  $\Rightarrow$ `http://greensocs.sourceforge.net/pinapa/`

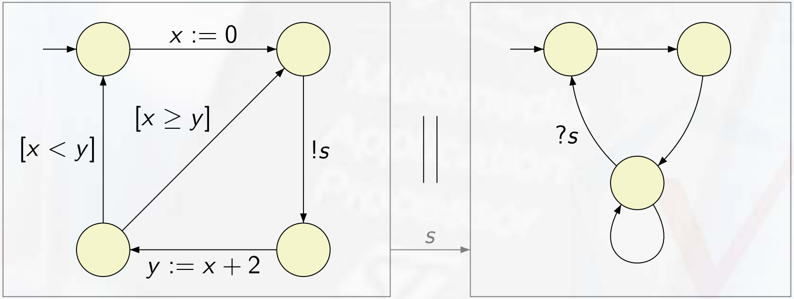
Outline

BISE: Back-end Independent Semantics Extraction



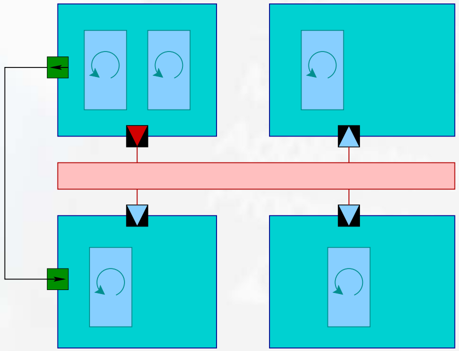
HPIOM: Heterogeneous Parallel Input/Output Machines

- A formalism of communicating automata
- With both explicit states and variables
- Using a synchronous product



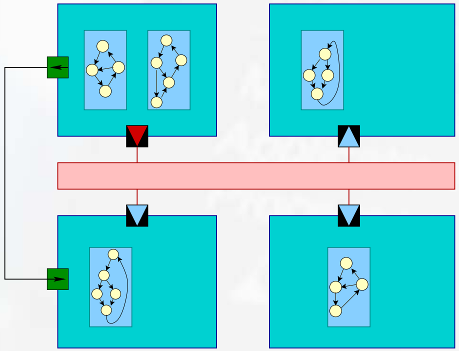
Semantics of SystemC in terms of automata

- Generate a set of automata including:



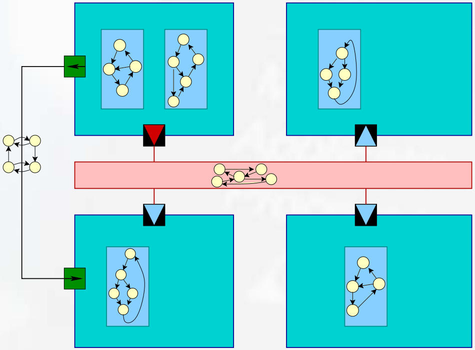
Semantics of SystemC in terms of automata

- Generate a set of automata including:
- One automaton per process (the control flow)



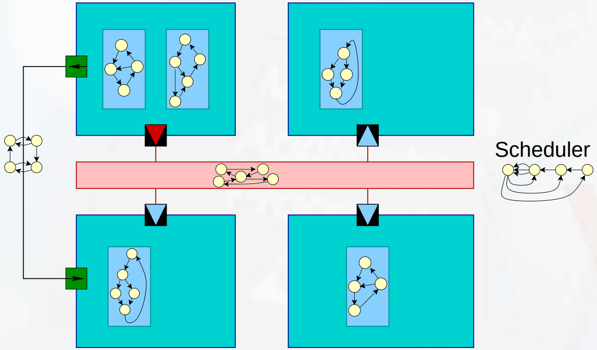
Semantics of SystemC in terms of automata

- Generate a set of automata including:
- One automaton per process (the control flow)
- Plus SystemC objects



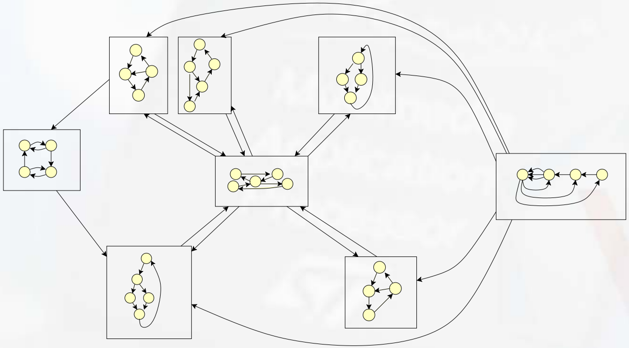
Semantics of SystemC in terms of automata

- Generate a set of automata including:
- One automaton per process (the control flow)
- Plus SystemC objects
- And the scheduler



Semantics of SystemC in terms of automata

- Generate a set of automata including:
- One automaton per process (the control flow)
- Plus SystemC objects
- And the scheduler

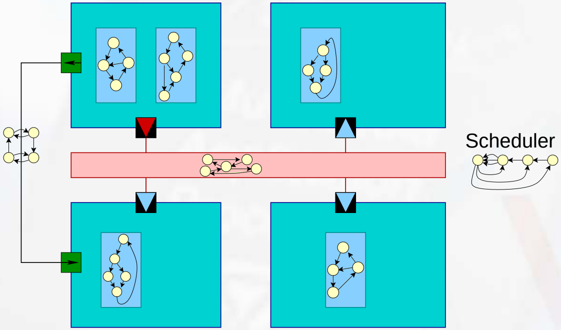


Direct Semantics Vs Translation

- Translation = Parse the source code, generate an automaton
- Direct semantics = Read the specification, instantiate an automaton

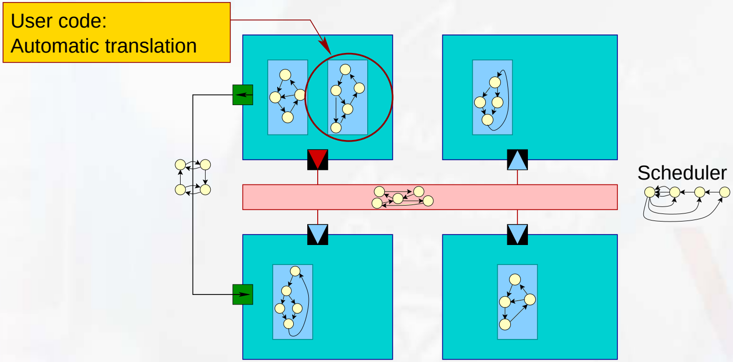
Direct Semantics Vs Translation

- Translation = Parse the source code, generate an automaton
- Direct semantics = Read the specification, instantiate an automaton



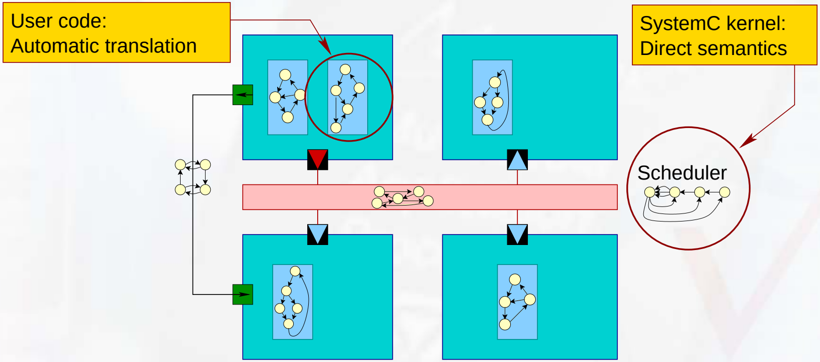
Direct Semantics Vs Translation

- Translation = Parse the source code, generate an automaton
- Direct semantics = Read the specification, instantiate an automaton



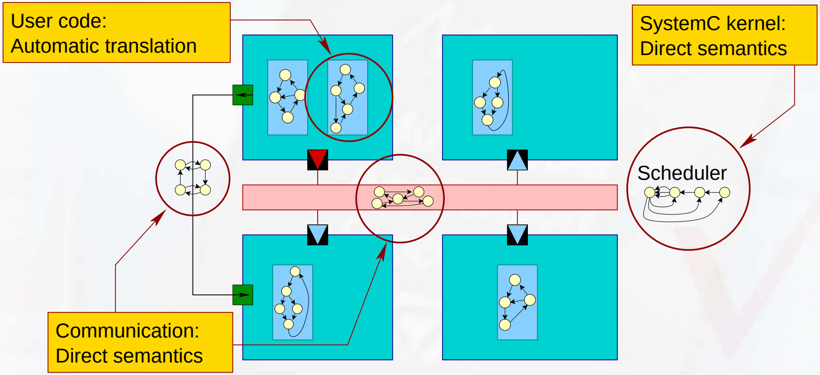
Direct Semantics Vs Translation

- Translation = Parse the source code, generate an automaton
- Direct semantics = Read the specification, instantiate an automaton



Direct Semantics Vs Translation

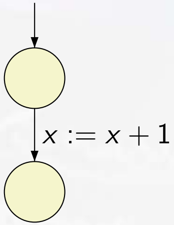
- Translation = Parse the source code, generate an automaton
- Direct semantics = Read the specification, instantiate an automaton



Translating C++ code into HPIOM

- Nothing new, but has to be done . . .

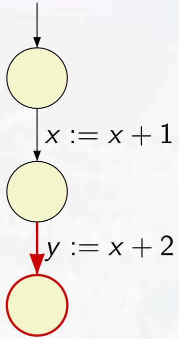
```
x = x + 1;
```



Translating C++ code into HPIOM

- Nothing new, but has to be done . . .

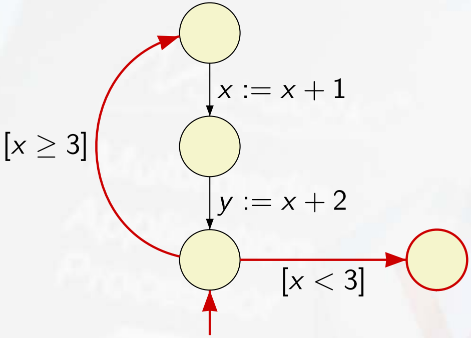
```
x = x + 1;
y = x + 2;
```



Translating C++ code into HPIOM

- Nothing new, but has to be done . . .

```
while (x <= 3) {
   x = x + 1;
   y = x + 2;
}
```



The SystemC scheduler

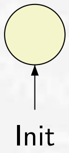
- Non-preemptive scheduler

The SystemC scheduler

- Non-preemptive scheduler
- Non-deterministic processes election

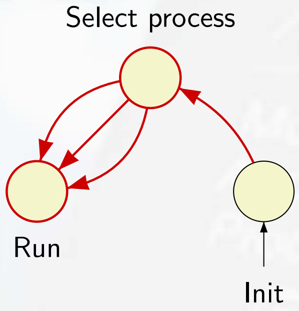
The SystemC scheduler

- Non-preemptive scheduler
- Non-deterministic processes election



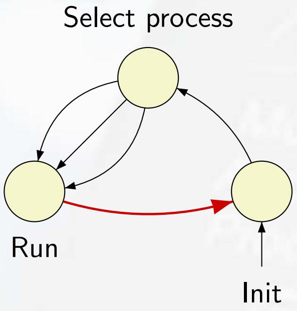
The SystemC scheduler

- Non-preemptive scheduler
- Non-deterministic processes election



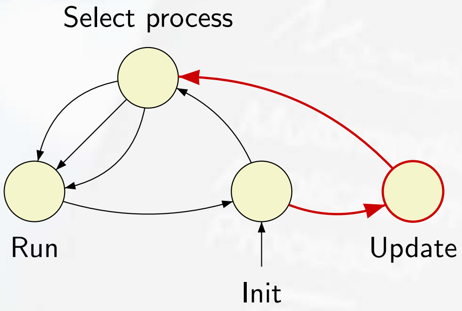
The SystemC scheduler

- Non-preemptive scheduler
- Non-deterministic processes election



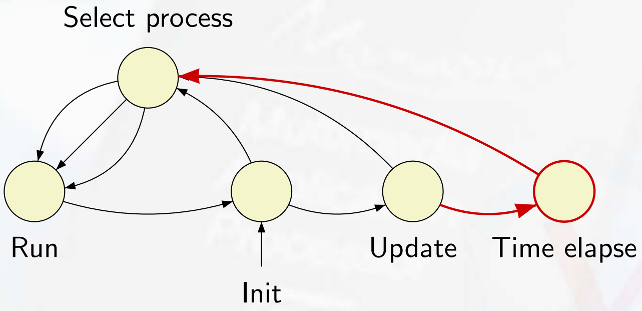
The SystemC scheduler

- Non-preemptive scheduler
- Non-deterministic processes election



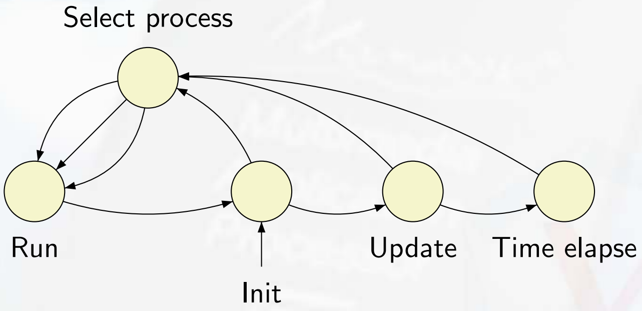
The SystemC scheduler

- Non-preemptive scheduler
- Non-deterministic processes election



The SystemC scheduler

- Non-preemptive scheduler
- Non-deterministic processes election



State of a process in the SystemC scheduler

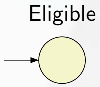
- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it

State of a process in the SystemC scheduler

- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it
- For each process, create an automaton representing its state in the scheduler ($\in (\textit{eligible}, \textit{run}, \textit{sleep})$).

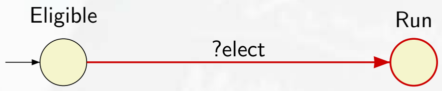
State of a process in the SystemC scheduler

- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it
- For each process, create an automaton representing its state in the scheduler ($\in$ (*eligible*, *run*, *sleep*)).



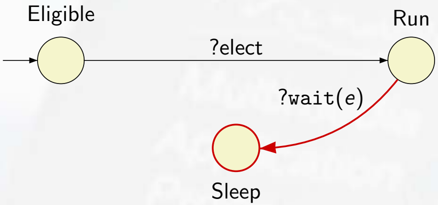
State of a process in the SystemC scheduler

- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it
- For each process, create an automaton representing its state in the scheduler ($\in$ (*eligible*, *run*, *sleep*)).



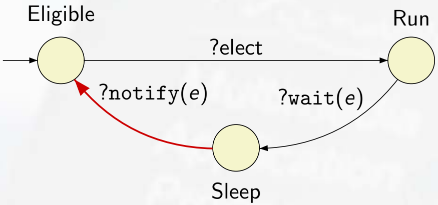
State of a process in the SystemC scheduler

- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it
- For each process, create an automaton representing its state in the scheduler ($\in (\mathit{eligible}, \mathit{run}, \mathit{sleep})$).



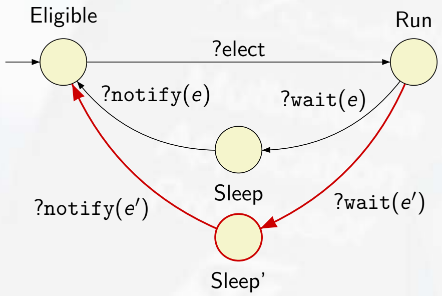
State of a process in the SystemC scheduler

- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it
- For each process, create an automaton representing its state in the scheduler ($\in$ (*eligible*, *run*, *sleep*)).



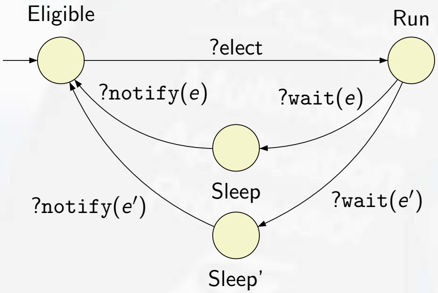
State of a process in the SystemC scheduler

- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it
- For each process, create an automaton representing its state in the scheduler ($\in (\textit{eligible}, \textit{run}, \textit{sleep})$).



State of a process in the SystemC scheduler

- Election of a process is done in two phases:
  1. Make the process eligible
  2. Run it
- For each process, create an automaton representing its state in the scheduler ($\in$ (*eligible*, *run*, *sleep*)).



Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.

Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.

Current value	
True	False

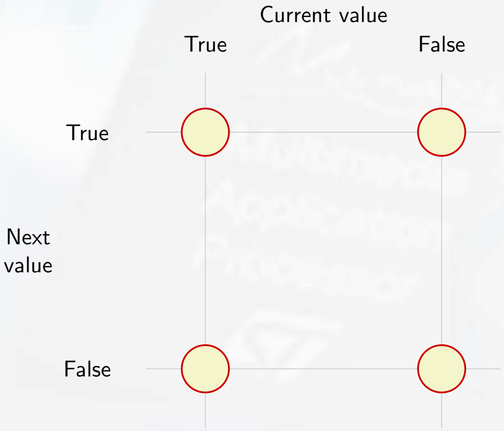
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.

		Current value	
		True	False
Next value	True		
	False		

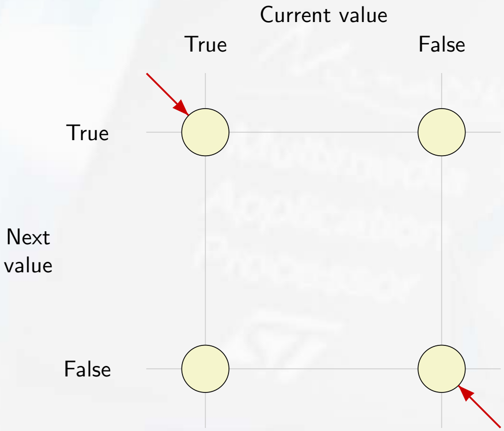
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



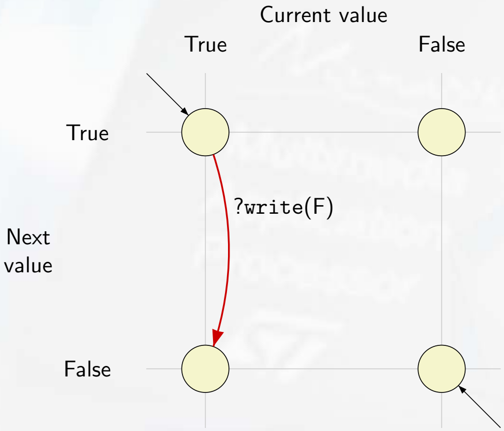
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



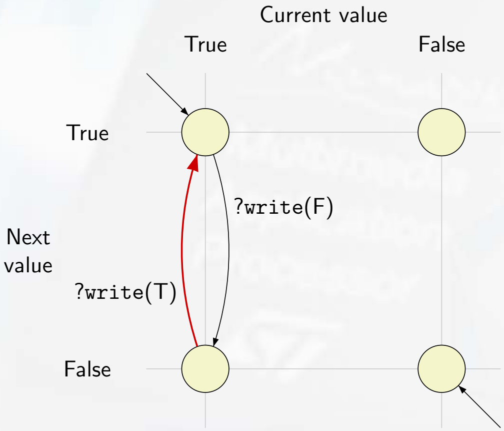
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



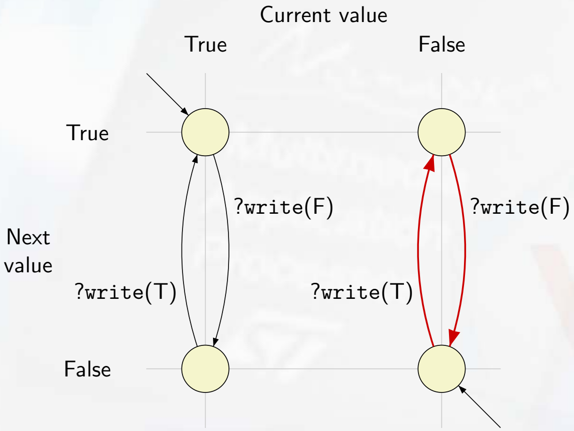
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



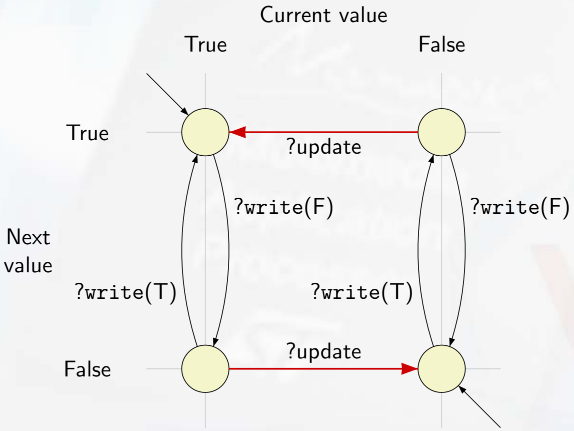
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



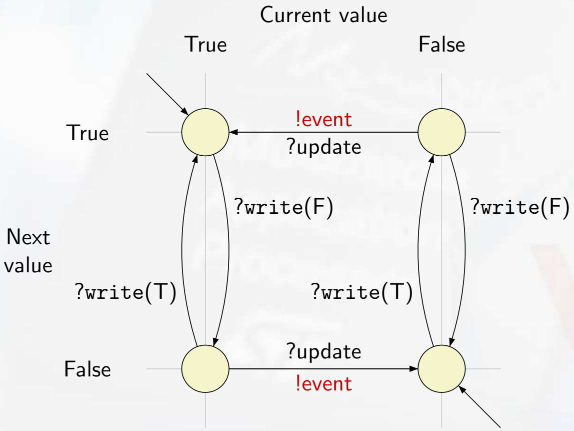
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



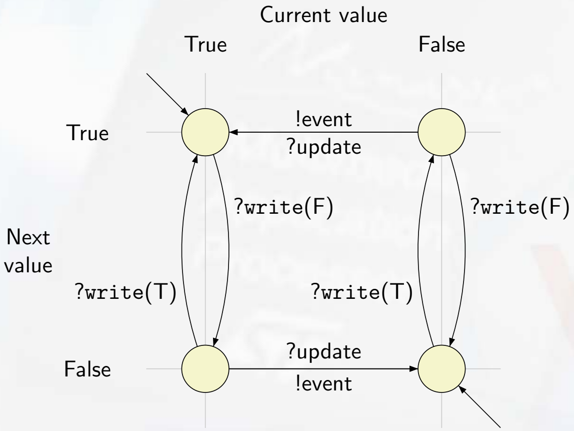
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



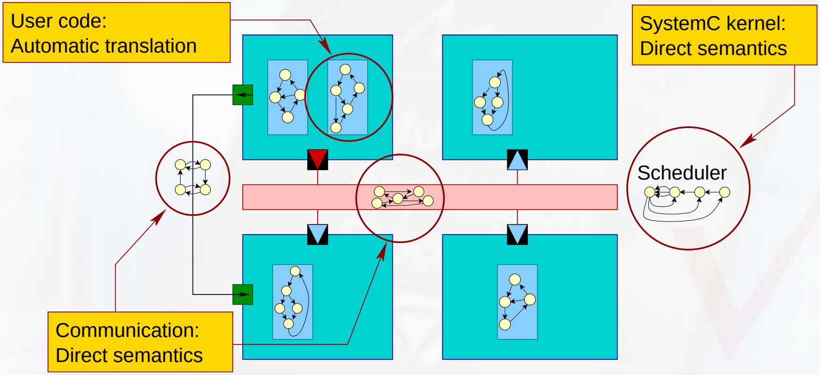
Semantics of SystemC Boolean Signals

- SystemC `sc_signal` avoids scheduling dependencies:
  1. `read` takes the current value, but
  2. `write` will be taken into account at the next update phase.



Direct Semantics Vs Translation

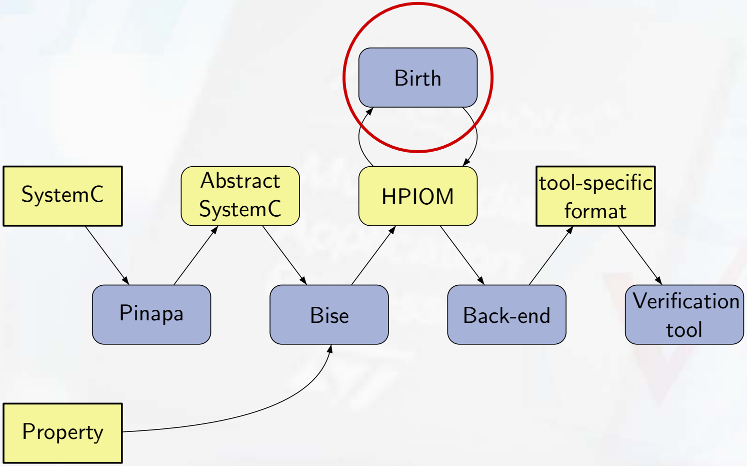
- Translation = Parse the source code, generate an automaton
- Direct semantics = Read the specification, instantiate an automaton



Outline

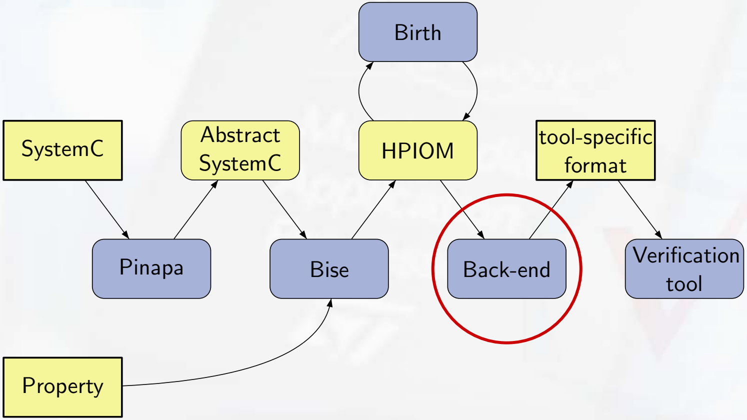
Other components not detailed here

- BIRTH : HPIOM to HPIOM transformations
  (**B**ack-end **I**ndependent **R**eduction and **T**ransformation of **H**piom)



Other components not detailed here

- BIRTH : HPIOM to HPIOM transformations
  (**B**ack-end **I**ndependent **R**eduction and **T**ransformation of **H**piom)
- Back-ends: Connection to verification tools



Results

- The complete LUSSY tool-chain is operational
  From SystemC program to “yes/no+diagnosis”
- Methodology to debug and validate LUSSY itself
- Accurate modeling of SystemC semantics

Results

- The complete LUSSY tool-chain is operational
  From SystemC program to “yes/no+diagnosis”
- Methodology to debug and validate LUSSY itself
- Accurate modeling of SystemC semantics
- Results of the verification tools
  - Able to find well hidden bugs in small programs

Results

- The complete LUSSY tool-chain is operational
  From SystemC program to “yes/no+diagnosis/state explosion”
- Methodology to debug and validate LUSSY itself
- Accurate modeling of SystemC semantics
- Results of the verification tools
  - Able to find well hidden bugs in small programs
  - But state explosion on medium and large programs

Results

- The complete LUSSY tool-chain is operational
  From SystemC program to “yes/no+diagnosis/state explosion”
- Methodology to debug and validate LUSSY itself
- Accurate modeling of SystemC semantics
- Results of the verification tools
  - Able to find well hidden bugs in small programs
  - But state explosion on medium and large programs

So, is this work useless in practice?

Results

- The complete LUSSY tool-chain is operational
  From SystemC program to “yes/no+diagnosis/state explosion”
- Methodology to debug and validate LUSSY itself
- Accurate modeling of SystemC semantics
- Results of the verification tools
  - Able to find well hidden bugs in small programs
  - But state explosion on medium and large programs
- LUSSY provides the building blocks, but a lot is still to be done . . .

Perspectives

- Better formal verification

Perspectives

- Better formal verification
  - Extensive use of abstract interpretation

    ⇒ Mathias Peron, Ph.D at Verimag

Perspectives

- Better formal verification
  - Extensive use of abstract interpretation
    $\Rightarrow$ Mathias Peron, Ph.D at Verimag
  - Compositional verification
    $\Rightarrow$ Experiments with PROMETHEUS (Yvan Roux, INRIA)
    $\Rightarrow$ Towards a joint project with Edmund M. Clarke (CMU)

Perspectives

- Better formal verification
  - Extensive use of abstract interpretation
    $\Rightarrow$ Mathias Peron, Ph.D at Verimag
  - Compositional verification
    $\Rightarrow$ Experiments with PROMETHEUS (Yvan Roux, INRIA)
    $\Rightarrow$ Towards a joint project with Edmund M. Clarke (CMU)
- Run-time verification: Improve test coverage with a minimal test-bench (will use PINAPA for instrumentation)
  $\Rightarrow$ Claude Helmstetter, Ph.D at Verimag/STMicroelectronics
  $\Rightarrow$ Yussef Bouzouzou, DRT at Verimag/Silicomp

Perspectives

- Better formal verification
  - Extensive use of abstract interpretation
    $\Rightarrow$ Mathias Peron, Ph.D at Verimag
  - Compositional verification
    $\Rightarrow$ Experiments with PROMETHEUS (Yvan Roux, INRIA)
    $\Rightarrow$ Towards a joint project with Edmund M. Clarke (CMU)
- Run-time verification: Improve test coverage with a minimal test-bench (will use PINAPA for instrumentation)
  $\Rightarrow$ Claude Helmstetter, Ph.D at Verimag/STMicroelectronics
  $\Rightarrow$ Yussef Bouzouzou, DRT at Verimag/Silicomp
- Other tools based on the LUSSY tool-chain

Perspectives

- Better formal verification
  - Extensive use of abstract interpretation
    $\Rightarrow$ Mathias Peron, Ph.D at Verimag
  - Compositional verification
    $\Rightarrow$ Experiments with PROMETHEUS (Yvan Roux, INRIA)
    $\Rightarrow$ Towards a joint project with Edmund M. Clarke (CMU)
- Run-time verification: Improve test coverage with a minimal test-bench (will use PINAPA for instrumentation)
  $\Rightarrow$ Claude Helmstetter, Ph.D at Verimag/STMicroelectronics
  $\Rightarrow$ Yussef Bouzouzou, DRT at Verimag/Silicomp
- Other tools based on the LUSSY tool-chain
  - SPINAPA: A prototype of SPIRIT back-end for PINAPA developed in STMicroelectronics (allows for graphical visualization in particular)
    $\Rightarrow$ Frédéric Saunier, Silicomp/STMicroelectronics

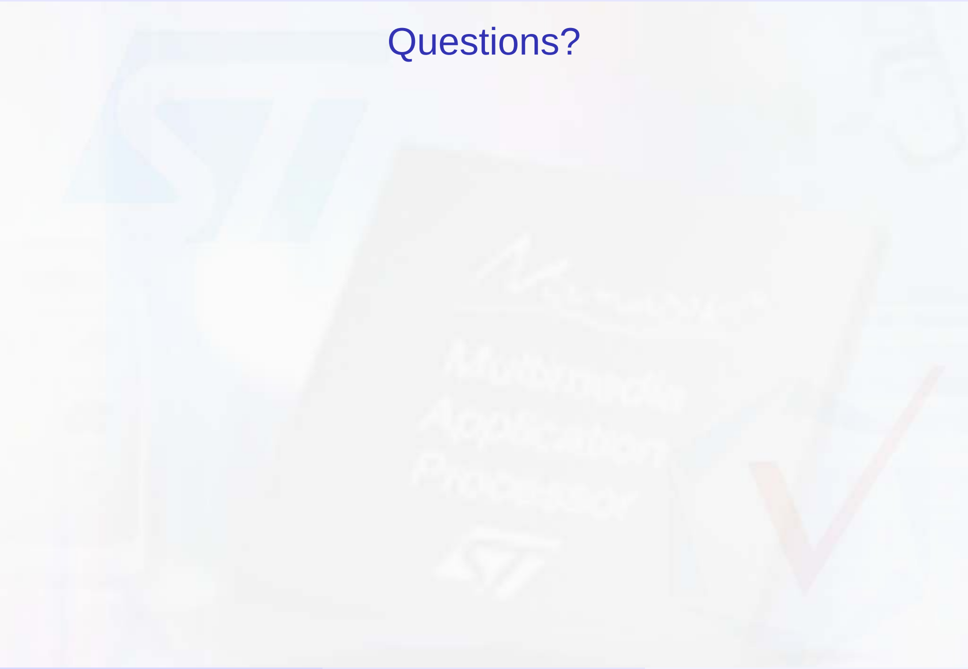
Perspectives

- Better formal verification
  - Extensive use of abstract interpretation
    $\Rightarrow$ Mathias Peron, Ph.D at Verimag
  - Compositional verification
    $\Rightarrow$ Experiments with PROMETHEUS (Yvan Roux, INRIA)
    $\Rightarrow$ Towards a joint project with Edmund M. Clarke (CMU)
- Run-time verification: Improve test coverage with a minimal test-bench (will use PINAPA for instrumentation)
  $\Rightarrow$ Claude Helmstetter, Ph.D at Verimag/STMicroelectronics
  $\Rightarrow$ Yussef Bouzouzou, DRT at Verimag/Silicomp
- Other tools based on the LUSSY tool-chain
  - SPINAPA: A prototype of SPIRIT back-end for PINAPA developed in STMicroelectronics (allows for graphical visualization in particular)
    $\Rightarrow$ Frédéric Saunier, Silicomp/STMicroelectronics
  - Several PINAPA back-ends outside ST/Verimag

Perspectives

- Better formal verification
  - ▸ Extensive use of abstract interpretation
    $\Rightarrow$ Mathias Peron, Ph.D at Verimag
  - ▸ Compositional verification
    $\Rightarrow$ Experiments with PROMETHEUS (Yvan Roux, INRIA)
    $\Rightarrow$ Towards a joint project with Edmund M. Clarke (CMU)
- Run-time verification: Improve test coverage with a minimal test-bench (will use PINAPA for instrumentation)
  $\Rightarrow$ Claude Helmstetter, Ph.D at Verimag/STMicroelectronics
  $\Rightarrow$ Yussef Bouzouzou, DRT at Verimag/Silicomp
- Other tools based on the LUSSY tool-chain
  - ▸ SPINAPA: A prototype of SPIRIT back-end for PINAPA developed in STMicroelectronics (allows for graphical visualization in particular)
    $\Rightarrow$ Frédéric Saunier, Silicomp/STMicroelectronics
  - ▸ Several PINAPA back-ends outside ST/Verimag
  - ▸ OpenTLM: A Minalogic project including STMicroelectronics, Verimag, Silicomp-AQL and others

Questions?



Backup slides

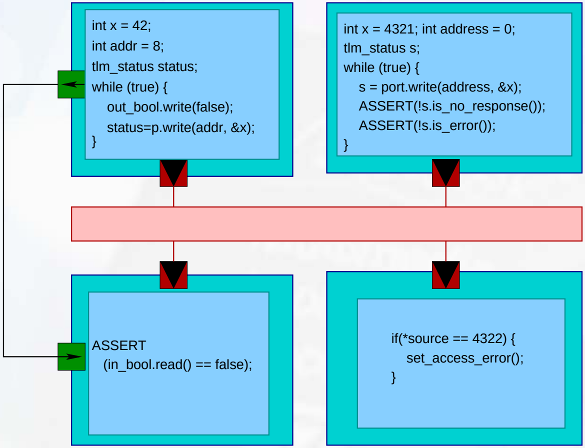
Users of PINAPA

- SPINAPA: SPIRIT back-end (Frédéric Saunier, STMicroelectronics/Silicomp)
- Theorem proving (Primrose Mbanefo)
- Platform Based Design Methodology (Humberto Rocha)
- Introspection in SystemC (Diogo Alves)
- Connection to PROMETHEUS (Yvan Roux, INRIA)
- 9 other subscribers on the mailing list.

PINAPA and dynamic objects

- No management of pointers/references to SystemC objects
- `port_array[42]` managed exactly as a normal port (because 42 is a constant)
- `port_array[x + y]`: PINAPA attaches a pointer to `port_array[0]` and the AST for “`x + y`”

Examples of Properties on a TLM



```
int x = 42;
int addr = 8;
tlm_status status;
while (true) {
    out_bool.write(false);
    status=p.write(addr, &x);
}
```

```
int x = 4321; int address = 0;
tlm_status s;
while (true) {
    s = port.write(address, &x);
    ASSERT(!s.is_no_response());
    ASSERT(!s.is_error());
}
```

```
ASSERT
  (in_bool.read() == false);
```

```
if(*source == 4322) {
    set_access_error();
}
```