tel-00539648, version 1 - 24 Nov 2010

42: A Component-Based Approach to Virtual Prototyping of Heterogeneous Embedded Systems

Ph.D. Defense

Tayeb BOUHADIBA

Verimag Grenoble INP

Directrice de thèse : Florence MARANINCHI

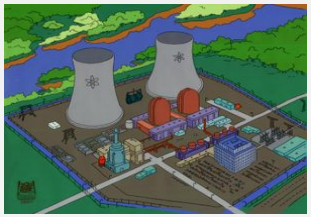
Jury:

Marc POUZET	Rapporteur
Lionel SEINTURIER	Rapporteur
Jean-Bernard STEFANI	Examinateur

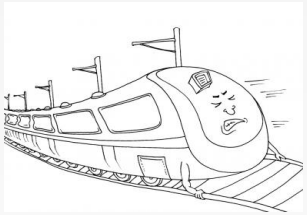
September 15th 2010

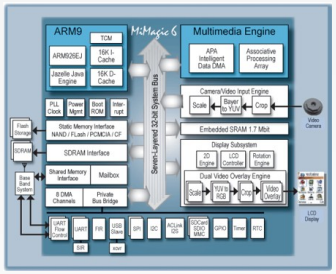
tel-00539648, version 1 - 24 Nov 2010

Embedded Systems

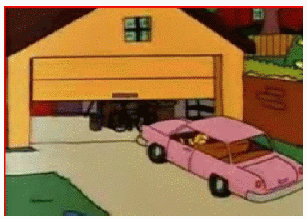


Consumer Electronics



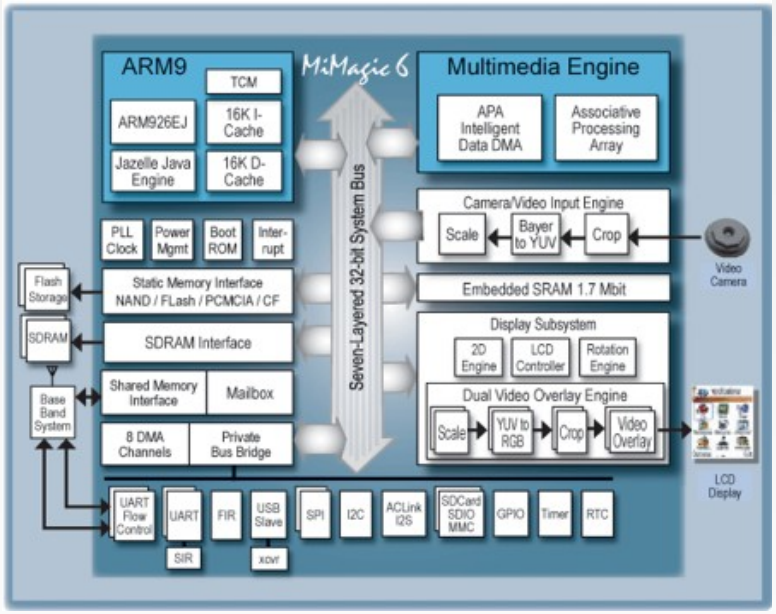


Safety Critical Systems



tel-00539648, version 1 - 24 Nov 2010

Embedded Systems, Components & Heterogeneity



Software

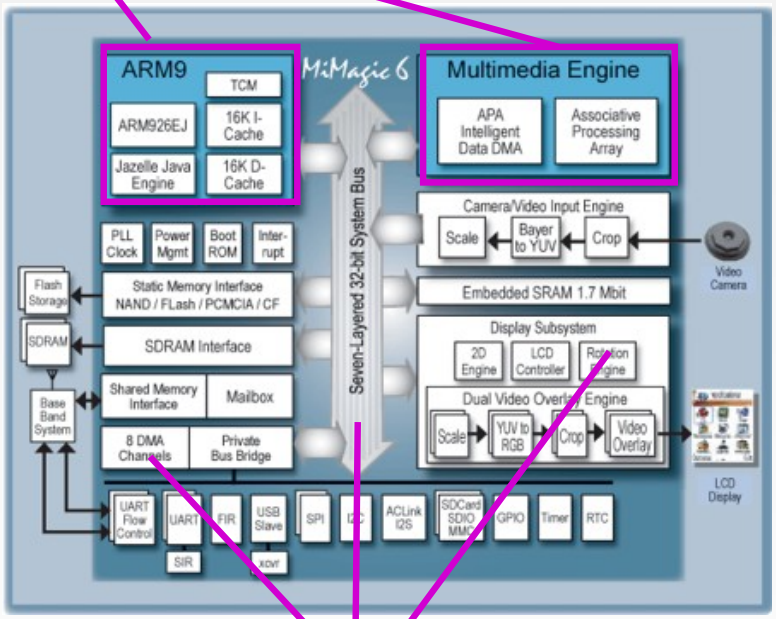
+

Hardware

tel-00539648, version 1 - 24 Nov 2010

Embedded Systems, Components & Heterogeneity

Processors



Hardware IP's
(Intellectual Properties)

Software

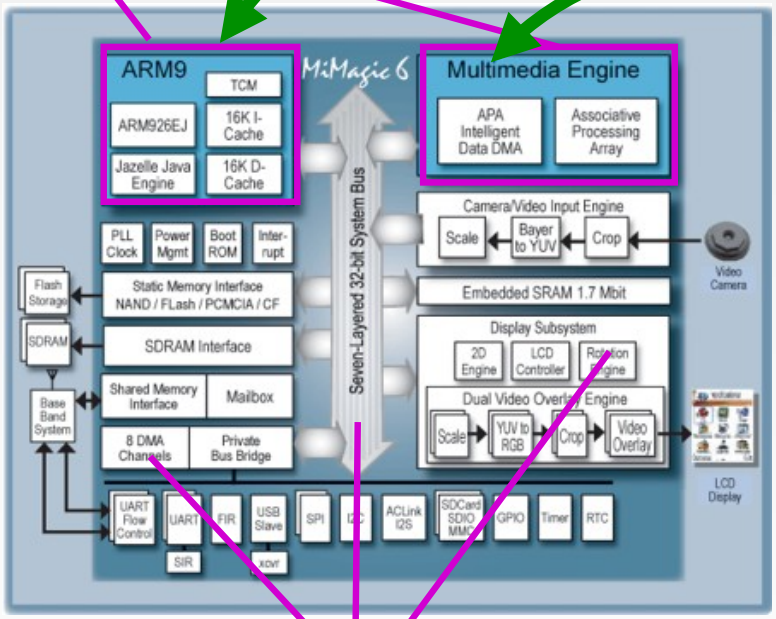
+

Hardware

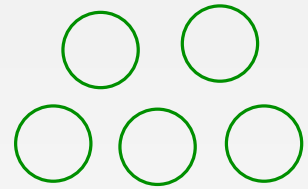
tel-00539648, version 1 - 24 Nov 2010

Embedded Systems, Components & Heterogeneity

Processors



Hardware IP's
(Intellectual Properties)



Software

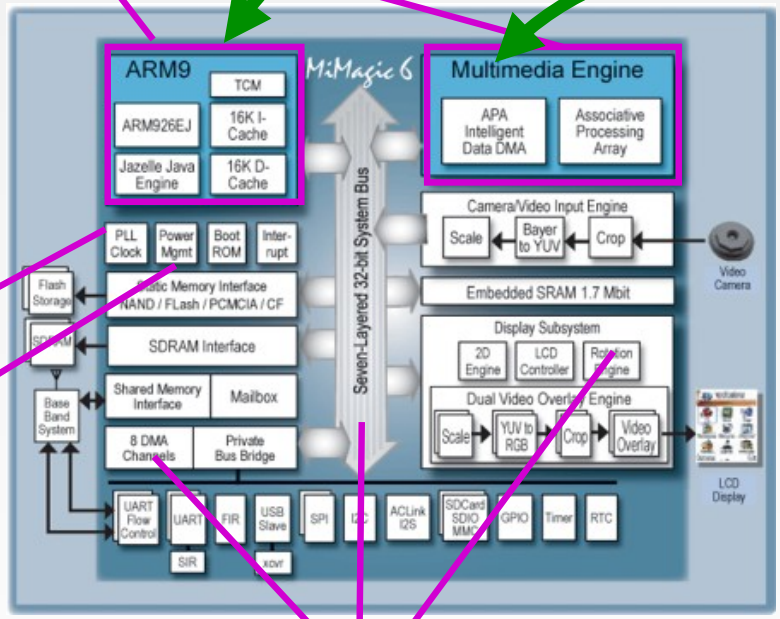
+

Hardware

tel-00539648, version 1 - 24 Nov 2010

Embedded Systems, Components & Heterogeneity

Processors



Analog

Digital

Hardware IP's
(Intellectual Properties)

Software

+

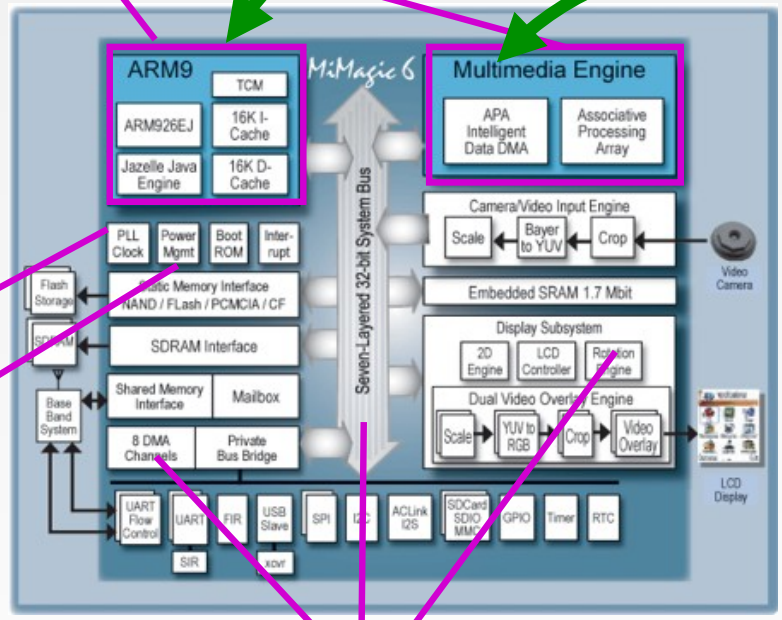
Hardware

tel-00539648, version 1 - 24 Nov 2010

Embedded Systems, Components & Heterogeneity

Processors

Asynchronous



Analog

Digital

Software

+

Hardware

Hardware IP's
(Intellectual Properties)

Synchronous

tel-00539648, version 1 - 24 Nov 2010

Virtual Prototyping

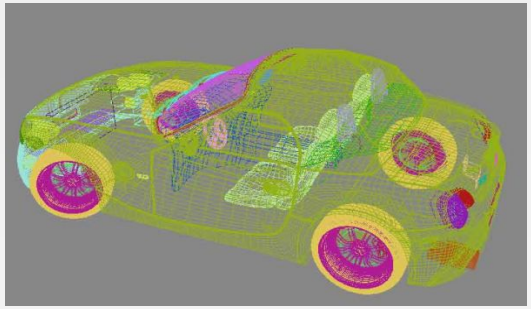
Virtual Prototyping

=

An Executable Model Before the System is Manufactured

Virtual Prototyping

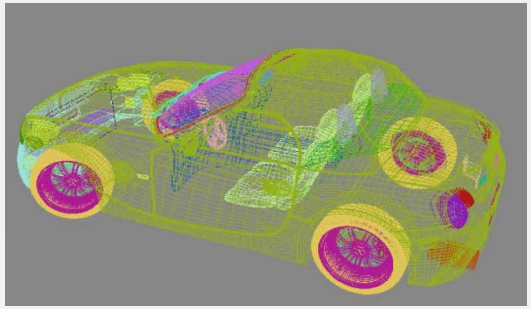
Modeling



tel-00539648, version 1 - 24 Nov 2010

Virtual Prototyping

Modeling

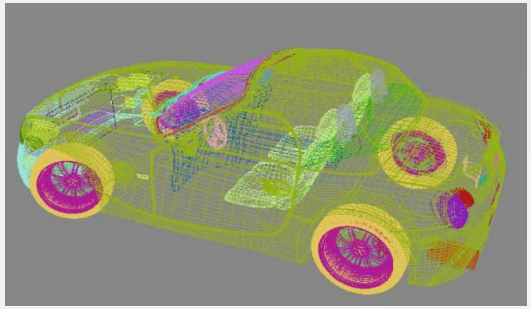


Software

tel-00539648, version 1 - 24 Nov 2010

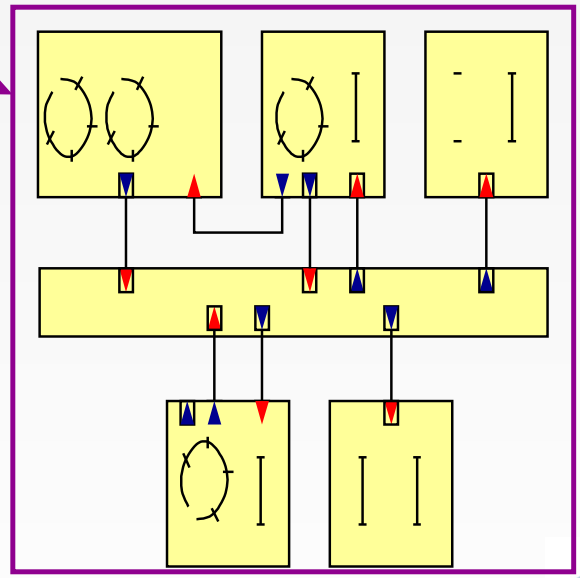
Virtual Prototyping

Modeling



Virtual Prototyping

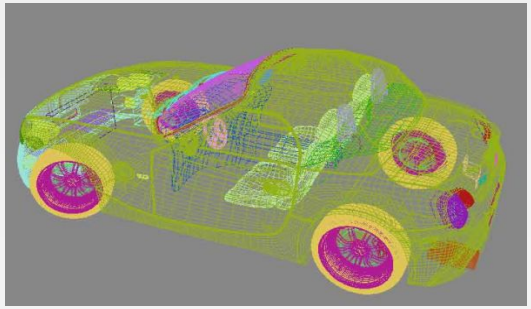
Software



tel-00539648, version 1 - 24 Nov 2010

Virtual Prototyping

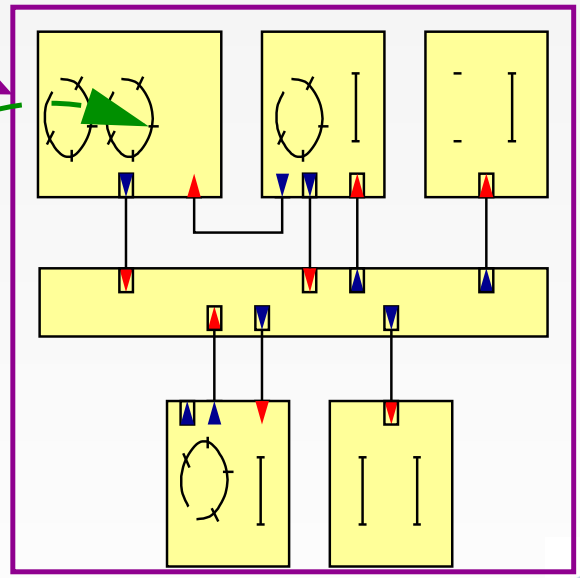
Modeling



Virtual Prototyping

Executability

Software



tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration
- Virtual Prototyping of Embedded Systems
- Modeling Hardware/Software with Synchronous Languages
- Ptolemy
- Specifying Components: Contracts

2 Overview of 42 & Examples [GPCE07]

3 Hardware Simulation by Interpreting Contracts [COORD09]

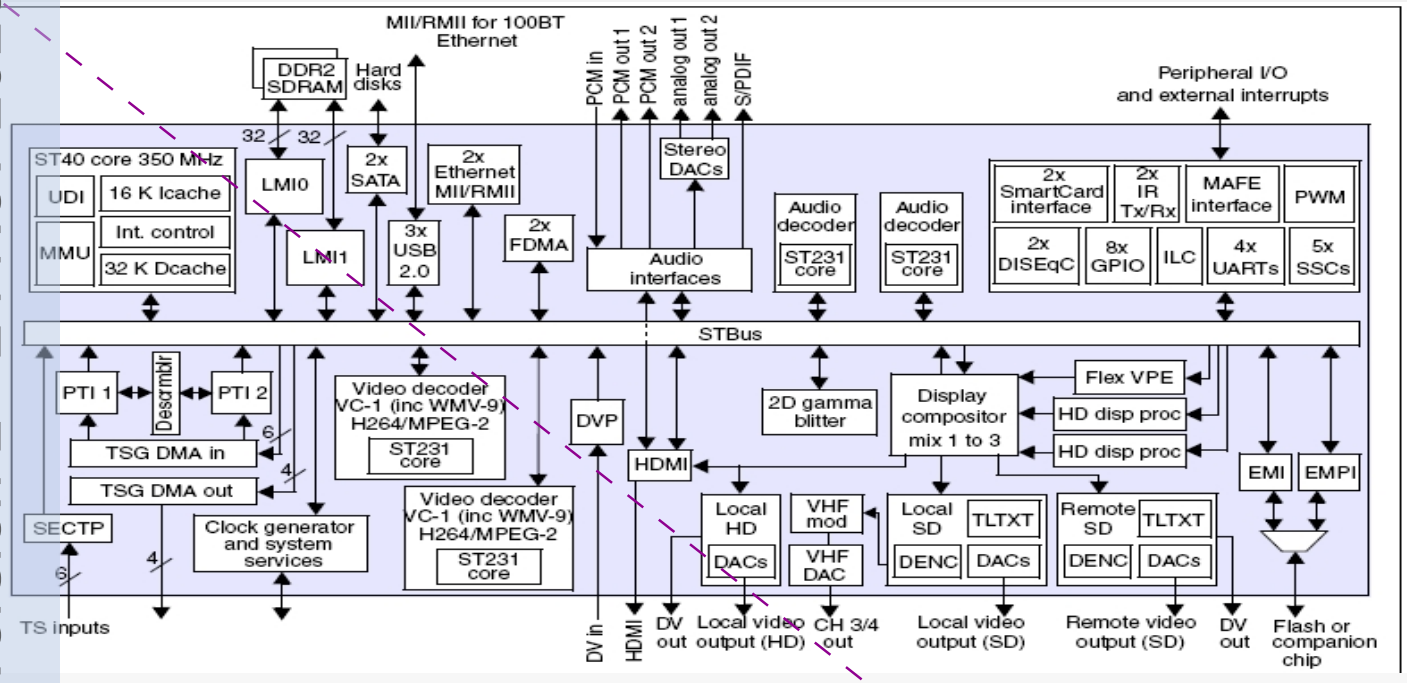
4 Using 42 Together with Existing Approaches [EMSOFT09]

5 Some Related Work

tel-00539648, version 1 - 24 Nov 2010

SystemC/TLM for Systems-on-a-Chip

tel-00539648, version 1 - 24 Nov 2010



Hardware Architecture

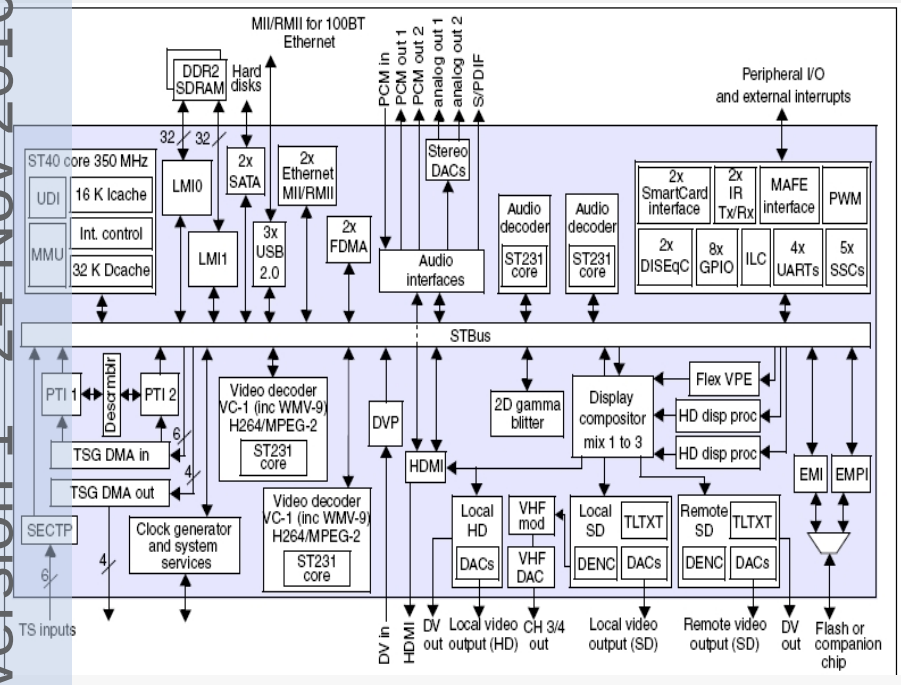
```
while(true)
x = 42;
while (x neq 0)
y = read(addrM + x);
write(addrM , x);
...
write(addrD+0x0, addrM);
write(addrD+0x1, 42);
write(addrD+0x4, 0x01);
...
...
```

Embedded Software

A SoC (System-on-a-Chip)

SystemC/TLM for Systems-on-a-Chip

tel-00539648, version 1 - 24 Nov 2010

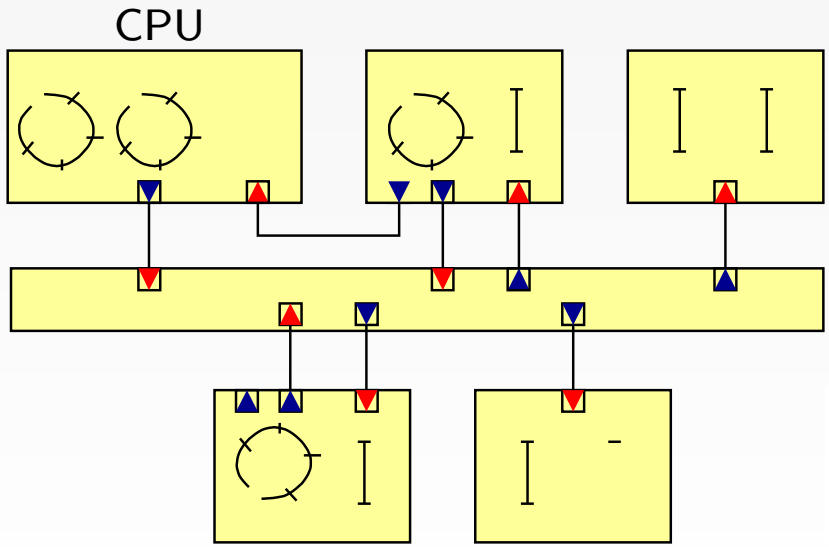



```
while(true)
x = 42;
while (x neq 0)
y = read(addrM + x);
write(addrM , x);
...
write(addrD+0x0, addrM);
write(addrD+0x1, 42);
write(addrD+0x4, 0x01);
...
...
```

Embedded Software

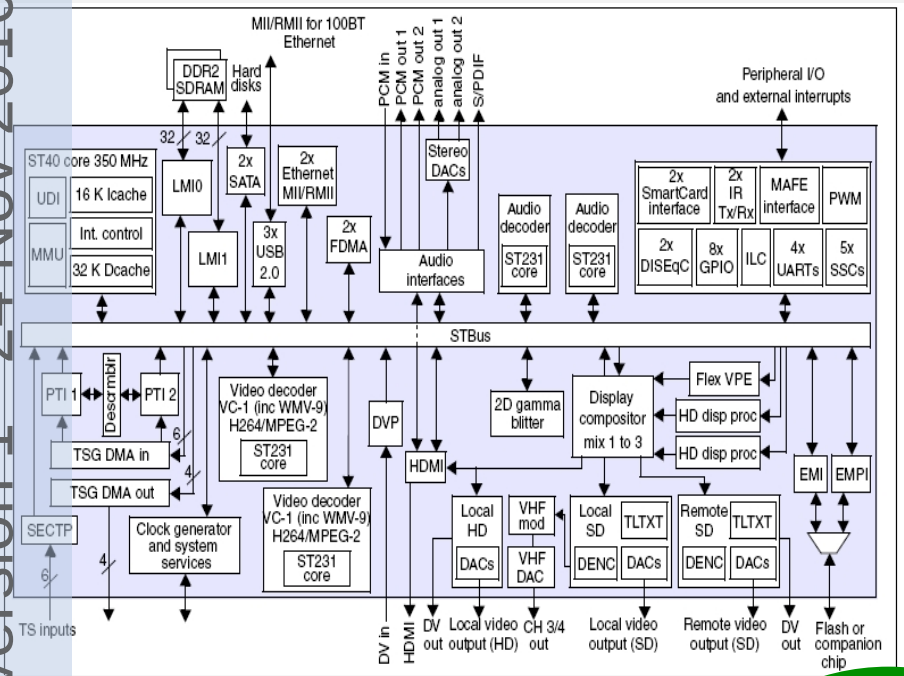
TLM (Transaction-Level Modeling) provides

Virtual Prototype



SystemC/TLM for Systems-on-a-Chip

tel-00539648, version 1 - 24 Nov 2010

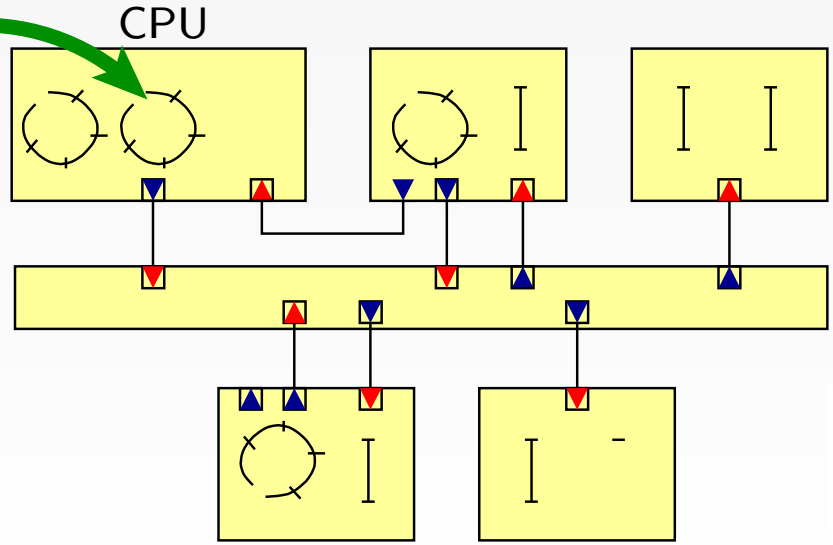


TLM (Transaction-Level Modeling) provides

```
while(true)
x = 42;
while (x neq 0)
y = read(addrM + x);
write(addrM , x);
...
write(addrD+0x0, addrM);
write(addrD+0x1, 42);
write(addrD+0x4, 0x01);
...
...
```

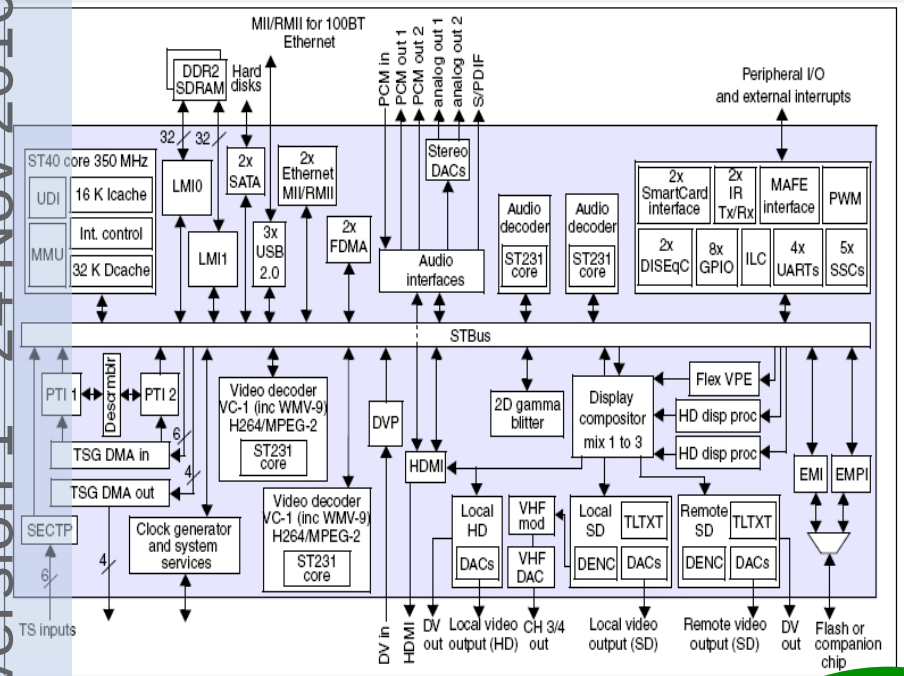
Embedded Software

Virtual Prototype



SystemC/TLM for Systems-on-a-Chip

tel-00539648, version 1 - 24 Nov 2010



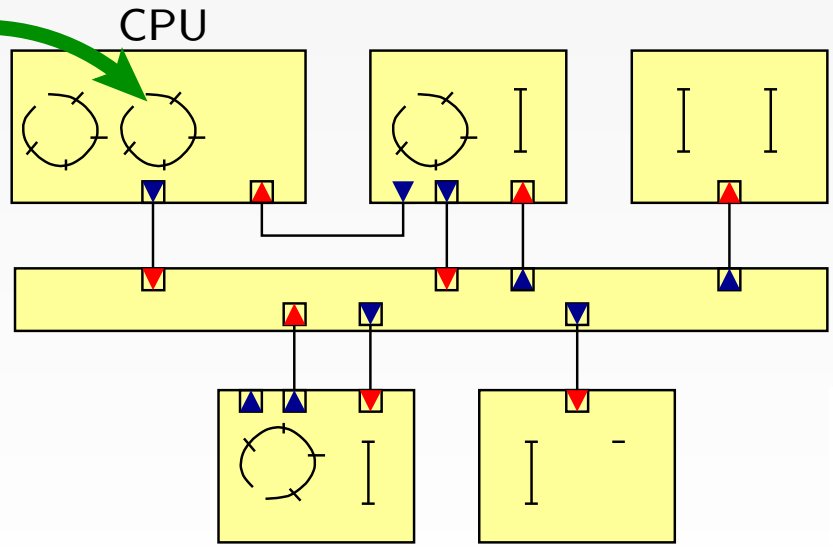

```
while(true)
x = 42;
while (x neq 0)
y = read(addrM + x);
write(addrM , x);
...
write(addrD+0x0, addrM);
write(addrD+0x1, 42);
write(addrD+0x4, 0x01);
...
...
```

Embedded Software

TLM (Transaction-Level Modeling) provides

- Early Available

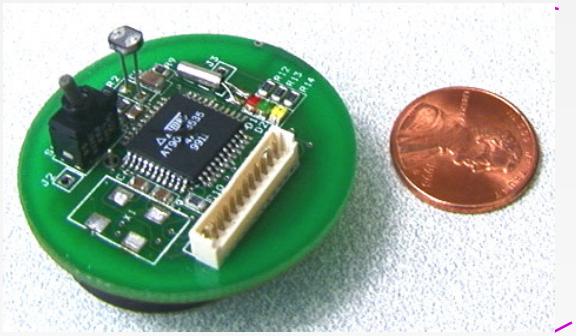
Virtual Prototype

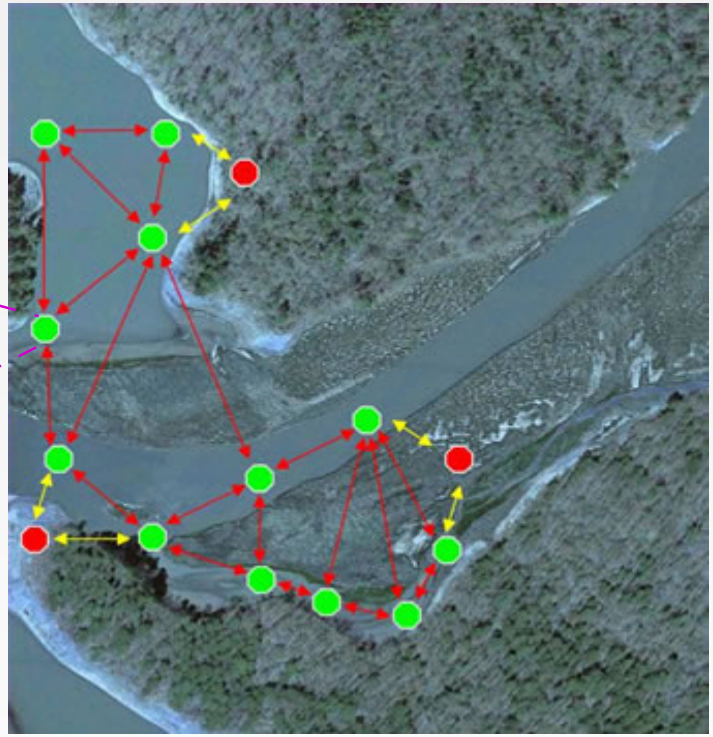


tel-00539648, version 1 - 24 Nov 2010

Virtual Prototyping of Sensor Networks

CPU, Sensor, Memory, Radio, Battery





- Several sensors communicating by radio
- Network lifetime depends on energy consumption
- Use of virtual prototyping to study non-functional aspects (e.g., energy)

tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration
- Virtual Prototyping of Embedded Systems
- Modeling Hardware/Software with Synchronous Languages
- Ptolemy
- Specifying Components: Contracts

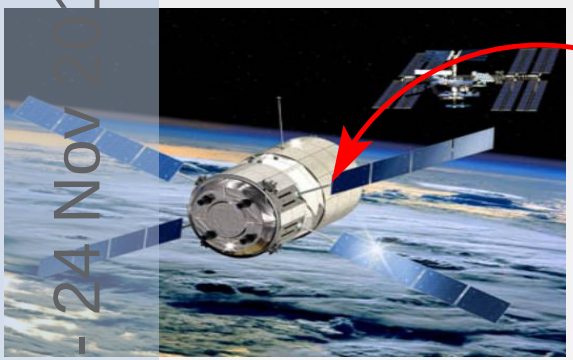
2 Overview of 42 & Examples [GPCE07]

3 Hardware Simulation by Interpreting Contracts [COORD09]

4 Using 42 Together with Existing Approaches [EMSOFT09]

5 Some Related Work

Modeling HW/SW with Synchronous Languages

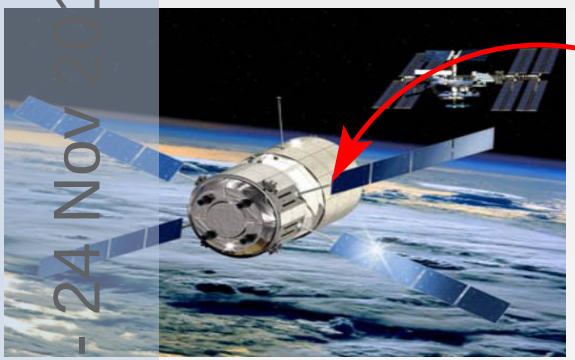


PFS (Proximity Flight Safety) is Embedded in the ATV

ATV (Automated Transfer Vehicule)

tel-00539648, version 1 - 24 Nov 2010

Modeling HW/SW with Synchronous Languages



PFS (Proximity Flight Safety)
is Embedded in the ATV

ATV (Automated Transfer Vehicule)

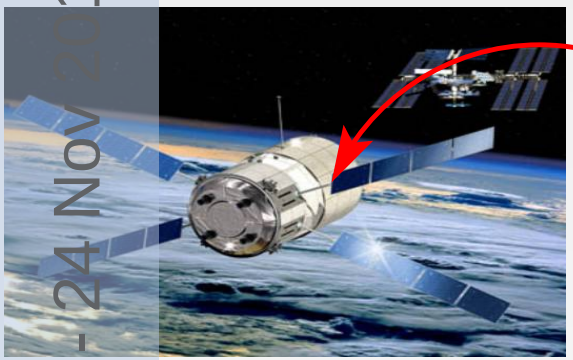
The original model of the PFS
is written in AADL

(Architecture Analysis and Design Language)
(Not Executbale)

tel-00539648, version 1 - 24 Nov 2010

Modeling HW/SW with Synchronous Languages

tel-00539648, version 1 - 24 Nov 2010



PFS (Proximity Flight Safety)
is Embedded in the ATV

ATV (Automated Transfer Vehicule)

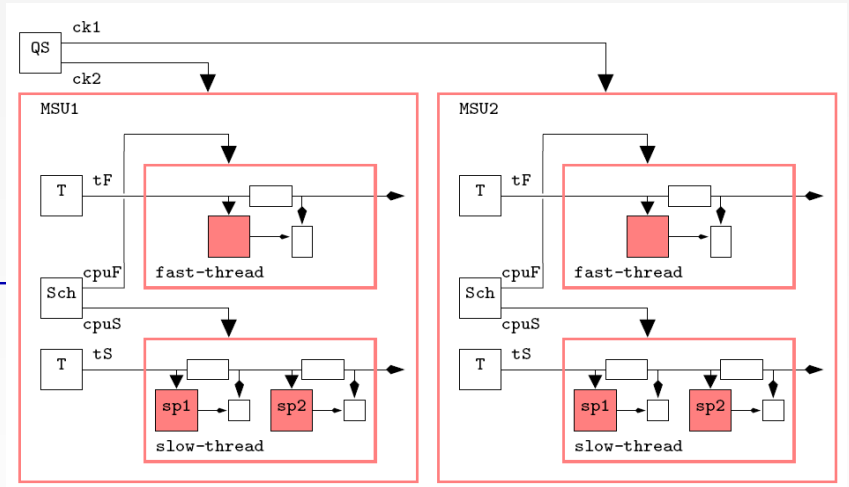
The original model of the PFS
is written in AADL

(Architecture Analysis and Design Language)
(Not Executbale)

Automatic Translation
into Lustre

Simulation

Automatic Testing



PFS description in Lustre

"E. Jahier, N. Halbwachs, P. Raymond, X. Nicollin, D. Lesens, Virtual Execution of AADL Models via a Translation into Synchronous Programs" [EMSOFT07]

tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration
- Virtual Prototyping of Embedded Systems
- Modeling Hardware/Software with Synchronous Languages
- Ptolemy
- Specifying Components: Contracts

2 Overview of 42 & Examples [GPCE07]

3 Hardware Simulation by Interpreting Contracts [COORD09]

4 Using 42 Together with Existing Approaches [EMSOFT09]

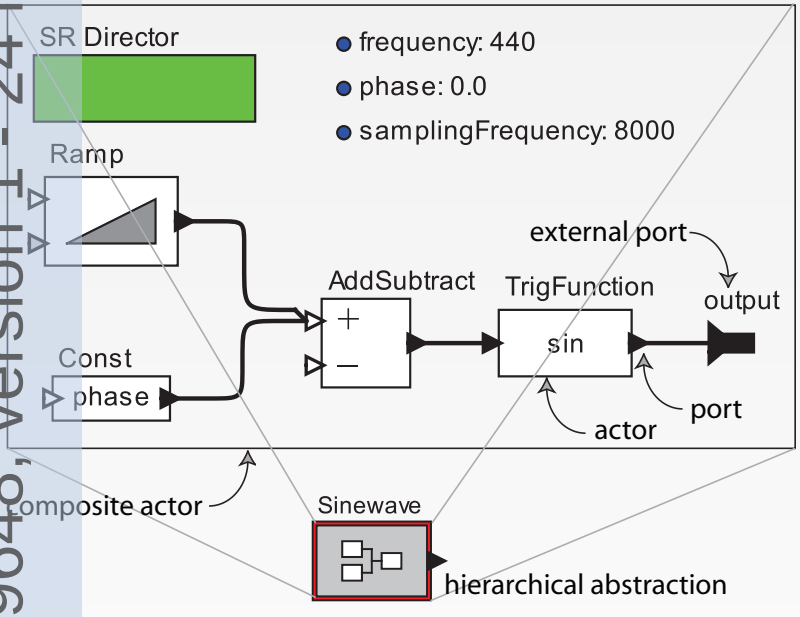
5 Some Related Work

Ptolemy

ptolemy.eecs.berkeley.edu

tel-00539648, version 1 - 24 Nov 2010

MoCC : Model of Computation and Communication



- Components are actors
- The director implements a MoCC
- Hierarchical framework

- ∃ a catalogue of predefined MoCCs:

  Synchronous Reactive, Discrete Event, Continuous Time, etc.

tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration
- Virtual Prototyping of Embedded Systems
- Modeling Hardware/Software with Synchronous Languages
- Ptolemy
- Specifying Components: Contracts

2 Overview of 42 & Examples [GPCE07]

3 Hardware Simulation by Interpreting Contracts [COORD09]

4 Using 42 Together with Existing Approaches [EMSOFT09]

5 Some Related Work

tel-00539648, version 1 - 24 Nov 2010

Classification of Contracts in the CBSE Community

(Component-Based Software Engineering)

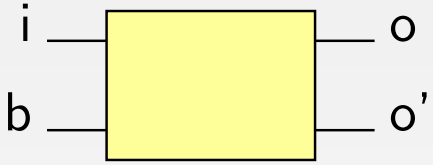
- Syntactic contracts
  - Interface Description languages (Java-IDL, WSDL, IDL, etc.)
- Behavioral contracts
  - pre and post conditions, assertions, etc. (Design by Contracts^{TM}, Eiffel, iContracts, etc.)
- Synchronization contracts
  - Sequence of method-calls, Component synchronization, etc. (PROCOL, Interface automata, Session types, etc.)
- Quality of service contracts
  - Resource consumption, Image quality, etc.

tel-00539648, version 1 - 24 Nov 2010

Contracts in the Hardware Community

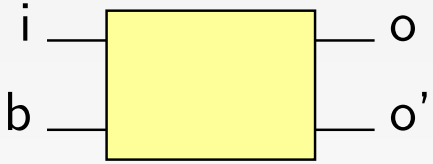
and the Synchronous Languages

- Don't care conditions
  - Hardware optimization



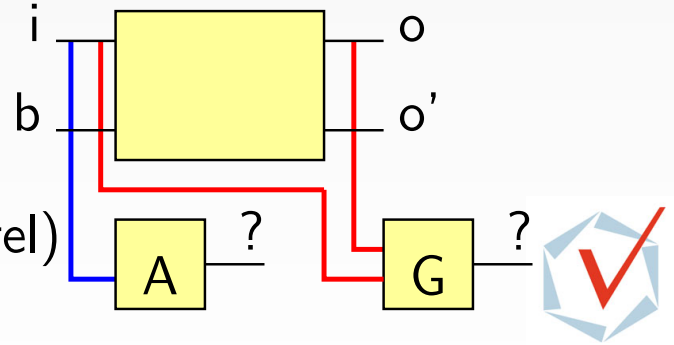
Input (ib)=(01) never occurs

- Conditional dependencies in Signal
  - Cycle analysis in synchronous circuits



o depends on i when b=true

- Assume/Guarantee reasoning
  - Modular verification (K. McMillan)
  - Executable specifications for Lustre (L. Morel)



tel-00539648, version 1 - 24 Nov 2010

Observation & Motivations of 42

In each context of Virtual Prototyping of embedded systems, there exists a notion of components and MoCCs related issues

42 aims at providing:

- A language-independent component-based framework for modeling hardware/software systems.
- Support for a clean definition of components, and help enforcing the FAMAPASAP (Forget As Much As Possible As Soon As Possible) principle.
- Support for integration of existing code and models in open virtual prototyping environments.

tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration

2 Overview of 42 & Examples [GPCE07]
- Components & Assemblies
- Explicit Specification of Components (Contracts)

3 Hardware Simulation by Interpreting Contracts [COORD09]

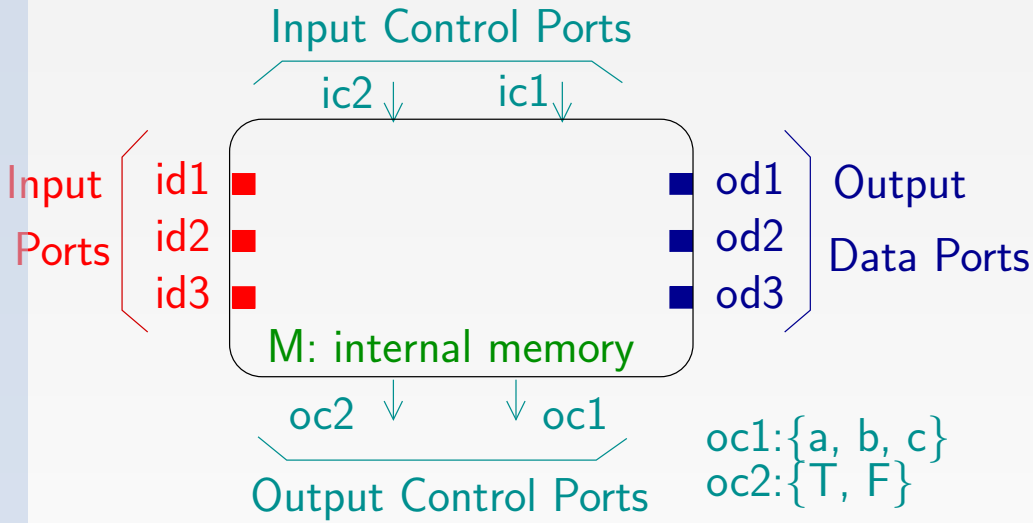
4 Using 42 Together with Existing Approaches [EMSOFT09]

5 Some Related Work

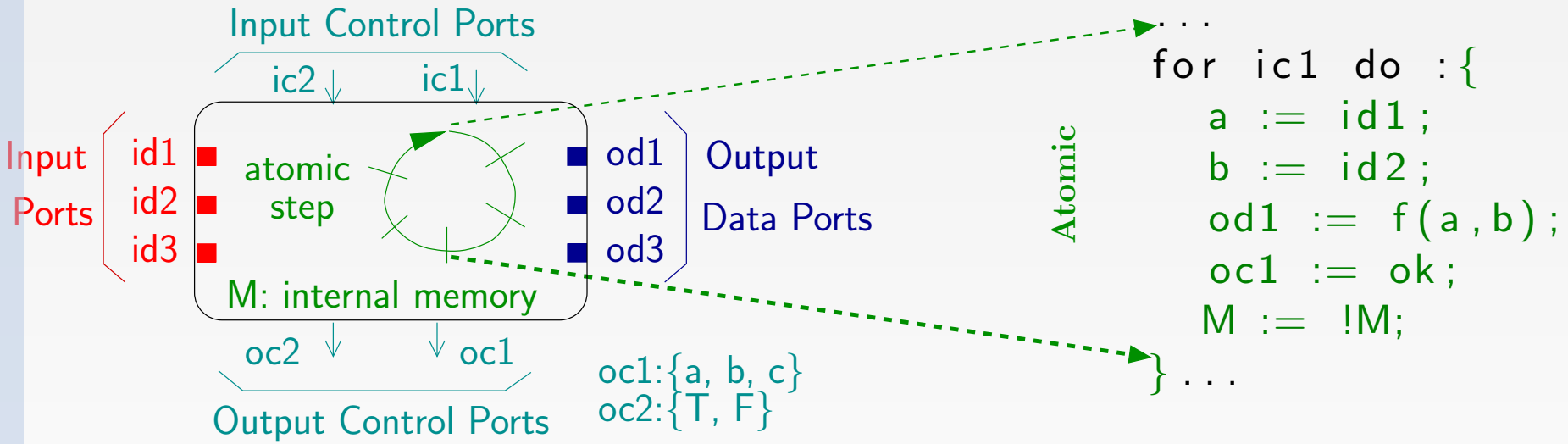
6 Summary

tel-00539648, version 1 - 24 Nov 2010

42 in a Nutshell: Basic Components



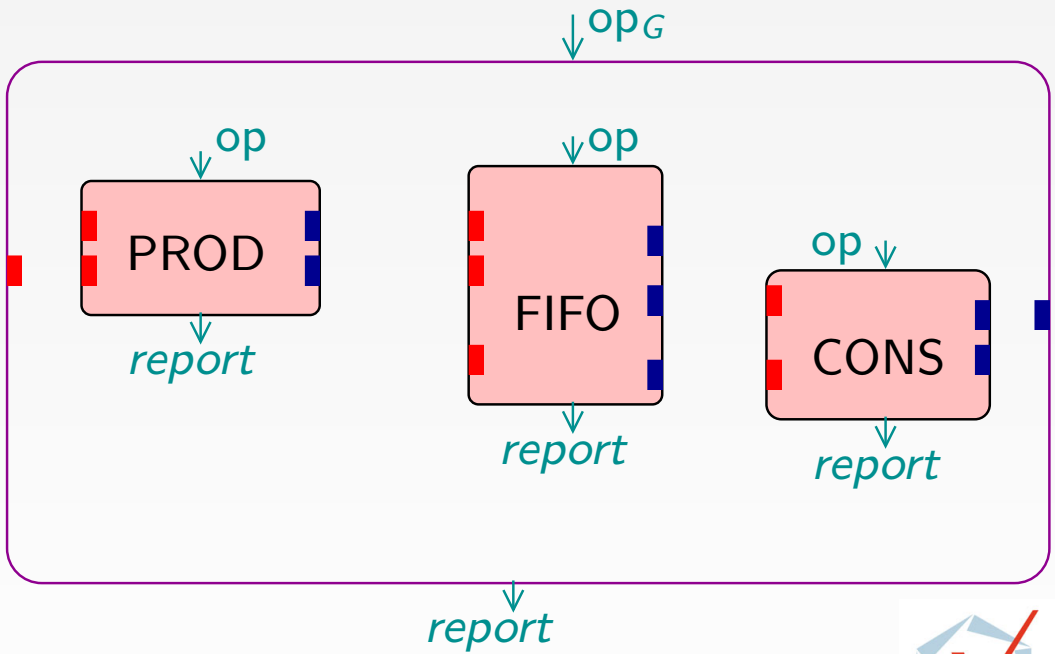
42 in a Nutshell: Basic Components



tel-00539648, version 1 - 24 Nov 2010

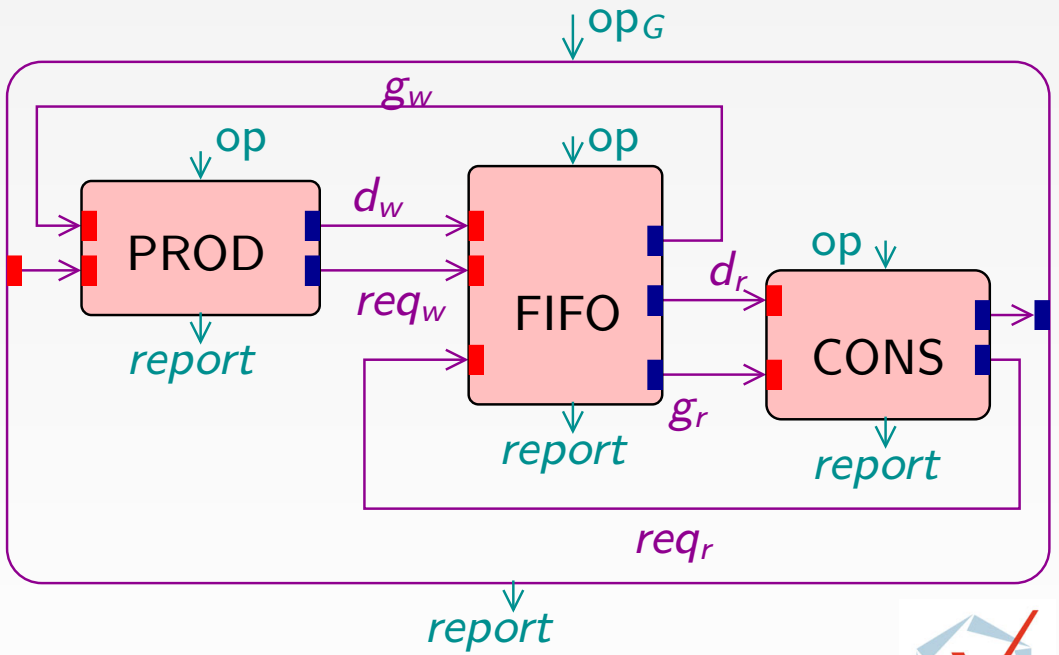
tel-00539648, version 1 - 24 Nov 2010

42 in a Nutshell: Assembling Components



tel-00539648, version 1 - 24 Nov 2010

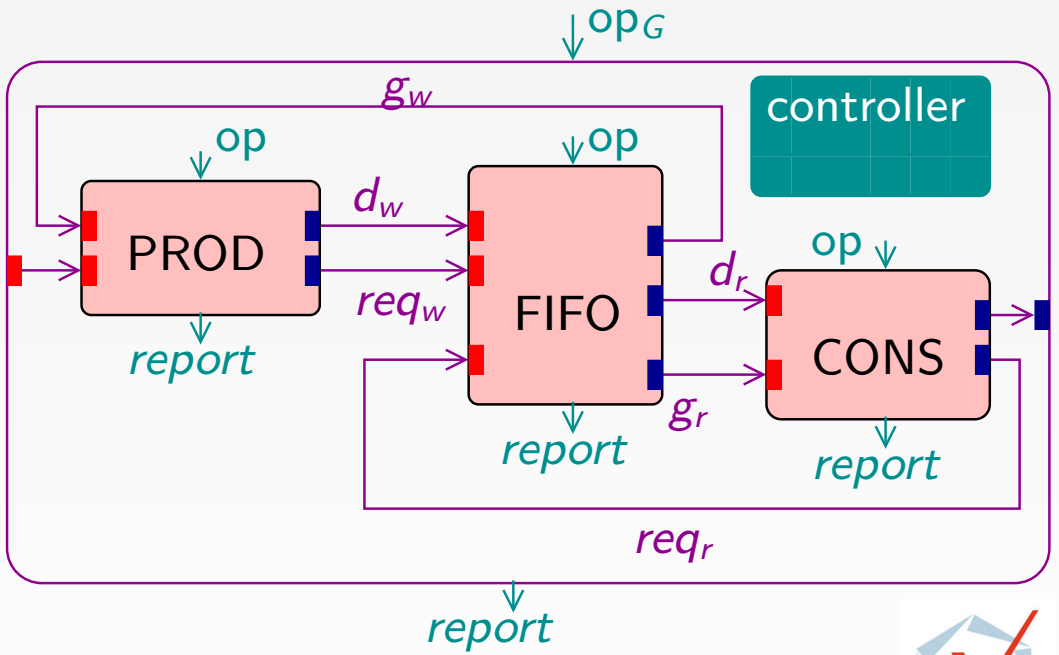
42 in a Nutshell: Assembling Components



tel-00539648, version 1 - 24 Nov 2010

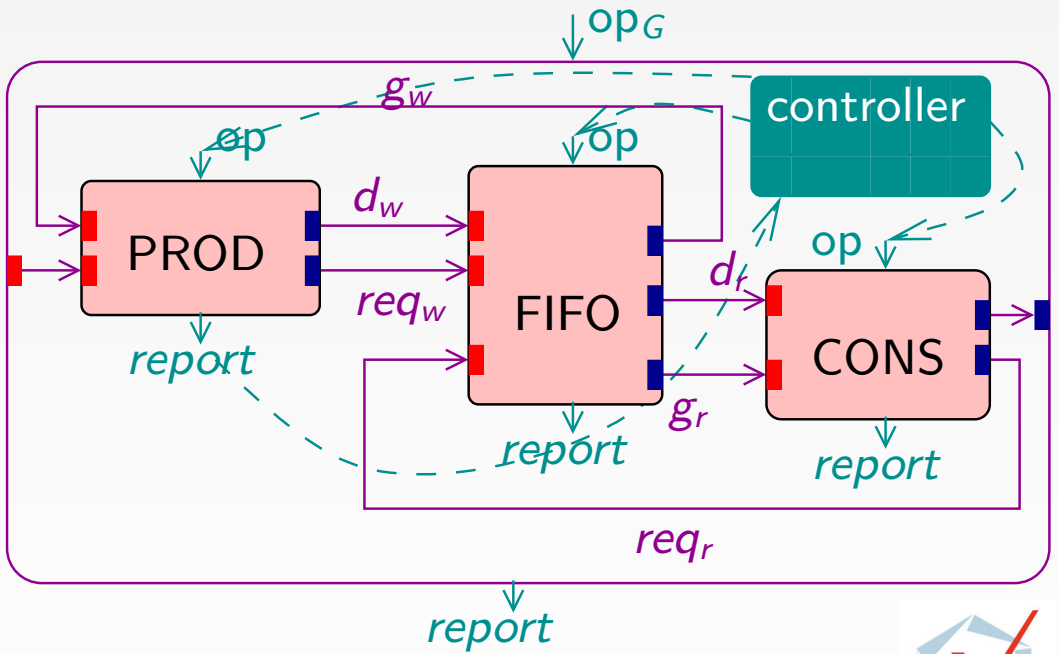
42 in a Nutshell: Assembling Components

For each OP_G the controller:



42 in a Nutshell: Assembling Components

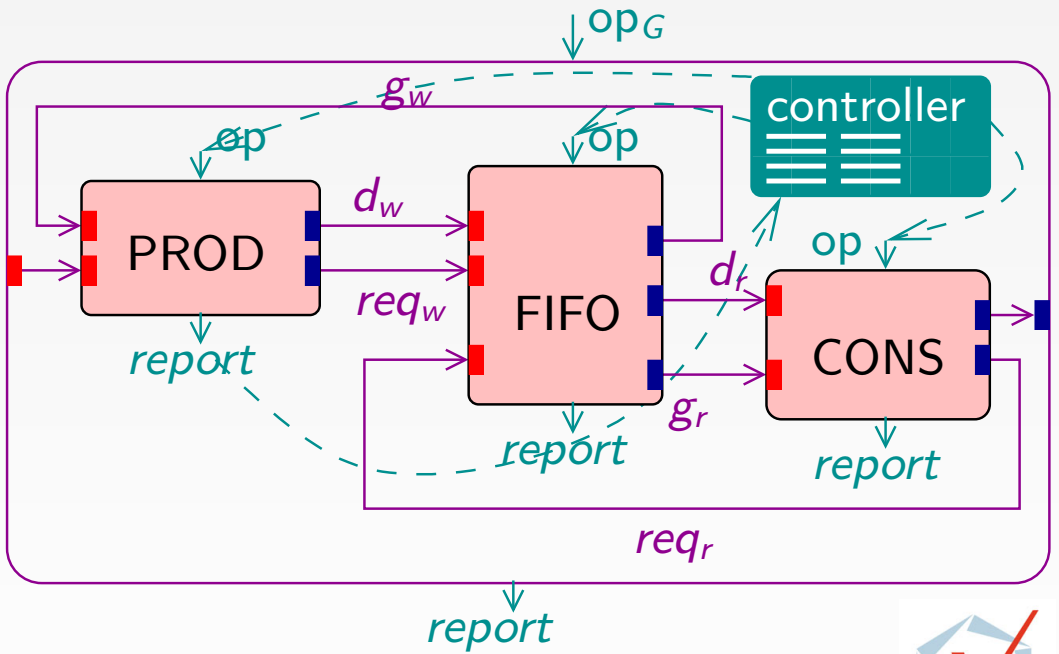
For each OP_G the controller:
- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)



42 in a Nutshell: Assembling Components

For each OP_G the controller:

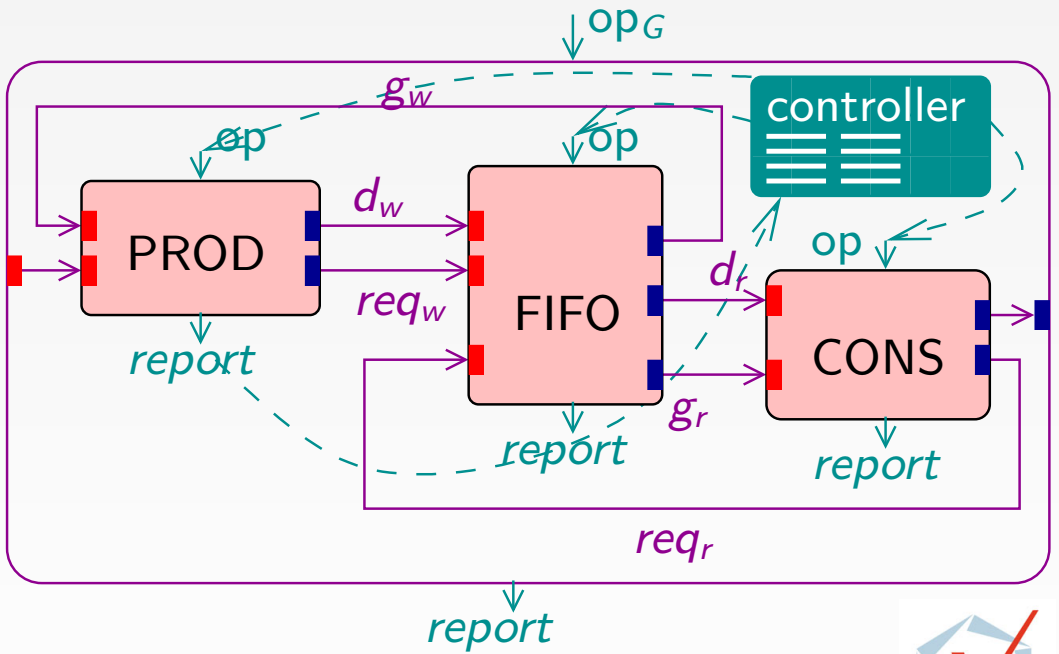
- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr,...)



42 in a Nutshell: Assembling Components

For each OP_G the controller:

- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr,...)
- Sets global output values



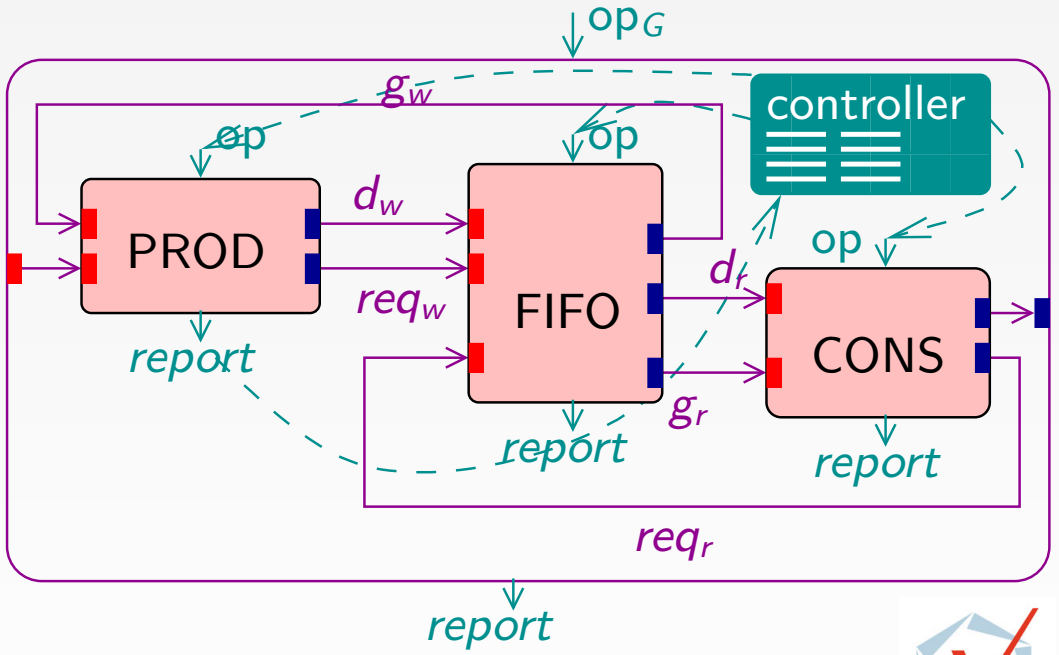
42 in a Nutshell: Assembling Components

tel-00539648, version 1 - 24 Nov 2010

```
Controller is {
 var M : bool;
 for OP_G do :{ /*defines opg.
  dw, dr, reqw,...: fifo(1,int);
  M := random();
  if (M) {
   PROD.op ; reqw.put; reqw.get;
   FIFO.op ; gw.put; gw.get;
   a := FIFO.report; /*reads oc.
   if(a==ok){ /*output control
    PROD.op;dw.put; dw.get;
    FIFO.op;/*activates FIFO.
     ...
   ...
  else{
   CONS.op; reqr.put; reqr.get;
   FIFO.op ; gr.put; gr.get;
   a := FIFO.report;
   ...
 }
}
```

For each OP_G the controller:

- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr,...)
- Sets global output values

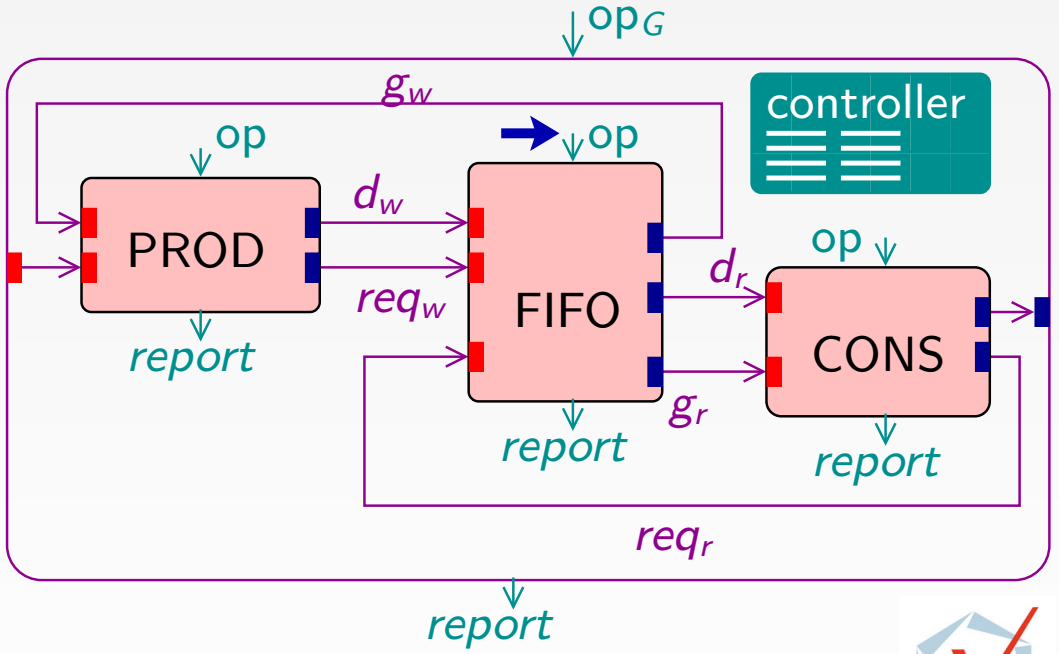


42 in a Nutshell: Assembling Components

```
Controller is {
 var M : bool;
 for OP_G do :{ /*defines opg.
  dw, dr, reqw,...: fifo(1,int);
  M := random();
  if (M) {
   PROD.op ; reqw.put; reqw.get;
   FIFO.op ; gw.put; gw.get;
   a := FIFO.report; /*reads oc.
   if(a==ok){ /*output control
    PROD.op;dw.put; dw.get;
    FIFO.op;/*activates FIFO.
     ...
   ...
   else{
    CONS.op; reqr.put; reqr.get;
    FIFO.op ; gr.put; gr.get;
    a := FIFO.report;
   ...
 }
}
```

For each OP_G the controller:

- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr,...)
- Sets global output values



tel-00539648, version 1 - 24 Nov 2010

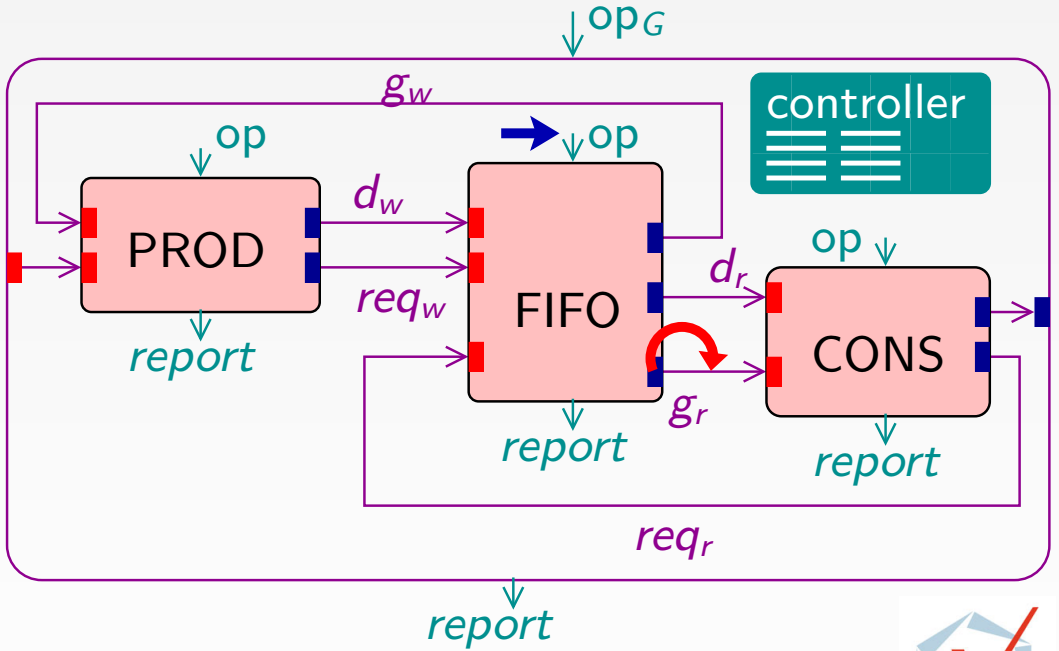
42 in a Nutshell: Assembling Components

tel-00539648, version 1 - 24 Nov 2010

```
Controller is {
 var M : bool;
 for OP_G do :{ /*defines opg.
  gw, dr, reqw,...: fifo(1,int);
  M := random();
  if (M) {
   PROD.op ; reqw.put; reqw.get;
   FIFO.op ; gw.put; gw.get;
   a := FIFO.report; /*reads oc.
   if(a==ok){ /*output control
    PROD.op;dw.put; dw.get;
    FIFO.op;/*activates FIFO.
     ...
   ...
   else{
    CONS.op; reqr.put; reqr.get;
    FIFO.op ; gr.put; gr.get;
    a := FIFO.report;
   ...
 }
}
```

For each OP_G the controller:

- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr,...)
- Sets global output values



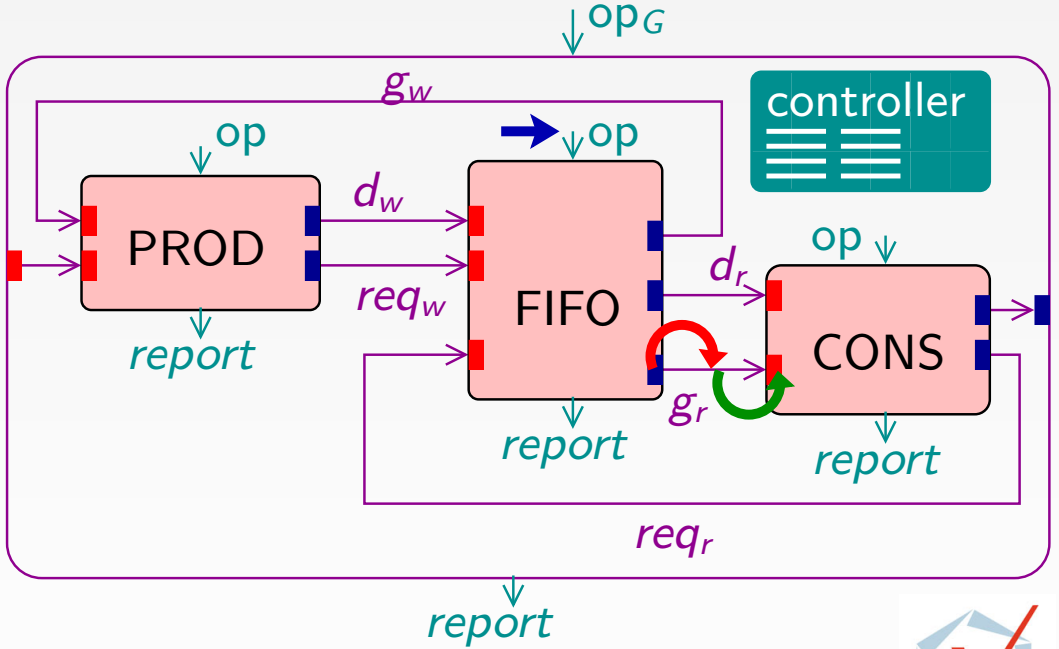
42 in a Nutshell: Assembling Components

tel-00539648, version 1 - 24 Nov 2010

```
Controller is {
 var M : bool;
 for OP_G do :{ /*defines opg.
  dw, dr, reqw,...: fifo(1,int);
  M := random();
  if (M) {
   PROD.op ; reqw.put; reqw.get;
   FIFO.op ; gw.put; gw.get;
   a := FIFO.report; /*reads oc.
   if(a==ok){ /*output control
    PROD.op;dw.put; dw.get;
    FIFO.op;/*activates FIFO.
     ...
   ...
   else{
    CONS.op; reqr.put; reqr.get;
    FIFO.op ; gr.put; gr.get;
    a := FIFO.report;
   ...
 }
}
```

For each OP_G the controller:

- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr,...)
- Sets global output values



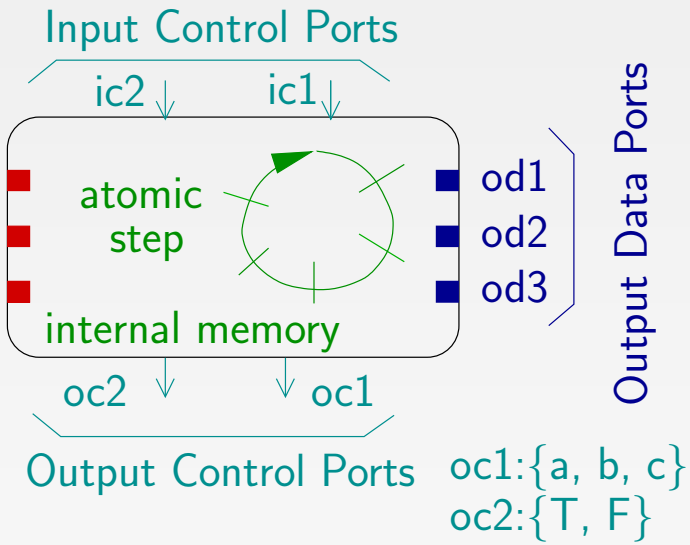
Summary of 42 Basics

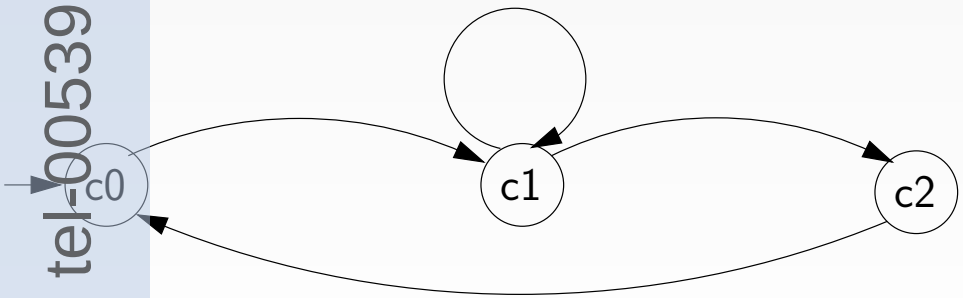
tel-00539648, version 1 - 24 Nov 2010

Summary of 42

- Hierarchical model $\Longrightarrow$ modeling heterogeneity
  in the same spirit as Ptolemy
- Controllers are expressed as programs
  identification of the basic primitives for describing MoCCs
- Separation of control/data to enforce the FAMAPASAP
  more details later

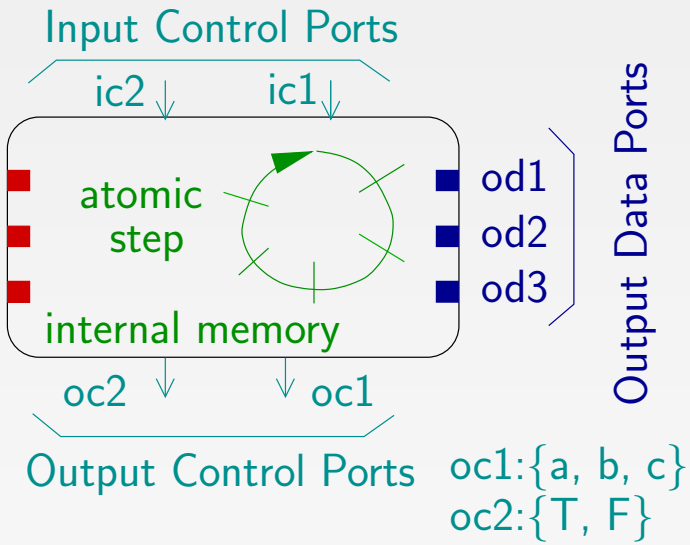
Control Contracts for 42 Components



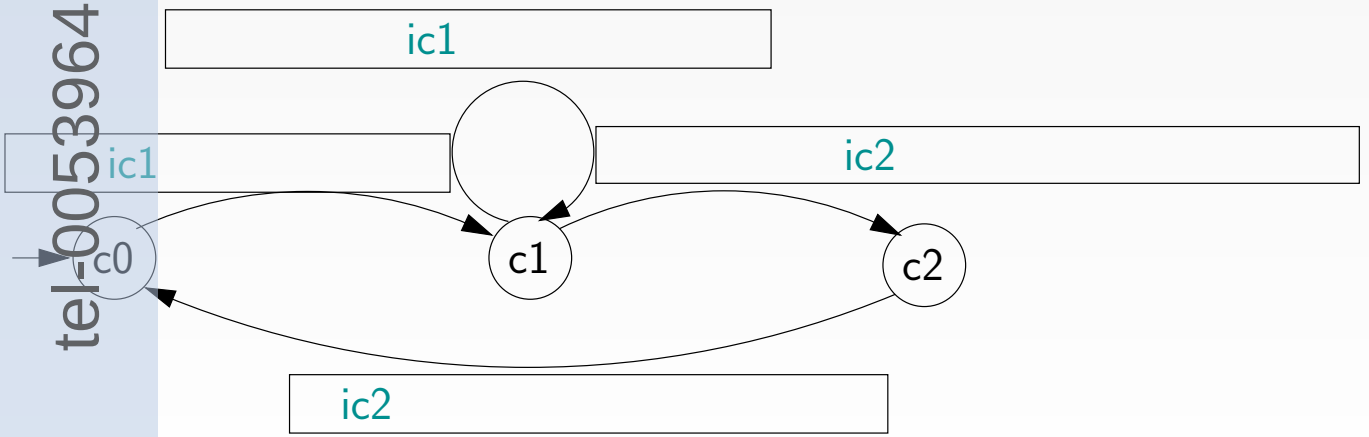


Input Data Ports
tel-00539648, version 1 - 24 Nov 2010

Control Contracts for 42 Components



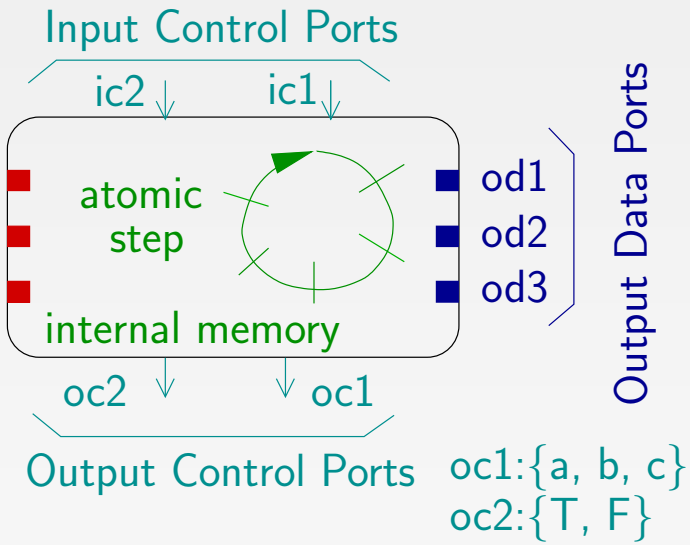
- Allowed activation sequences



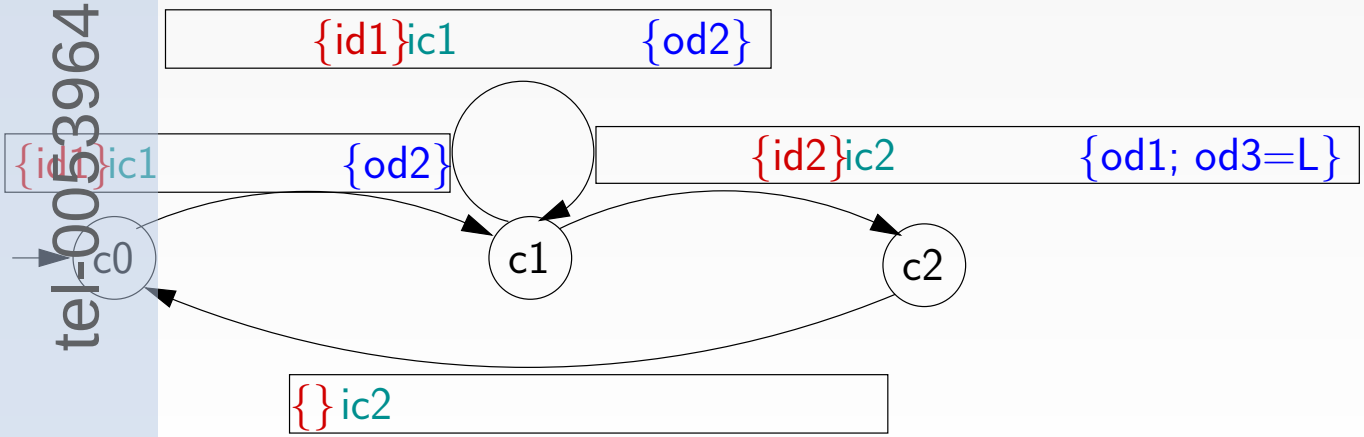
Input Data Ports
tel-00539648, version 1 - 24 Nov 2010

Control Contracts for 42 Components

Input Data Ports
tel-00539648, version 1 - 24 Nov 2010

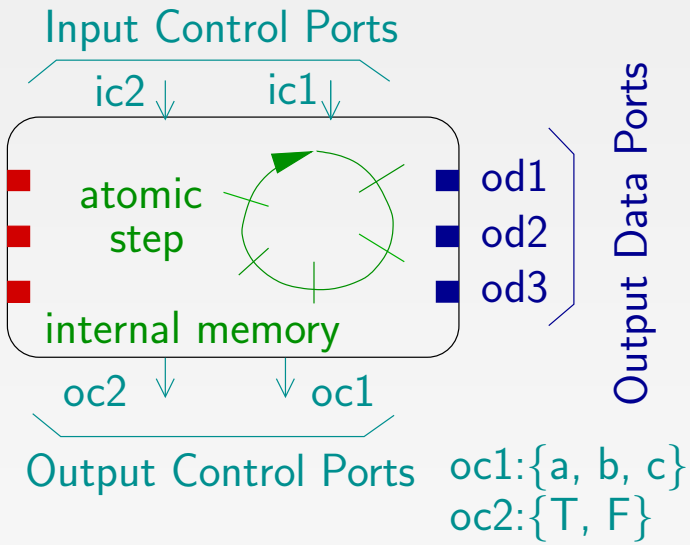


- Allowed activation sequences
- Data dependencies (Required, Provided)

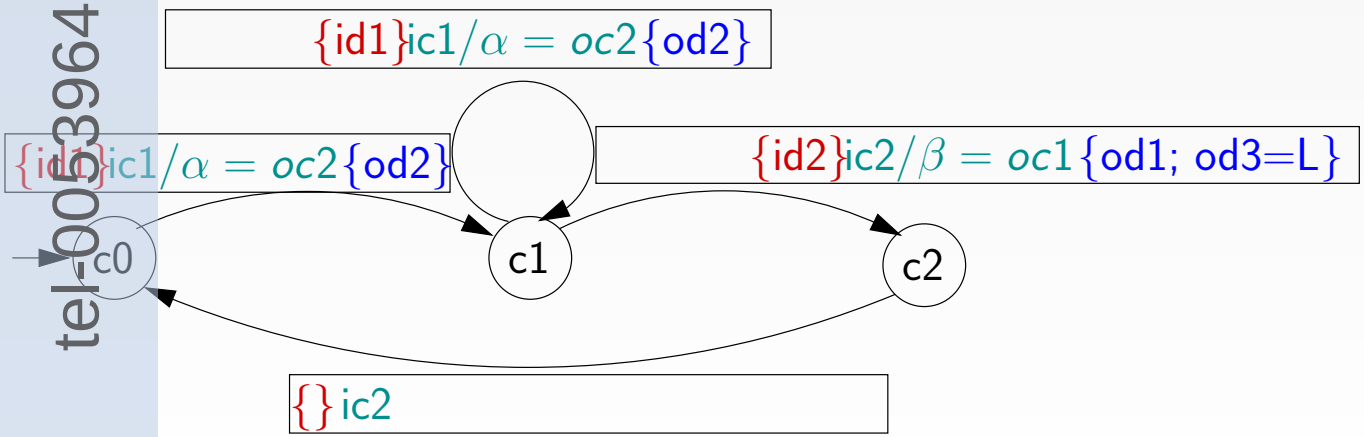


Control Contracts for 42 Components

Input Data Ports
tel-00539648, version 1 - 24 Nov 2010

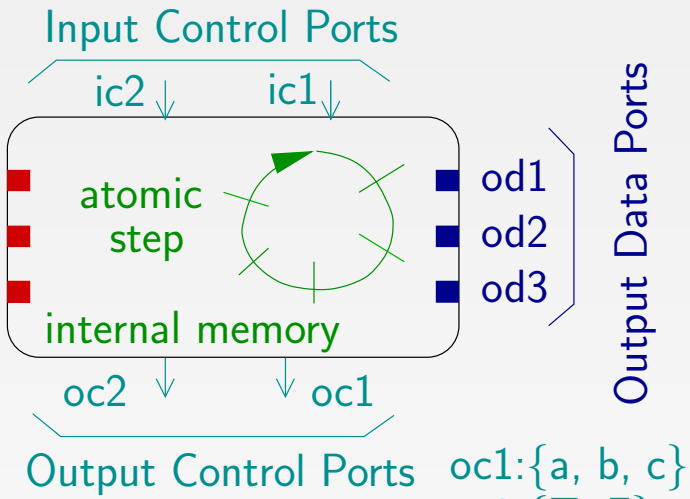


- Allowed activation sequences
- Data dependencies (Required, Provided)
- Control information

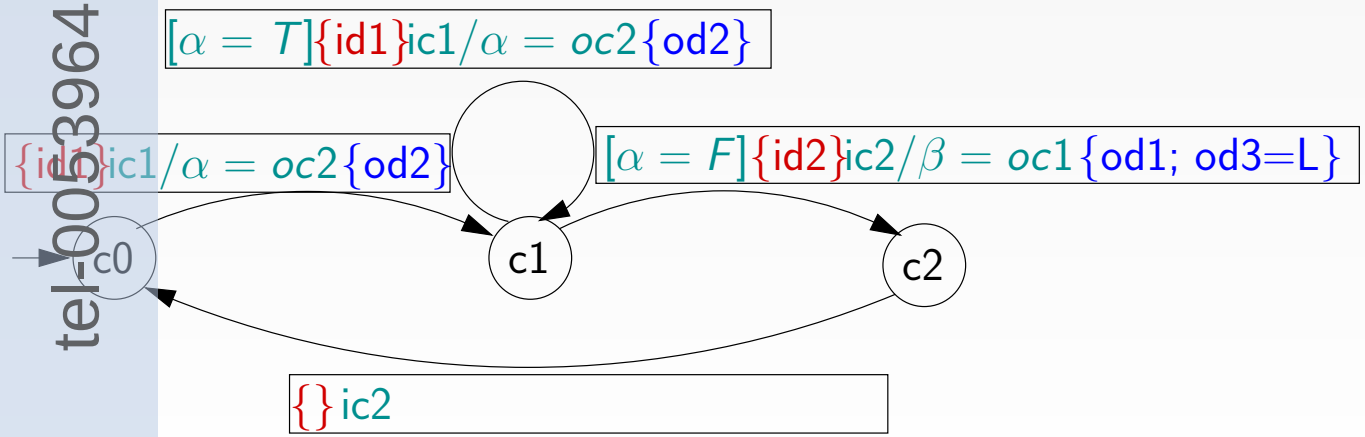


Control Contracts for 42 Components

Input Data Ports
tel-00539648, version 1 - 24 Nov 2010

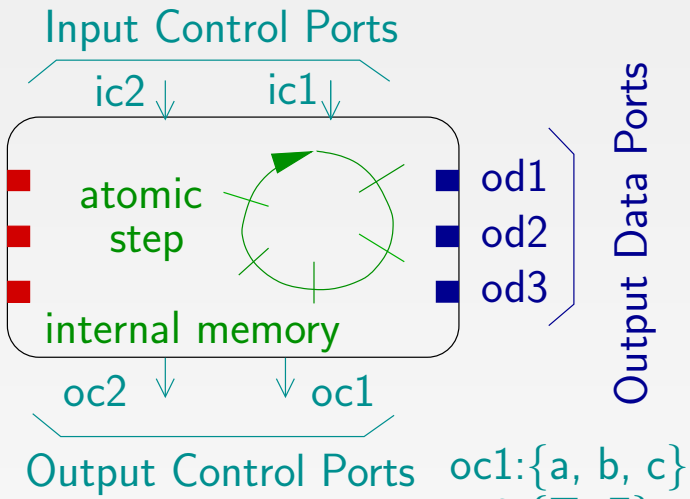


- Allowed activation sequences
- Data dependencies (Required, Provided)
- Control information
- Conditional activations.

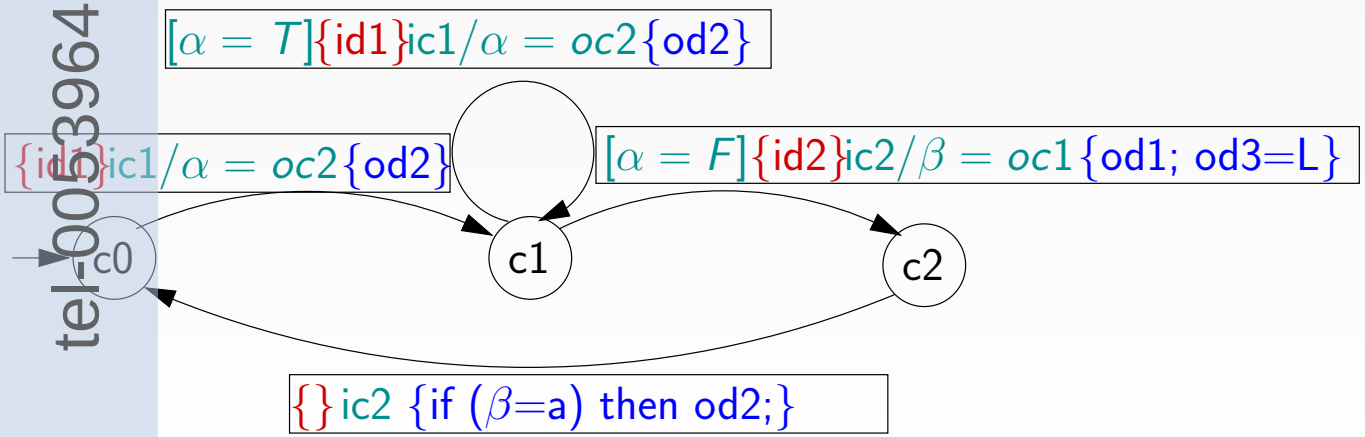


Control Contracts for 42 Components

Input Data Ports
tel-00539648, version 1 - 24 Nov 2010



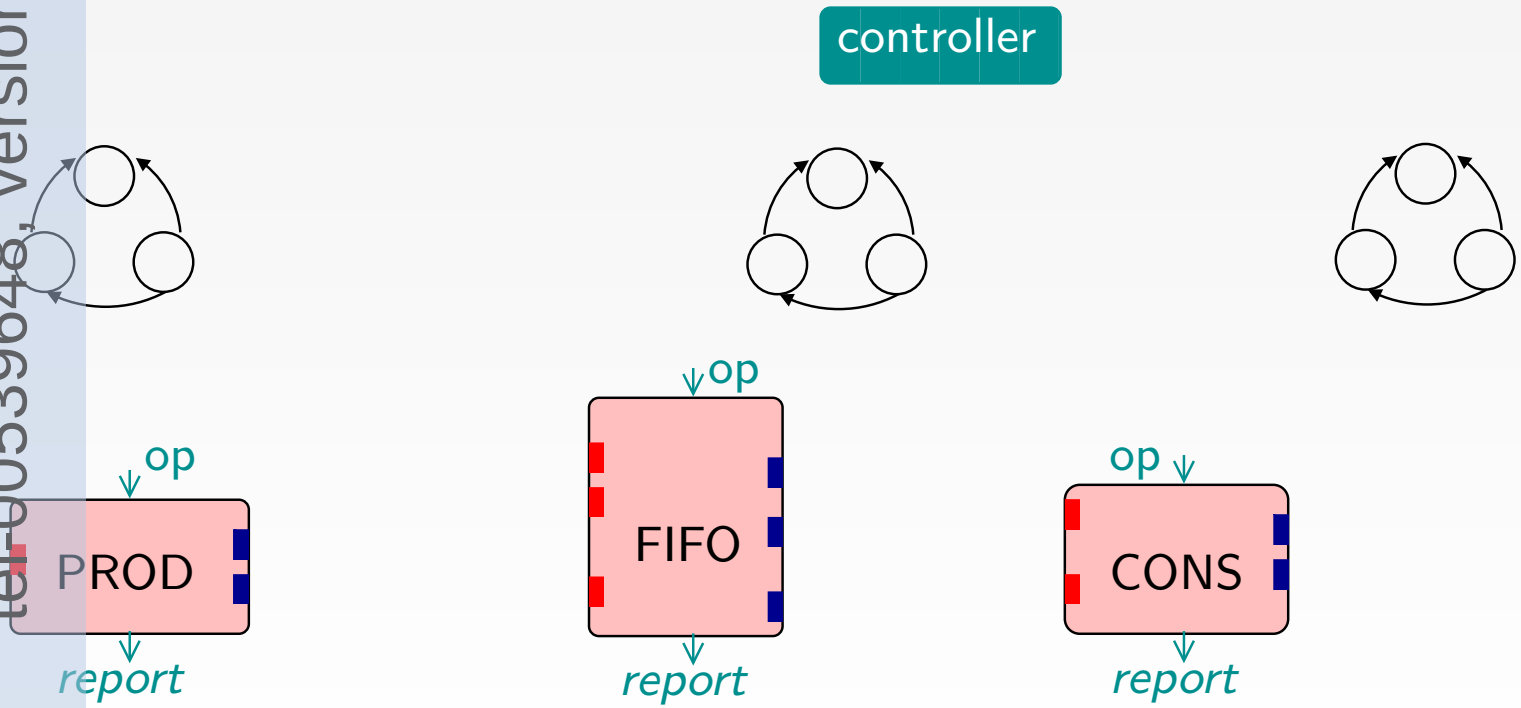
- Allowed activation sequences
- Data dependencies (Required, Provided)
- Control information
- Conditional activations.
- Conditional data dependencies.



What are 42 Contracts used for?

42 contracts (+ the Architecture Description Language)

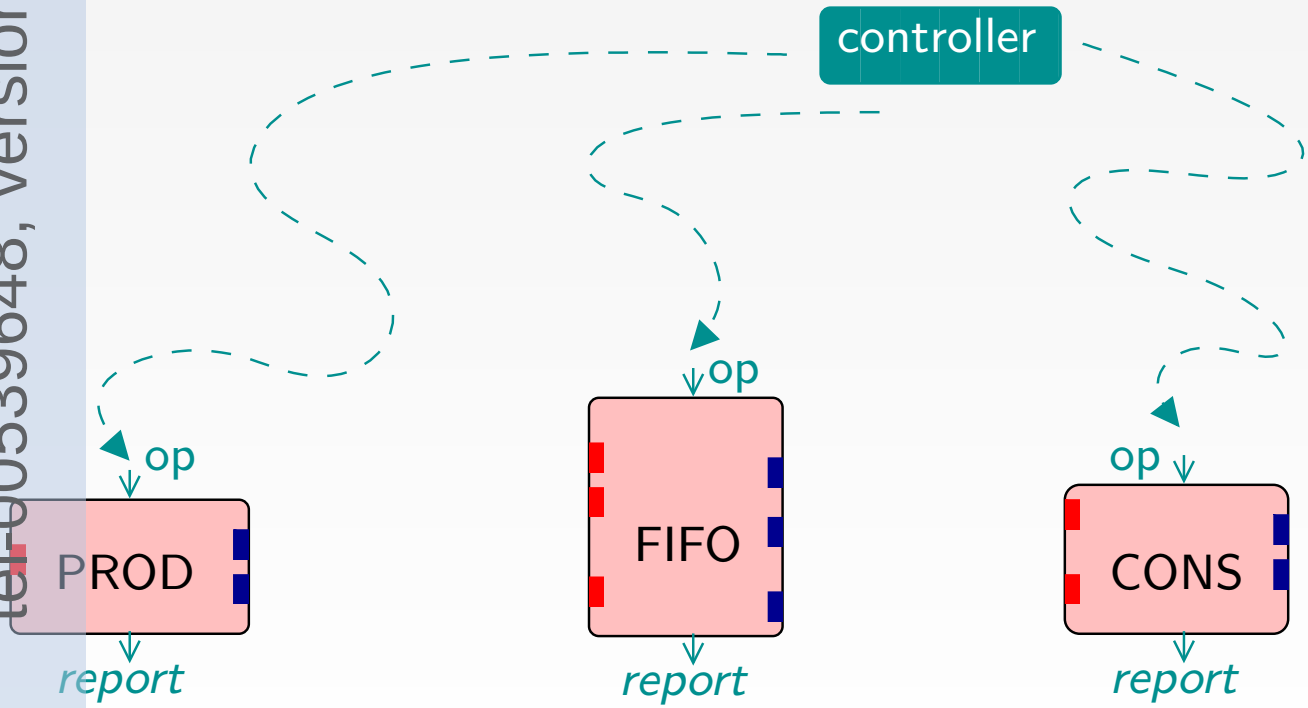
tel-00539648, version 1 - 24 Nov 2010



What are 42 Contracts used for?

42 contracts (+ the Architecture Description Language)

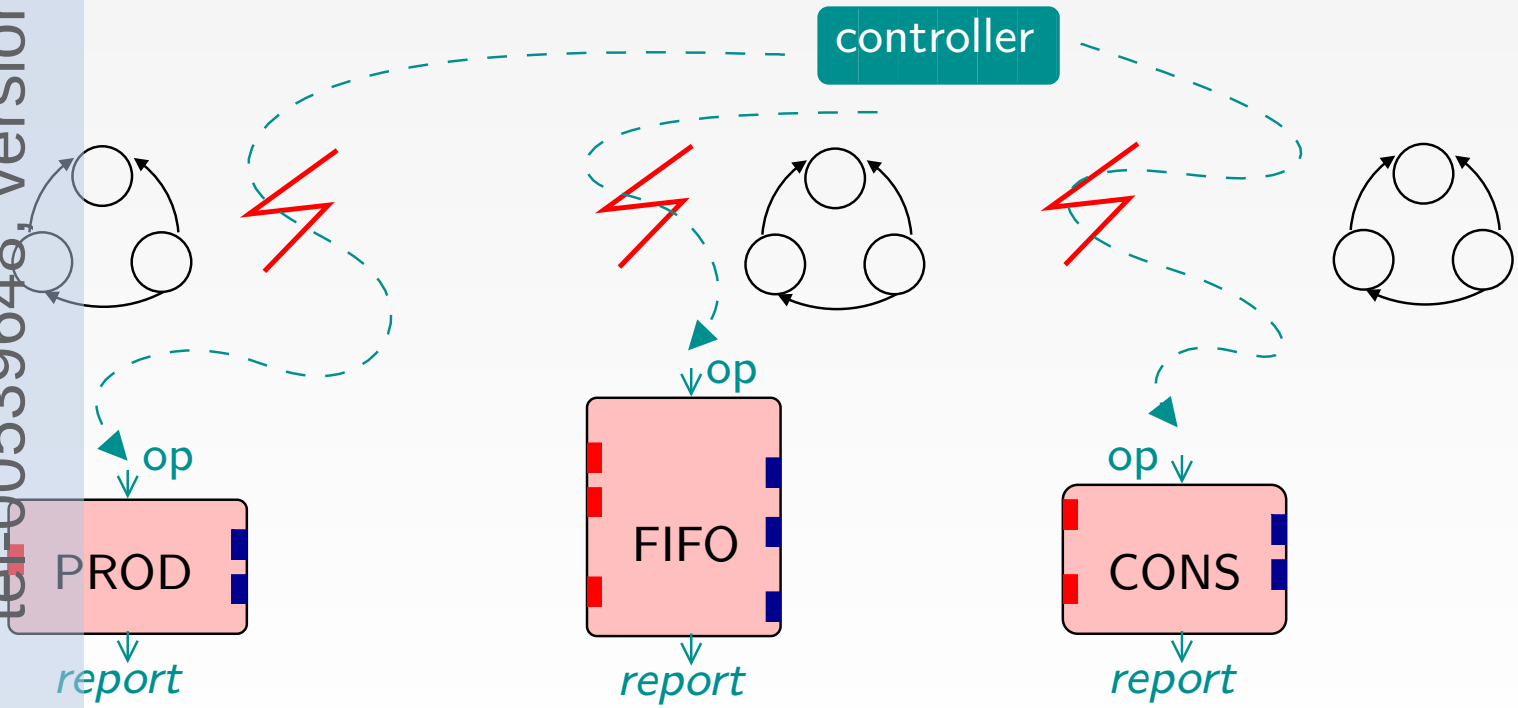
tel-00539648, version 1 - 24 Nov 2010



What are 42 Contracts used for?

42 contracts (+ the Architecture Description Language)

- Testing controllers/components consistency

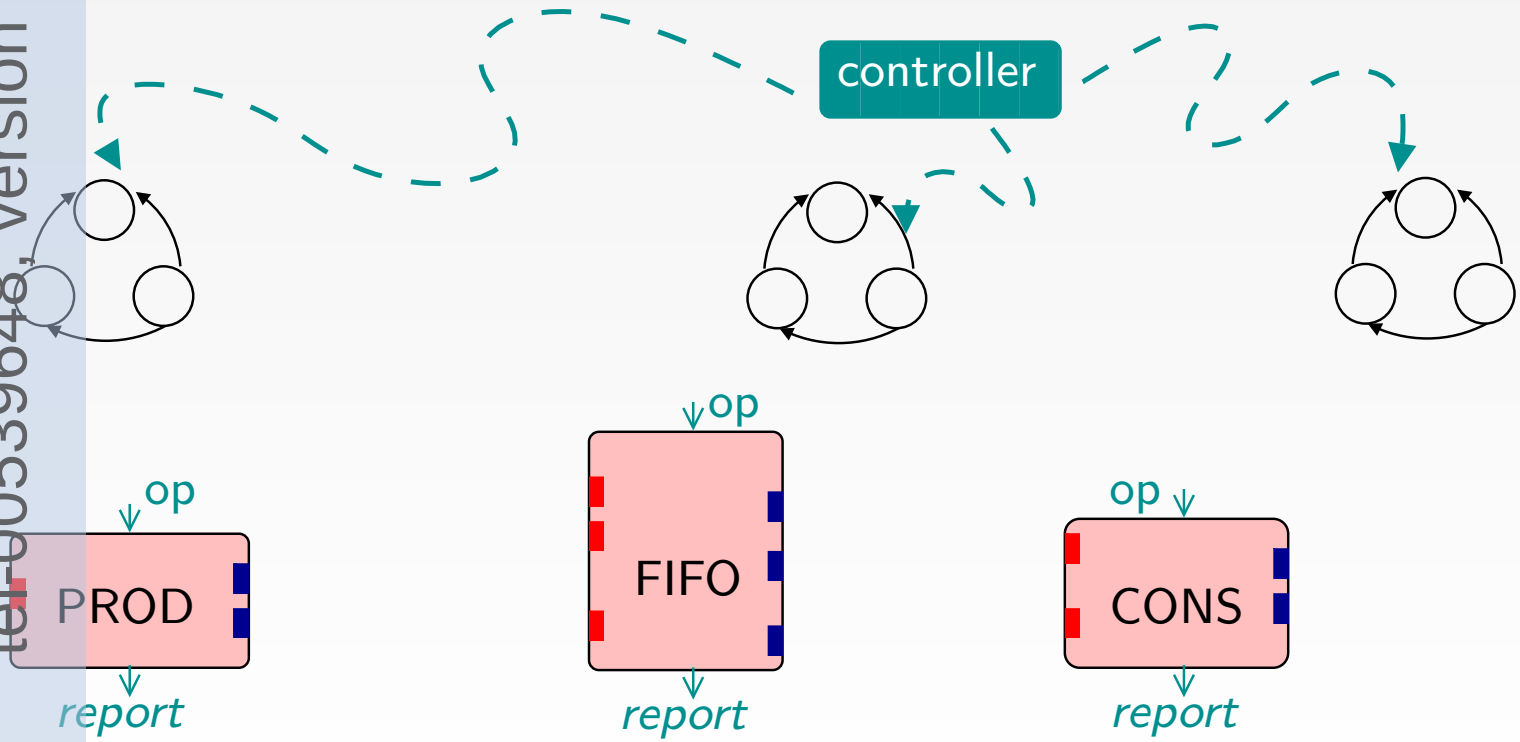


tel-00539648, version 1 - 24 Nov 2010

What are 42 Contracts used for?

42 contracts (+ the Architecture Description Language)

- Testing controllers/components consistency
- Contract Interpretation

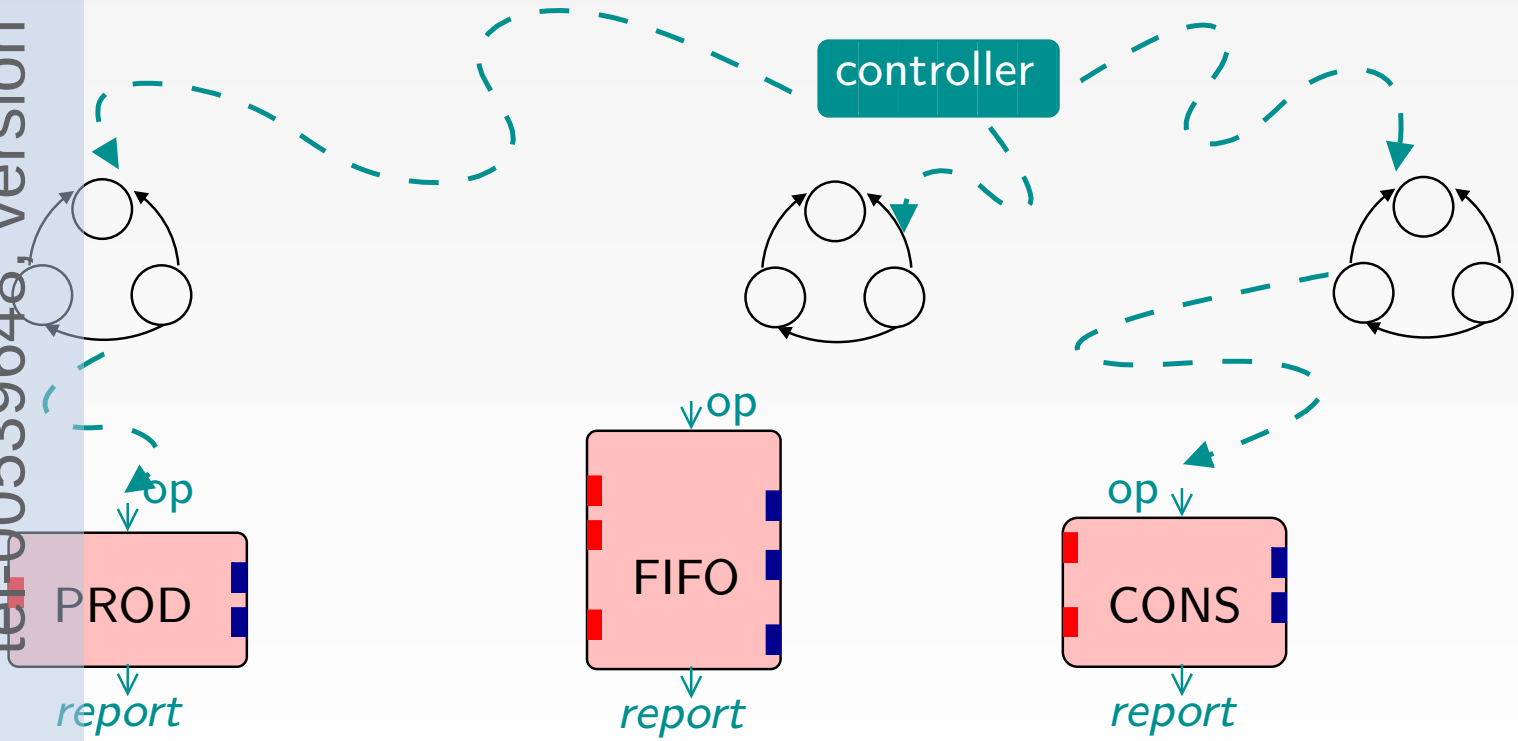


tel-00539648, version 1 - 24 Nov 2010

What are 42 Contracts used for?

42 contracts (+ the Architecture Description Language)

- Testing controllers/components consistency
- Contract Interpretation



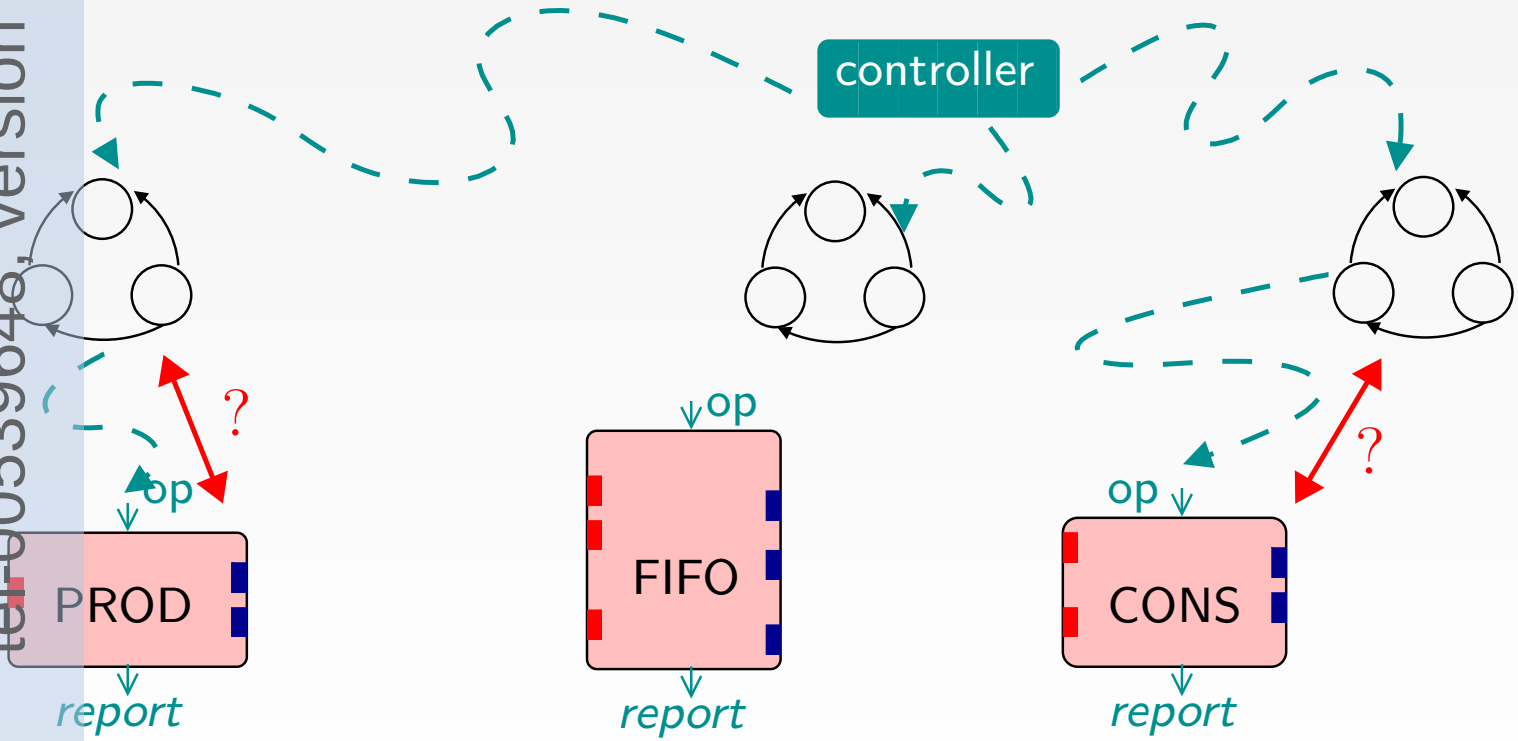
tel-00539648, version 1 - 24 Nov 2010

What are 42 Contracts used for?

42 contracts (+ the Architecture Description Language)

tel-00539648, version 1 - 24 Nov 2010

- Testing controllers/components consistency
- Contract Interpretation
- Testing component/contract consistency



tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration

2 Overview of 42 & Examples [GPCE07]

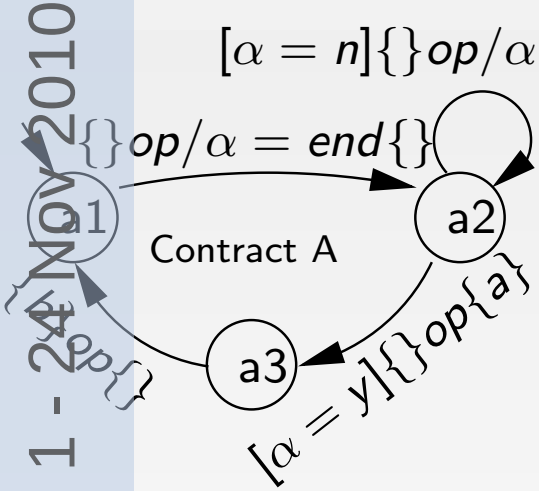
3 Hardware Simulation by Interpreting Contracts [COORD09]
- Contract Interpretation
- Contract Interpretation for Hardware Simulation
- Executing Embedded Software on Hardware Models

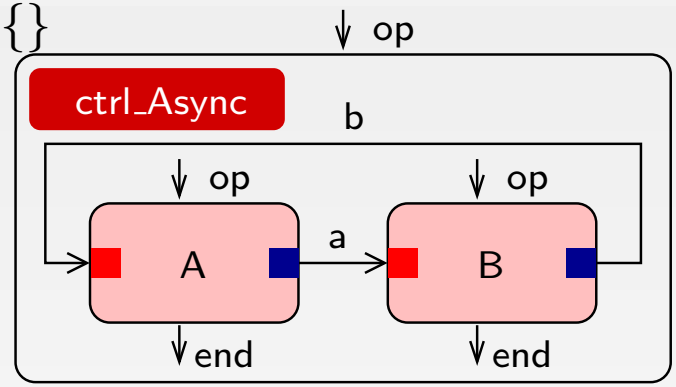
4 Using 42 Together with Existing Approaches [EMSOFT09]

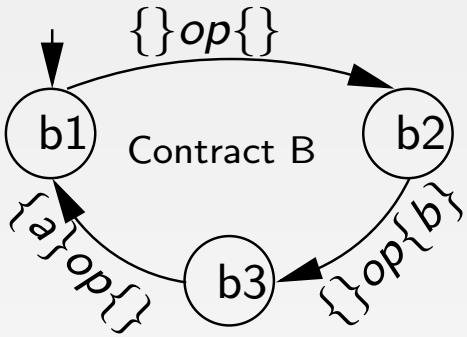
5 Some Related Work

6 Summary

Contract Interpretation: Principle

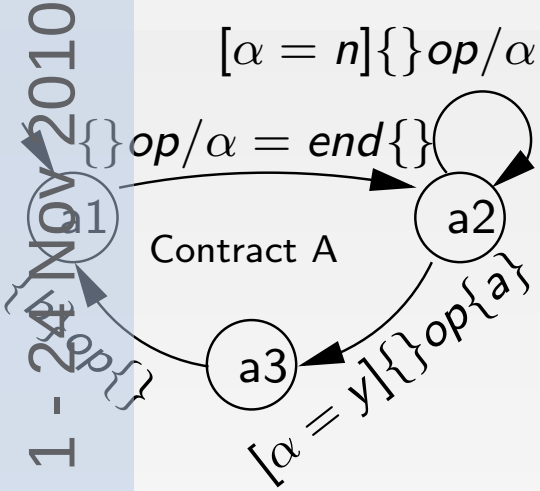


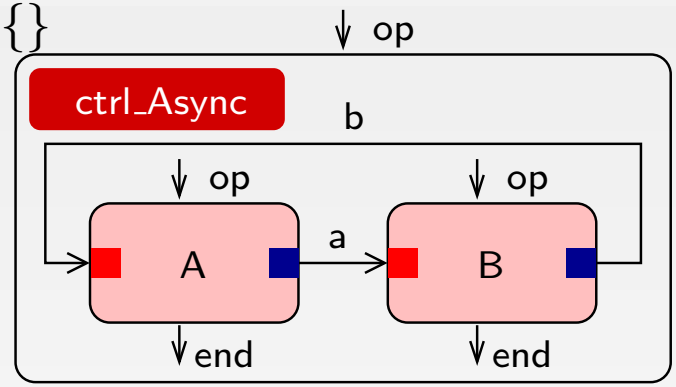


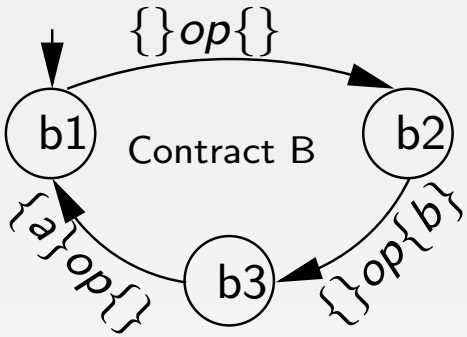


tel-00539648, version 1 - 24 Nov 2010

Contract Interpretation: Principle







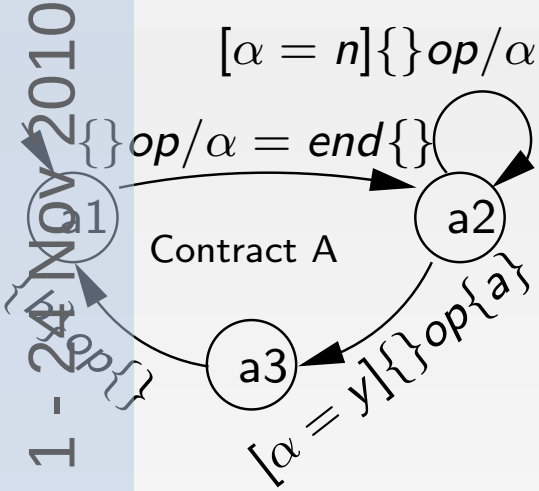
a=b1
available={}
$\alpha = \perp$

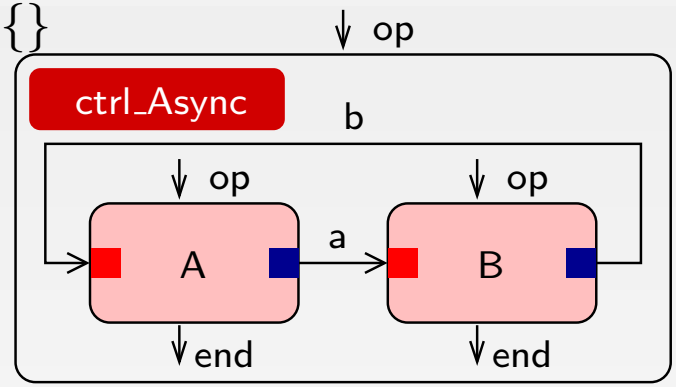
The controller maintains:

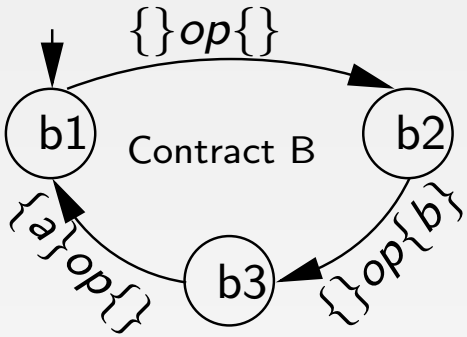
- The current state of each contract
- The set of available data
- The values of the variables(α)

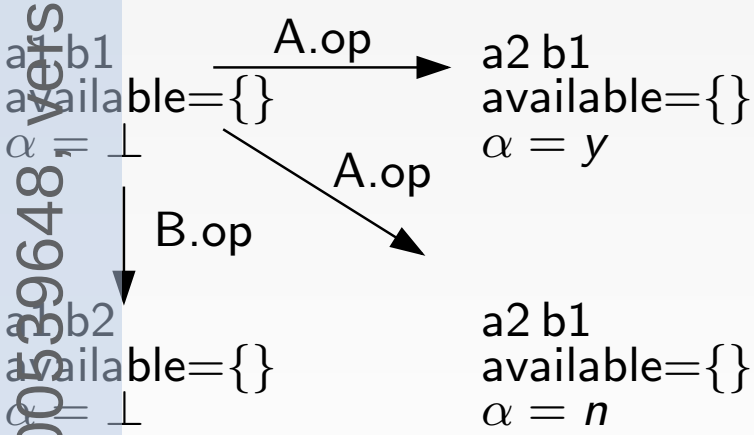
tel-00539648, version 1 - 24 Nov 2010

Contract Interpretation: Principle







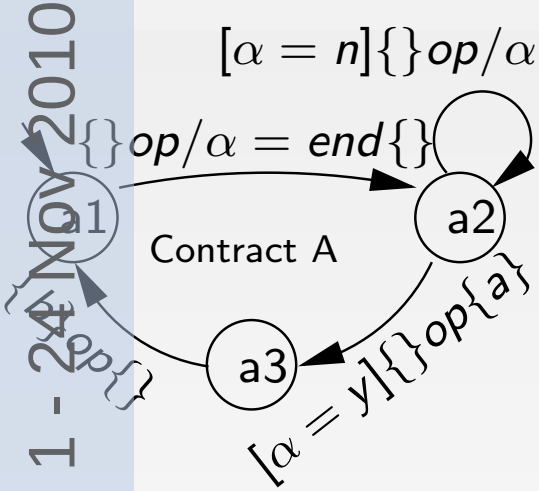


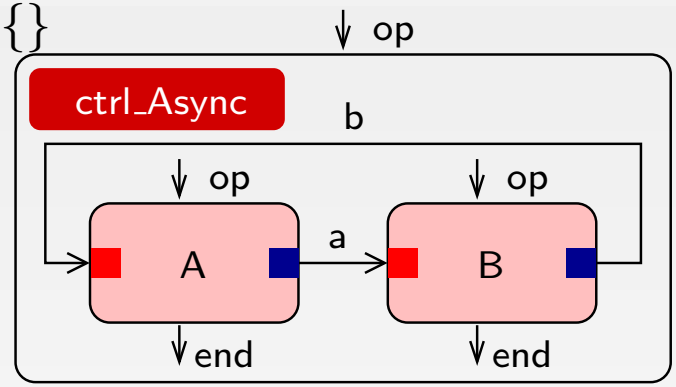
For each global activation :

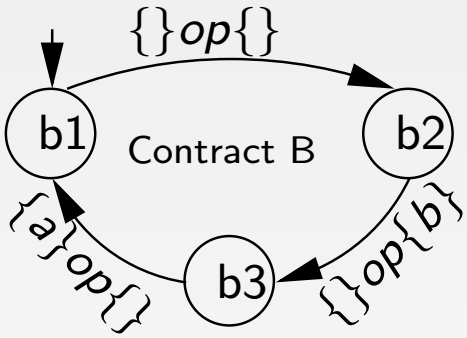
- Depending on the available data it selects a component and activates it
- The set of available data is updated
- The output controls are given non-deterministic values

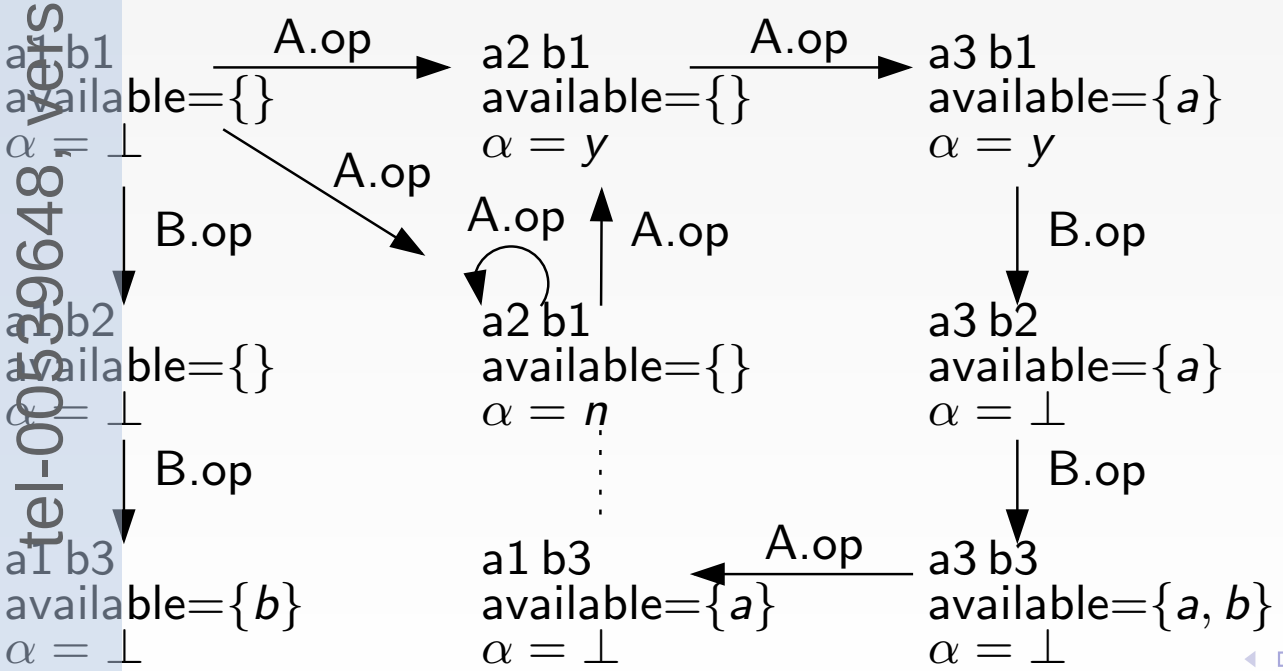
tel-00539648, version 1 - 24 Nov 2010

Contract Interpretation: Principle





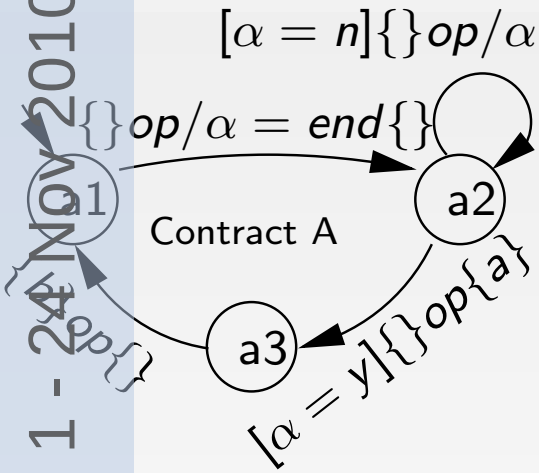


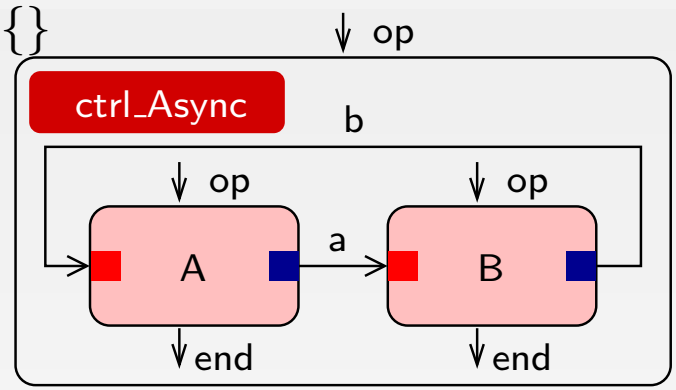


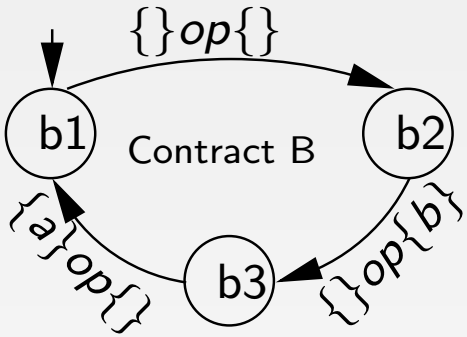
tel-00539648, version 1 - 24 Nov 2010

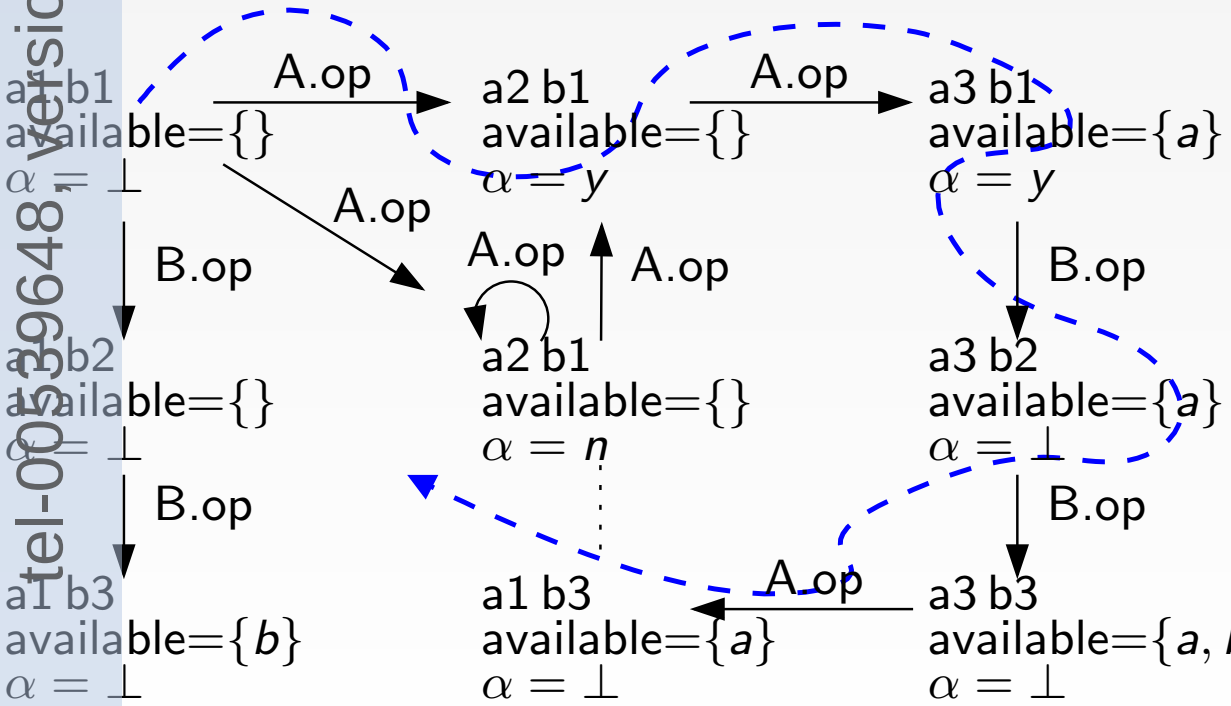
Contract Interpretation: Principle

tel-00539648, version 1 - 24 Nov 2010



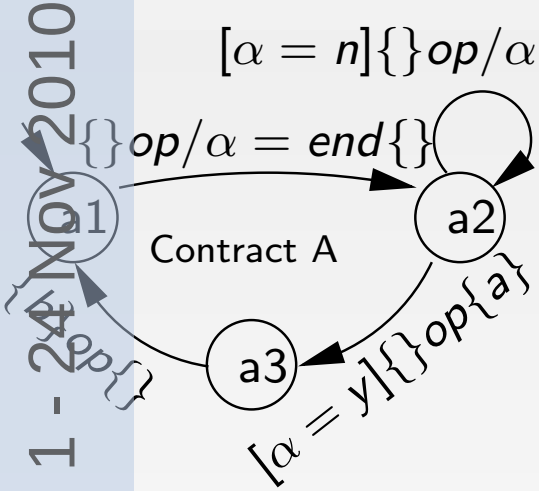


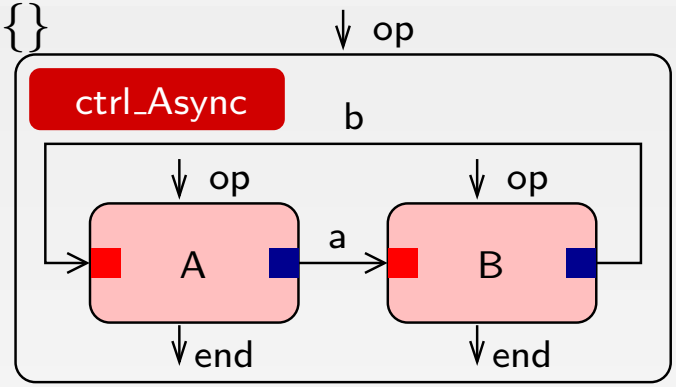


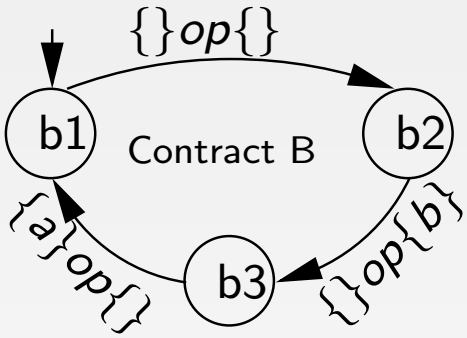


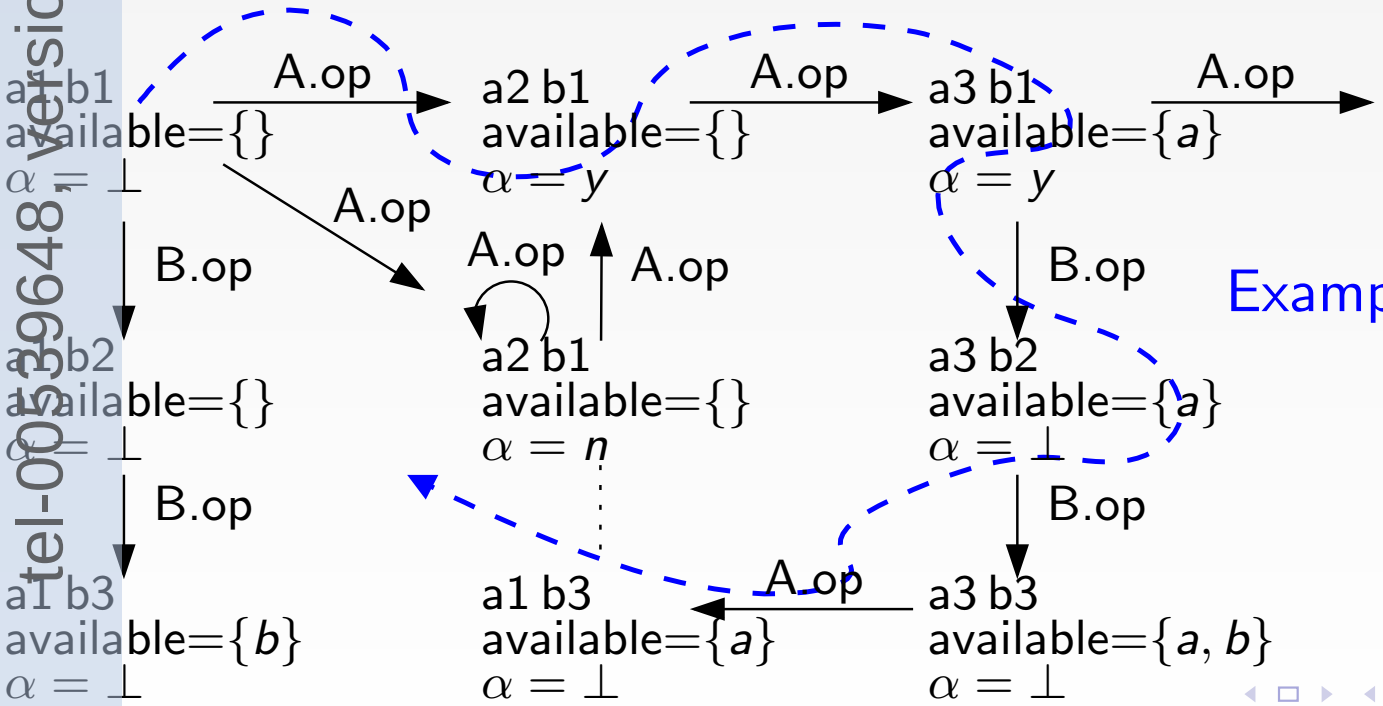
Example execution path

Contract Interpretation: Principle





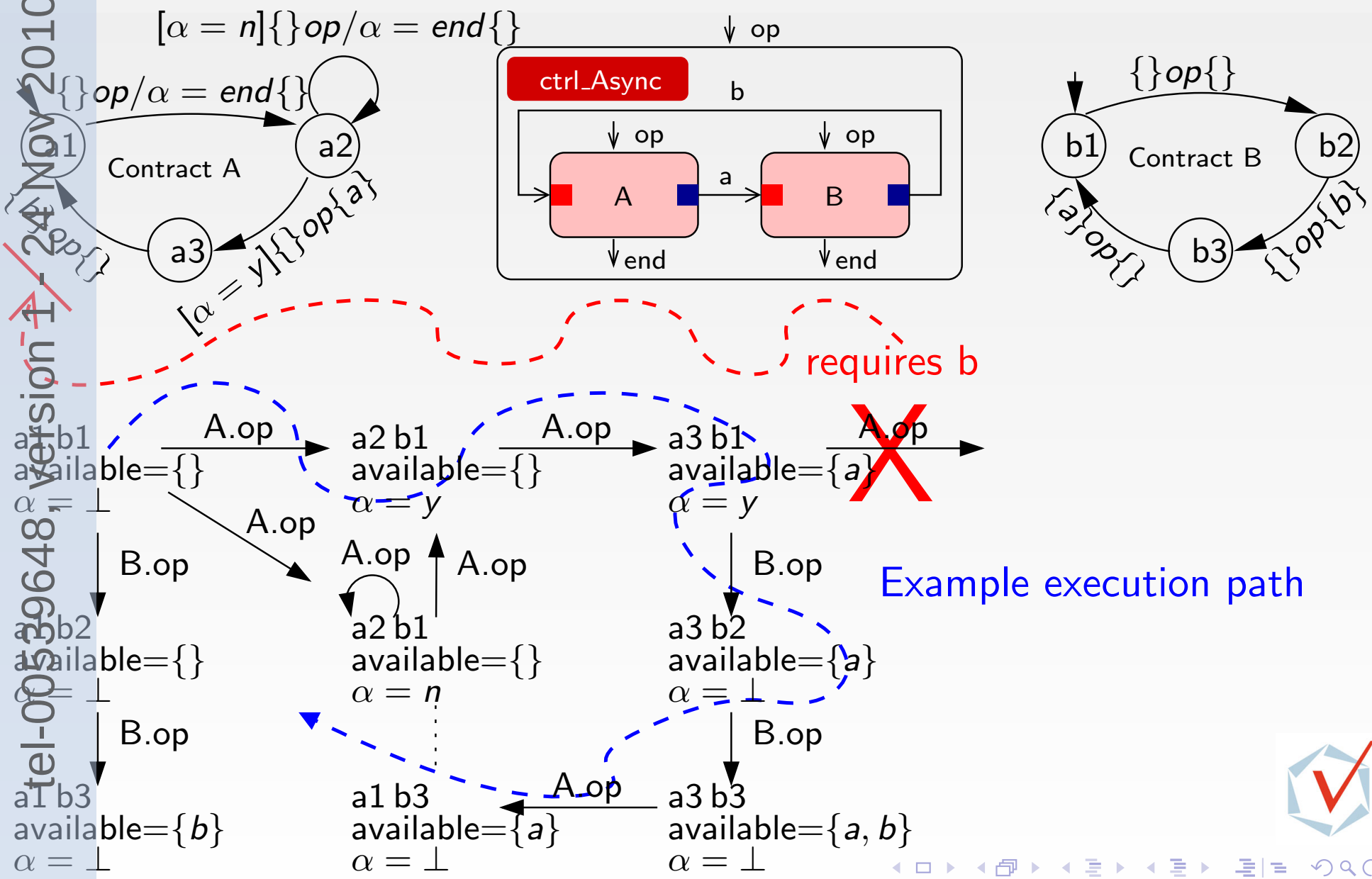




Example execution path

tel-00539648, version 1 - 24 Nov 2010

Contract Interpretation: Principle



tel-00539648, version 1 - 24 Nov 2010

Modeling Hardware with 42: Case Study

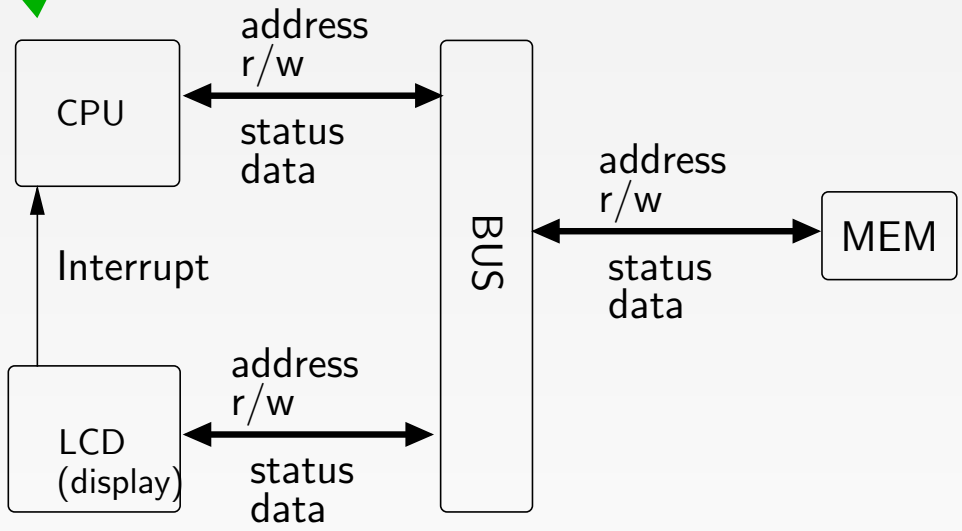
tel-00539648, version 1 - 24 Nov 2010

Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```



Modeling Hardware with 42: Case Study

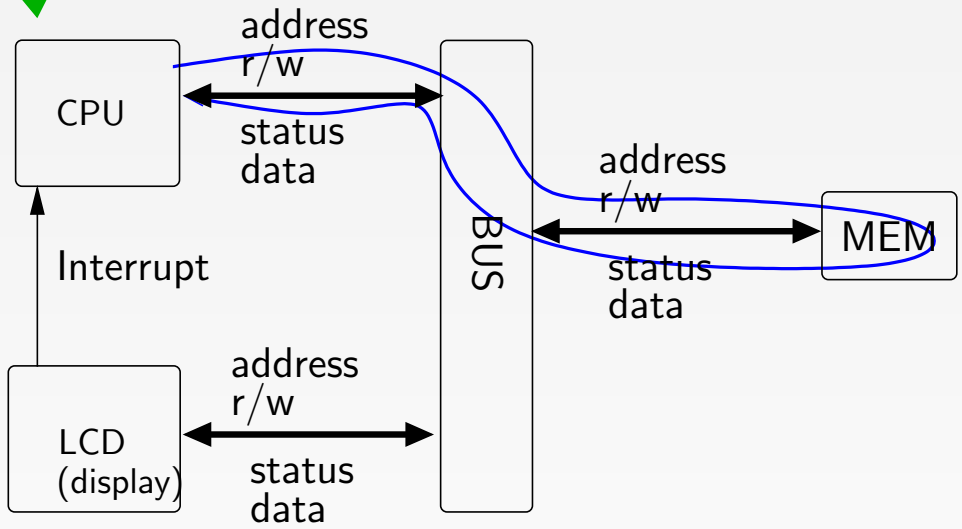
tel-00539648, version 1 - 24 Nov 2010

Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```



Modeling Hardware with 42: Case Study

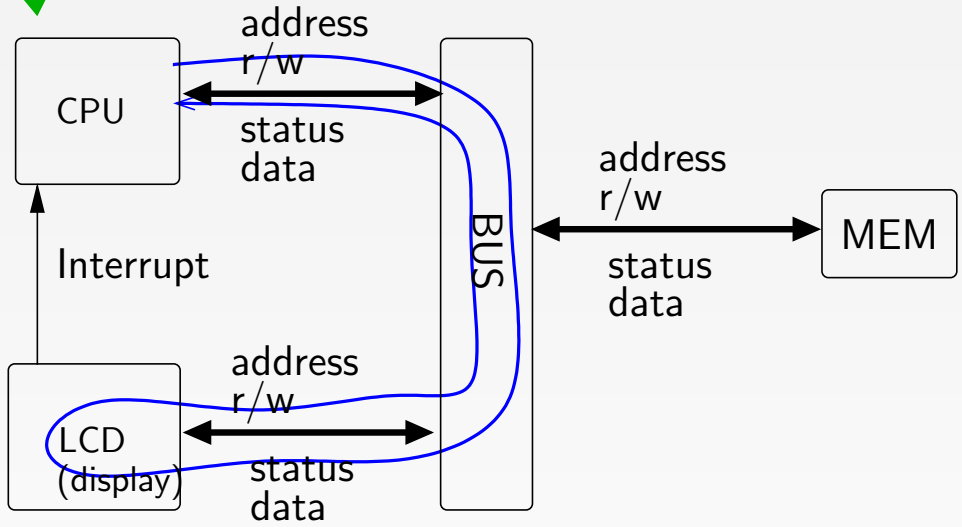
tel-00539648, version 1 - 24 Nov 2010

Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```



Modeling Hardware with 42: Case Study

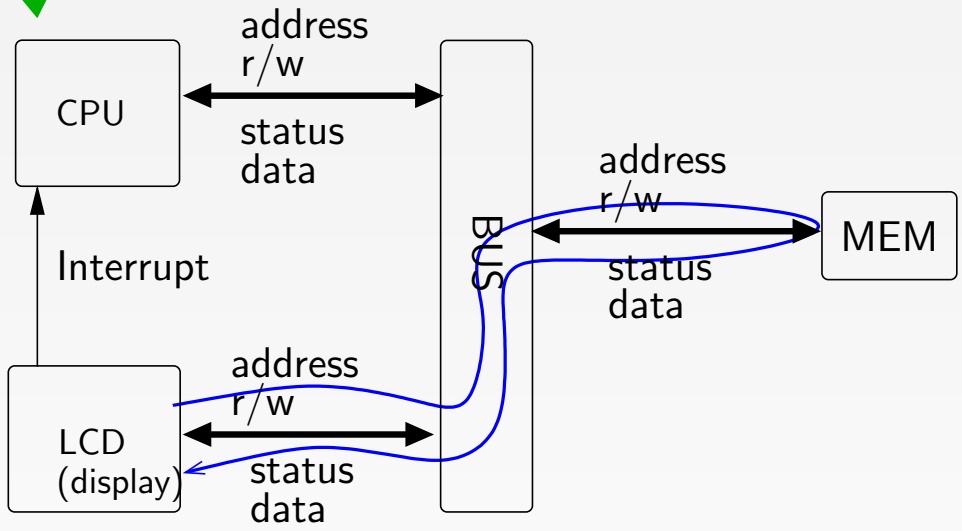
tel-00539648, version 1 - 24 Nov 2010

Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```



Modeling Hardware with 42: Case Study

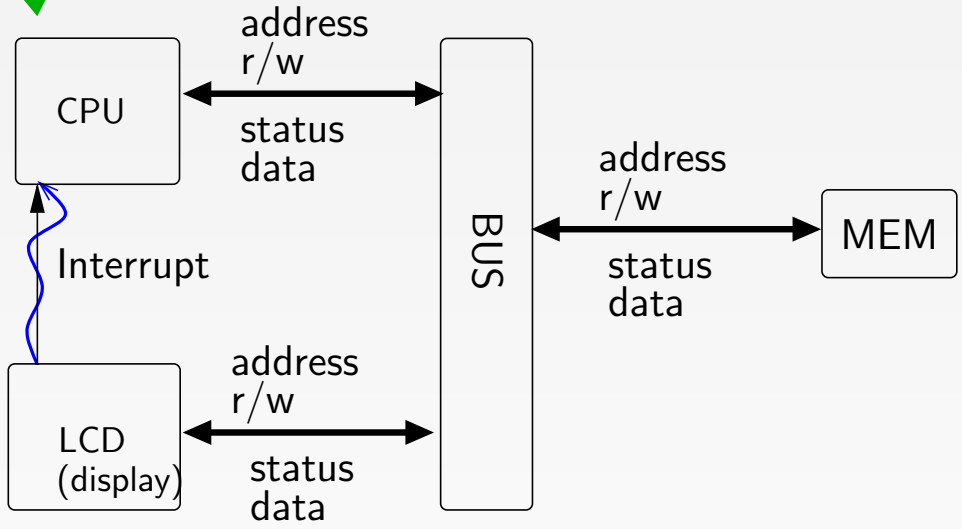
tel-00539648, version 1 - 24 Nov 2010

Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```



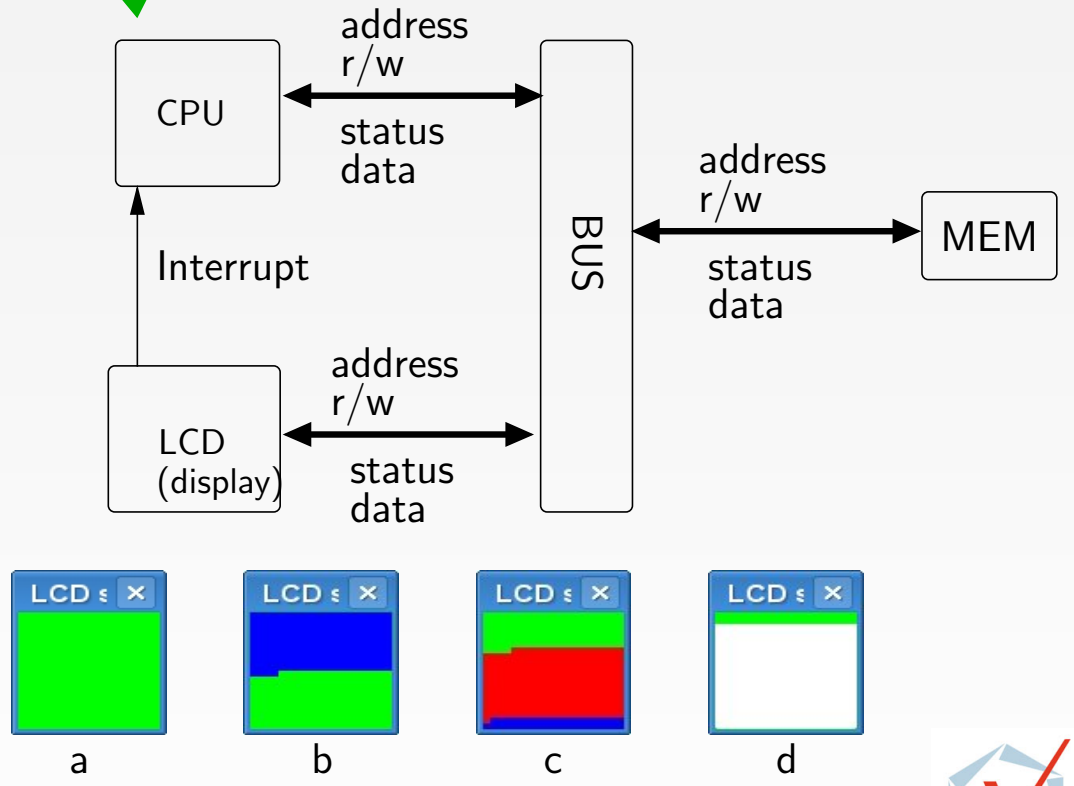
Modeling Hardware with 42: Case Study

Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```



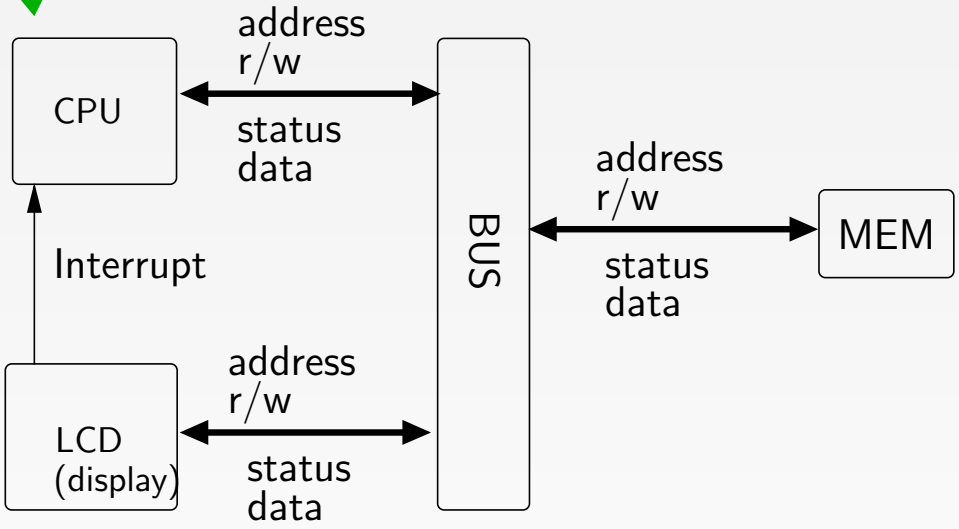
Modeling Hardware with 42: Case Study

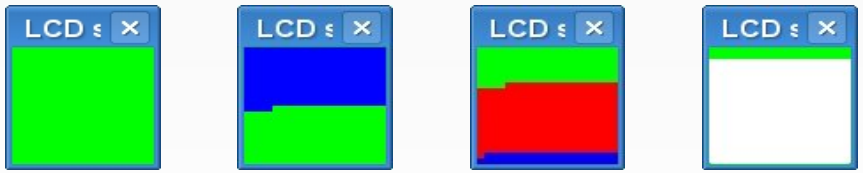
Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```





a b c d

ok

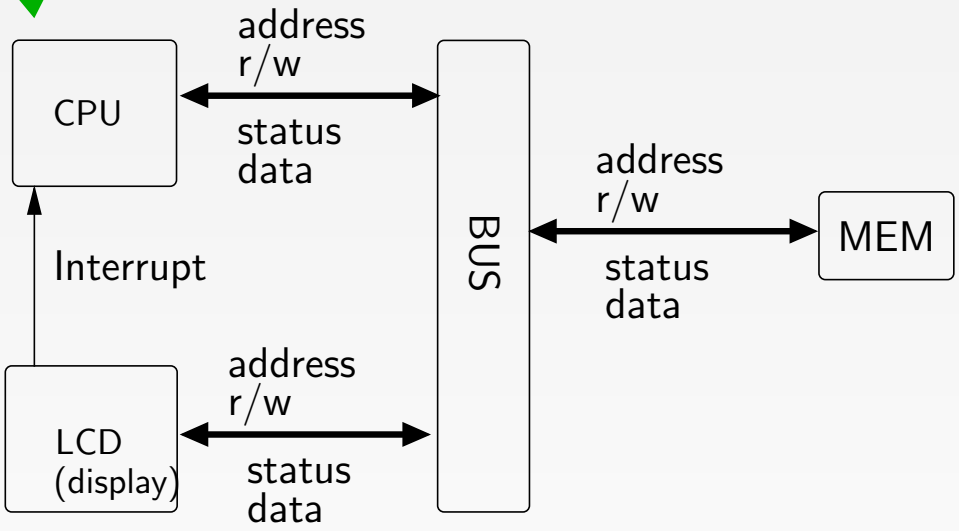
Modeling Hardware with 42: Case Study

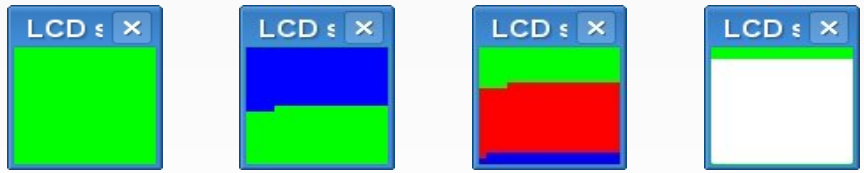
Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   wait_interrupt();
}
}
```





a b c d

ok ok

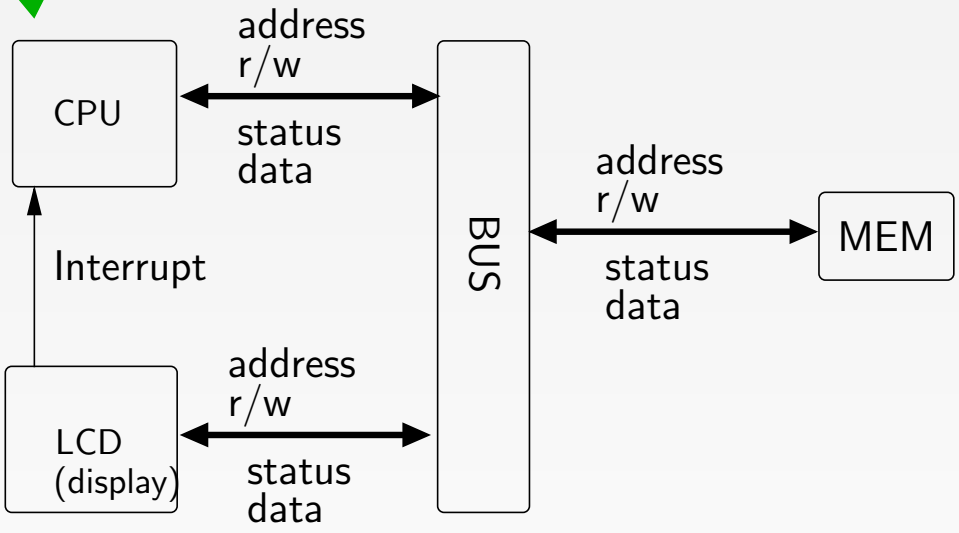
Modeling Hardware with 42: Case Study

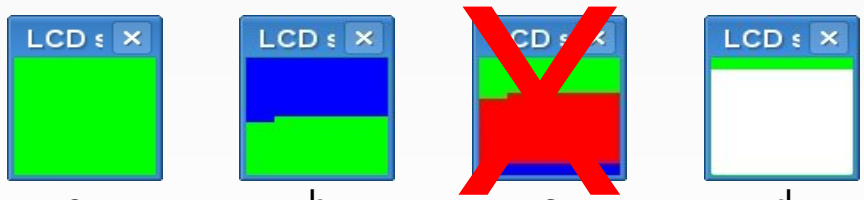
Embedded Software

```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   // wait_interrupt() ;
}
}
```





ok ok bug (Synchronization)

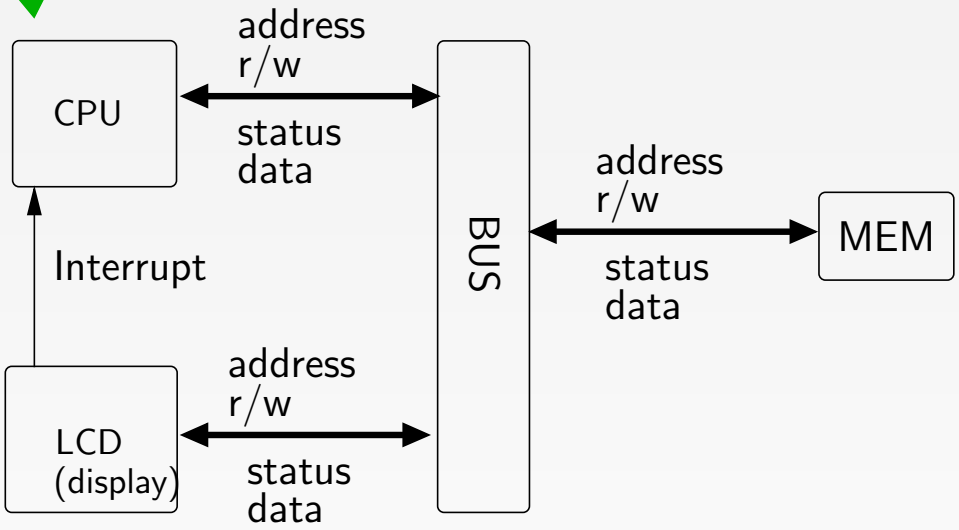
Modeling Hardware with 42: Case Study

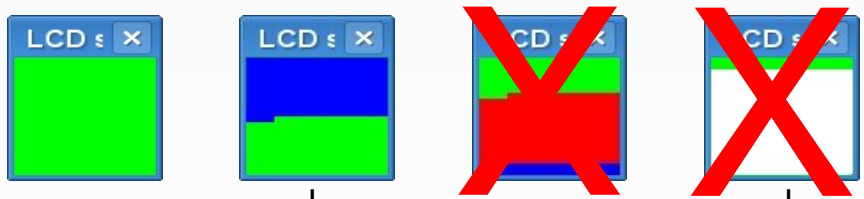
Embedded Software

```
#define width 240
#define height 24
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*writing the blue image*/
   for(int x=0; x<width * height)
       write_mem(blue);
   write_lcd(0x01,0x1);
   wait_interrupt();

//*writing the red image*/
   for(int x=0; x<width * height)
       write_mem(red);
   write_lcd(0x01,0x1);
   // wait_interrupt() ;
}
}
```





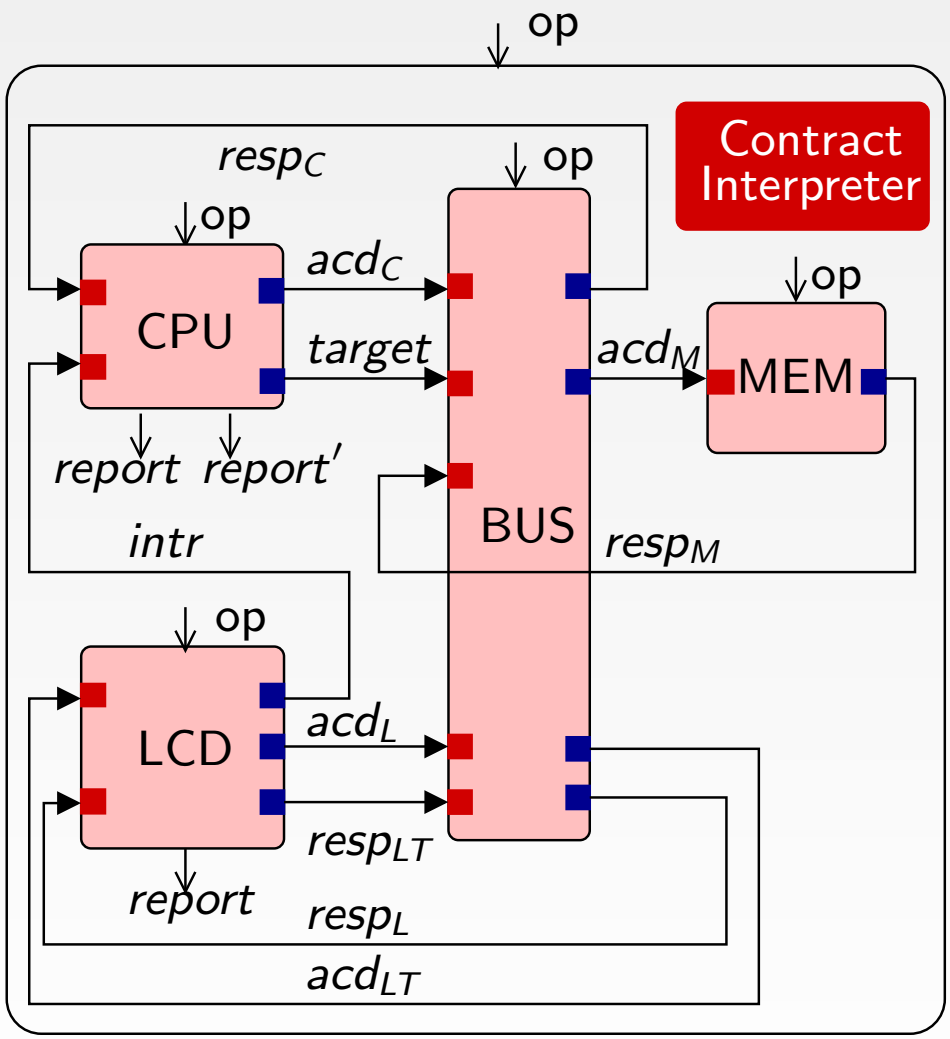
a ok

b ok

c bug (Synchronization)

d bug (Data)

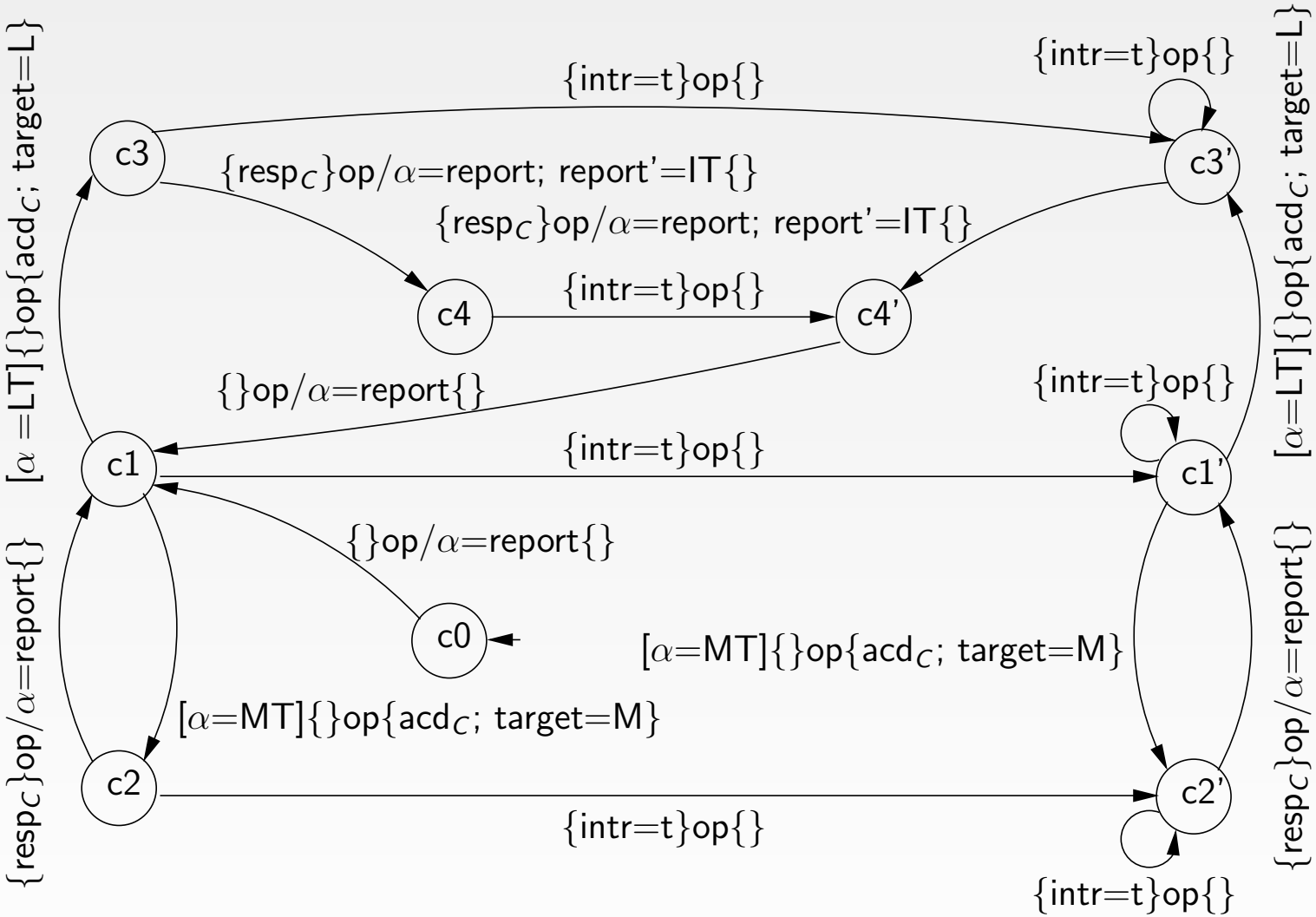
Modeling Hardware with 42: The 42 Model



tel-00539648, version 1 - 24 Nov 2010

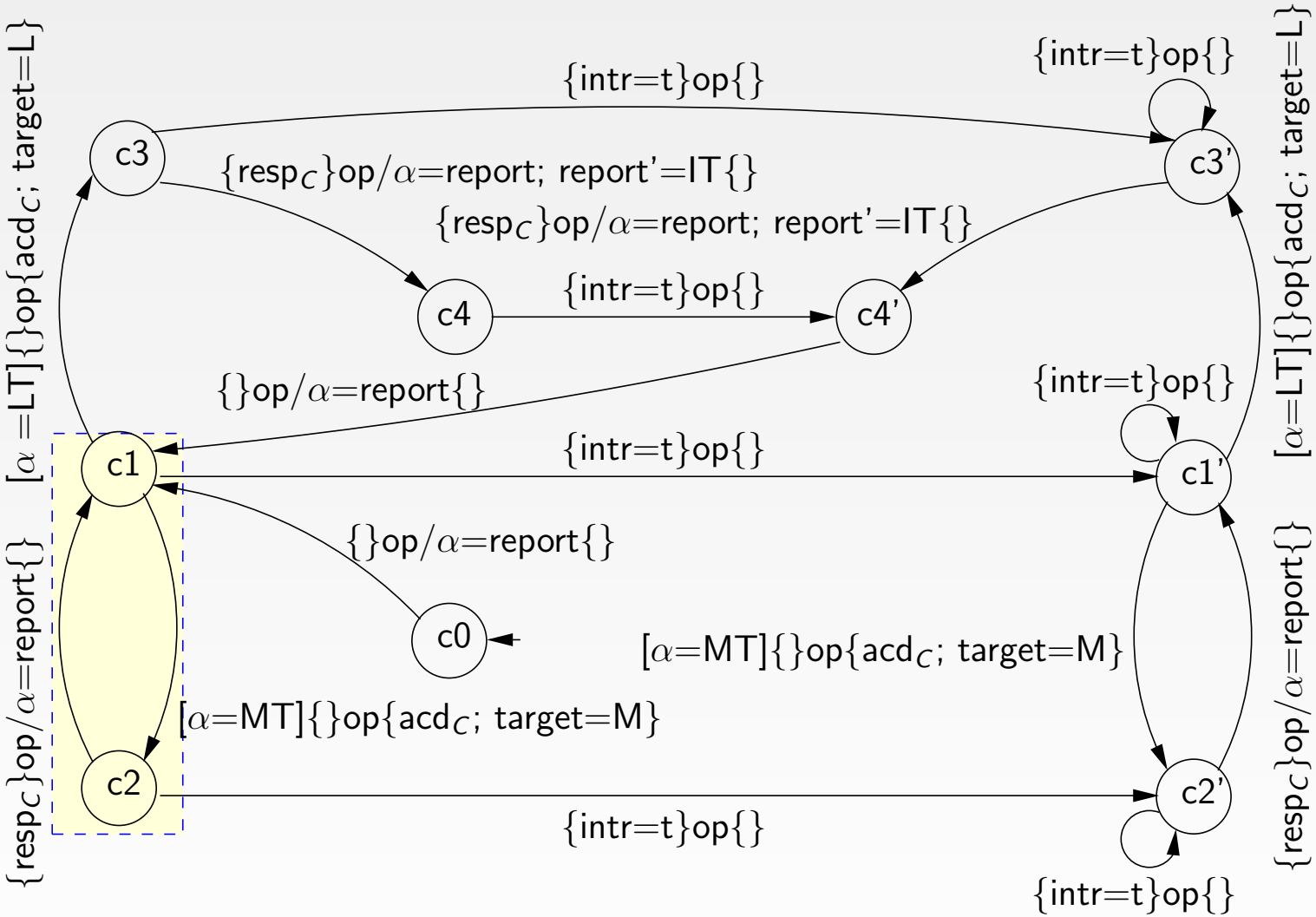
Modeling Hardware with 42: The contracts (CPU)

tel-00539648, version 1 - 24 Nov 2010



The contract of the CPU is in fact the contract of (CPU + Software)

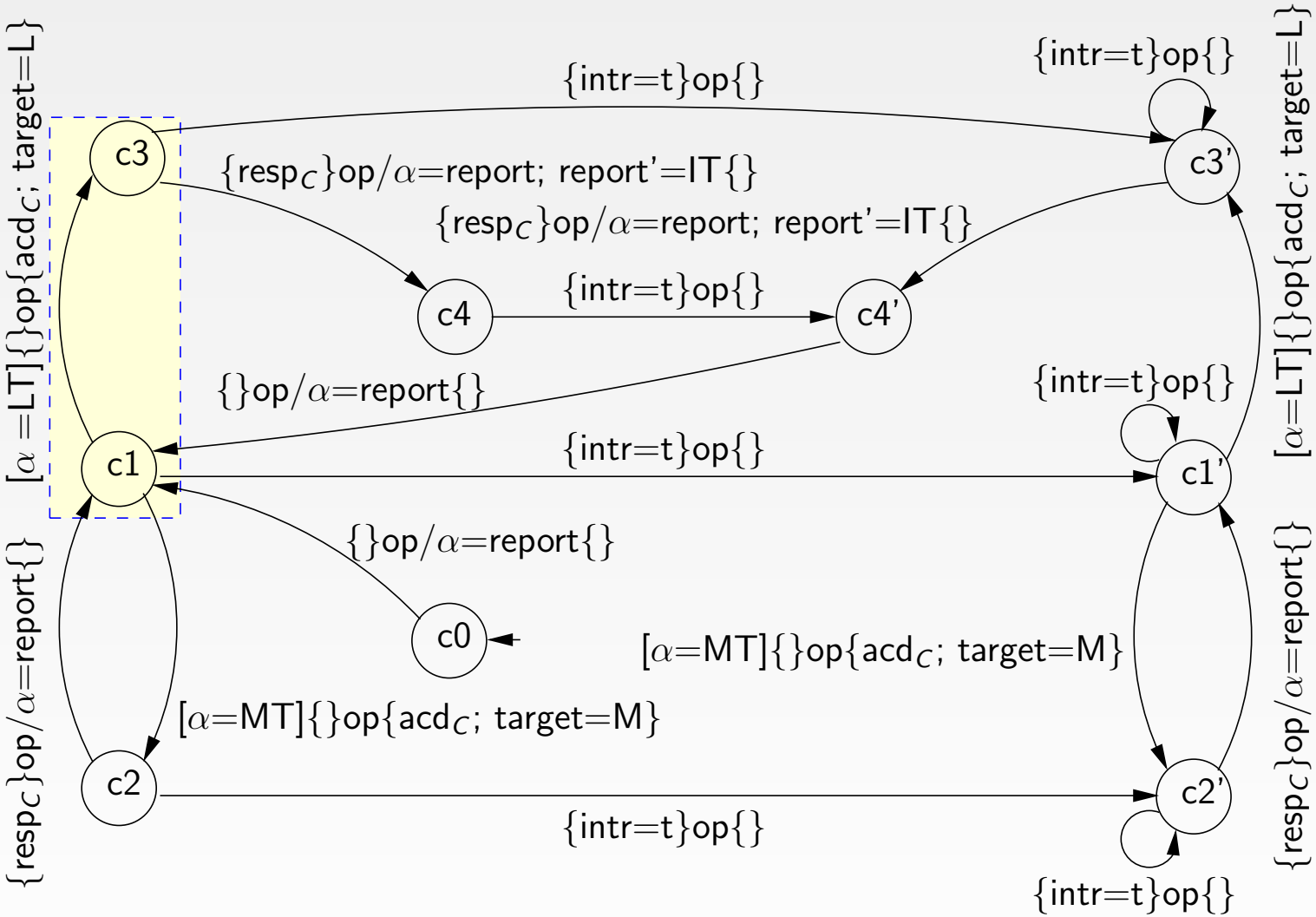
Modeling Hardware with 42: The contracts (CPU)



The contract of the CPU is in fact the contract of (CPU + Software)

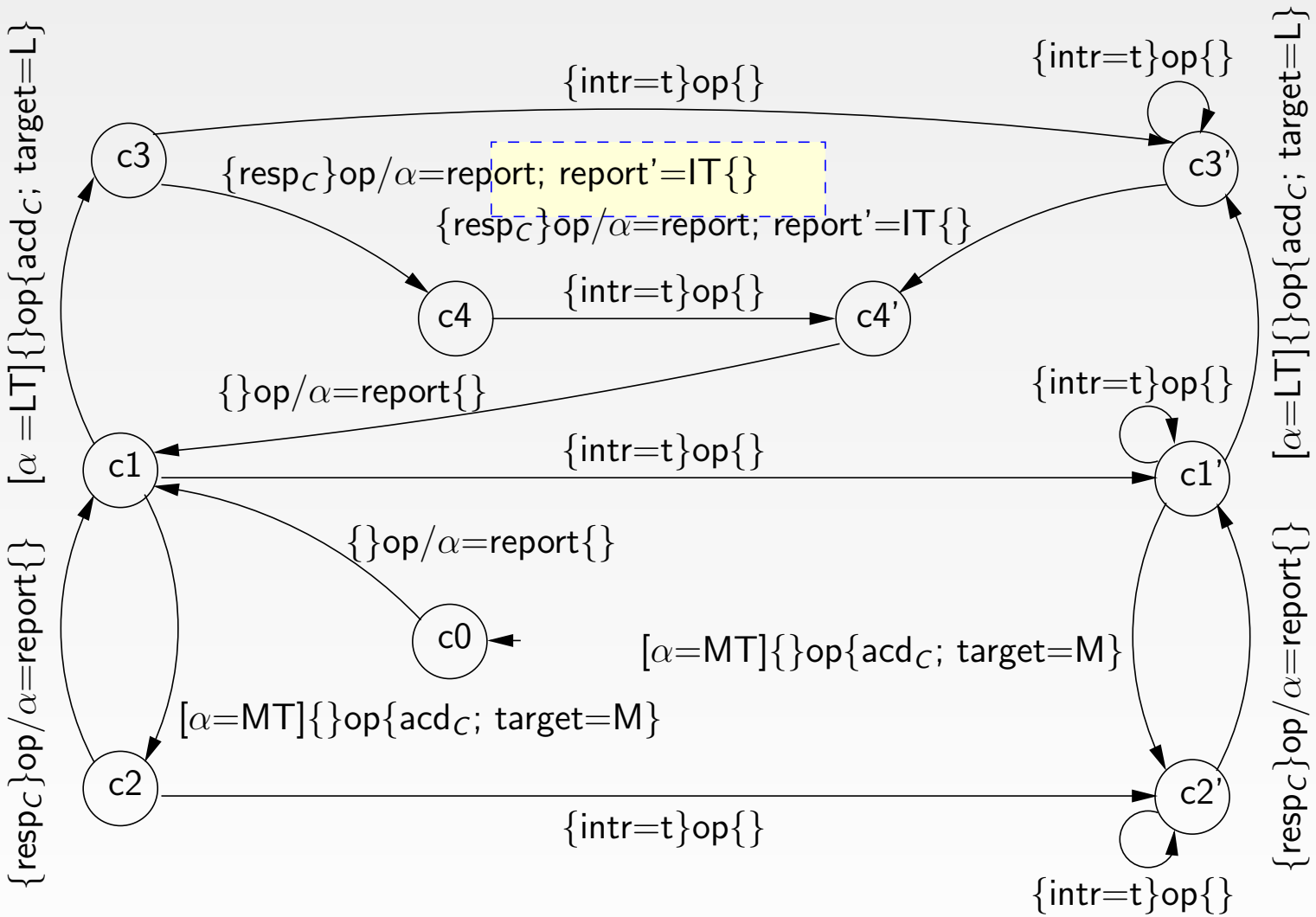
Modeling Hardware with 42: The contracts (CPU)

tel-00539648, version 1 - 24 Nov 2010



The contract of the CPU is in fact the contract of (CPU + Software)

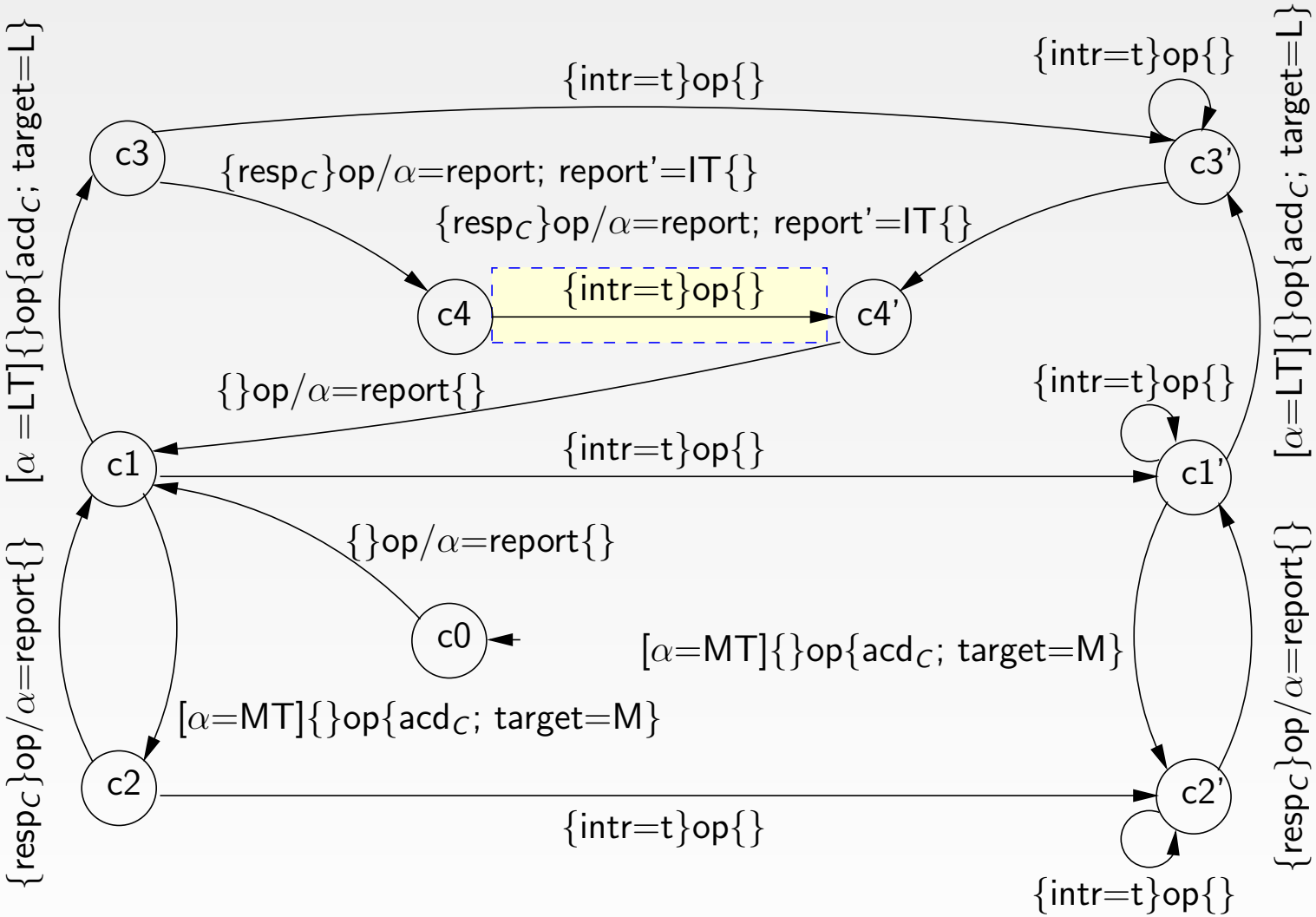
Modeling Hardware with 42: The contracts (CPU)



The contract of the CPU is in fact the contract of (CPU + Software)

Modeling Hardware with 42: The contracts (CPU)

tel-00539648, version 1 - 24 Nov 2010



The contract of the CPU is in fact the contract of (CPU + Software)

Detecting Synchronization Bugs by Executing Contracts

tel-00539648, version 1 - 24 Nov 2010

If the software doesn't wait for the interrupt $\Longrightarrow$

We don't have the software yet. We execute only the contracts.

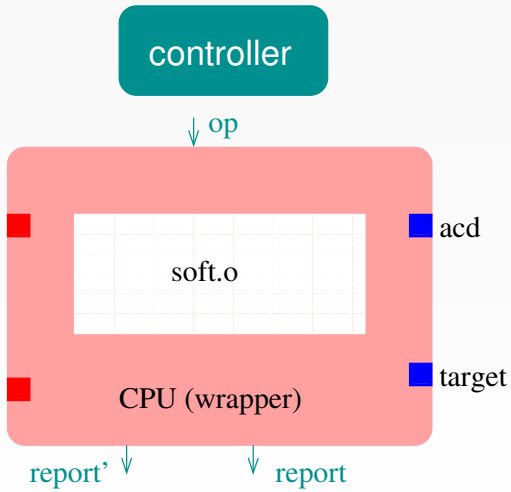
- Synchronization Bug: The contract doesn't wait for interrupts
- Detection: Deadlock during contract interpretation

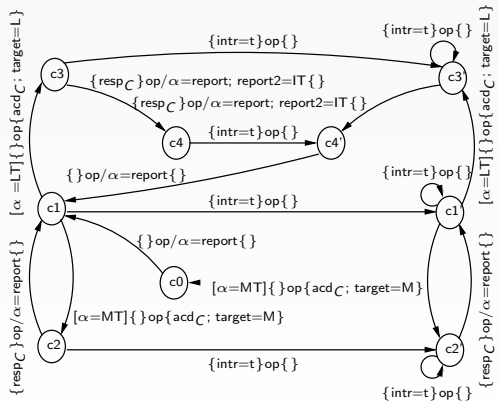
tel-00539648, version 1 - 24 Nov 2010

Detecting SW Bugs by Executing the SW Against the Contract

software wrapper controller

```
ⓐ int main(){
      while (true){
          int x = ....
          while(x>0){
              x - -;
              ...
          ⓑ write_mem(adr, data);
          ...
      ⓓ write_lcd(adr2, data2);
          ...
      ⓔ wait_interrupt();
          if(y!=0)
      ⓕ write_mem(adr, data)
          ...
      }
}
```





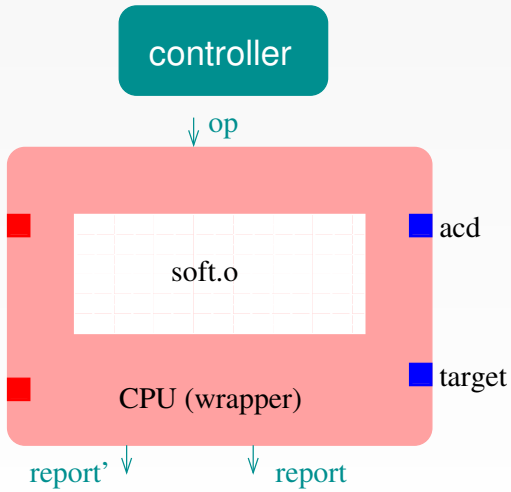
tel-00539648, version 1 - 24 Nov 2010

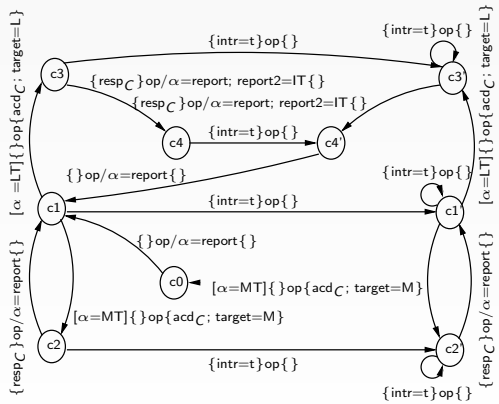
Detecting SW Bugs by Executing the SW Against the Contract

software wrapper controller

cpu.op

```
ⓐ int main(){
      while (true){
          int x = ....
          while(x>0){
              x - -;
              ...
          ⓑ write_mem(adr, data);
          ...
      ➤ ⓓ write_lcd(adr2, data2);
          ...
          ⓔ wait_interrupt();
          if(y!=0)
          ⓕ write_mem(adr, data)
          ...
      }   }
}
```

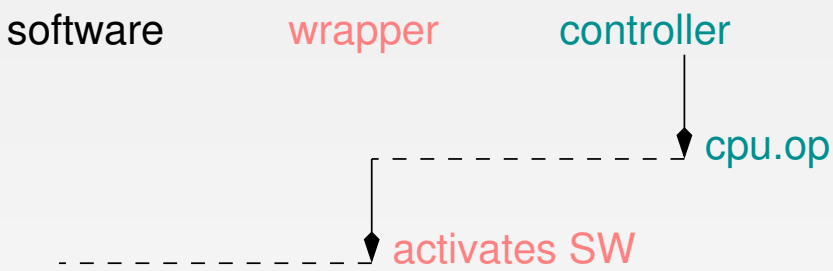


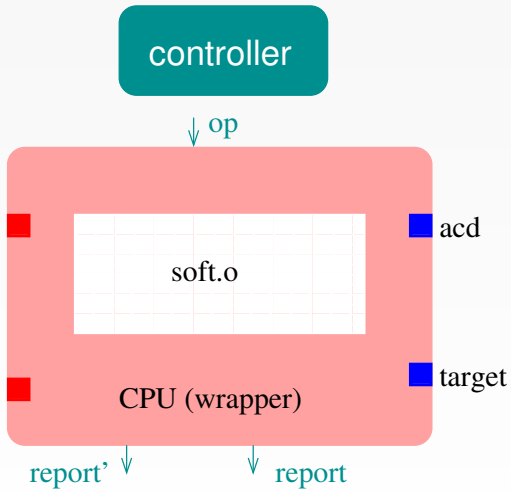


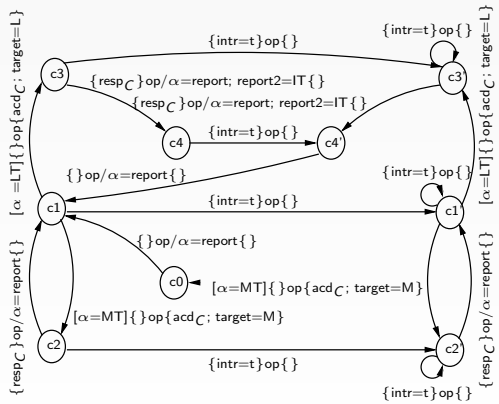
tel-00539648, version 1 - 24 Nov 2010

Detecting SW Bugs by Executing the SW Against the Contract

```
ⓐ int main(){
      while (true){
          int x = ....
          while(x>0){
              x - -;
              ...
          ⓑ write_mem(adr, data);
          ...
      ⓓ write_lcd(adr2, data2);
          ...
          ⓔ wait_interrupt();
          if(y!=0)
          ⓕ write_mem(adr, data)
          ...
      }
}
```



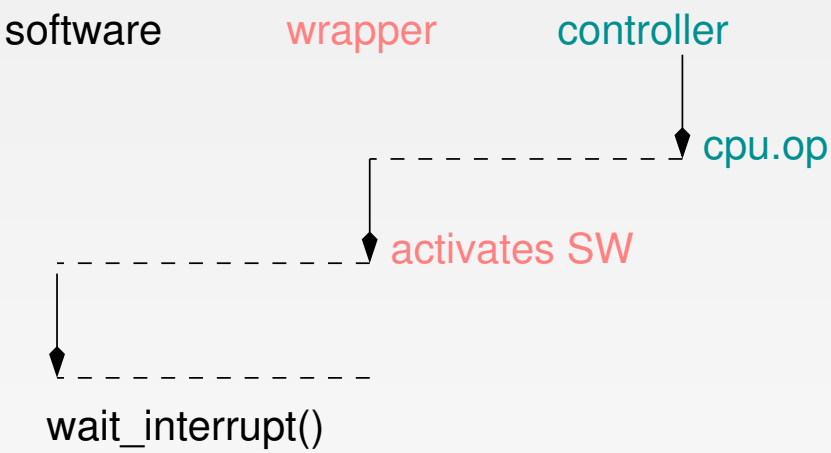


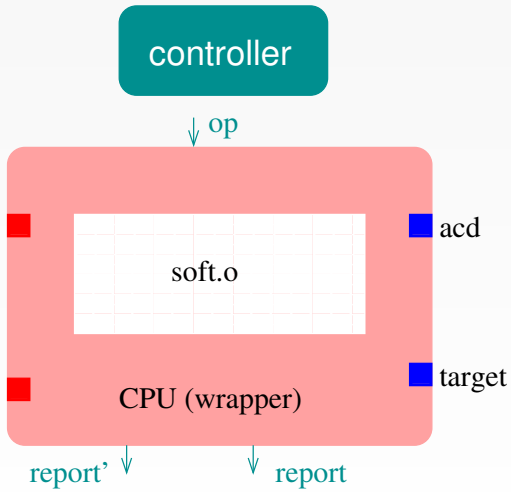


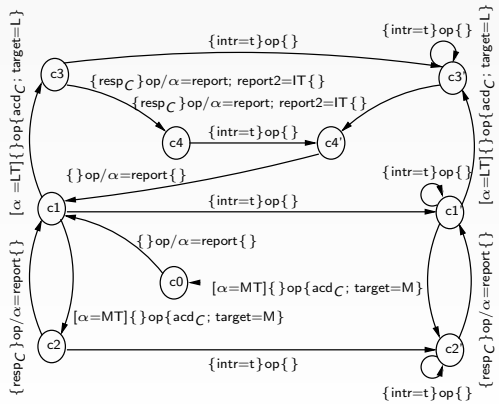
Detecting SW Bugs by Executing the SW Against the Contract

tel-00539648, version 1 - 24 Nov 2010

```
ⓐ int main(){
      while (true){
          int x = ....
          while(x>0){
              x - -;
              ...
          ⓑ write_mem(adr, data);
          ...
atomic ⓓ write_lcd(adr2, data2);
          ...
       ⓔ wait_interrupt();
          if(y!=0)
       ⓕ write_mem(adr, data)
          ...
      }  }
}
```



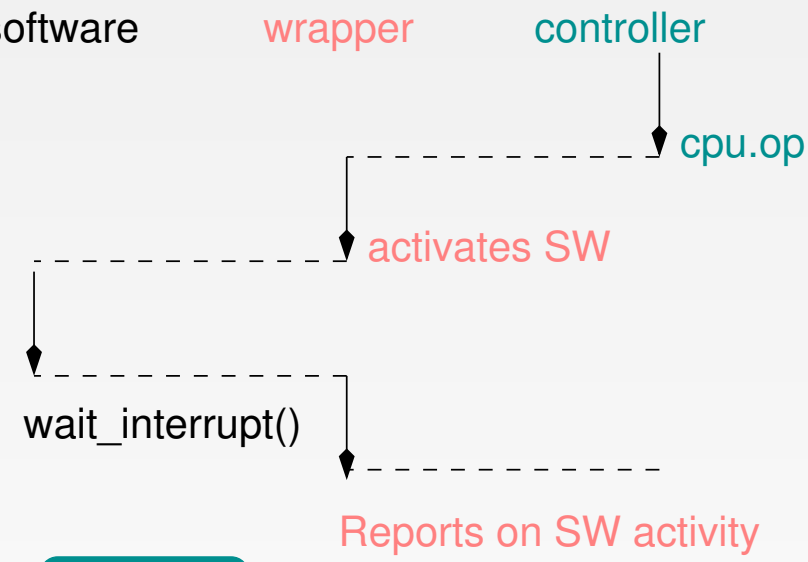


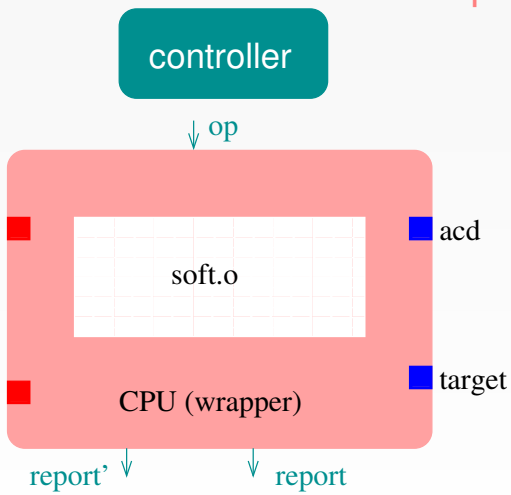


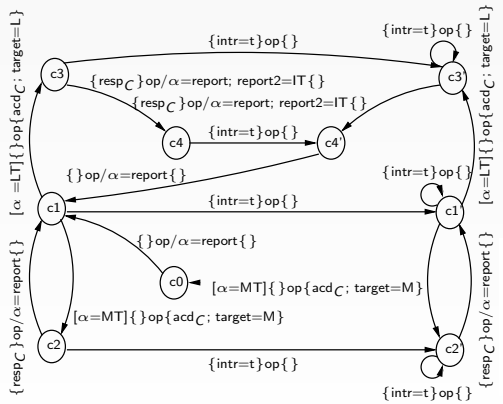
tel-00539648, version 1 - 24 Nov 2010

Detecting SW Bugs by Executing the SW Against the Contract

```
ⓐ int main(){
      while (true){
          int x = ....
          while(x>0){
              x - -;
              ...
          ⓑ write_mem(adr, data);
          ...
atomic ⓓ write_lcd(adr2, data2);
          ...
       ⓔ wait_interrupt();
          if(y!=0)
       ⓕ write_mem(adr, data)
          ...
      }  }
}
```



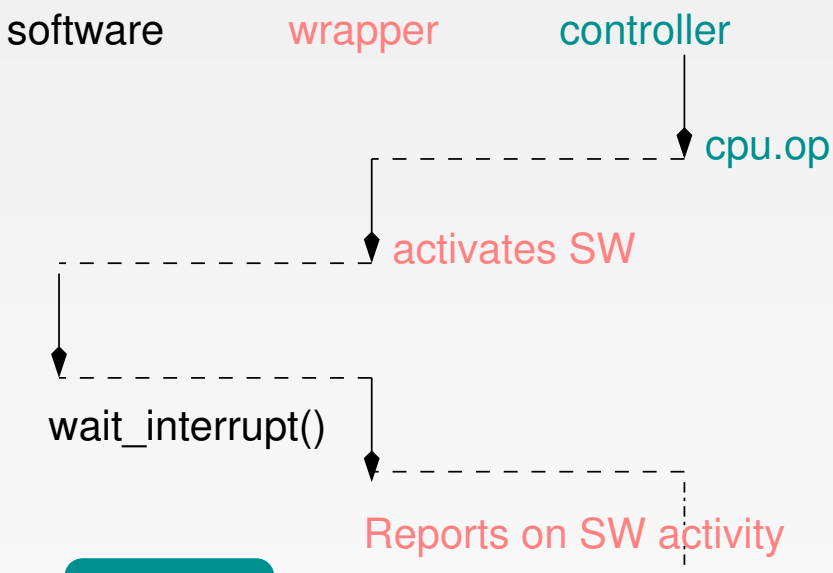


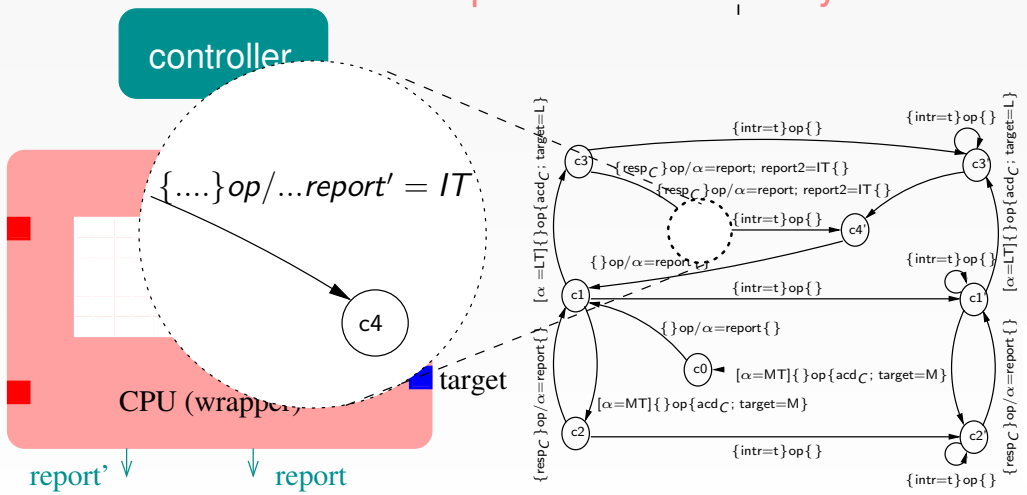


tel-00539648, version 1 - 24 Nov 2010

Detecting SW Bugs by Executing the SW Against the Contract

```
ⓐ int main(){
      while (true){
          int x = ....
          while(x>0){
              x - -;
              ...
          ⓑ write_mem(adr, data);
          ...
atomic ⓓ write_lcd(adr2, data2);
          ...
       ⓔ wait_interrupt();
          if(y!=0)
       ⓕ write_mem(adr, data)
          ...
      }  }
```





tel-00539648, version 1 - 24 Nov 2010

Summary

- Contracts are non-deterministic abstractions of the behavior of components and allow for early execution
- Contract execution detects synchronization bugs
- Executing contract together with implementation allows to check their consistency

tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration

2 Overview of 42 & Examples [GPCE07]

3 Hardware Simulation by Interpreting Contracts [COORD09]

4 Using 42 Together with Existing Approaches [EMSOFT09]
- TLM with SystemC
- 42-ization of SystemC/TLM
- Typical Uses of 42 Contracts with SystemC-TLM

5 Some Related Work

6 Summary

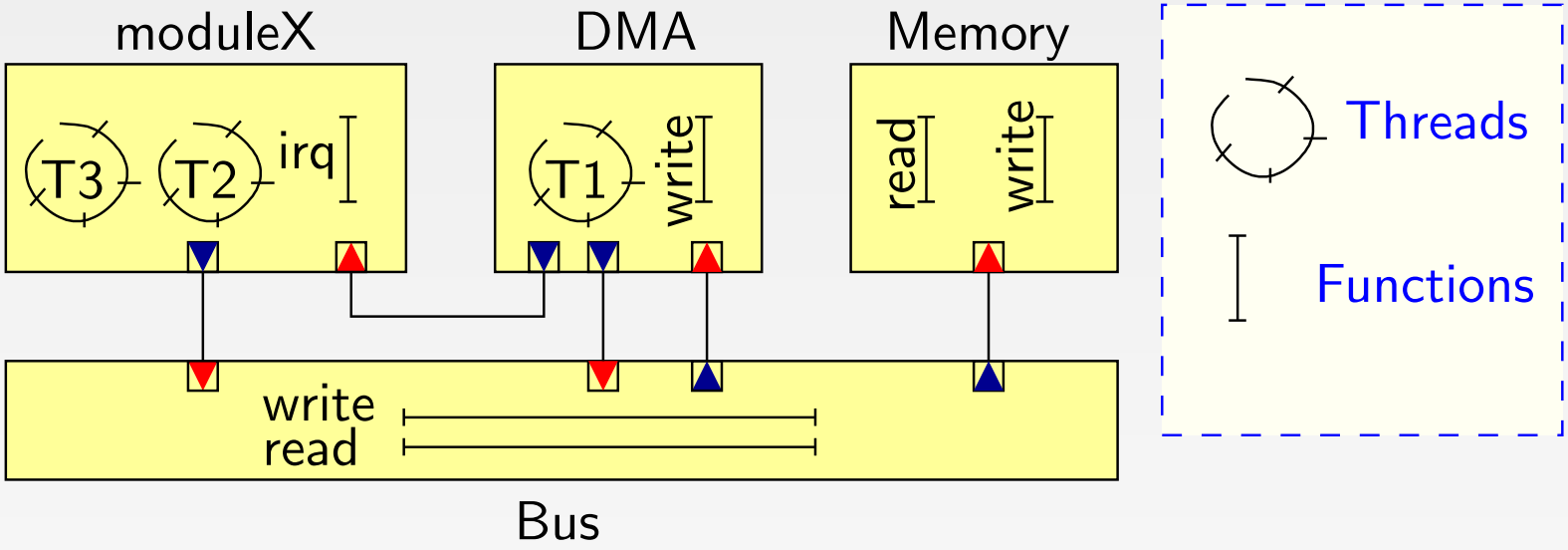
tel-00539648, version 1 - 24 Nov 2010

Motivation

- TLM = Transaction-Level Modeling
  - + Widely used
  - + Powerful (large systems, heterogeneous models, ...)
  - + Intrinsically component-based
  - − Very informal, models are mixed with the simulation mechanics

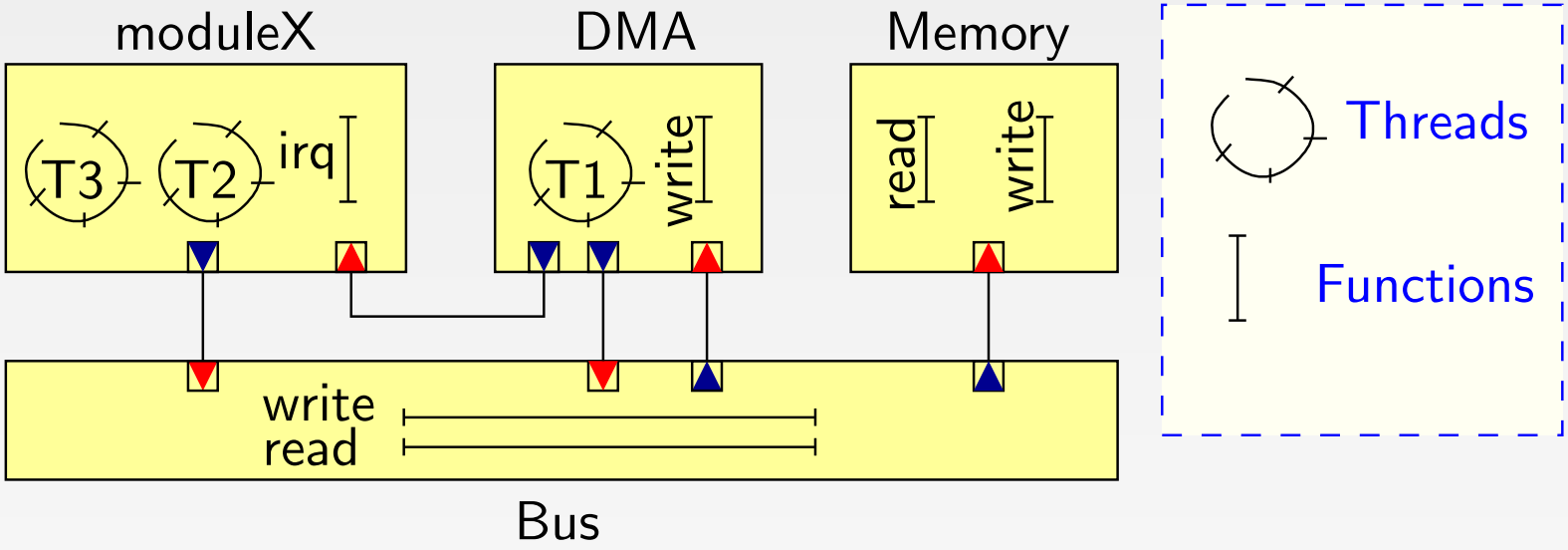
- Motivation
  - Propose a clear definition of TLM components
  - Simplify the simulation
  - Focus on Control Flow and Synchronizations
  - Find bugs earlier

TLM in practice (SystemC)



tel-00539648, version 1 - 24 Nov 2010

TLM in practice (SystemC)

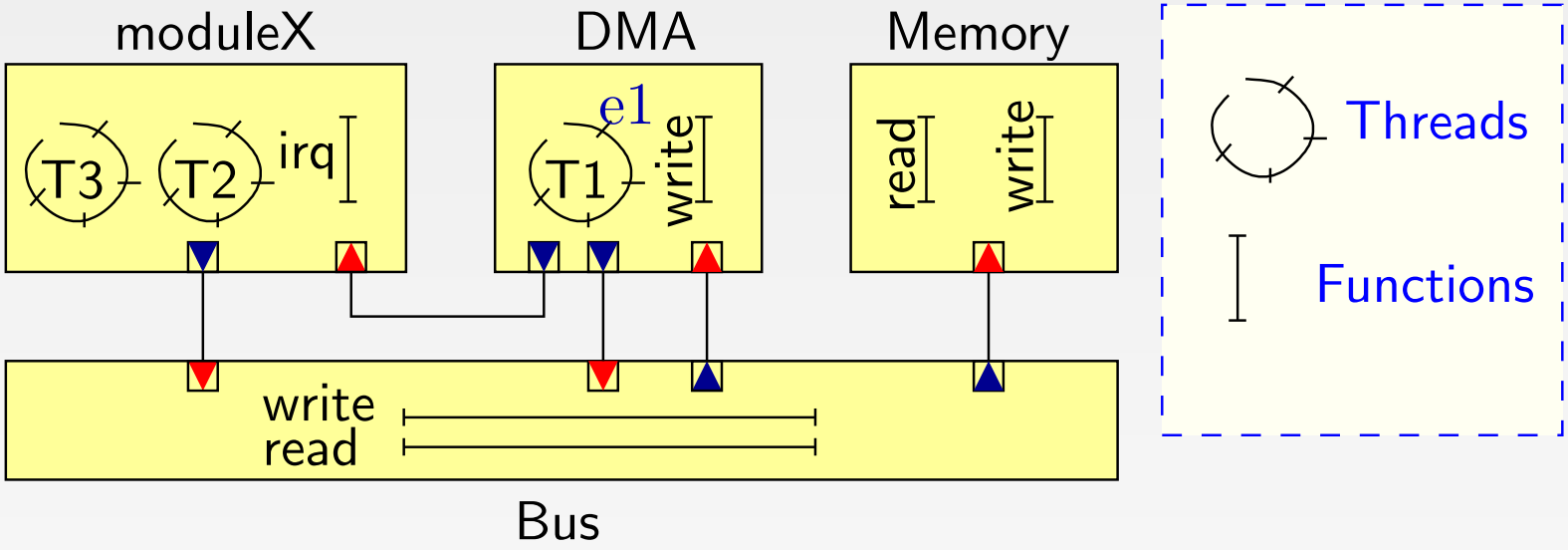


Some Guidelines:

tel-00539648, version 1 - 24 Nov 2010

tel-00539648, version 1 - 24 Nov 2010

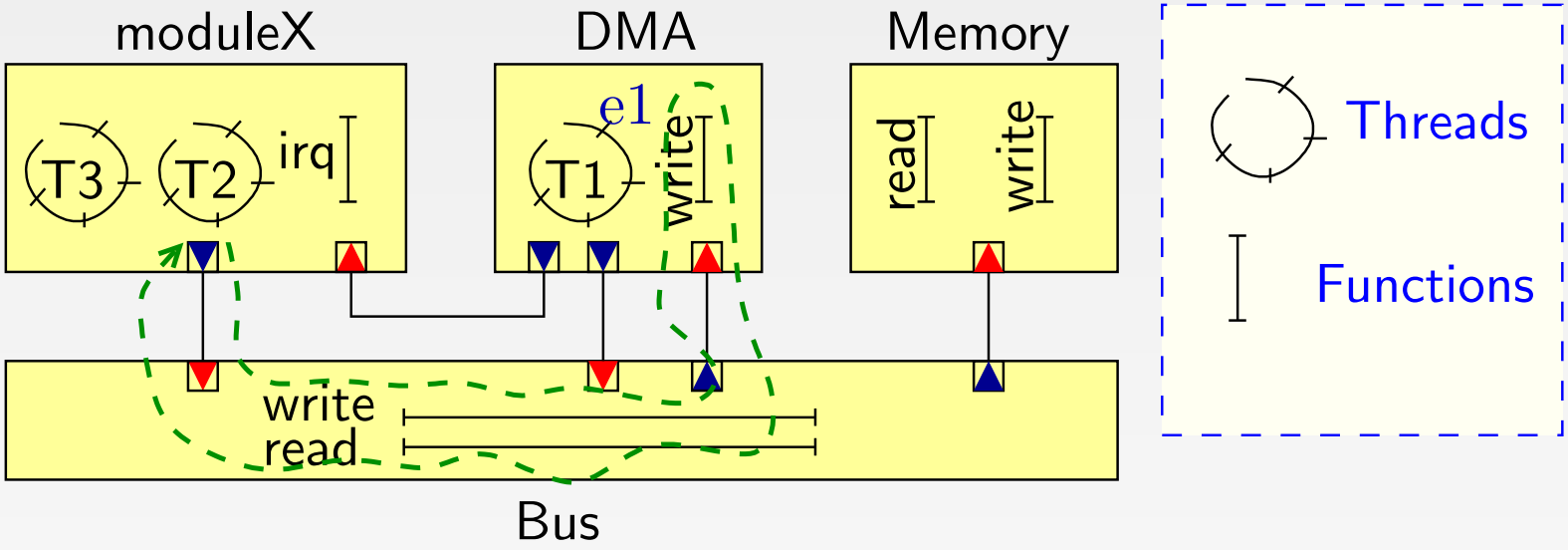
TLM in practice (SystemC)



Some Guidelines:

- Inside a module:
  - Shared variables
  - Events (wait, notify)

TLM in practice (SystemC)

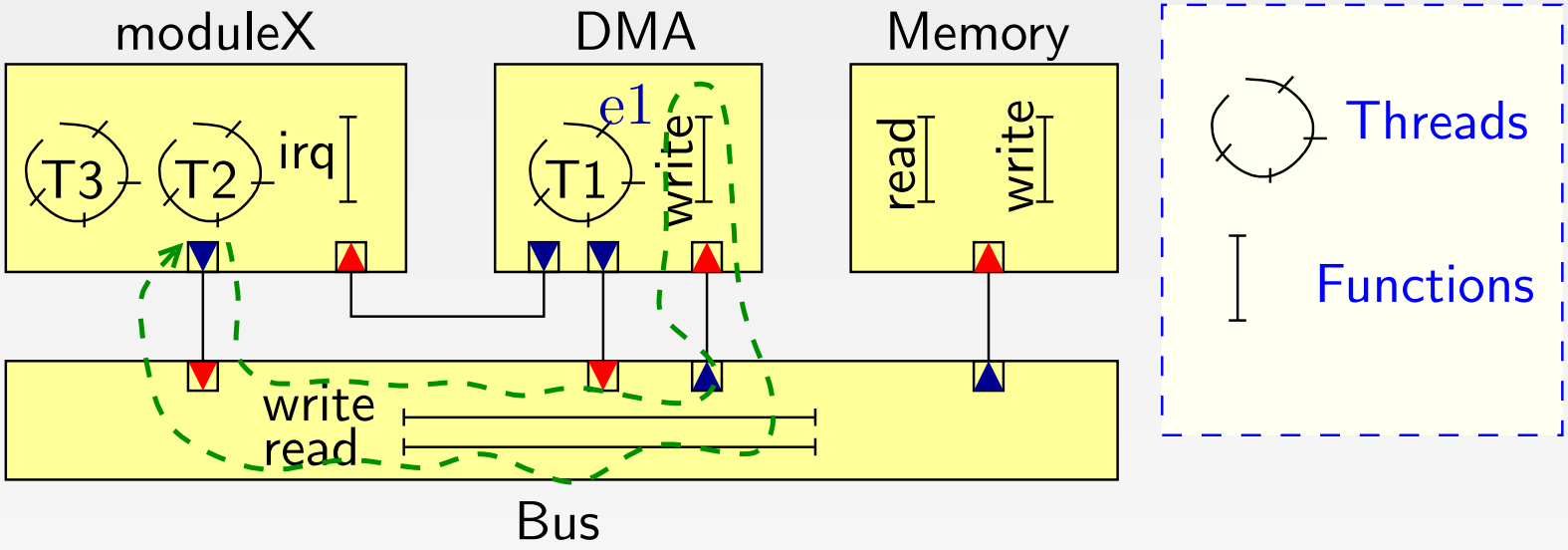


Some Guidelines:

- Inside a module:
  - Shared variables
  - Events (wait, notify)
- Between modules:
  - Function calls through ports
  - ...

tel-00539648, version 1 - 24 Nov 2010

TLM in practice (SystemC)



Some Guidelines:

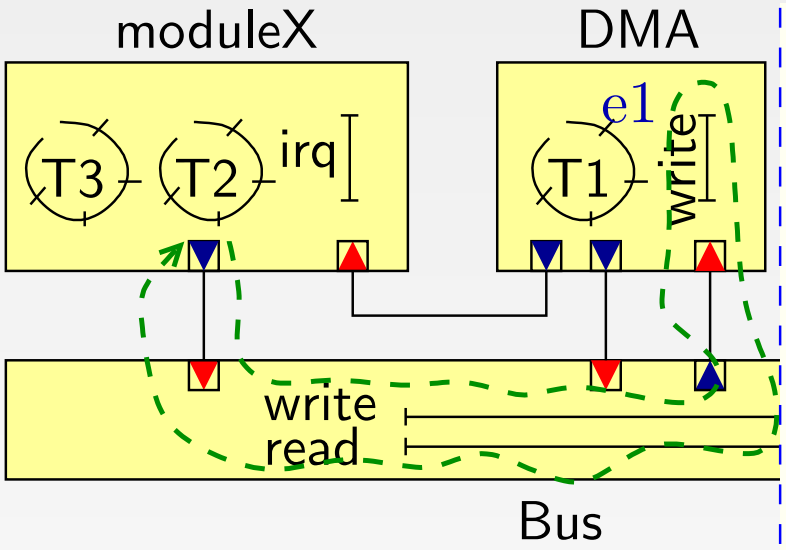
- Inside a module:
  - Shared variables
  - Events (wait, notify)
- Between modules:
  - Function calls through ports
  - ...

A non-preemptive scheduler manages the execution of threads

tel-00539648, version 1 - 24 Nov 2010

tel-00539648, version 1 - 24 Nov 2010

TLM in practice (SystemC)




```
moduleX::T2(){
  while(true){
    x = 42;
    while (x > 0){
      y = p.read(addrM + x)^0xFF;
      p.write(addrM , x);
    }
    p.write(addrD+0x0, addrM);
    p.write(addrD+0x1, 42);
    p.write(addrD+0x4, 0x01);
    wait(e0);
    ...
    e2.notify();
  }
  ...
}

moduleX::irq(int n){
  ...
  e0.notify();
  ...
  wait(e2);
}
```

Example Implementation of *moduleX*

Some Guidelines:

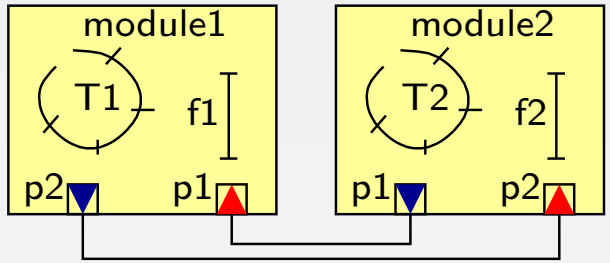
- Inside a module:
  - Shared variables
  - Events (wait, notify)
- Between modules:
  - Function calls through ports
  - ...

A non
ages th

Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
    }
}
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```



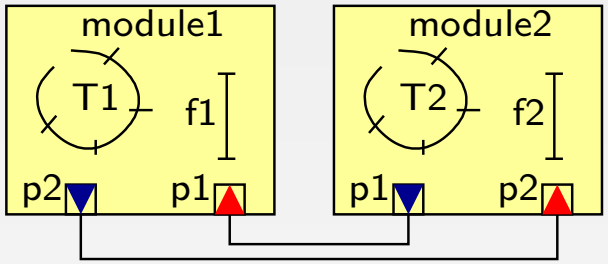
tel-00539648, version 1 - 24 Nov 2010

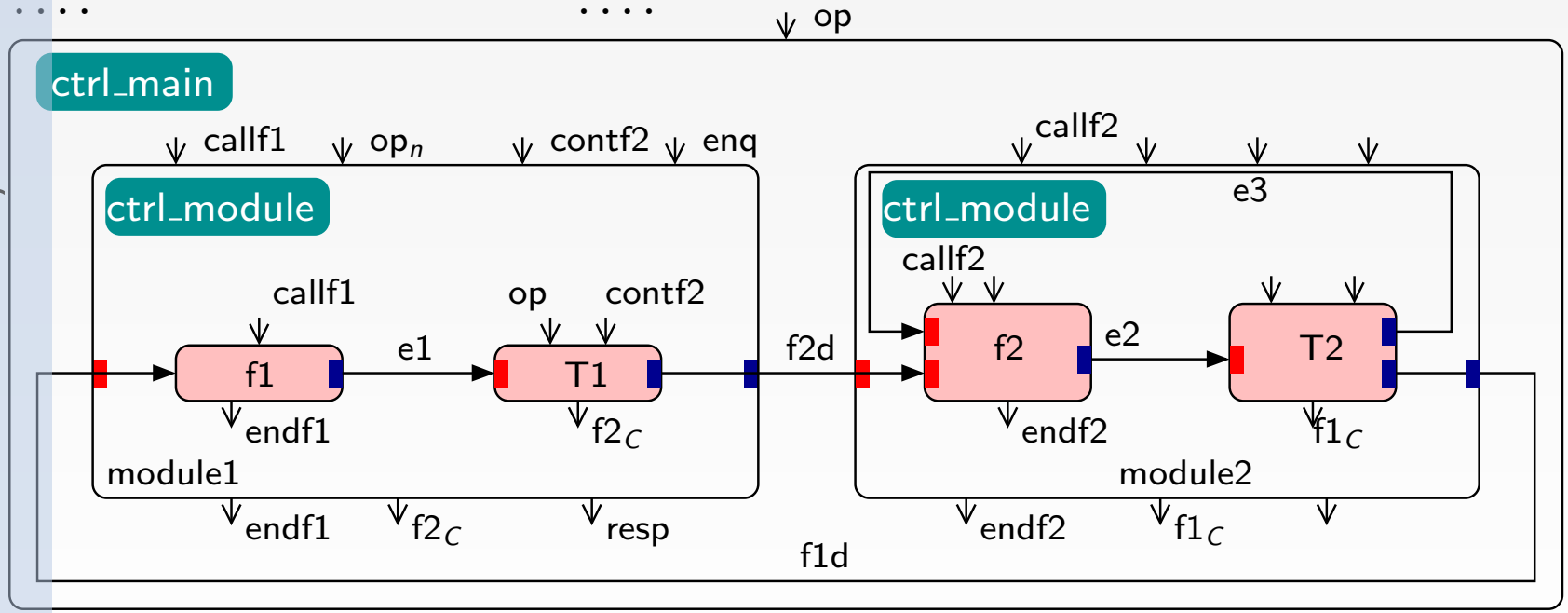
Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
    }
}
....
```

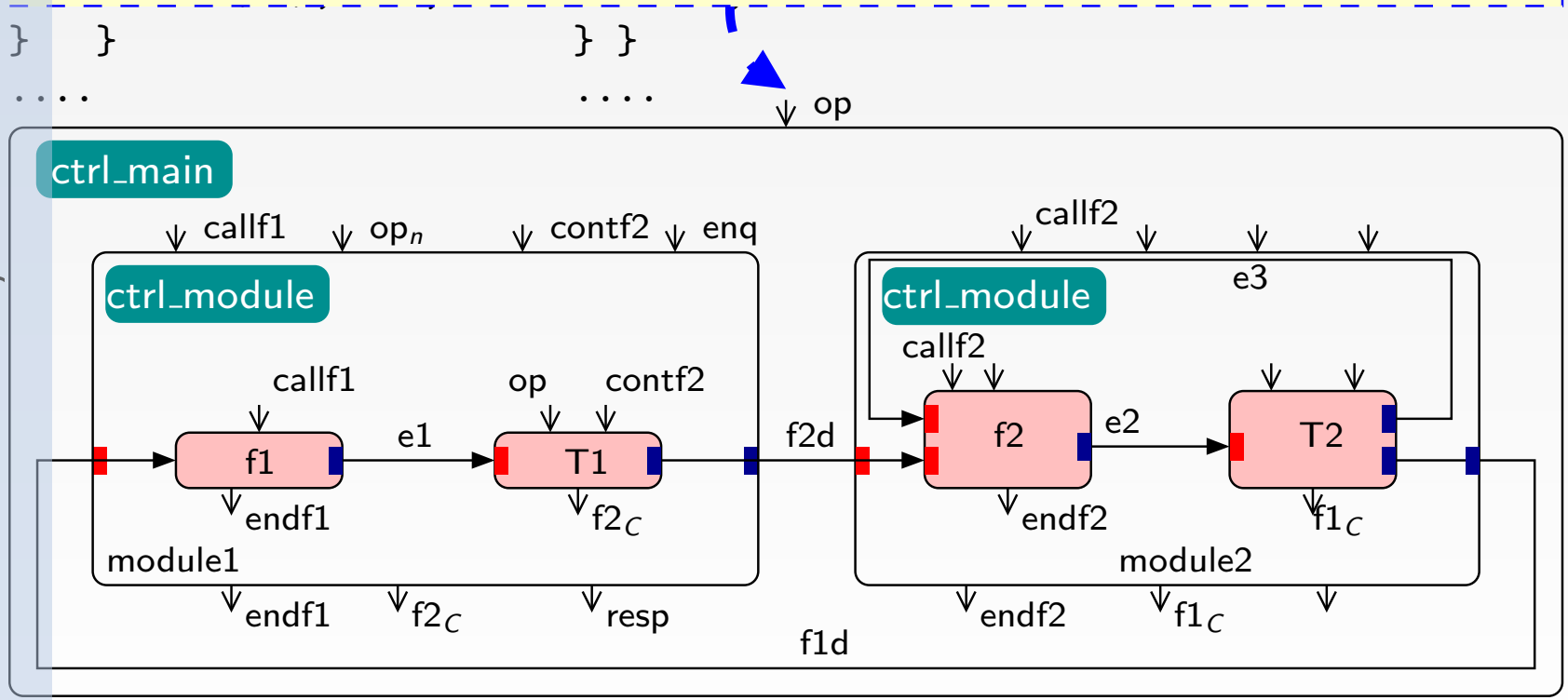
```
module2::f2(int x){
      ...;
      y = x * 42;
      ...;
      notify(e2);
      ...;
      ...;
} }
....
```





Structural Correspondence

An activation of the main component with OP corresponds to what happens in the SystemC-TLM model when the scheduler elects a new thread to execute.

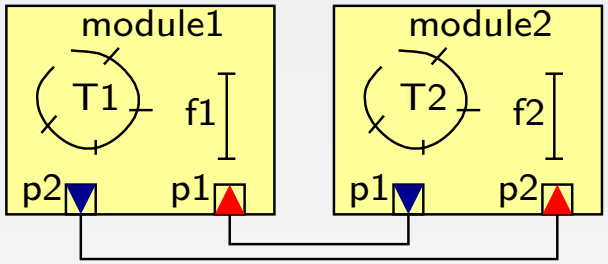


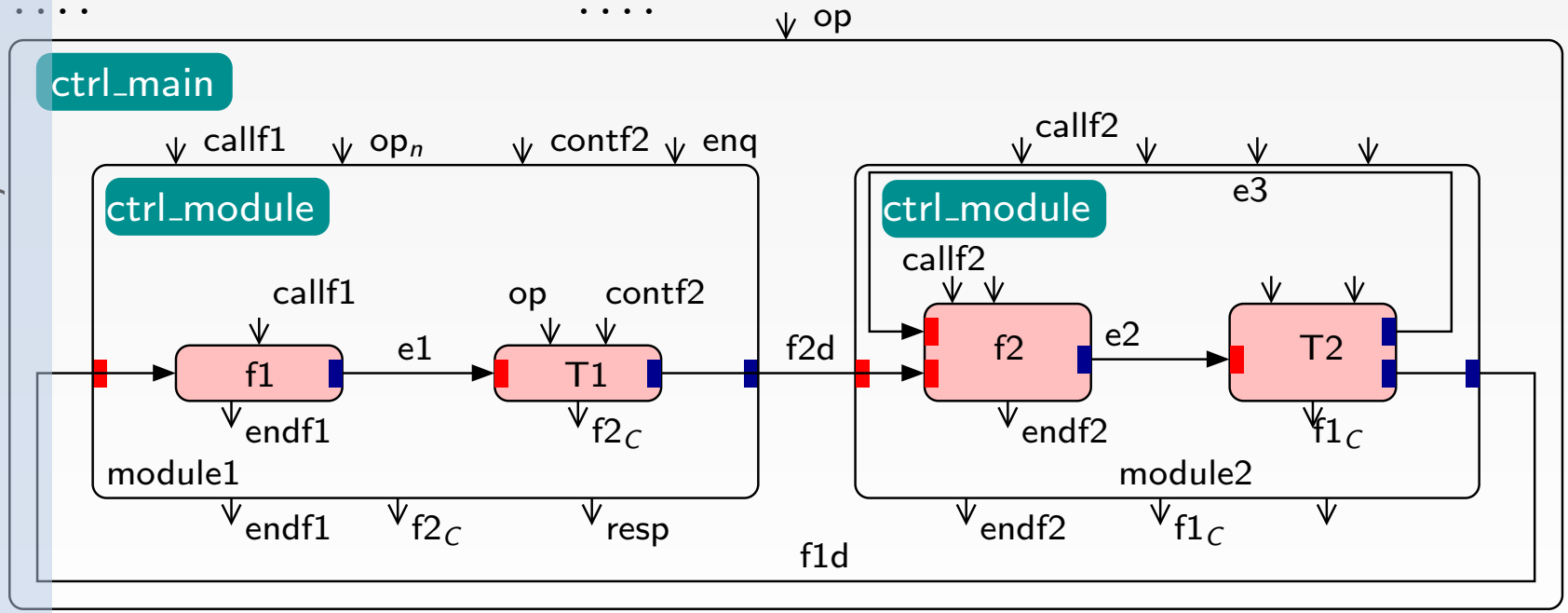
Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
    }
}
....
```

```
module2::f2(int x){
      ...;
      y = x * 42;
      ...;
      notify(e2);
      ...;
      ...;
} }
....
```



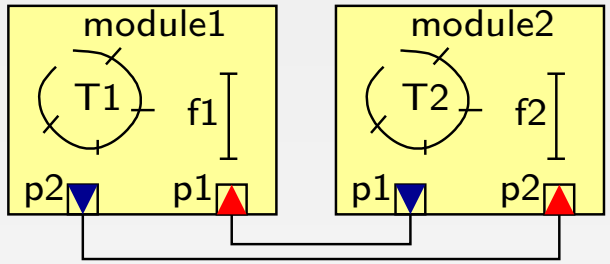


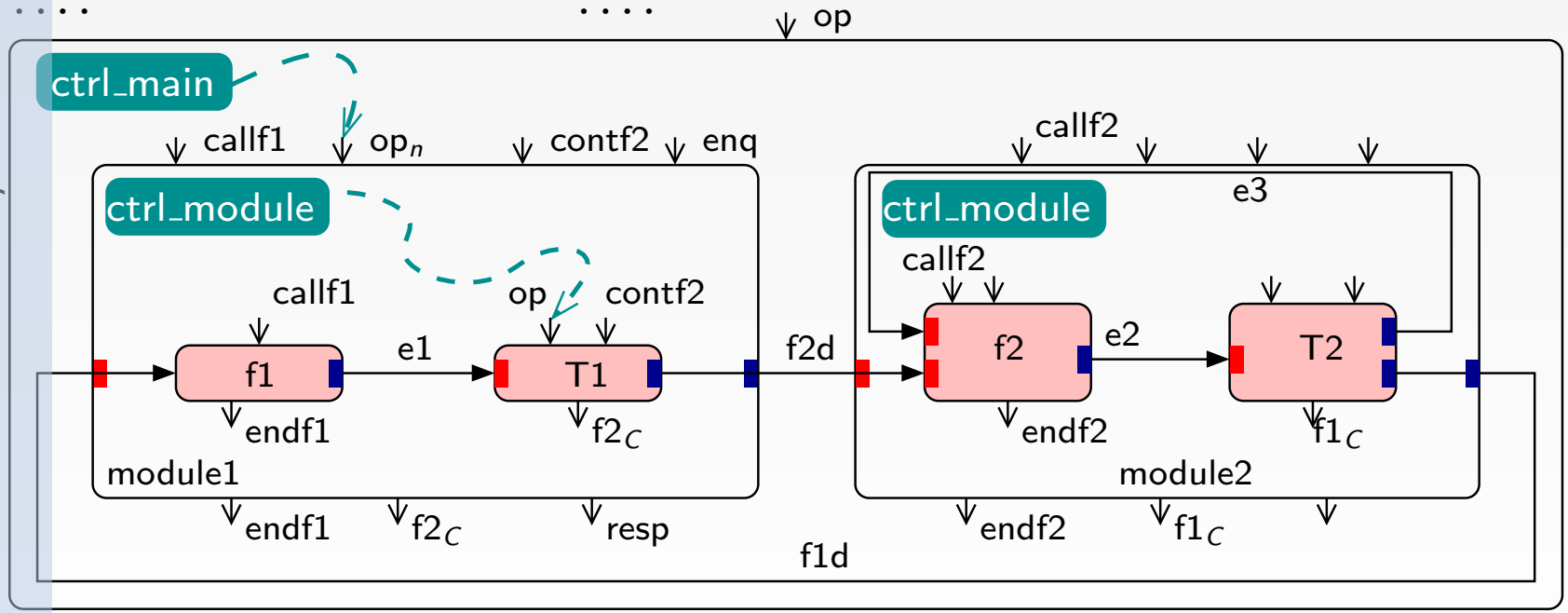
Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

```
module1::T1(){
  while(true){
    x++;
    ...;
    p2.f2(x);
    ...;
    wait(e1); ...;
  }
}
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

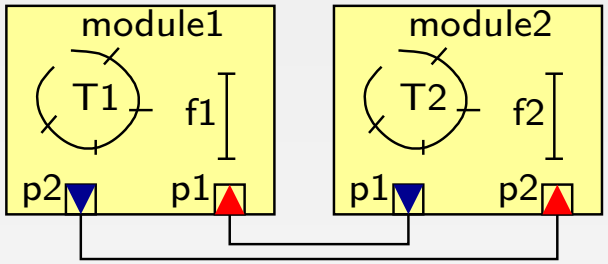


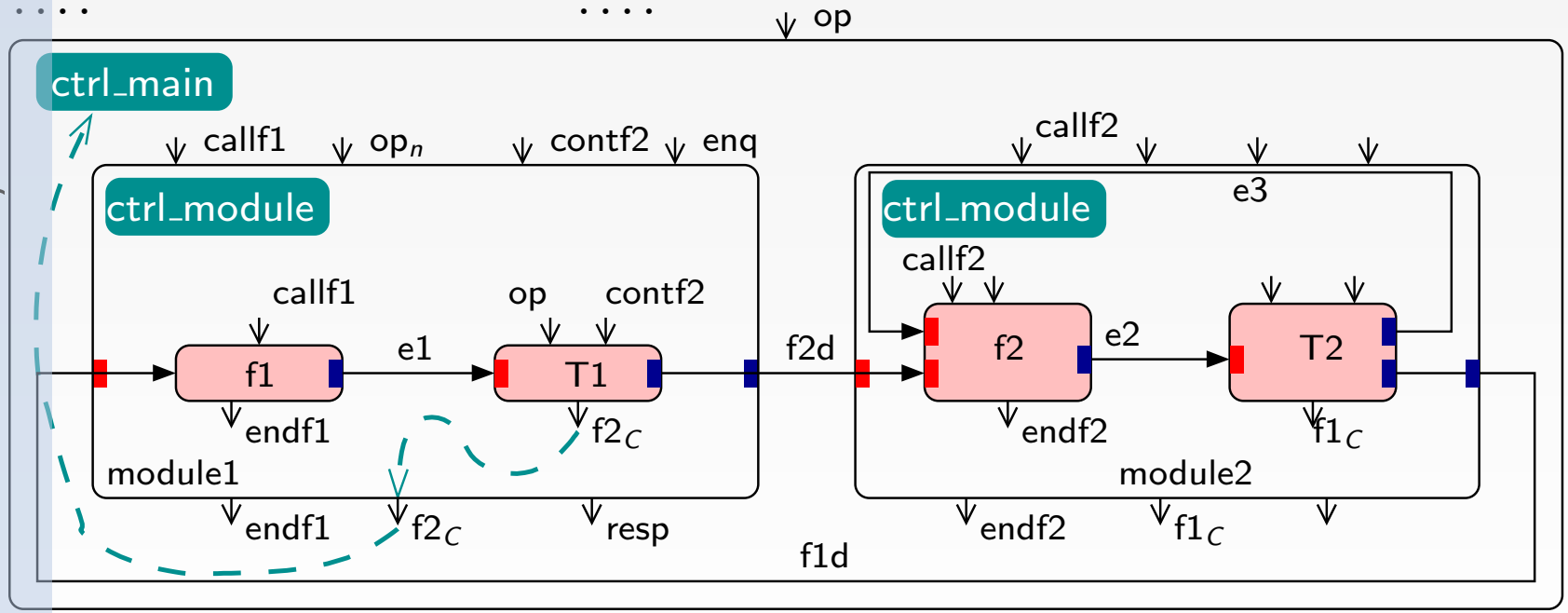


Structural Correspondence


```
module1::T1(){
  while(true){
    x++;
    ...;
    p2.f2(x);
    ...;
    wait(e1); ...;
  }
}
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

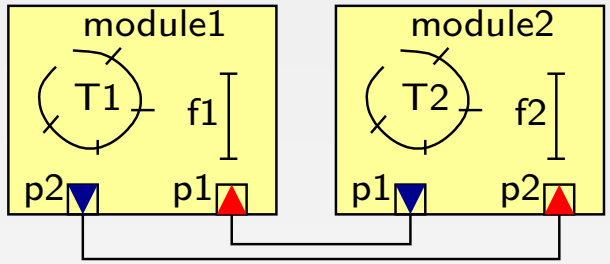


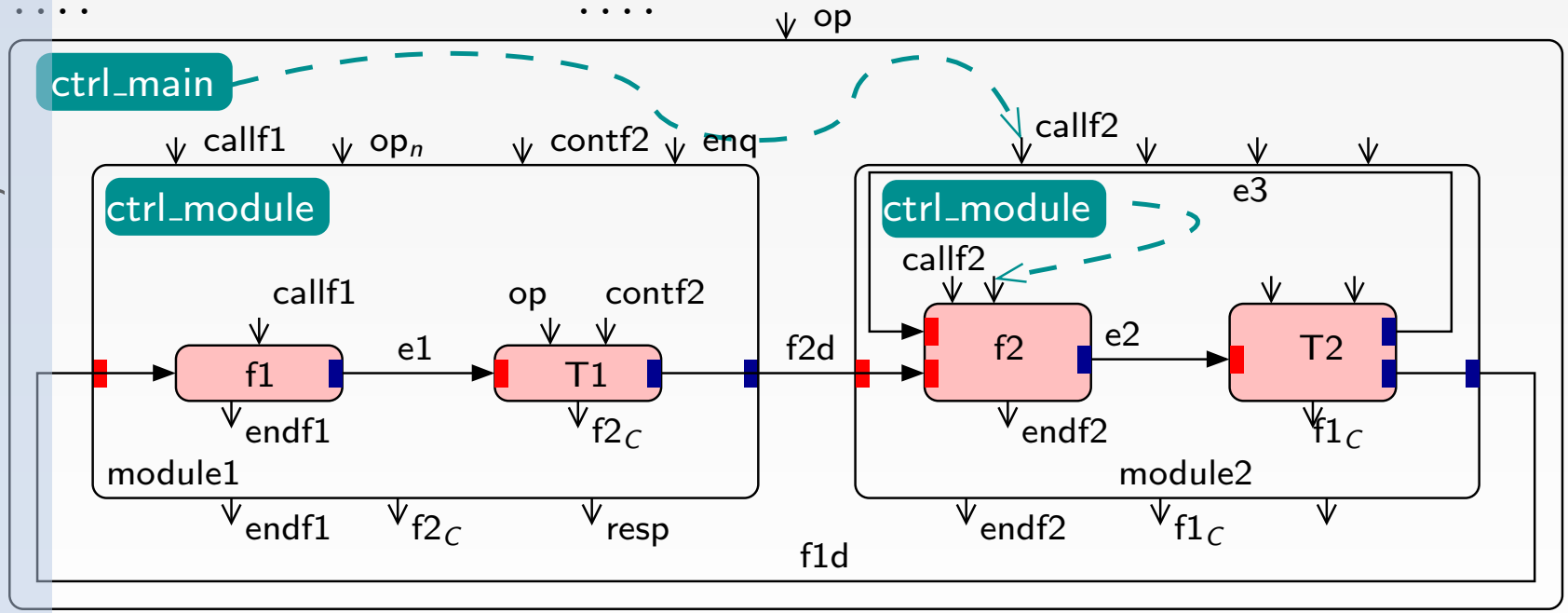


Structural Correspondence

```
module1::T1(){
  while(true){
    x++;
    ...;
    p2.f2(x);
    ...;
    wait(e1); ...;
  }
}
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```



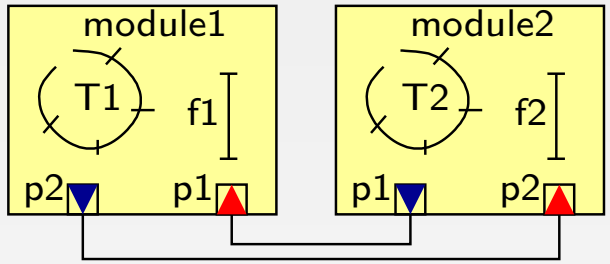


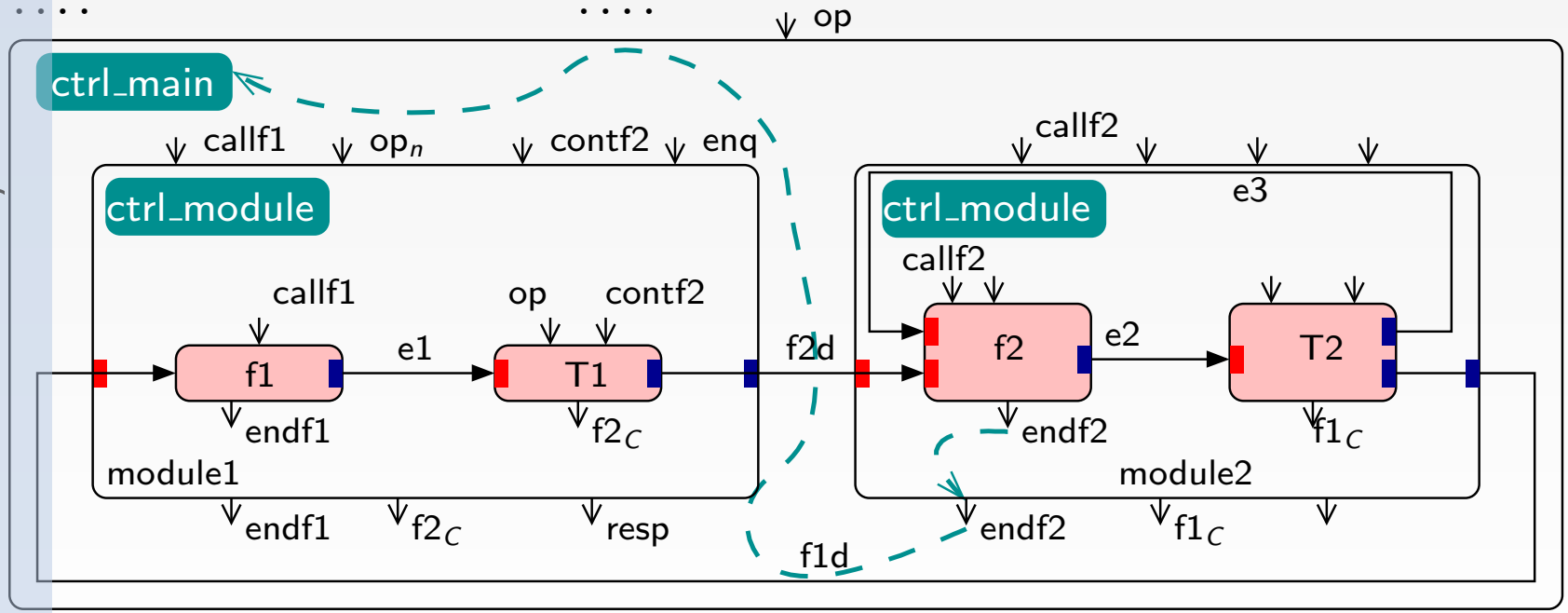
Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

```
module1::T1(){
  while(true){
    x++;
    ...;
    p2.f2(x);
    ...;
    wait(e1); ...;
  }
}
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
```



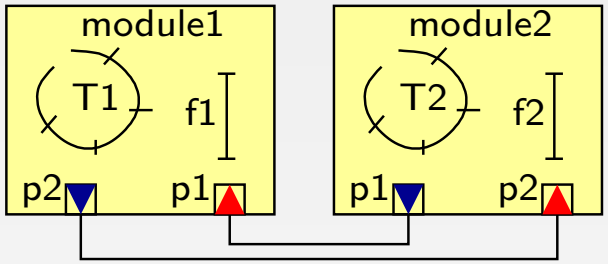


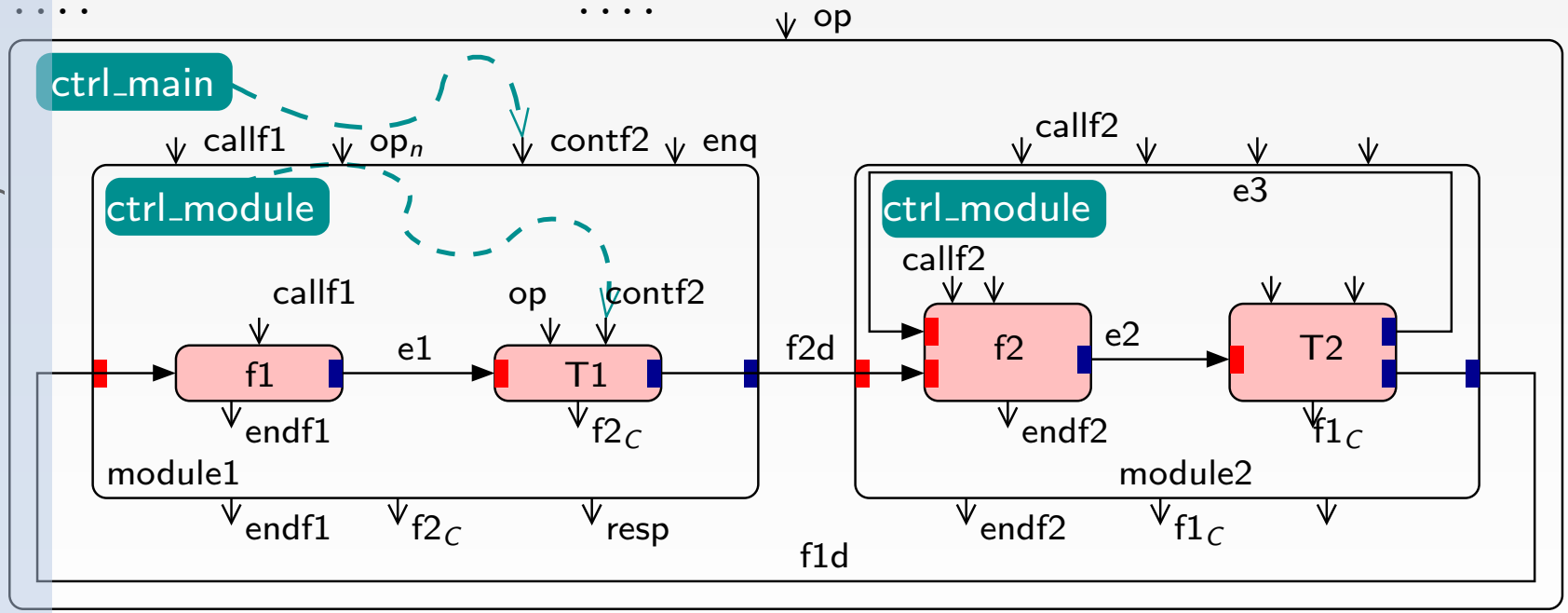
Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

```
module1::T1(){
  while(true){
    x++;
    ...;
    p2.f2(x);
    ...;
    wait(e1); ...;
  }
}
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```



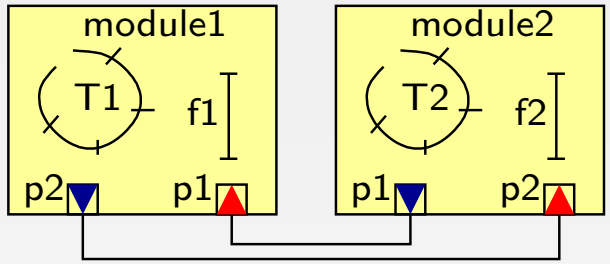


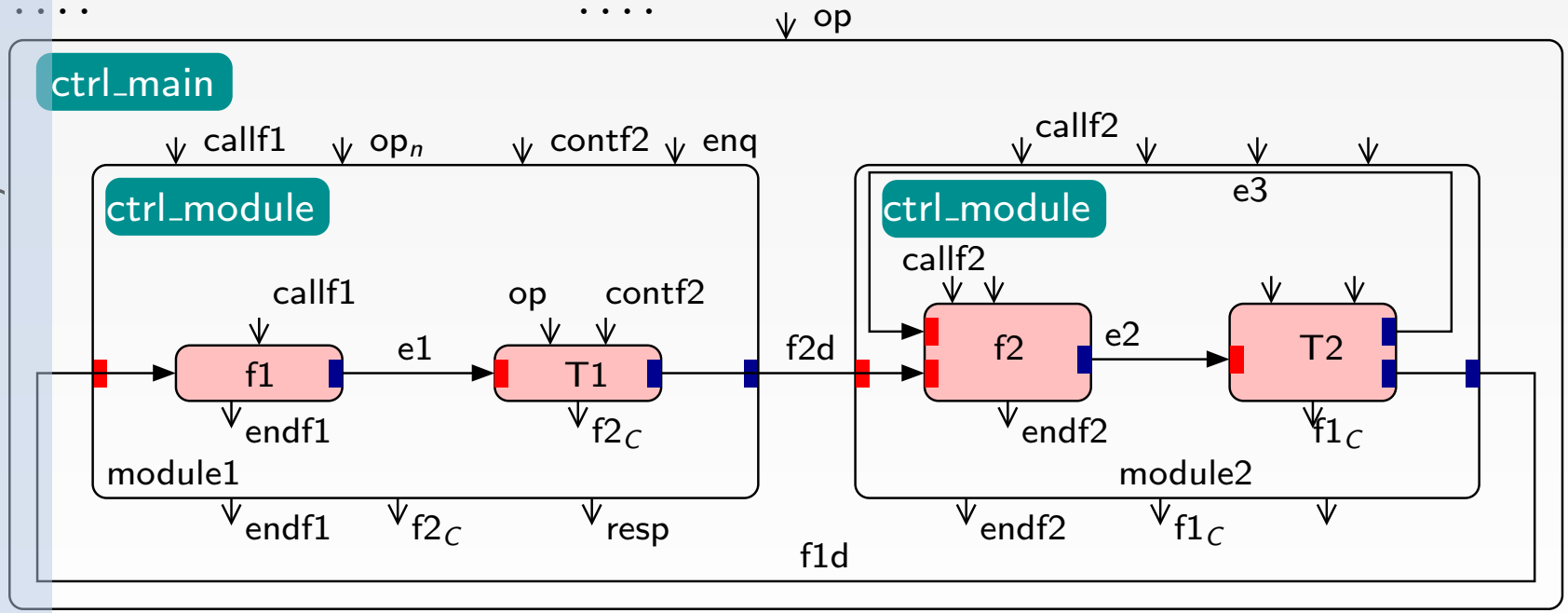
Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

```
module1::T1(){
  while(true){
    x++;
    ...;
    p2.f2(x);
    ...;
    wait(e1); ...;
  }
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```



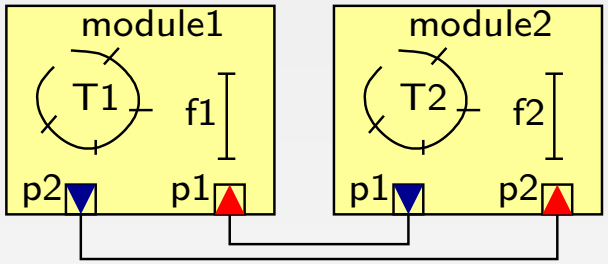


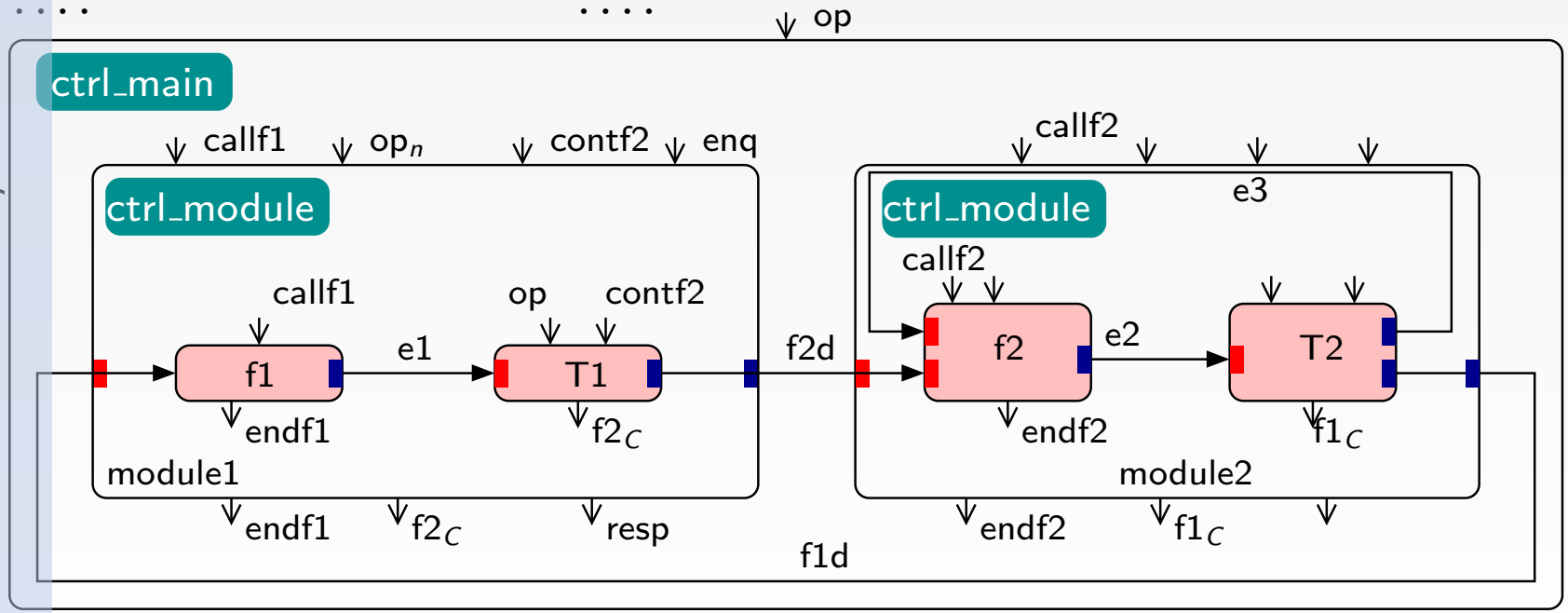
Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
    }
}
....
```

```
module2::f2(int x){
      ...;
      y = x * 42;
      ...;
      notify(e2);
      ...;
      ...;
} }
....
```

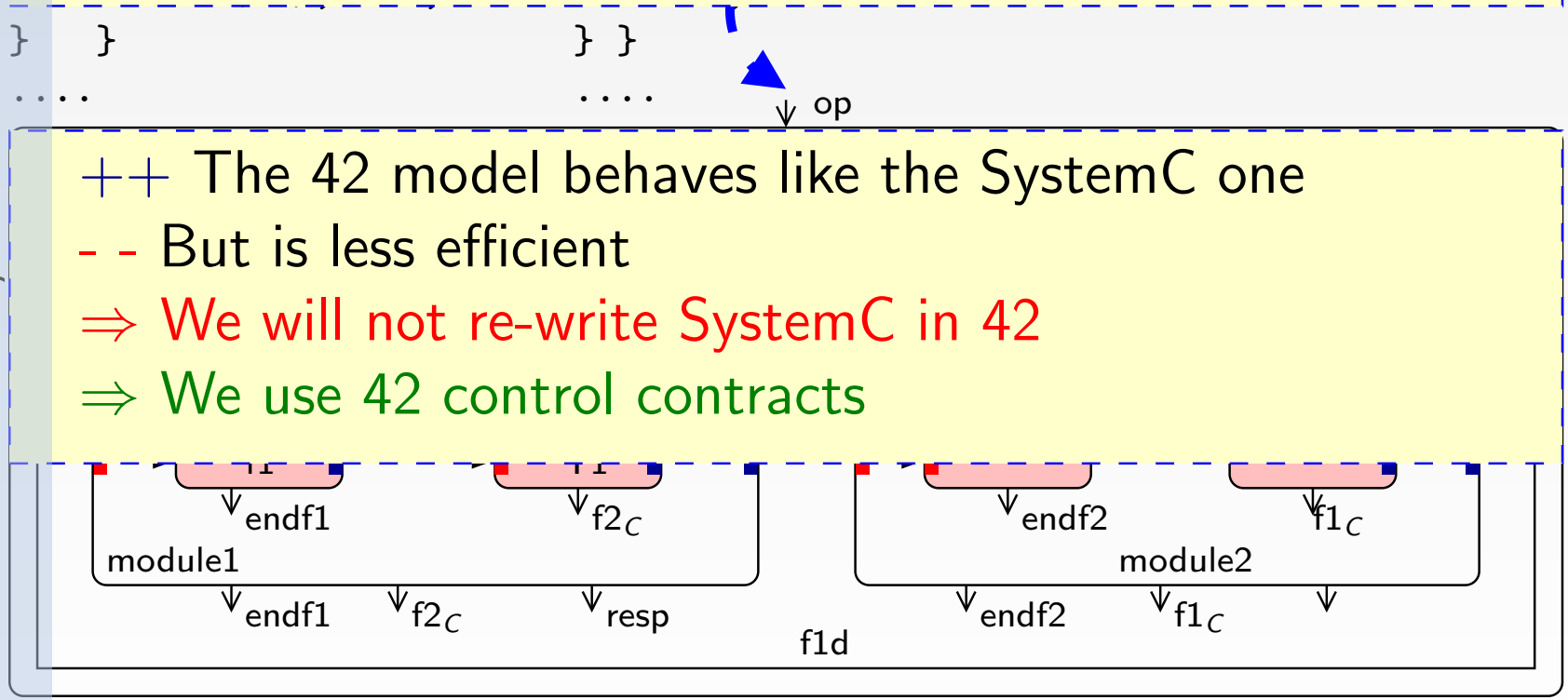




Structural Correspondence

tel-00539648, version 1 - 24 Nov 2010

An activation of the main component with OP corresponds to what happens in the SystemC-TLM model when the scheduler elects a new thread to execute.



++ The 42 model behaves like the SystemC one

- - But is less efficient

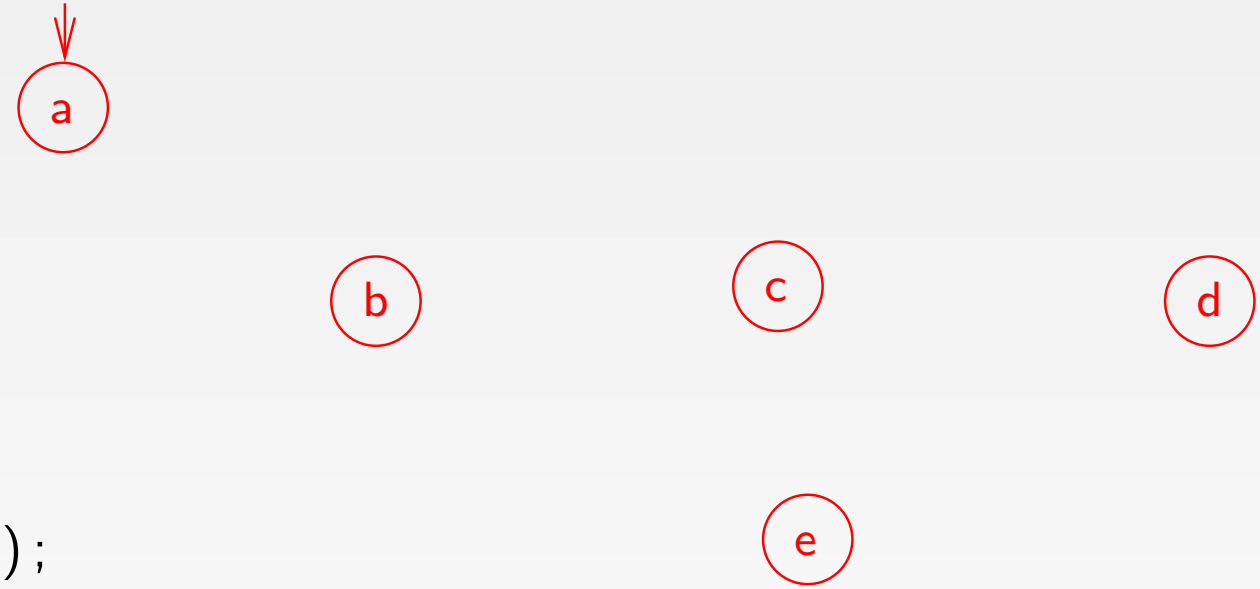
⇒ We will not re-write SystemC in 42

⇒ We use 42 control contracts

hal-00539648, version 1 - 24 Nov 2010

Extracting Control Contracts From SC-TLM Code

```
ⓐwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
        ⓒp.f(x);
        ⓓp.g(x);
    }
    y=x+1;
    while(y < 5){
        y++;
        ⓔwait(e2);
    }
```

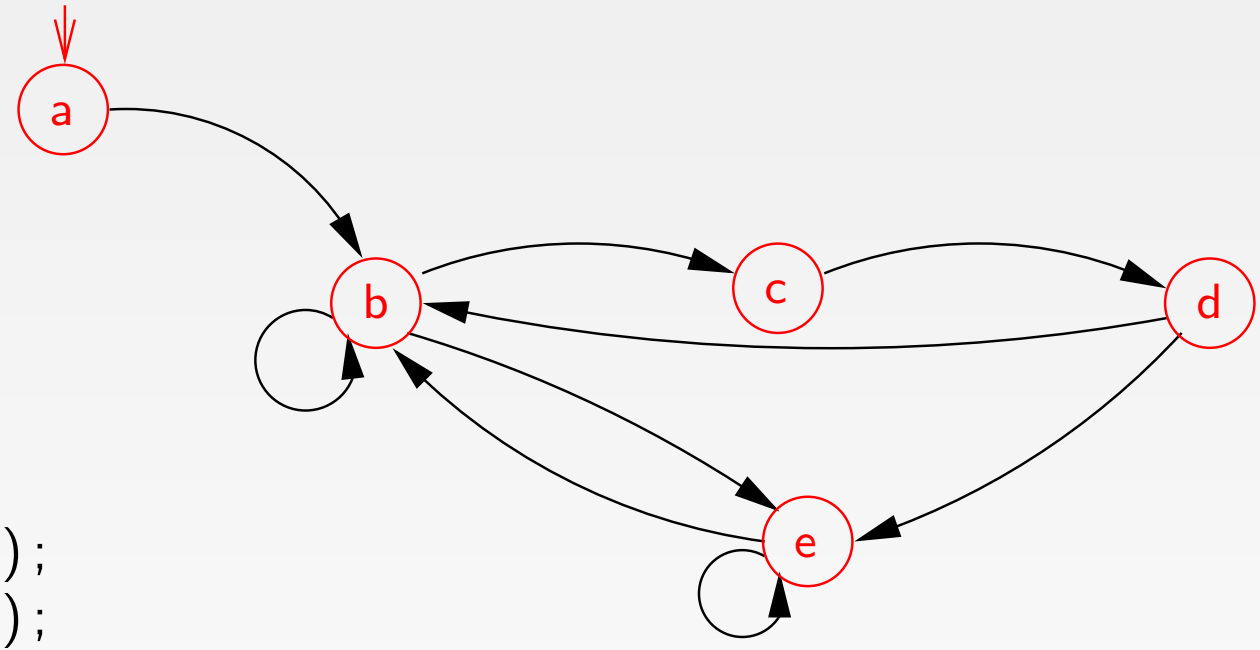


- States correspond to wait statements or function calls.

Extracting Control Contracts From SC-TLM Code

hal-00539648, version 1 - 24 Nov 2010

```
ⓐwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
        ⓒp.f(x);
        ⓓp.g(x);
    }
    y=x+1;
    while(y < 5){
        y++;
        ⓔwait(e2);
    }
```

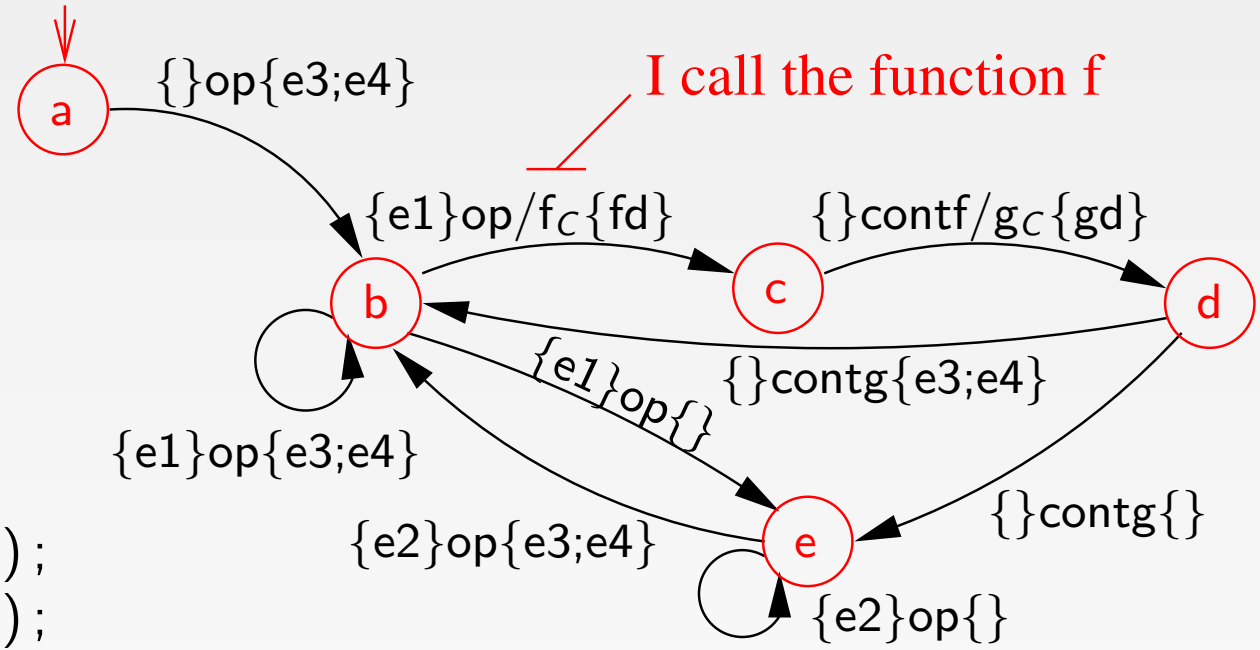


- States correspond to wait statements or function calls.
- Data values are abstracted

Extracting Control Contracts From SC-TLM Code

tel-00539648, version 1 - 24 Nov 2010

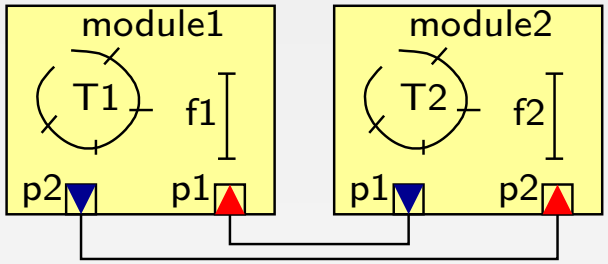
```
ⓑwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
        ⓒp.f(x);
        ⓓp.g(x);
    }
    y=x+1;
    while(y < 5){
        y++;
        ⓔwait(e2);
    }
```

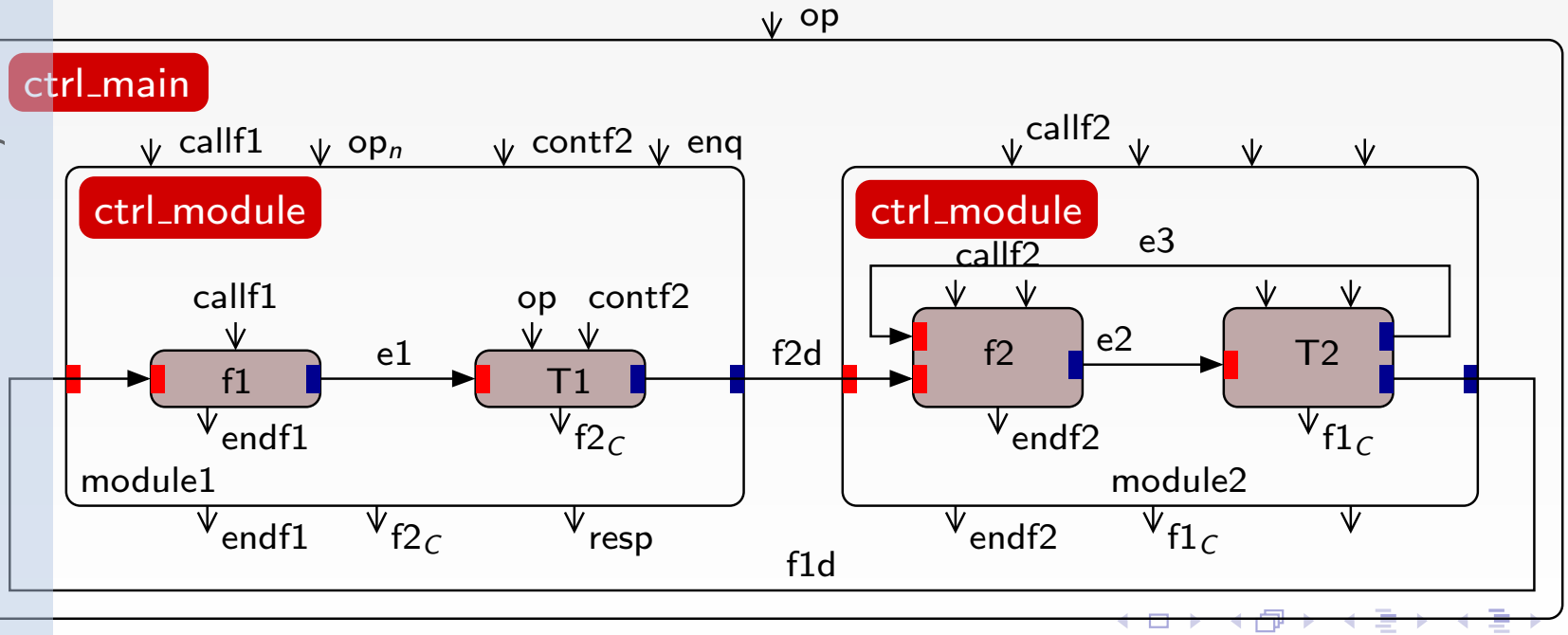


- States correspond to wait statements or function calls.
- Data values are abstracted

Lightweight Execution for SystemC-TLM

tel-00539648, version 1 - 24 Nov 2010

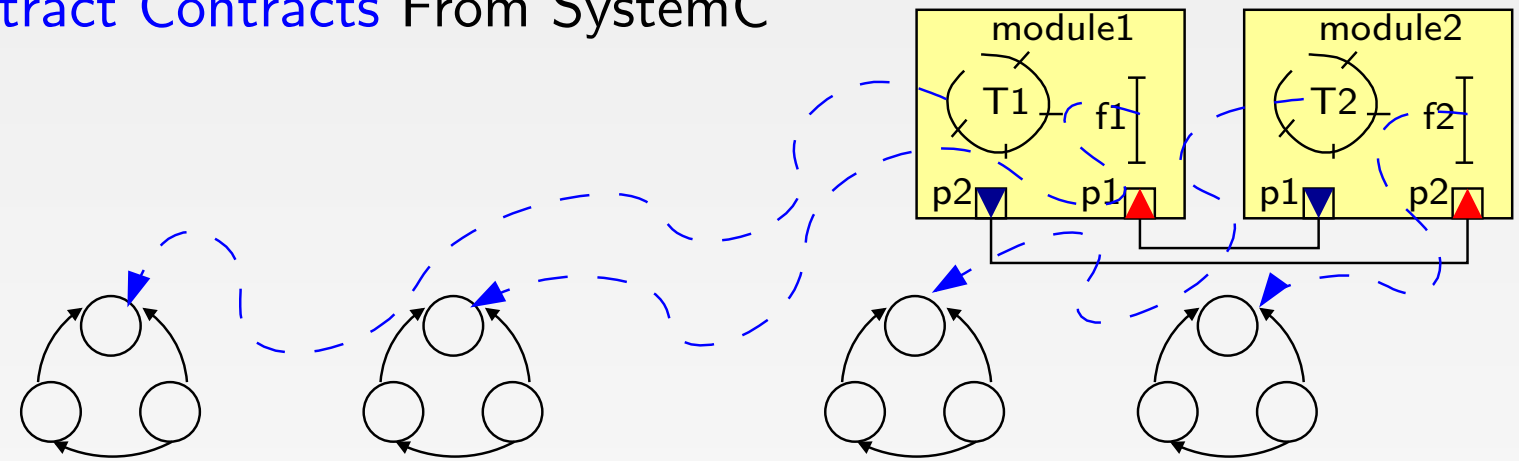


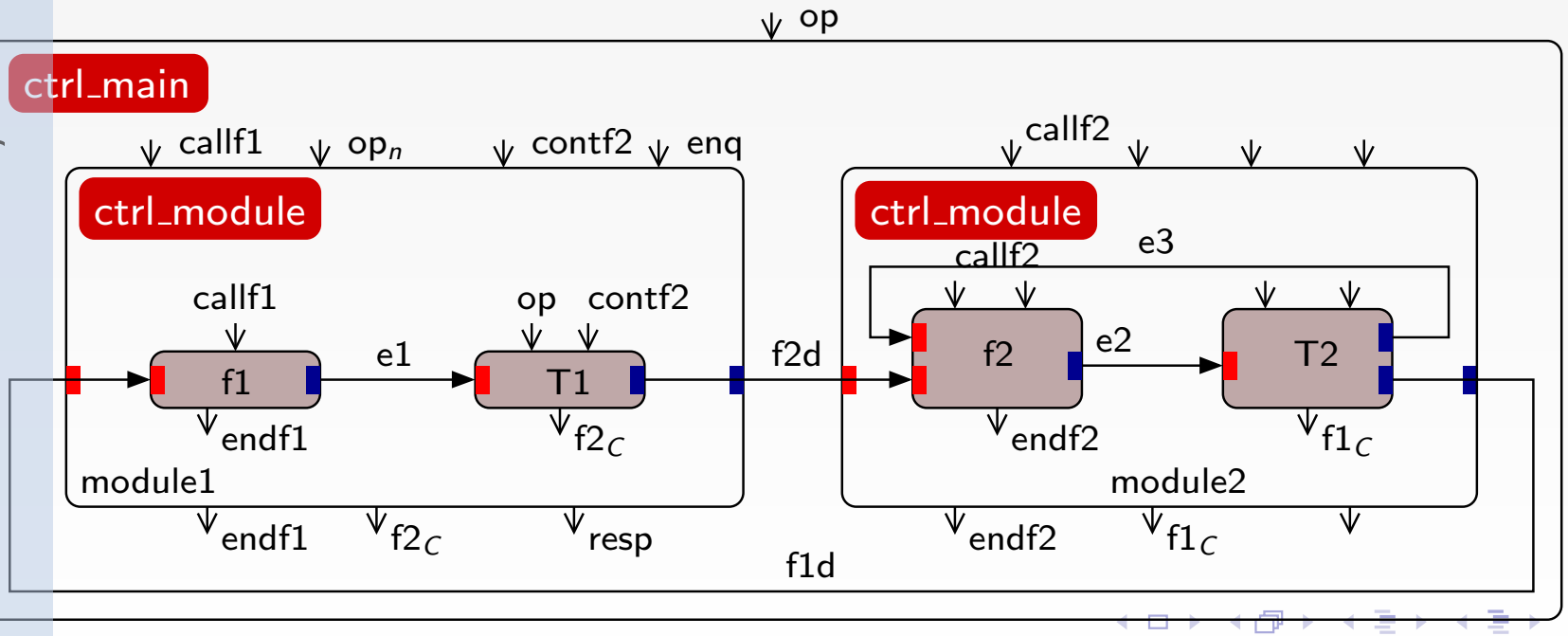


Lightweight Execution for SystemC-TLM

tel-00539648, version 1 - 24 Nov 2010

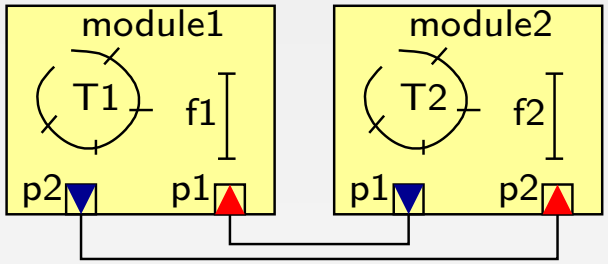
- Extract Contracts From SystemC

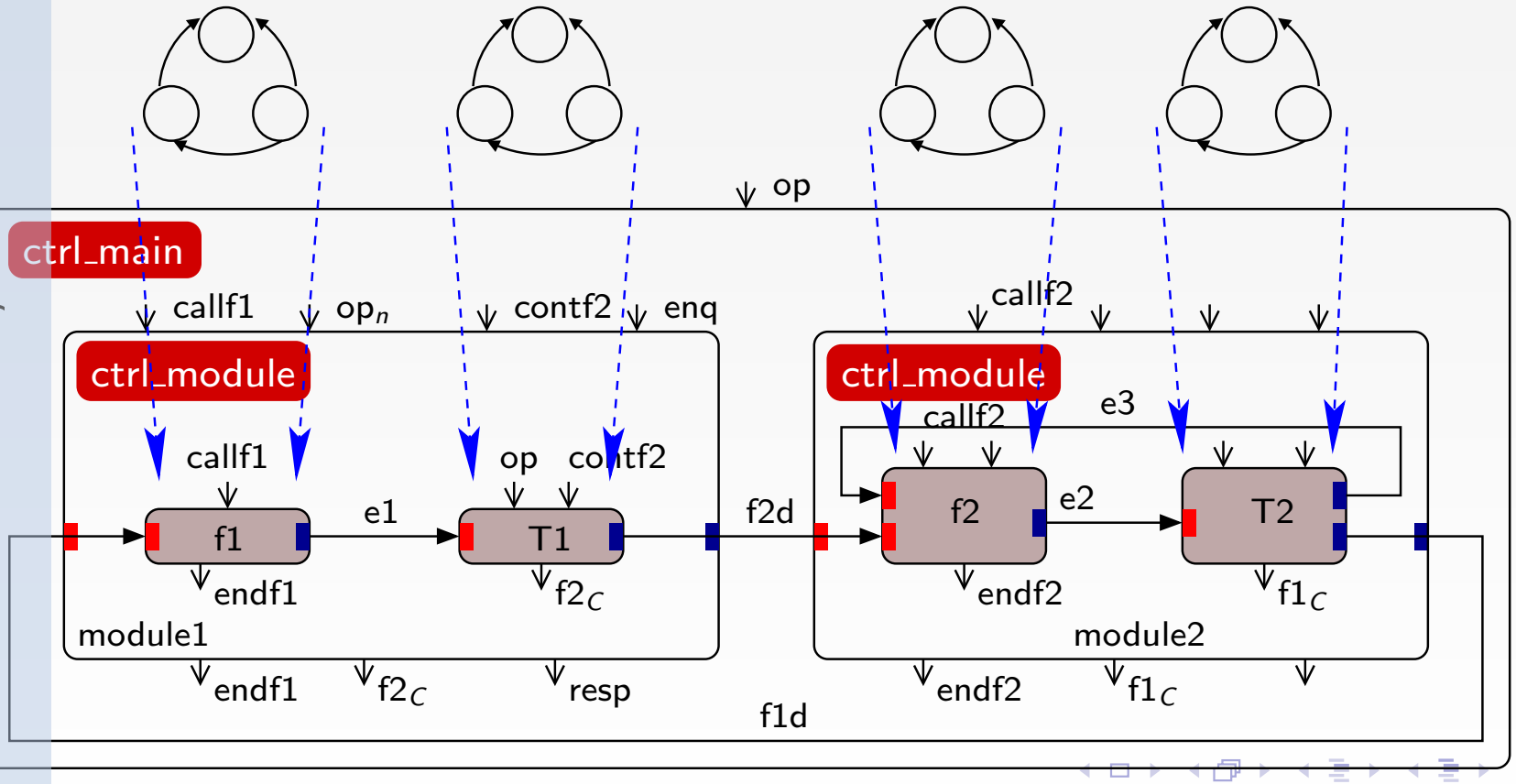




Lightweight Execution for SystemC-TLM

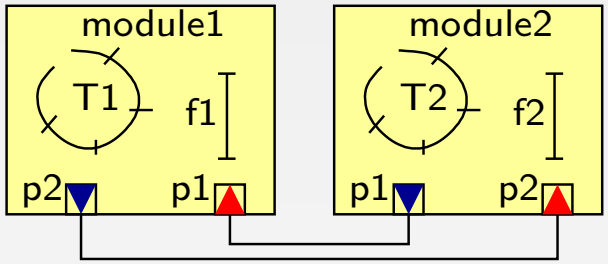
- Extract Contracts From SystemC
- Execute Contracts instead of Code

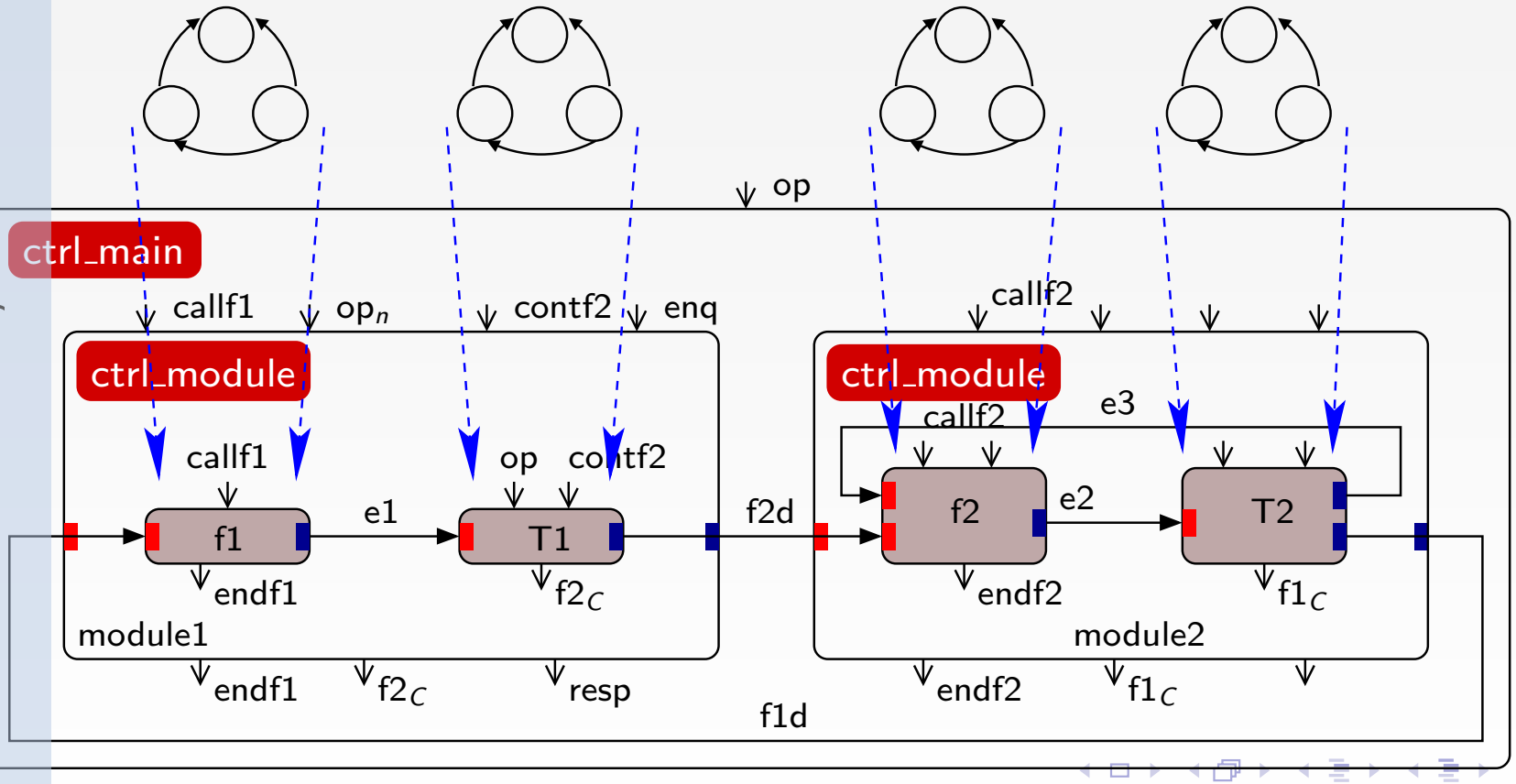




Lightweight Execution for SystemC-TLM

- Extract Contracts From SystemC
- Execute Contracts instead of Code
- Produces a super-set of SystemC executions





tel-00539648, version 1 - 24 Nov 2010

tel-00539648, version 1 - 24 Nov 2010

Contents

1 Introduction & Sources of Inspiration

2 Overview of 42 & Examples [GPCE07]

3 Hardware Simulation by Interpreting Contracts [COORD09]

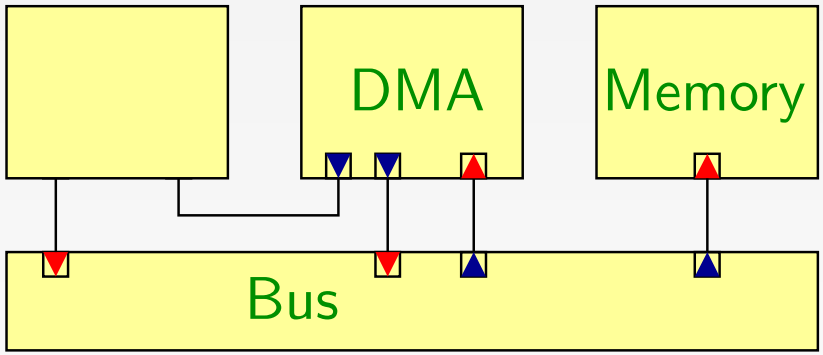
4 Using 42 Together with Existing Approaches [EMSOFT09]
- TLM with SystemC
- 42-ization of SystemC/TLM
- Typical Uses of 42 Contracts with SystemC-TLM

5 Some Related Work

6 Summary

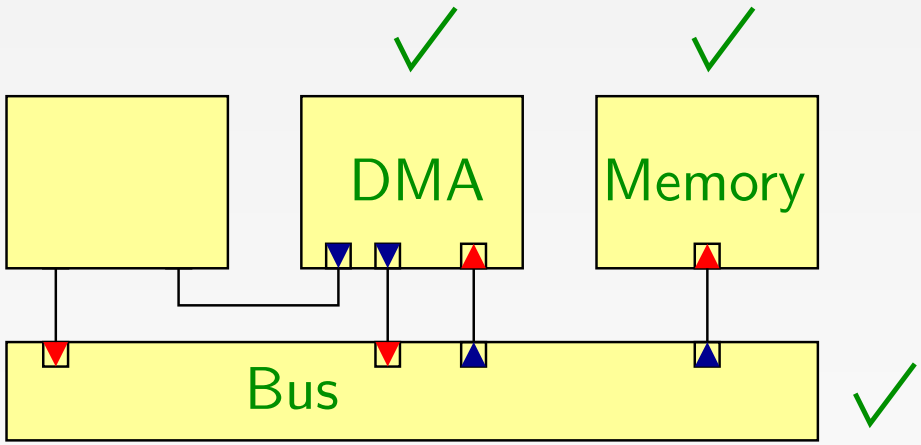
tel-00539648, version 1 - 24 Nov 2010

TL-Modeling with SystemC: an Example



tel-00539648, version 1 - 24 Nov 2010

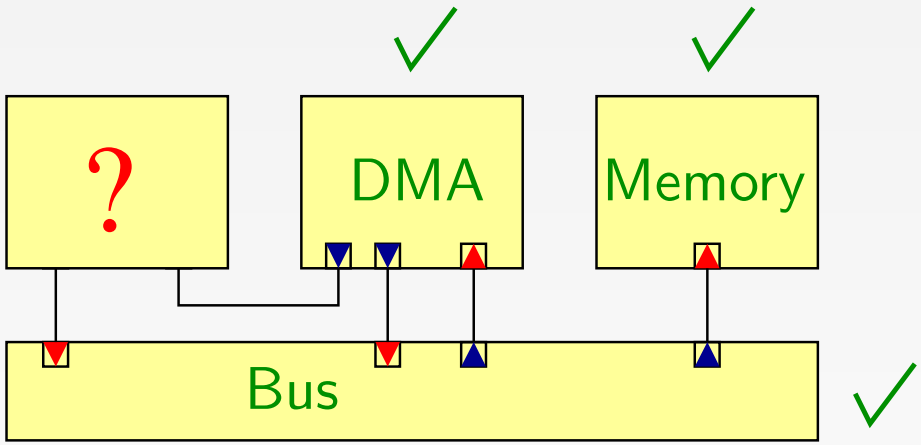
TL-Modeling with SystemC: an Example



- Reuse of existing components (e.g., DMA, Memory, Bus)

tel-00539648, version 1 - 24 Nov 2010

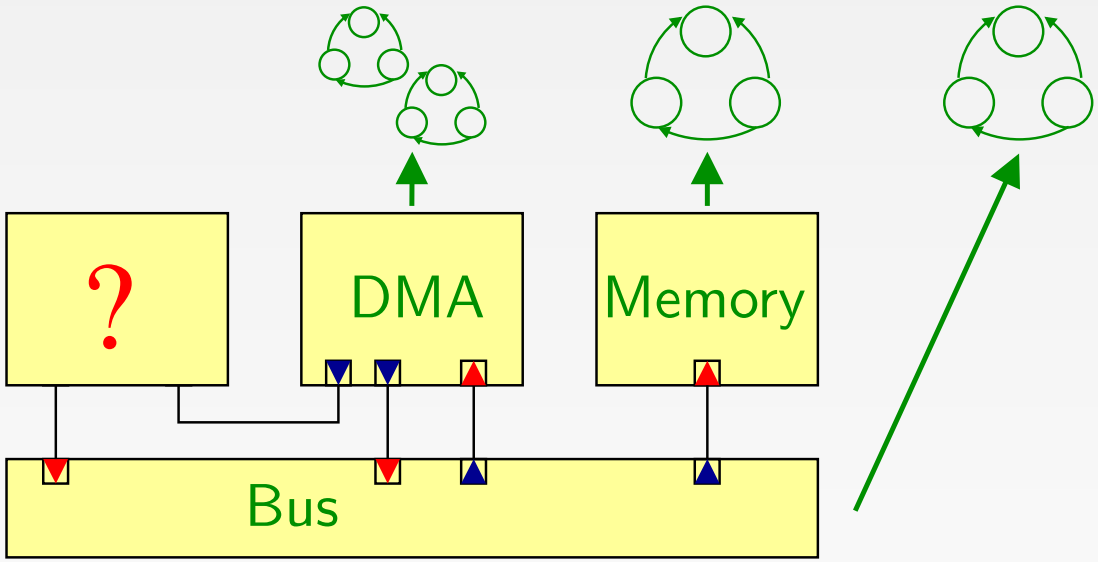
TL-Modeling with SystemC: an Example



- Reuse of existing components (e.g., DMA, Memory, Bus)
- Part of the functionality is not implemented

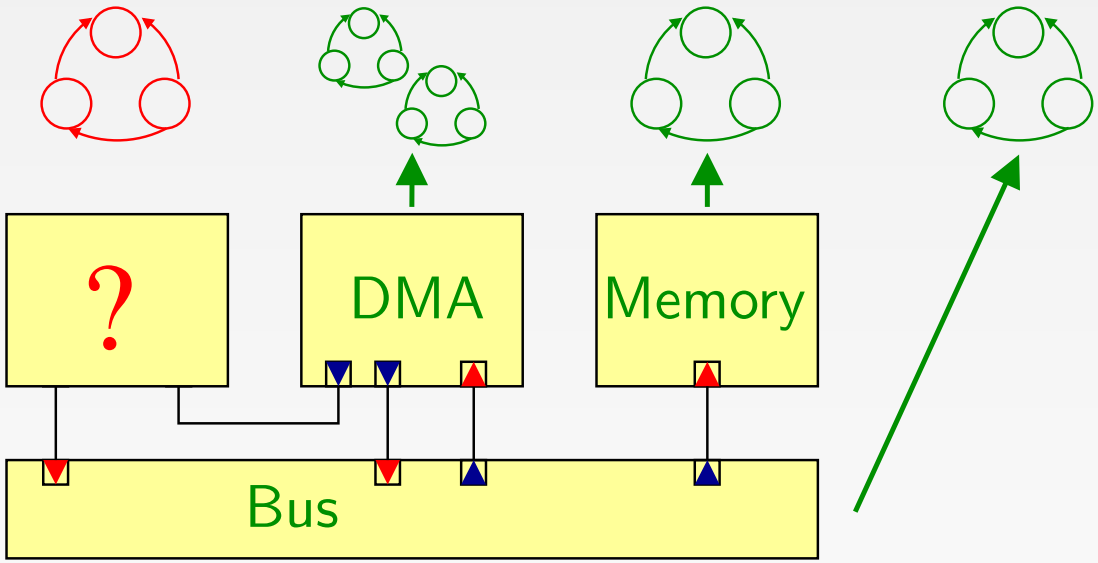
tel-00539648, version 1 - 24 Nov 2010

TL-Modeling with SystemC: an Example



- Extract control contracts from existing SC-TLM components

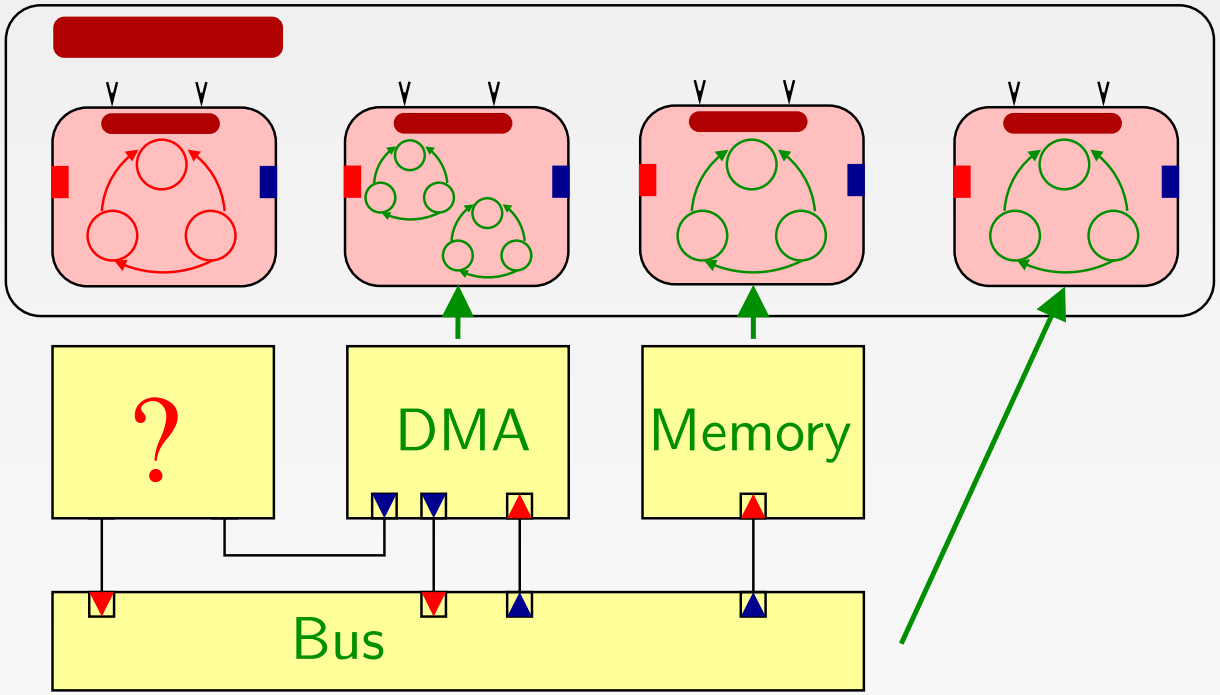
TL-Modeling with SystemC: an Example



- Extract control contracts from existing SC-TLM components
- Write new contracts for the missing ones

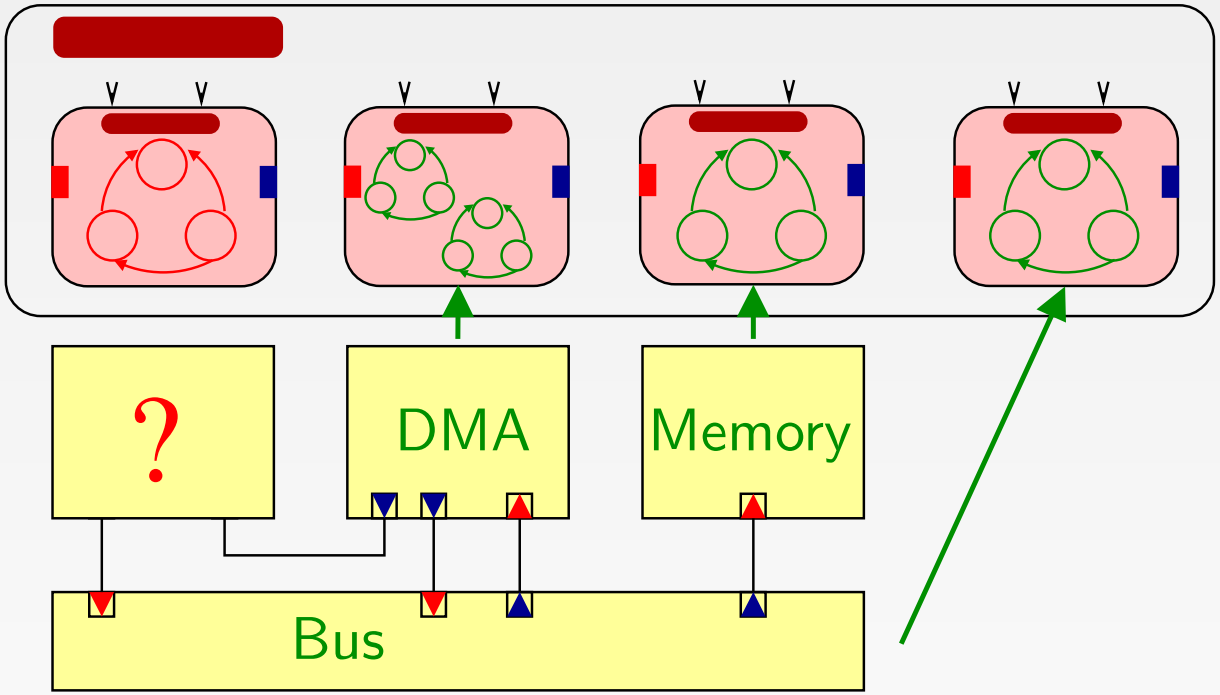
tel-00539648, version 1 - 24 Nov 2010

TL-Modeling with SystemC: an Example



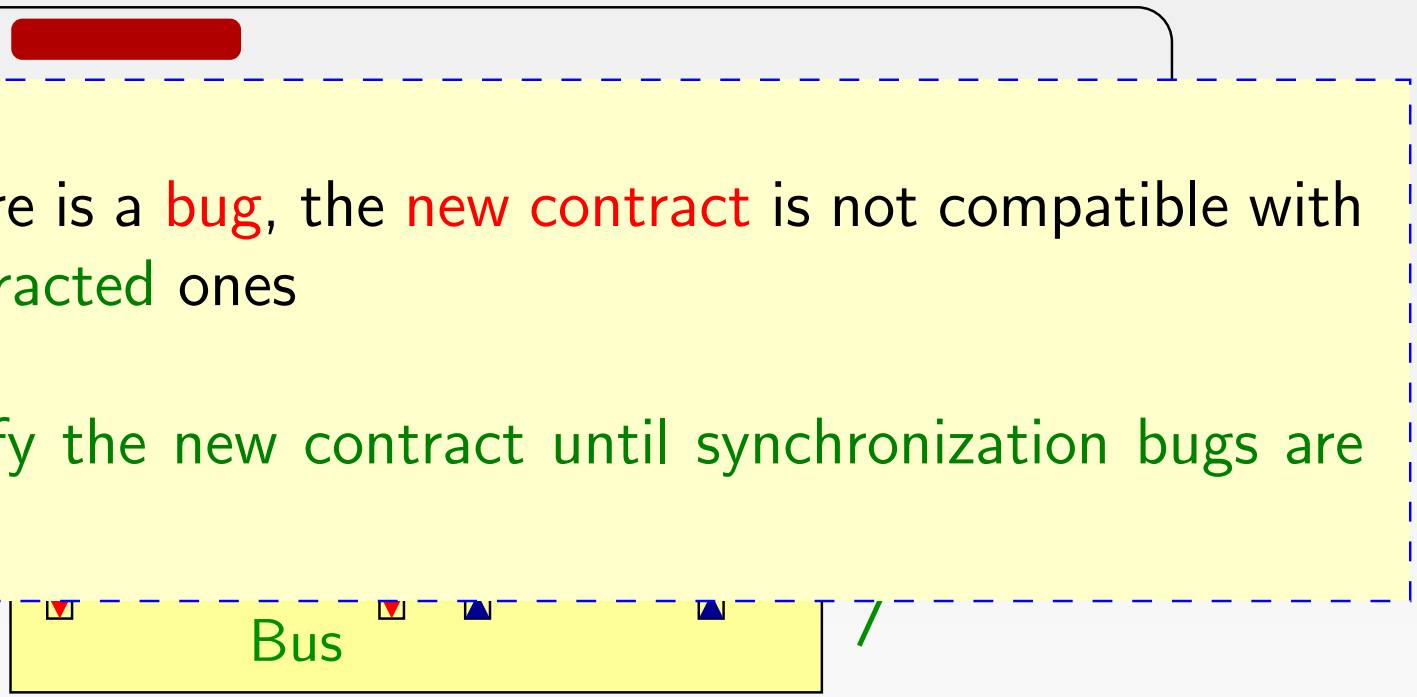
- Extract control contracts from existing SC-TLM components
- Write new contracts for the missing ones
- Execute the contracts with 42 controllers

TL-Modeling with SystemC: an Example



- Extract control contracts from existing SC-TLM components
- Write new contracts for the missing ones
- Execute the contracts with 42 controllers
- ⇒ Find Synchronization Bugs

TL-Modeling with SystemC: an Example



- Extract control contracts from existing SC-TLM components
- Write new contracts for the missing ones
- Execute the contracts with 42 controllers
- ⇒ Find Synchronization Bugs

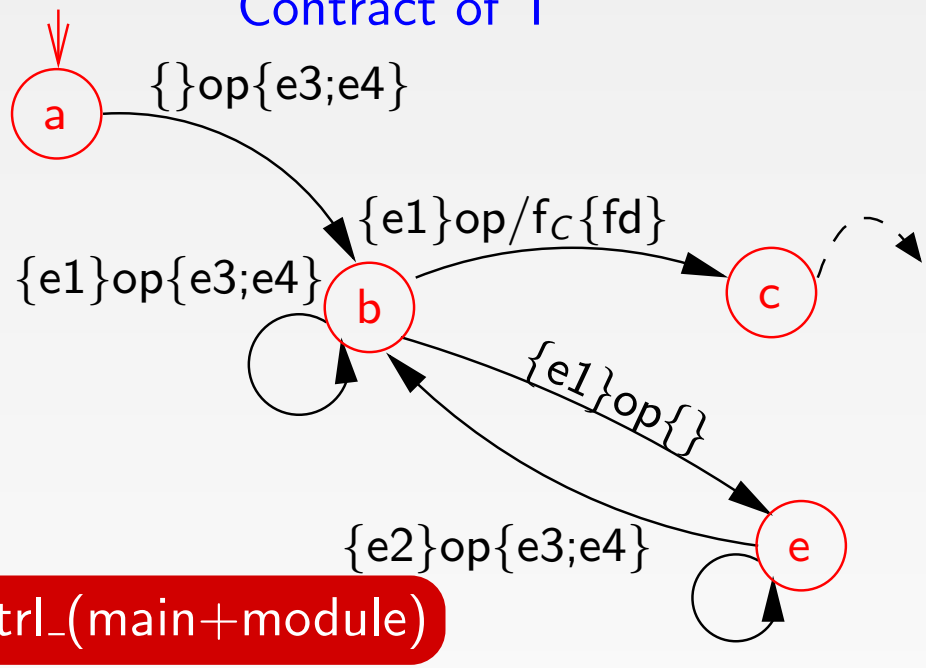
tel-00539648, version 1 - 24 Nov 2010

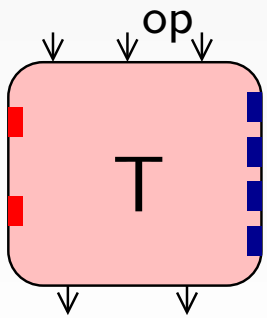
Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
      x++;
      e3.notify();
      e4.notify();
      ⓑwait(e1);
      if(x < 42){
            ⓒp.f(x);
            ⓓp.g(x);
      .....
```

Contract of T



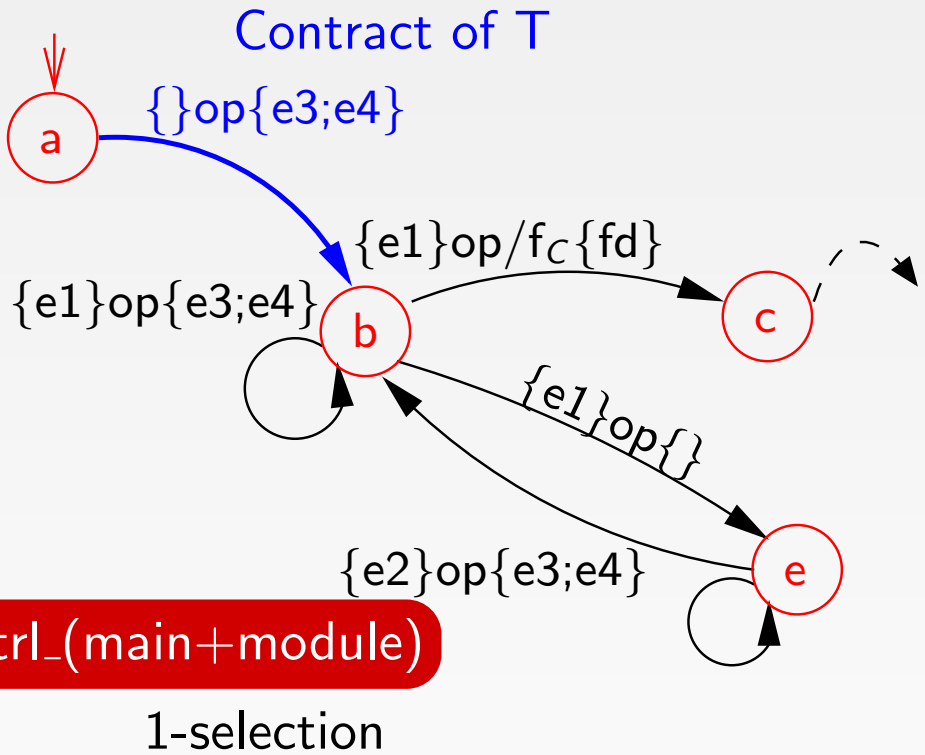


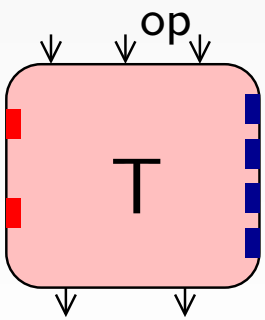
tel-00539648, version 1 - 24 Nov 2010

Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
      x++;
      e3.notify();
      e4.notify();
      ⓑwait(e1);
      if(x < 42){
            ⓒp.f(x);
            ⓓp.g(x);
      .....
```



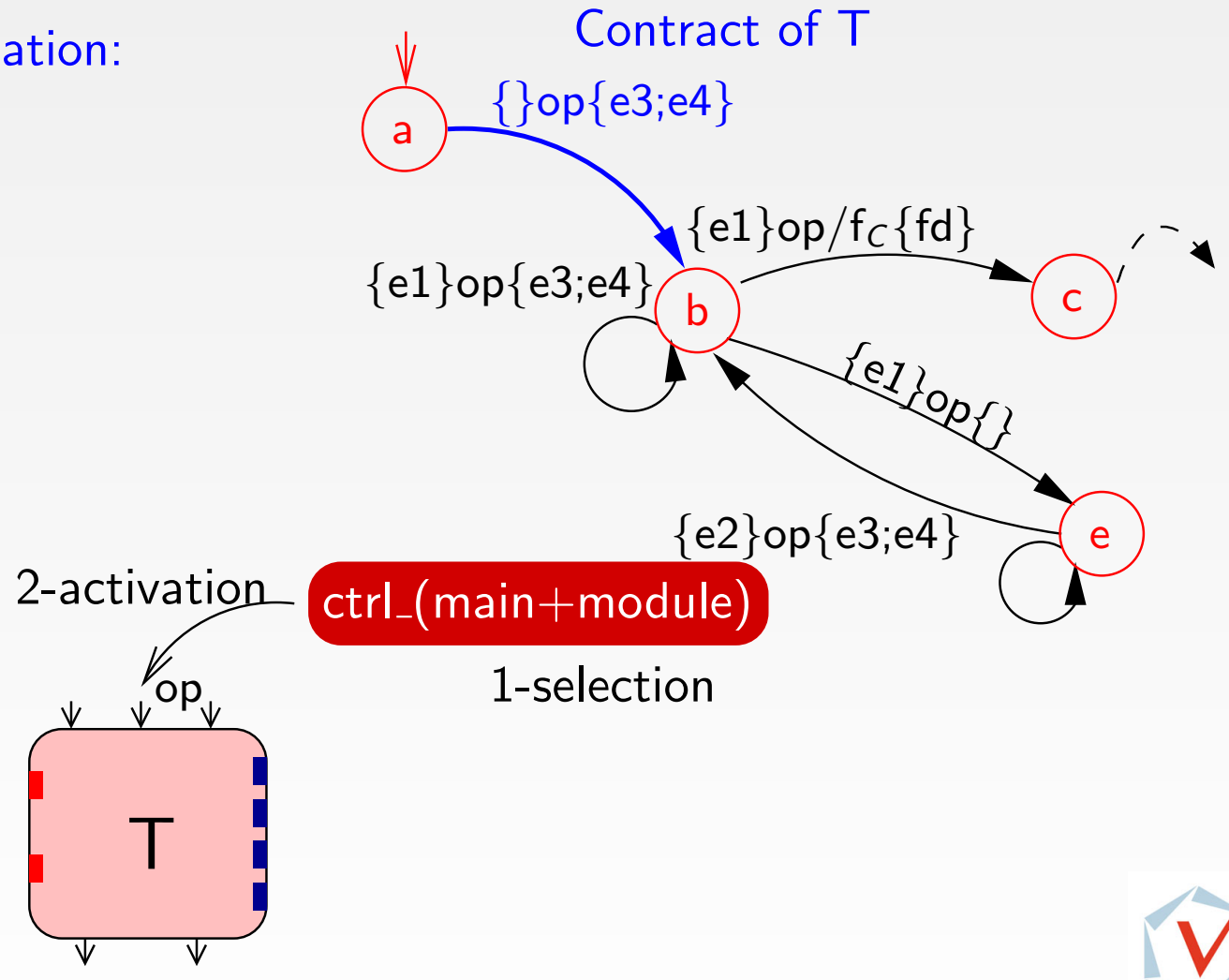


tel-00539648, version 1 - 24 Nov 2010

Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
      x++;
      e3.notify();
      e4.notify();
      ⓑwait(e1);
      if(x < 42){
            ⓒp.f(x);
            ⓓp.g(x);
      .....
```

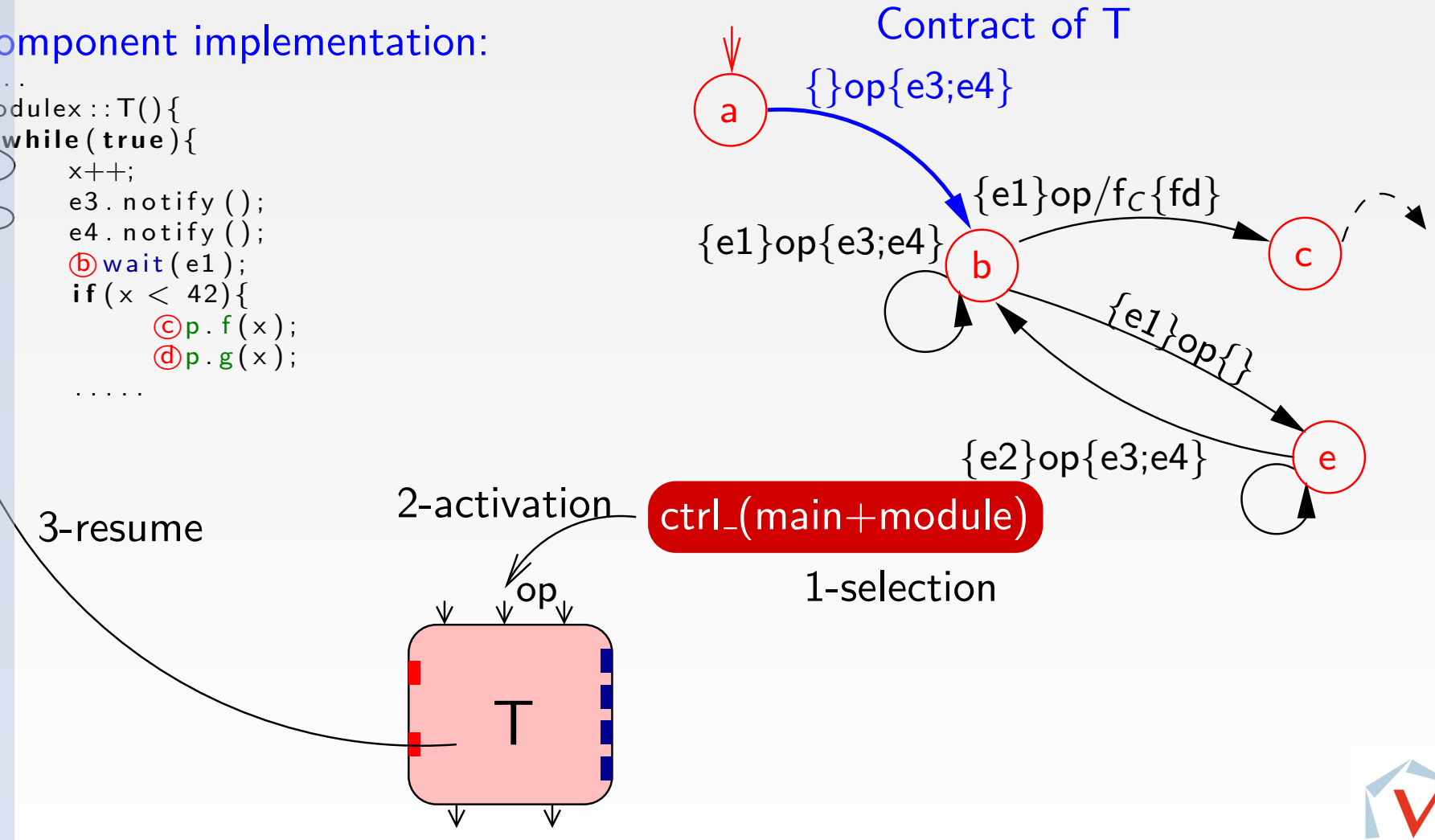


Executing the Implementation against the Contract

tel-00539648, version 1 - 24 Nov 2010

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
        ⓒp.f(x);
        ⓓp.g(x);
    .....
```



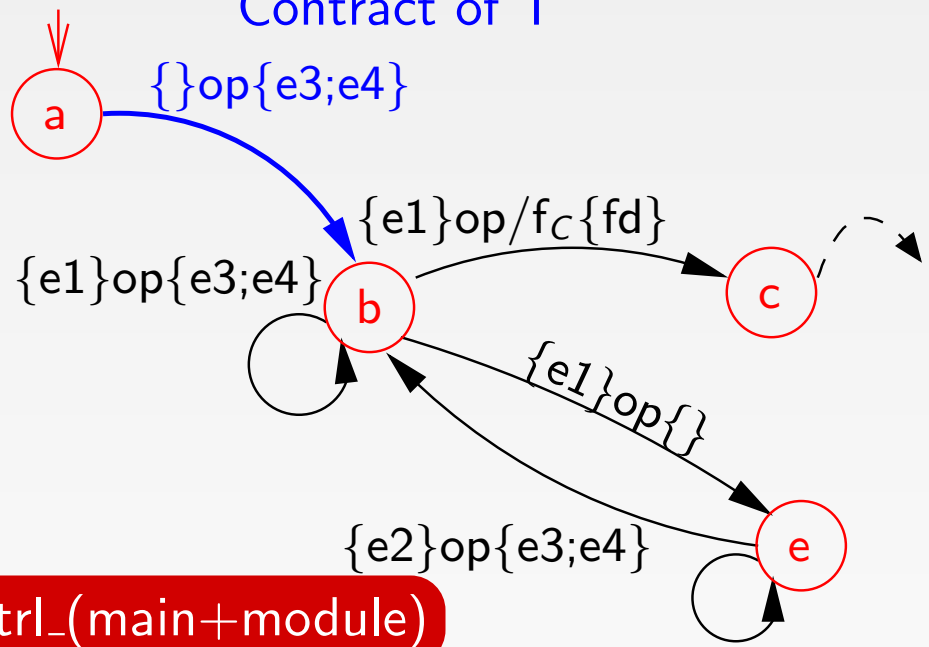
Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
        ⓒp.f(x);
        ⓓp.g(x);
    .....
```

...notify()

Contract of T



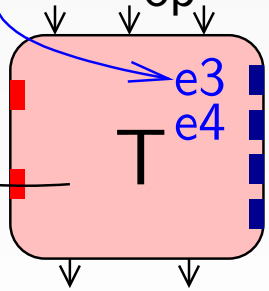
3-resume

2-activation

ctrl_(main+module)

1-selection

op

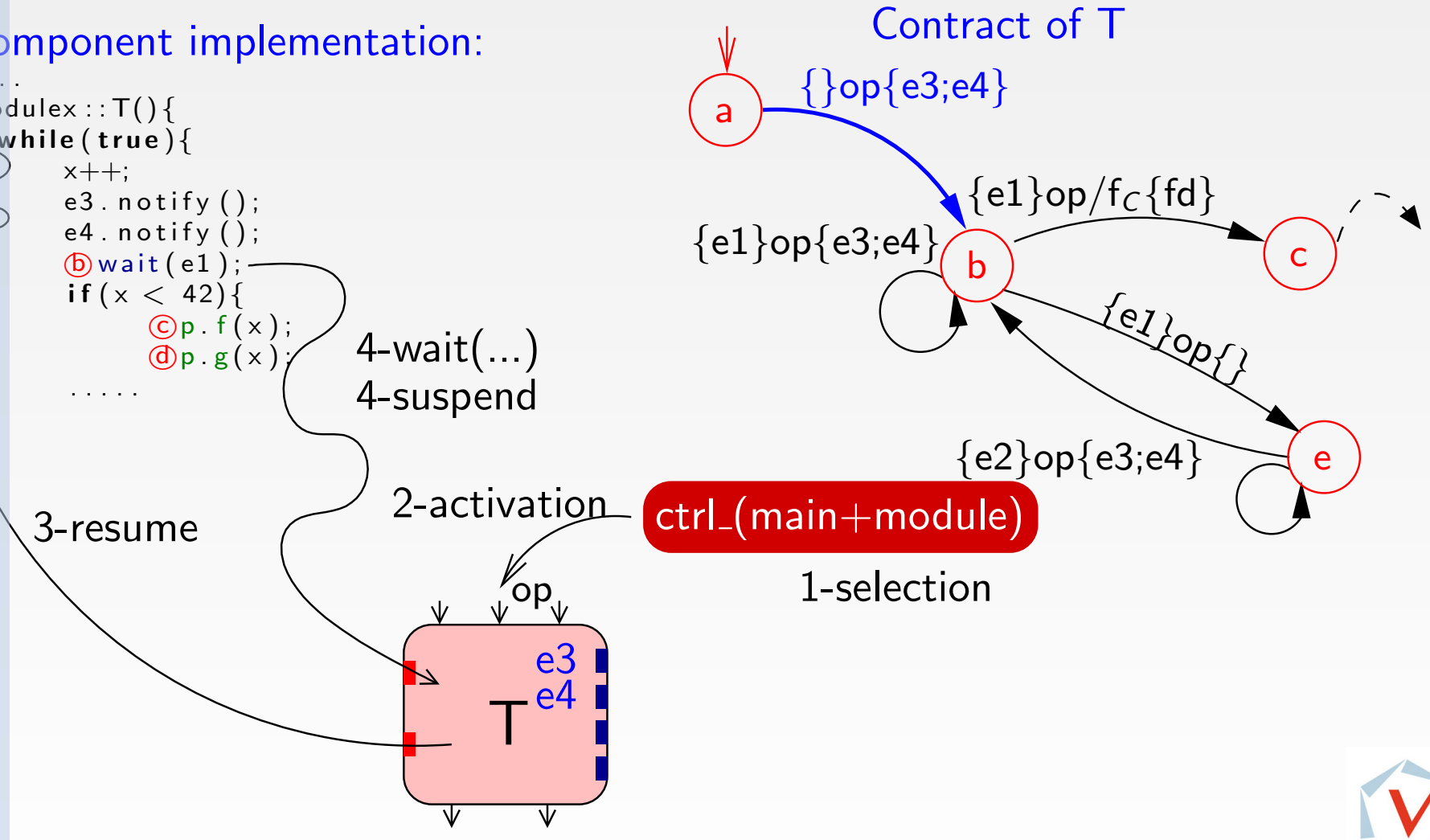


Executing the Implementation against the Contract

tel-00539648, version 1 - 24 Nov 2010

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
        ⓒp.f(x);
        ⓓp.g(x);
    .....
```

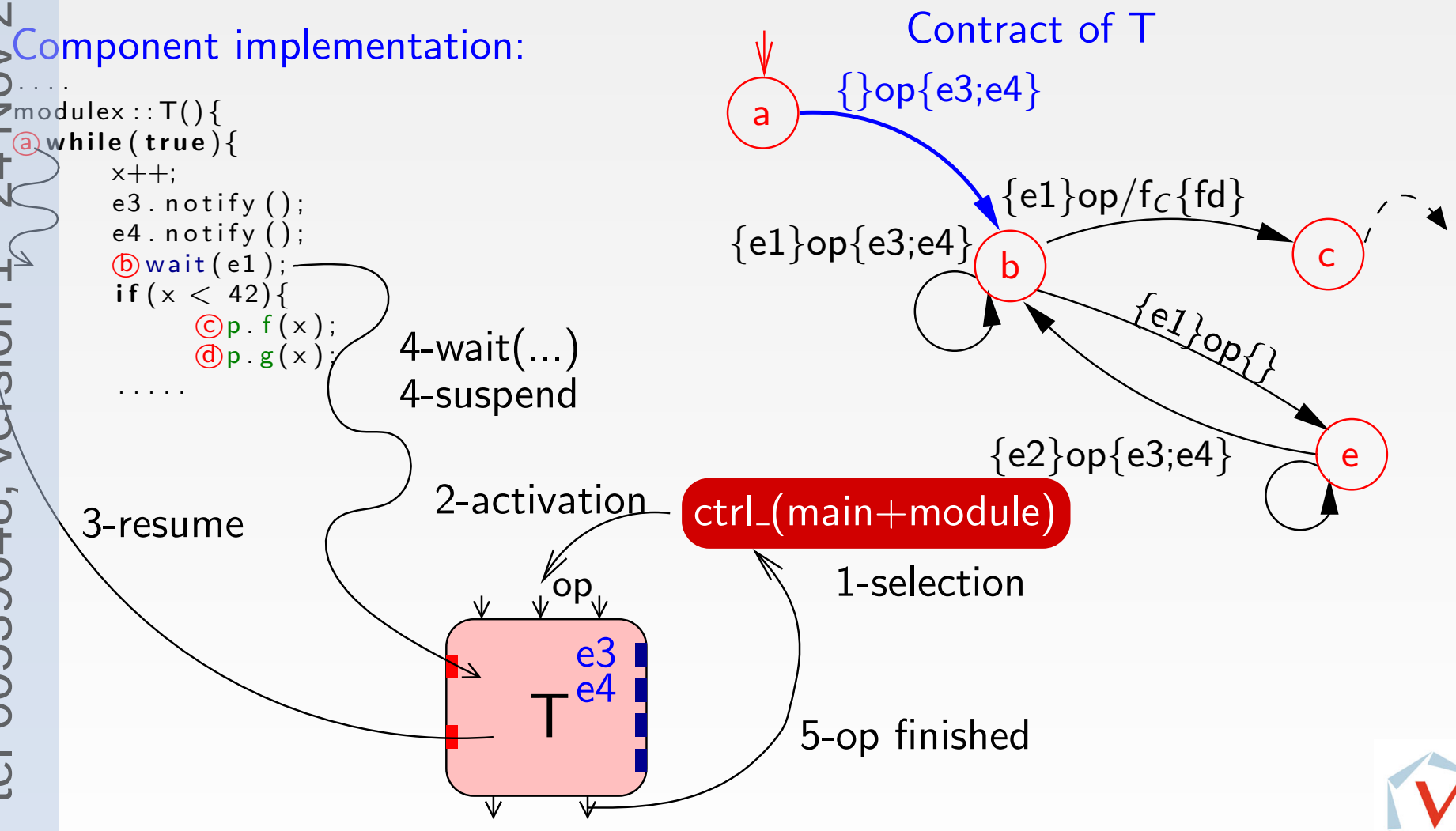


Executing the Implementation against the Contract

tel-00539648, version 1 - 24 Nov 2010

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
        ⓒp.f(x);
        ⓓp.g(x);
    .....
```

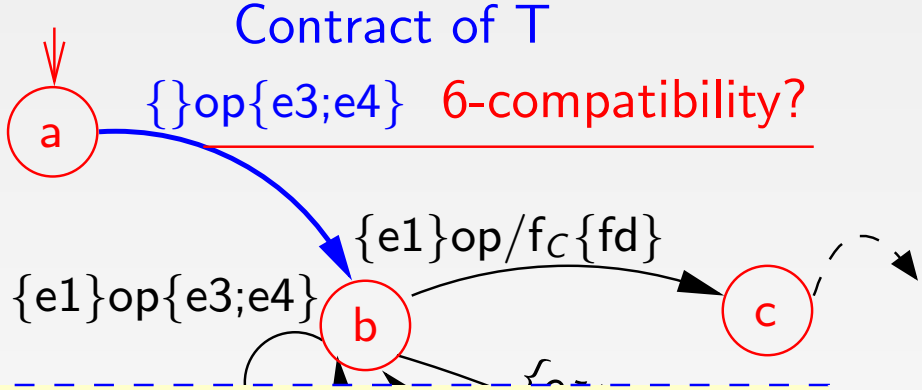


Executing the Implementation against the Contract

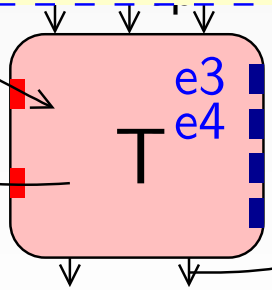
tel-00539648, version 1 - 24 Nov 2010

Component implementation:

```
....
modulex::T(){
ⓐ while(true){
    x++;
    e3.notify();
    e4.notify();
  ⓑ wait(e1);
    if(x < 42){
```



This execution step should notify *e3 and e4* (calls to *e.notify()*). Otherwise, the implementation does not respect the contract.



5-op finished

tel-00539648, version 1 - 24 Nov 2010

Summary

- Use of 42 with an existing approach (TLM)
- Separation of the semantics of models from the execution mechanics
- Support for reasoning on component synchronizations
- A tool for extracting control contracts from SystemC code [Master 1, P. Delahaye, 2010]
- Possibility of connection to verification tools

Contents

1 Introduction & Sources of Inspiration
2 Overview of 42 & Examples [GPCE07]
3 Hardware Simulation by Interpreting Contracts [COORD09]
4 Using 42 Together with Existing Approaches [EMSOFT09]
5 Some Related Work
6 Summary

tel-00539648, version 1 - 24 Nov 2010

Some Related Work

tel-00539648, version 1 - 24 Nov 2010

- Ptolemy — 42
  - Simulation tool for heterogeneous E.S. — Semantics of heterogeneity
  - Catalogue of predefined MoCCs — Programmable MoCCs
  - Modeling discrete/continuous systems — focuses on discrete ones
- Fractal
  - Language independent component-based approach — Similar
  - Dedicated to design and deployment — Virtual Prototyping
  - Does not deal with MoCCs — Central notion
- Formal Models for Embedded Systems
  - Formal Verification $\Longrightarrow$ Reduced expressiveness — Favor expressiveness
  - Describing MoCCs and Heterogeneity with TAG machines — Operational description of MoCCs

Contents

1 Introduction & Sources of Inspiration

2 Overview of 42 & Examples [GPCE07]

3 Hardware Simulation by Interpreting Contracts [COORD09]

4 Using 42 Together with Existing Approaches [EMSOFT09]

5 Some Related Work

6 Summary

tel-00539648, version 1 - 24 Nov 2010

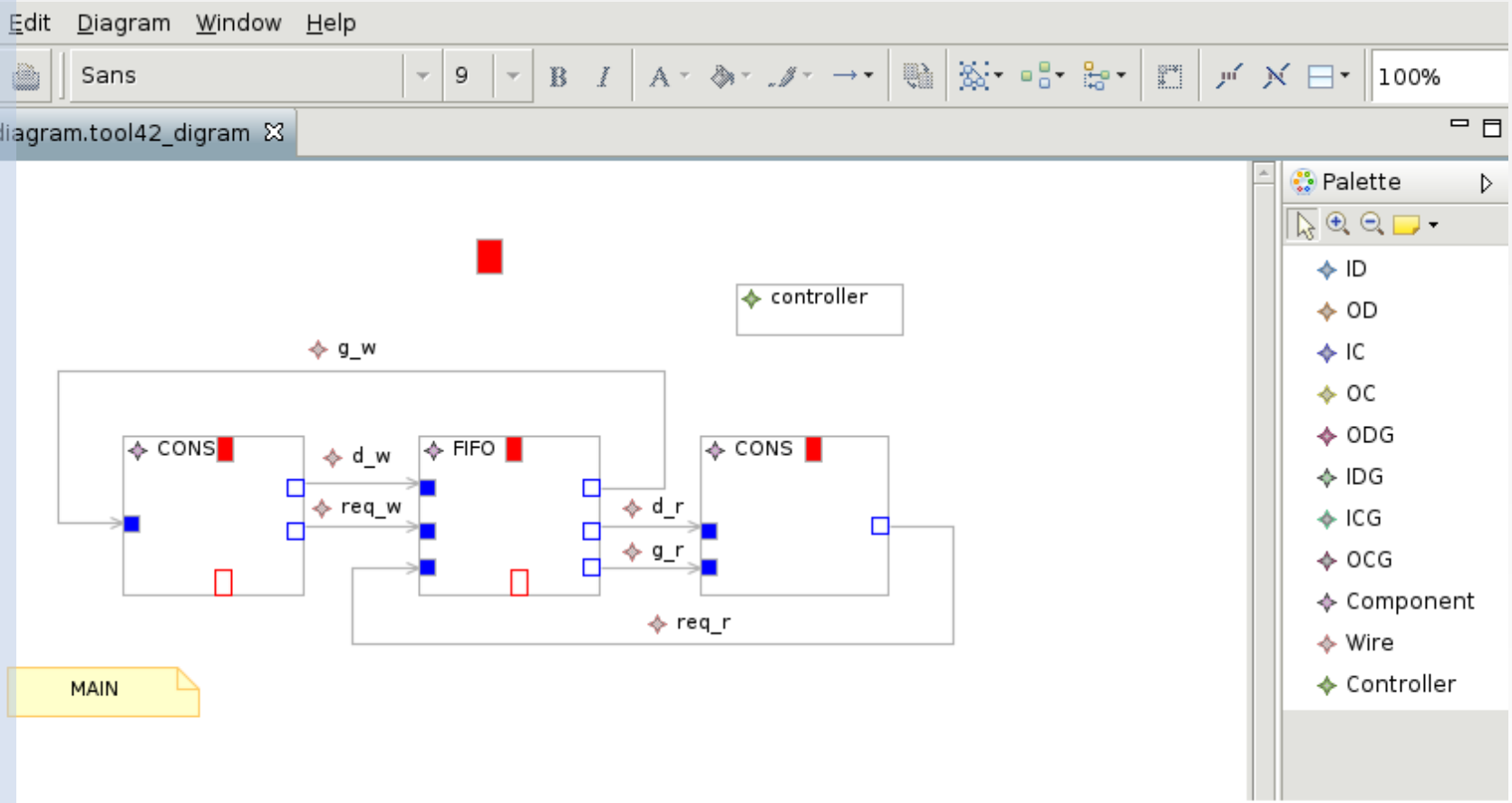
tel-00539648, version 1 - 24 Nov 2010

Contributions

- A language-independent component-based model for heterogeneous embedded systems, and reuse of existing code
- An executable specification language for components
- A rich set of examples for modeling MoCCs and heterogeneity
- A complete case-study on the use of 42 with existing approaches

A tool for describing/executing 42 models

tel-00539648, version 1 - 24 Nov 2010



tel-00539648, version 1 - 24 Nov 2010

Future Work

- Extending the semantics of 42 with a quantitative notion of time
- Investigate on expressiveness and readability of contracts
- Modeling non-functional aspects and their relation with functional models [MOCCs'08]

Communication on 42

tel-00539648, version 1 - 24 Nov 2010

Tayeb Bouhadiba and Florence Maraninchi, *Contract-based coordination of hardware components for the development of embedded software*, COORDINATION, Lecture Notes in Computer Science, vol. 5521, Springer, 2009, pp. 204–224.

Tayeb Bouhadiba, Florence Maraninchi, and Giovanni Funchal, *Formal and executable contracts for transaction-level modeling in systemc*, EMSOFT , ACM, 2009, pp. 97–106.

Florence Maraninchi and Tayeb Bouhadiba, *42: programmable models of computation for a component-based approach to heterogeneous embedded systems*, GPCE, ACM, 2007, pp. 53–62.

Florence Maraninchi and Tayeb Bouhadiba, *42: programmable models of computation for a component-based approach to heterogeneous embedded systems*, In SYNCHRON'07 International Open Workshop on Synchronous Programming.

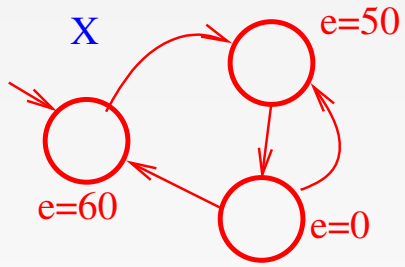
Tayeb Bouhadiba, Florence Maraninchi, Karine Altisen, Matthieu Moy. *Computational Modeling of Non-Functional Properties with the Component Model 42*, Position paper for the ARTIST MoCC'08 Workshop, july 2008, Eindhoven

Computational Modeling of Energy Consumption (Principle)

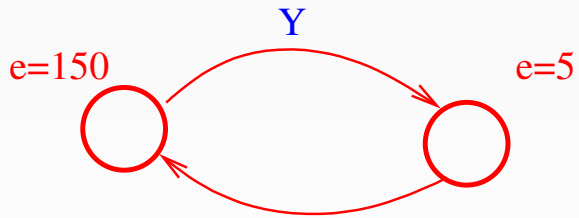
tel-00539648, version 1 - 24 Nov 2010

N–Functional

Radio consumption



CPU consumption

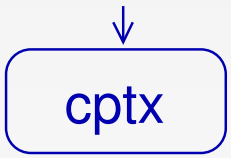


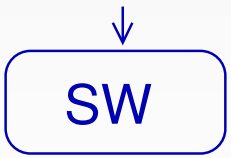
Computational Modeling of Energy Consumption (Principle)

tel-00539648, version 1 - 24 Nov 2010

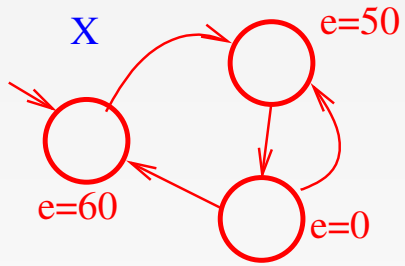
Functional

N–Functional

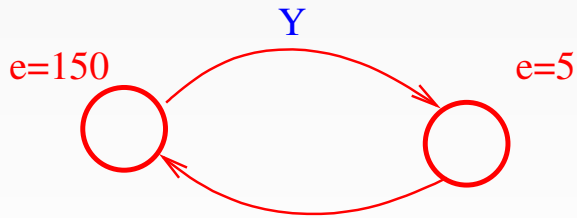




Radio consumption

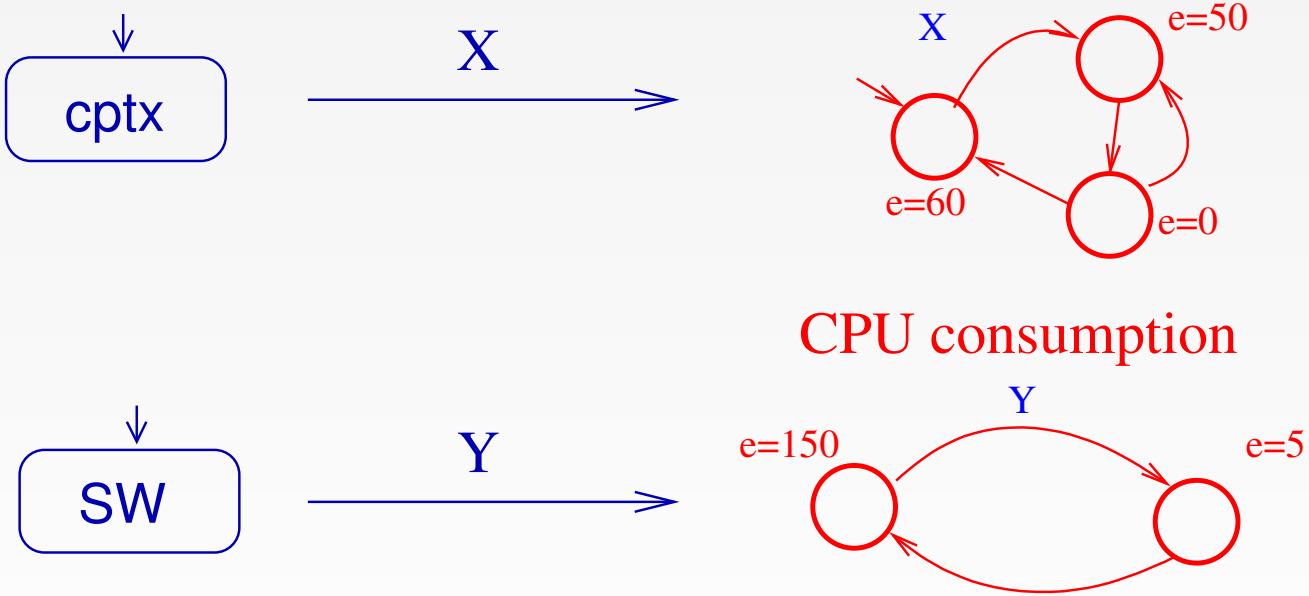


CPU consumption

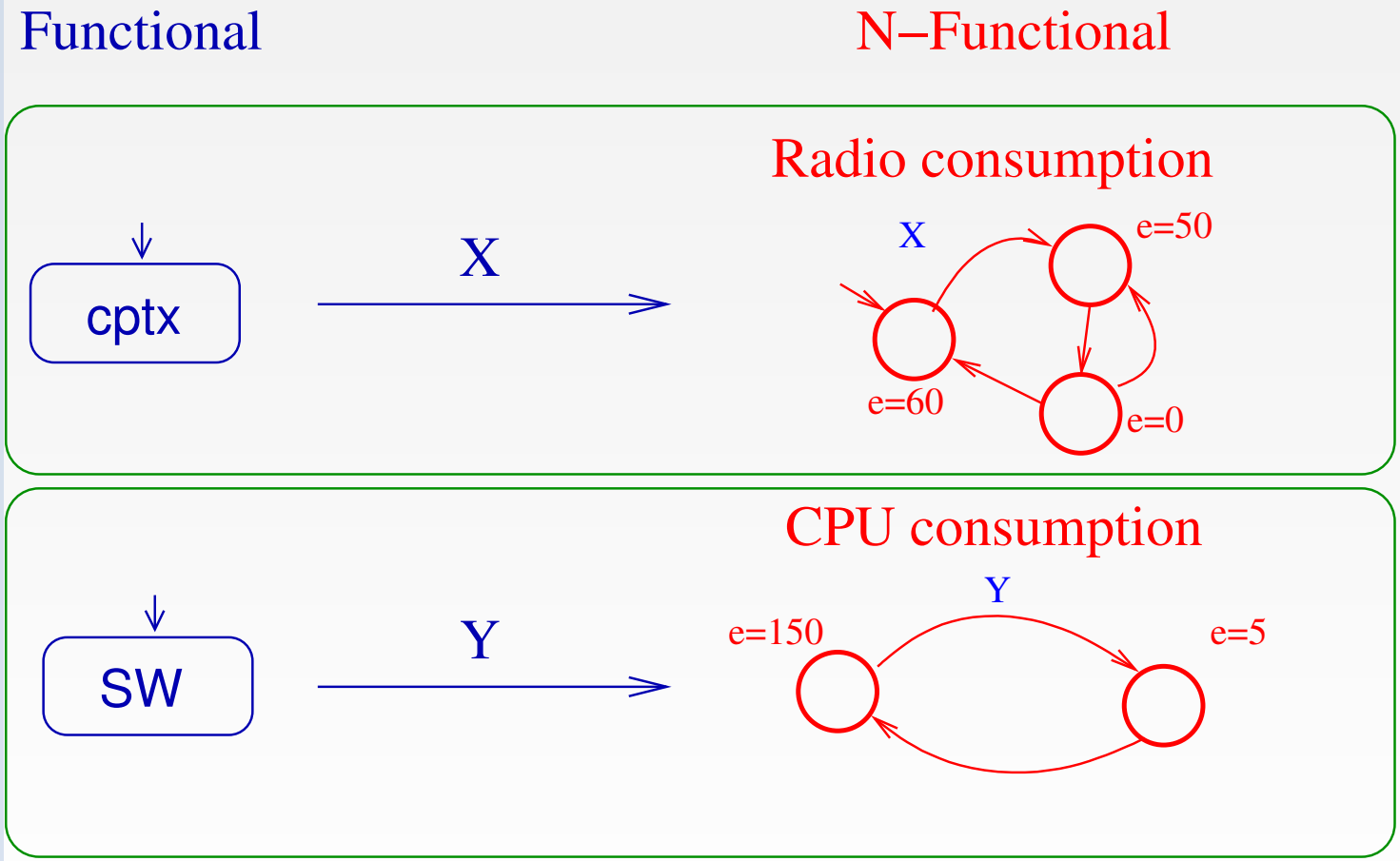


Computational Modeling of Energy Consumption (Principle)

tel-00539648, version 1 - 24 Nov 2010



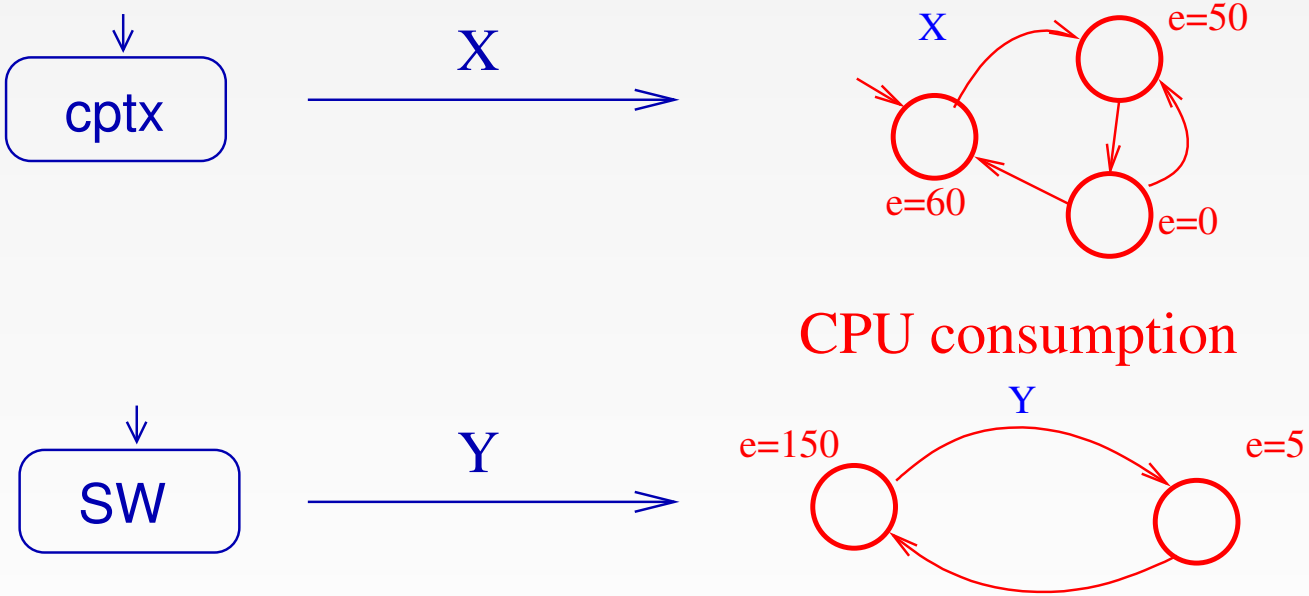
Computational Modeling of Energy Consumption (Principle)



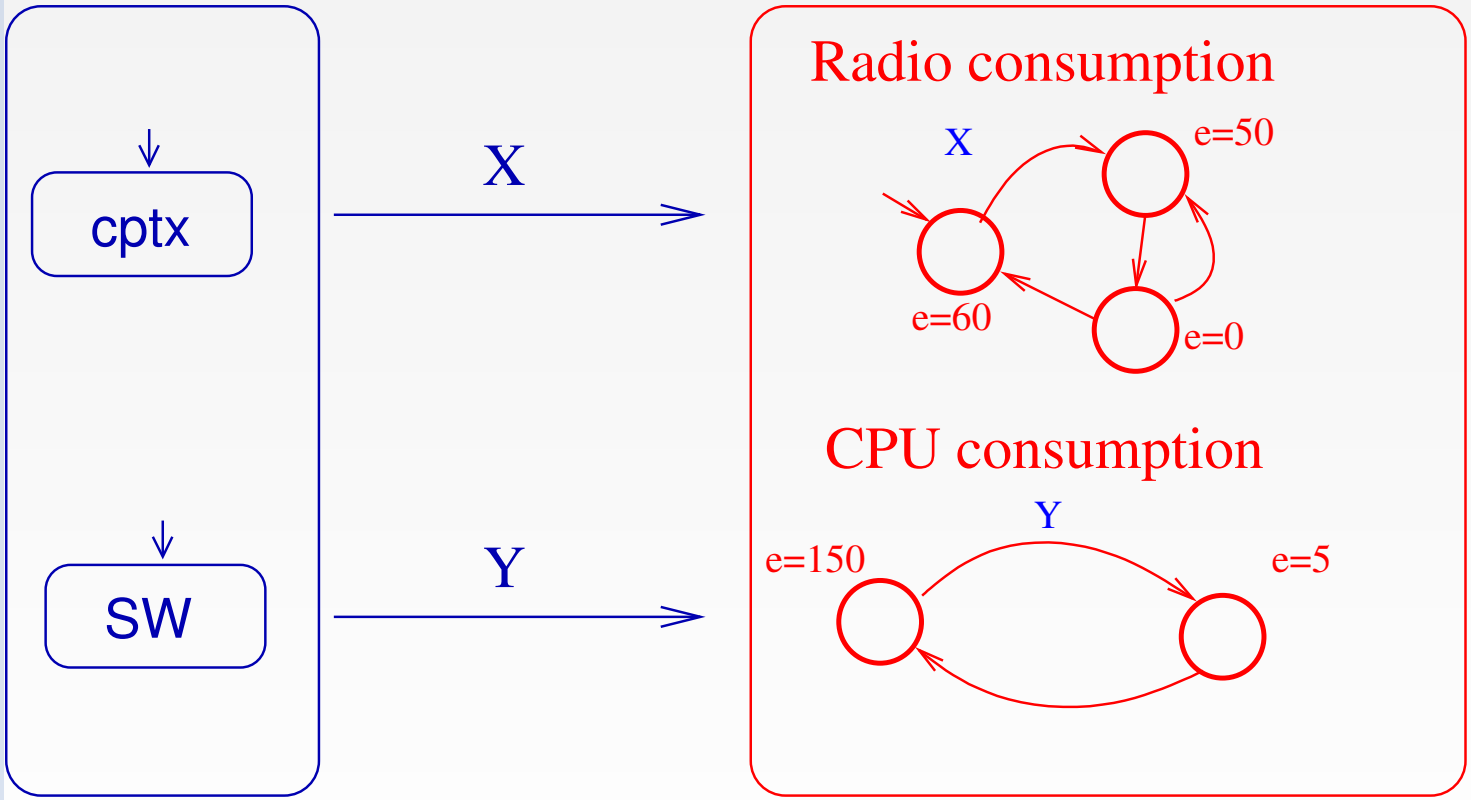
tel-00539648, version 1 - 24 Nov 2010

Computational Modeling of Energy Consumption (Principle)

tel-00539648, version 1 - 24 Nov 2010

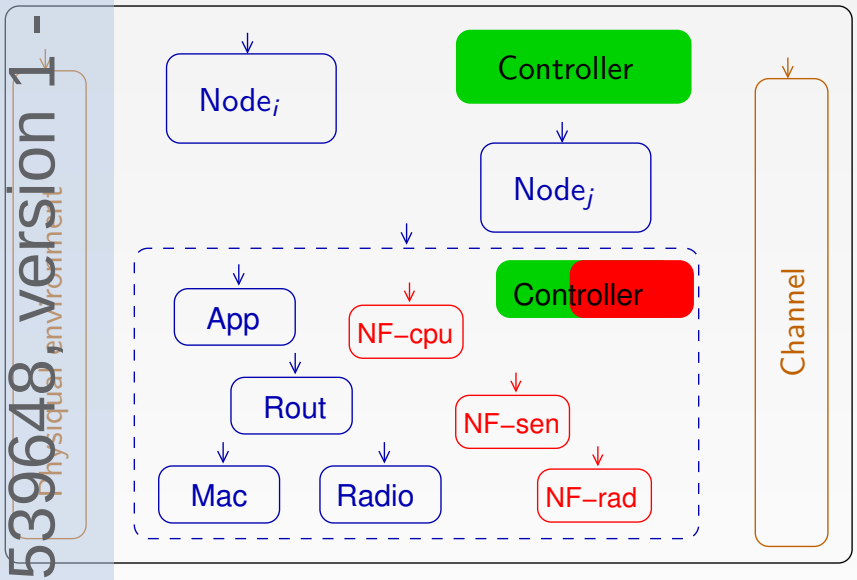


Computational Modeling of Energy Consumption (Principle)



tel-00539648, version 1 - 24 Nov 2010

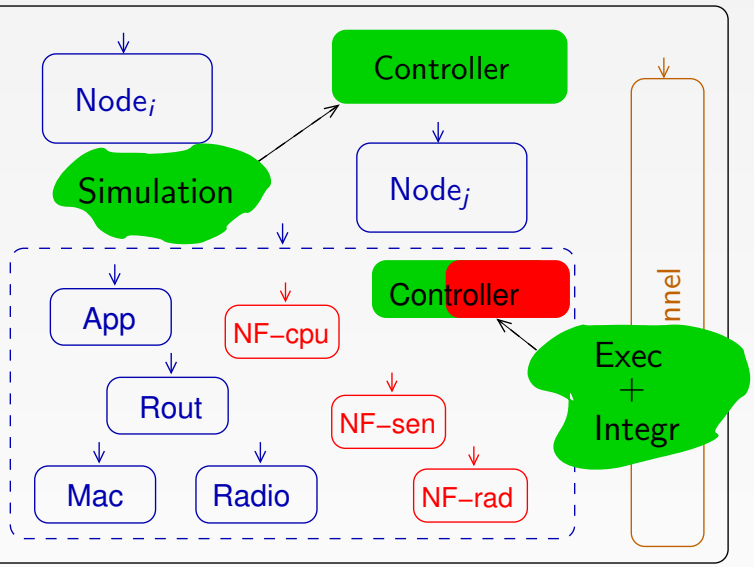
Integrating Functional/Non-Functional Model (option 1)



tel-00539648, version 1 - 24 Nov 2010

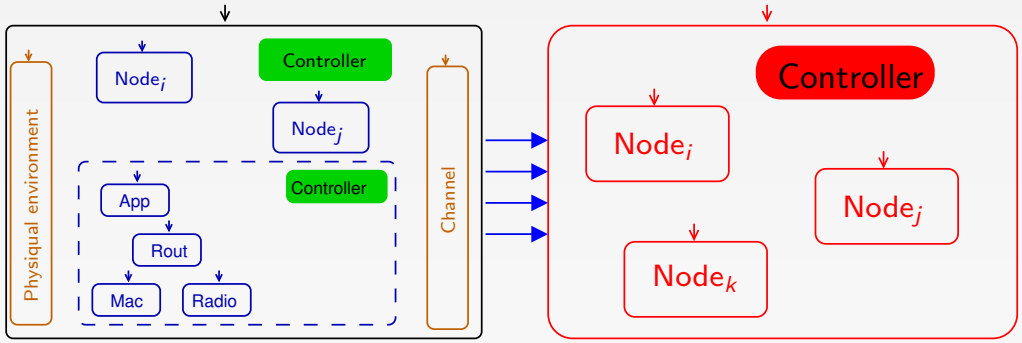
tel-00539648, version 1 - 24 Nov 2010

Integrating Functional/Non-Functional Model (option 1)



- The simulation MoCC is still the same.
- No separation between Non Functional / Functional
- Complicated MoCC.
- Each Component (node) will always be run as Func/N-Func component.

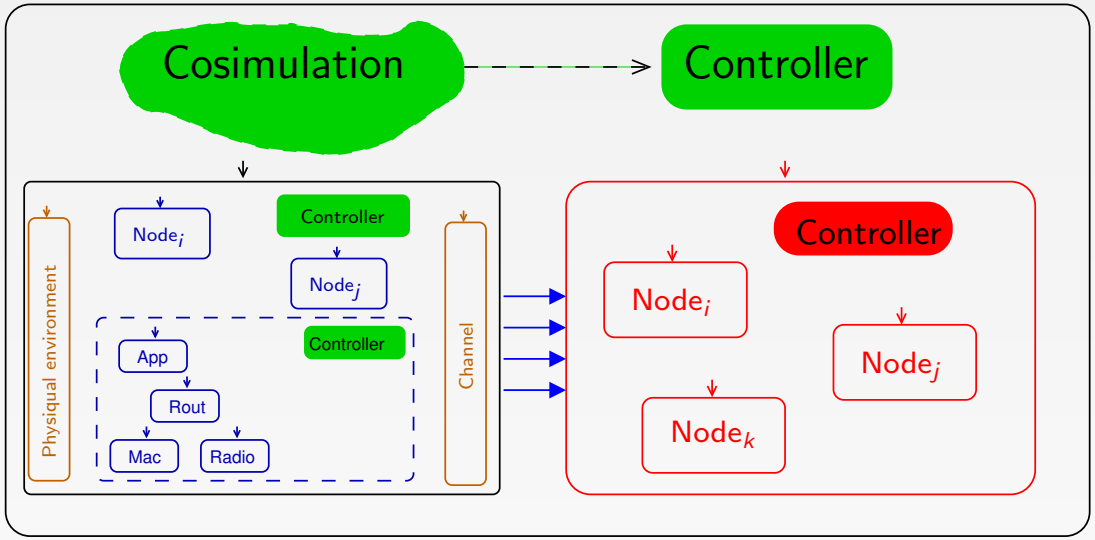
Integrating Functional/Non-Functional Model (option 2)



tel-00539648, version 1 - 24 Nov 2010

tel-00539648, version 1 - 24 Nov 2010

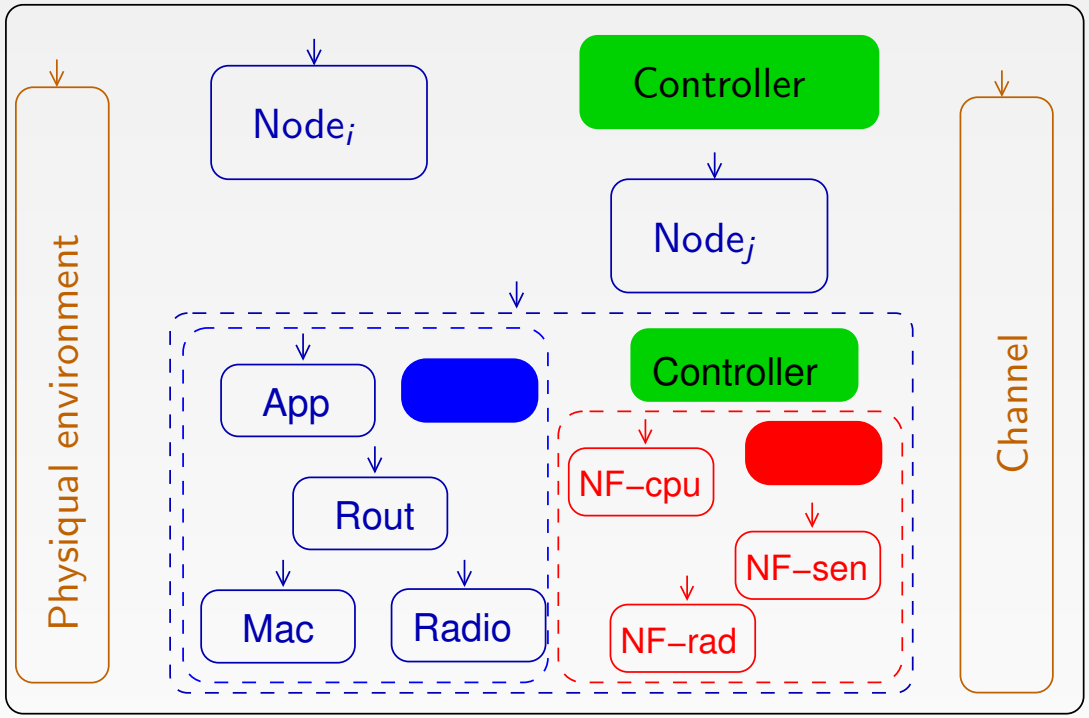
Integrating Functional/Non-Functional Model (option 2)



- Very clear separation Func/N-Func.
- Duplication of hierarchy (hard to maintain).
- To many wires to manage.
- Would be well suited for other applications.

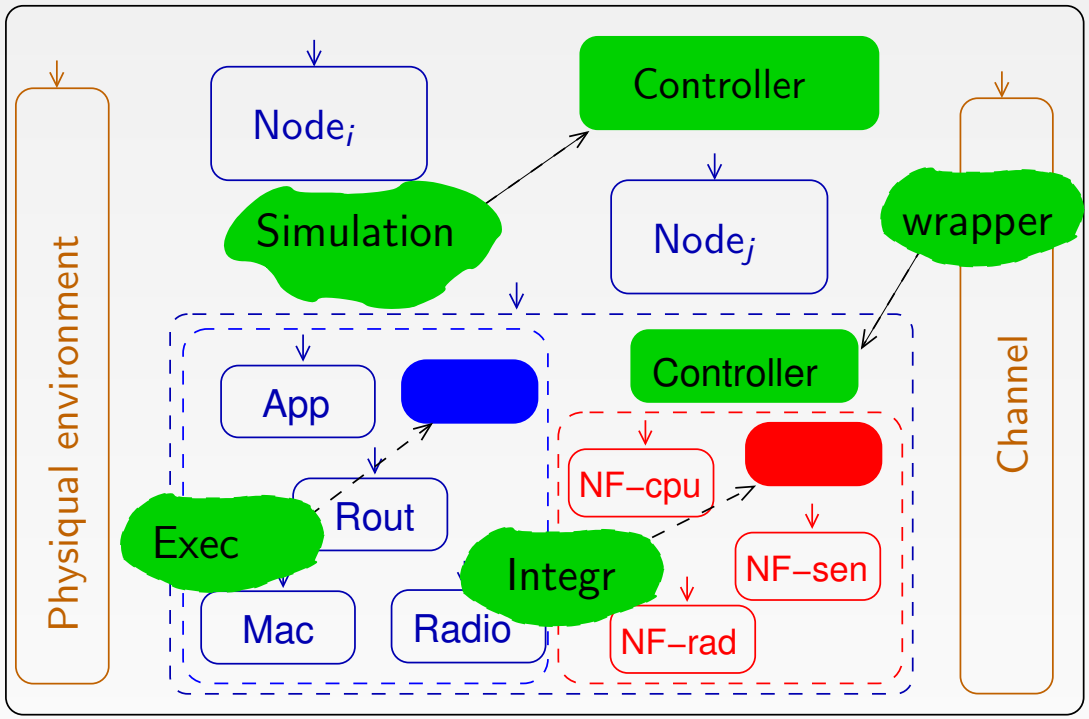
tel-00539648, version 1 - 24 Nov 2010

Integrating the Functional/Non-Functional Model (option 3)



tel-00539648, version 1 - 24 Nov 2010

Integrating the Functional/Non-Functional Model (option 3)



- Identification of Func/N-Func models.
- Functional models still unchanged.
- This structuring is well suited for WSN applications.
- Not adequate for applications where Energy models share resources.