Contract Based Coordination of Hardware Components for the Development of Embedded Software

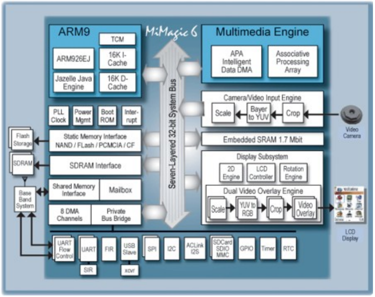
Tayeb Bouhadiba & Florence Maraninchi

VERIMAG

Grenoble INP ensimag

09-12 June 2009

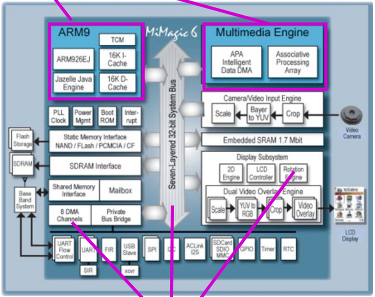
Embedded Software For Systems-on-a-Chip



Hardware

Embedded Software For Systems-on-a-Chip

Processors

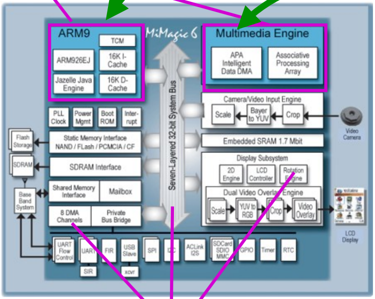


Hardware

Hardware IP's
(components)

Embedded Software For Systems-on-a-Chip

Processors



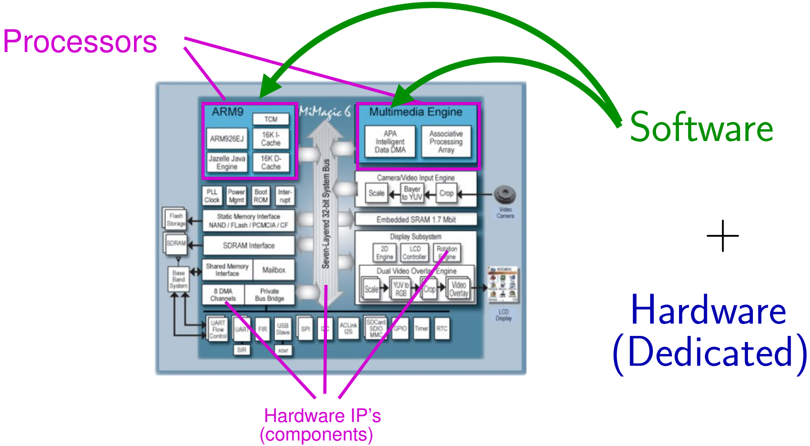
Hardware IP's
(components)

Software

+

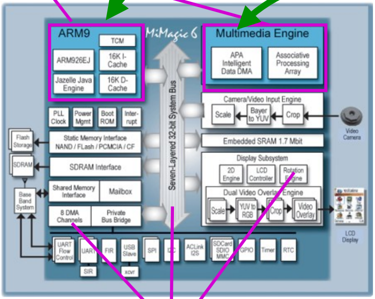
Hardware

Embedded Software For Systems-on-a-Chip



Embedded Software For Systems-on-a-Chip

Processors



Hardware IP's
(components)

Software

+

Hardware
(Dedicated)

⇒ Need for Virtual Prototypes of the Hardware
(Virtual Prototypes == Executable Models)

Executable HW Models for SW Development

- RTL (Register Transfer Level) Model
  - Accurate, Late availability, Low co-simulation speed

Executable HW Models for SW Development

- RTL (Register Transfer Level) Model
  - Accurate, Late availability, Low co-simulation speed
  - need to speed up simulation

Executable HW Models for SW Development

- RTL (Register Transfer Level) Model
  - Accurate, Late availability, Low co-simulation speed
  - need to speed up simulation
- High Level Models
  - Early available, Fast simulation, Abstract

Executable HW Models for SW Development

- RTL (Register Transfer Level) Model
  - Accurate, Late availability, Low co-simulation speed
  - need to speed up simulation
- High Level Models
  - Early available, Fast simulation, Abstract
  - SystemC-TLM is a standard in the industry (TLM: Transaction Level Modeling)

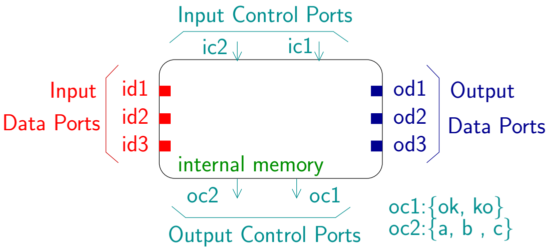
Motivation of the work

- Component-based Virtual Prototyping of Hardware
- As Abstract As Possible
- Software Execution on the Simulated Hardware
- Inspired From SystemC-TLM but Formal and Language-Independent (Using the 42 Component Model [GPCE'07])

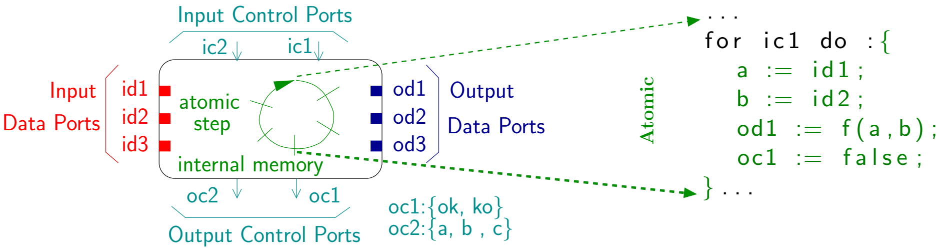
Content

1. Introduction & Motivations
2. Overview of the 42 Component Model
3. Contribution 1 :
   - Control Contracts
   - Executing Contracts (Alone)
4. Contribution 2 :
   - Modeling Hardware with 42
   - Executing Embedded Software on Hardware Models
5. Conclusion and Perspectives

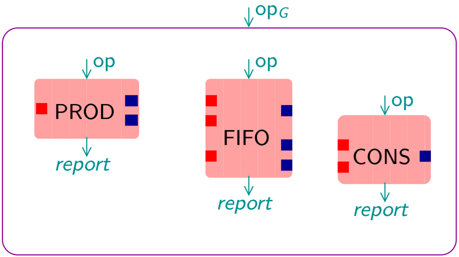
42 in a Nutshell: Basic Components



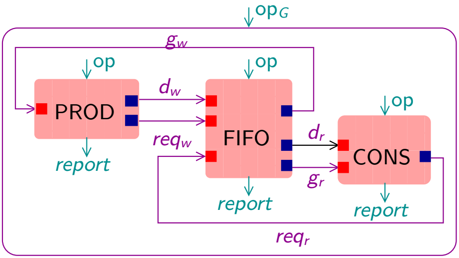
42 in a Nutshell: Basic Components



42 in a Nutshell: Assembling Components

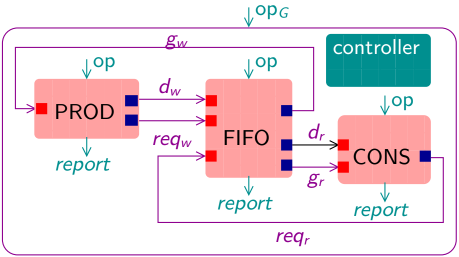


42 in a Nutshell: Assembling Components



42 in a Nutshell: Assembling Components

For each OP_G the controller:

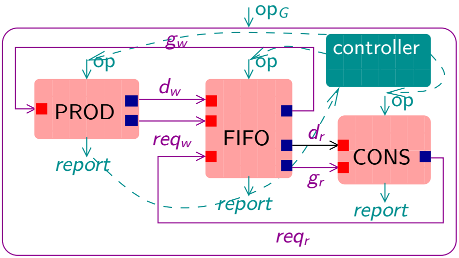


42 in a Nutshell: Assembling Components

For each OP_G the controller:

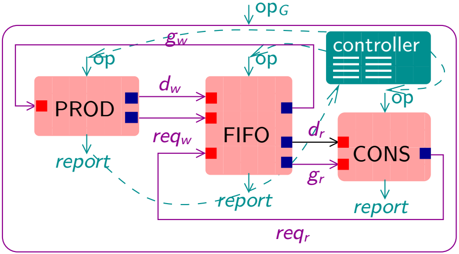
Activates PROD, CONS, FIFO through op

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, reqw, dw...)



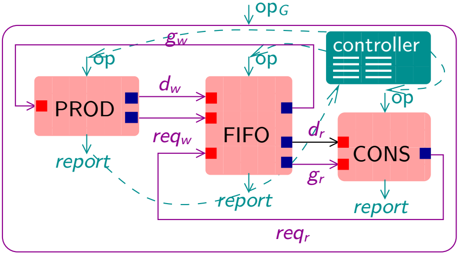
42 in a Nutshell: Assembling Components

For each OP_G the controller:

Activates PROD, CONS, FIFO through op

Reads their output control ports (report)

Manages a temporary memory (reqr, dr, reqw, dw...)

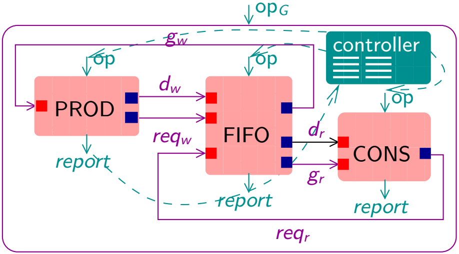


42 in a Nutshell: Assembling Components

```
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
          if(a==ok){     output control
              PROD.op;dw.put; dw.get;
              FIFO.op;/* activate 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, reqw, dw...)



Contents

1 Introduction & Motivations

2 Overview of the 42 Component Model

3 Contribution 1 :
- Control Contracts
- Executing Contracts (Alone)

4 Contribution 2 :
- Modeling Hardware with 42
- Executing Embedded Software on Hardware Models

5 Conclusion and Perspectives

Contents

1 Introduction & Motivations

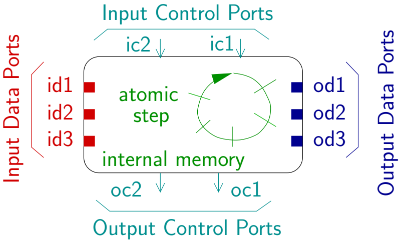
2 Overview of the 42 Component Model

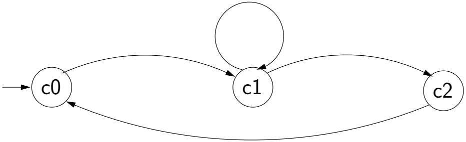
3 Contribution 1 :
- Control Contracts
- Executing Contracts (Alone)

4 Contribution 2 :
- Modeling Hardware with 42
- Executing Embedded Software on Hardware Models

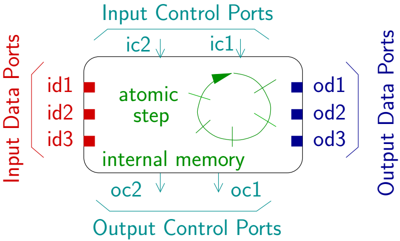
5 Conclusion and Perspectives

Control Contracts for 42 Components

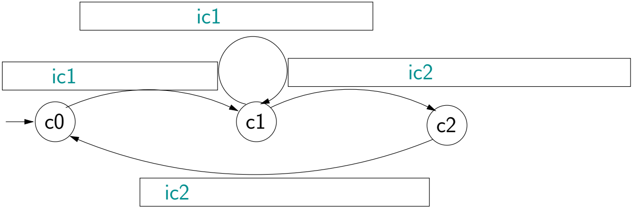




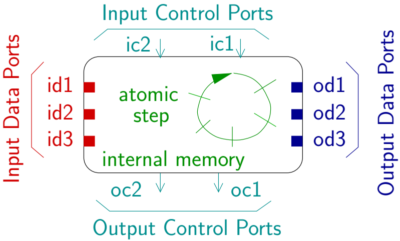
Control Contracts for 42 Components



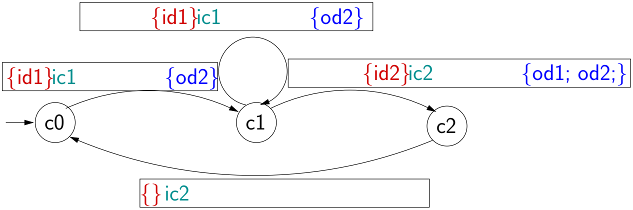
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state



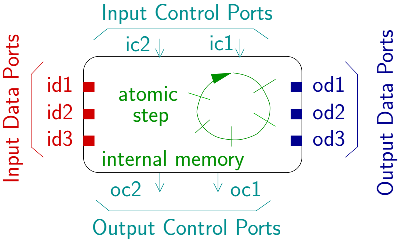
Control Contracts for 42 Components



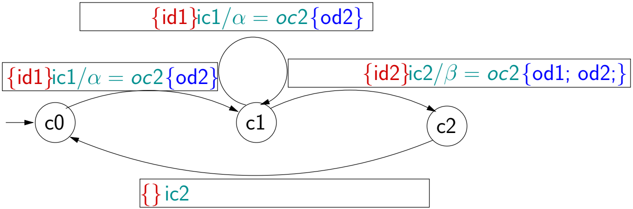
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state
- Data dependencies (Required, Provided)
  - Don't care about values except for particular data ports



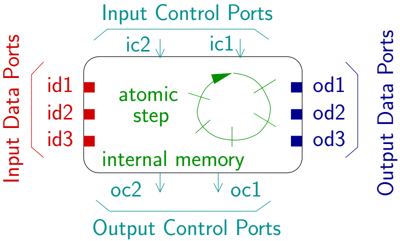
Control Contracts for 42 Components



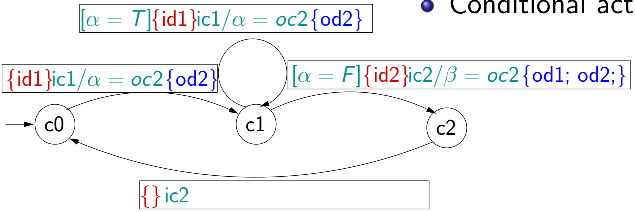
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state
- Data dependencies (Required, Provided)
  - Don't care about values except for particular data ports
- Control information



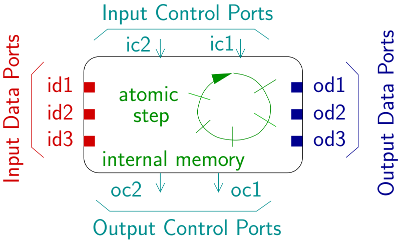
Control Contracts for 42 Components



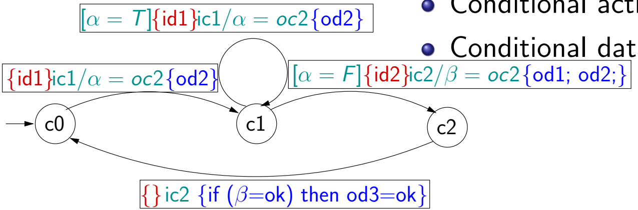
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state
- Data dependencies (Required, Provided)
  - Don't care about values except for particular data ports
- Control information
- Conditional activations.



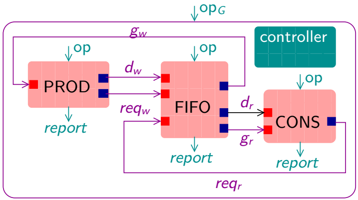
Control Contracts for 42 Components



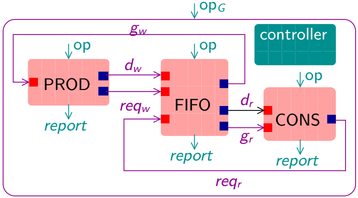
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state
- Data dependencies (Required, Provided)
  - Don't care about values except for particular data ports
- Control information
- Conditional activations.
- Conditional data dependencies.

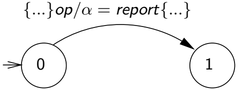


42 Contracts: Example



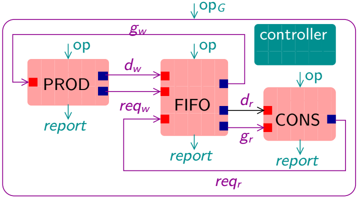
42 Contracts: Example

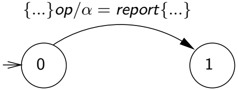




General form of transitions

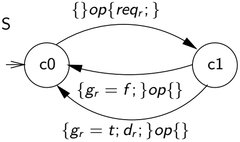
42 Contracts: Example



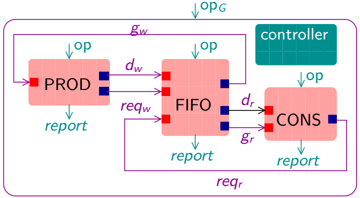


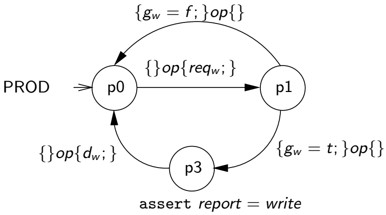
General form of transitions

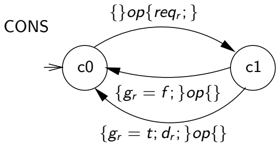
CONS

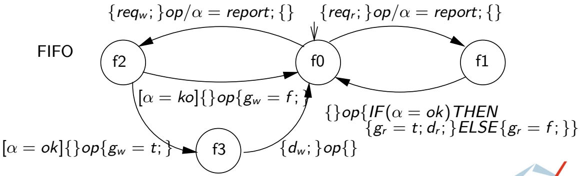


42 Contracts: Example









Contents

1 Introduction & Motivations

2 Overview of the 42 Component Model

3 Contribution 1 :
- Control Contracts
- Executing Contracts (Alone)

4 Contribution 2 :
- Modeling Hardware with 42
- Executing Embedded Software on Hardware Models

5 Conclusion and Perspectives

Controllers as Contracts Interpreters

The controller maintains:

- The current state of each contract

p0,f0,c0

Controllers as Contracts Interpreters

The controller maintains:

- The current state of each contract
- The set of available data

p0,f0,c0
$\mathit{available} = \{\}$

Controllers as Contracts Interpreters

The controller maintains:

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

p0,f0,c0
$\mathit{available} = \{\}$
$\alpha = \mathit{null}$

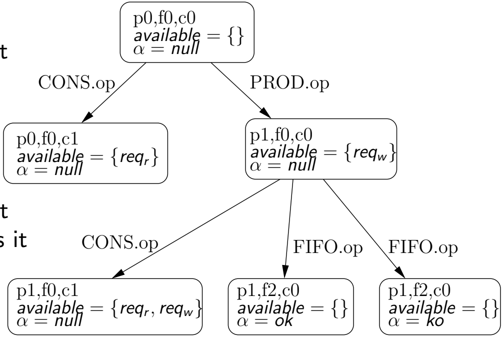
Controllers as Contracts Interpreters

The controller maintains:

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

For each global activation :

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



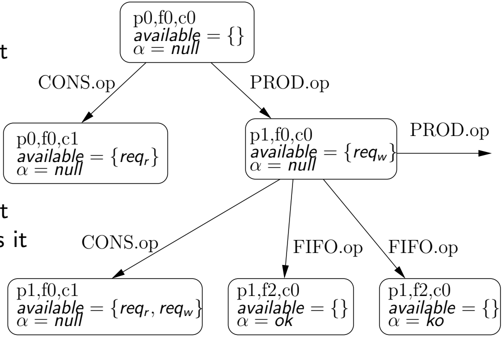
Controllers as Contracts Interpreters

The controller maintains:

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

For each global activation :

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



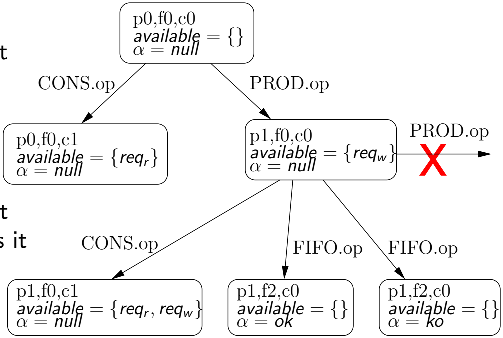
Controllers as Contracts Interpreters

The controller maintains:

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

For each global activation :

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



Summary

- Contracts are non-deterministic abstractions of the components' behaviors and allow for early execution
- Contracts interpreters expose the possible interleavings of the components' behaviors

Contents

1. Introduction & Motivations
2. Overview of the 42 Component Model
3. Contribution 1 :
   - Control Contracts
   - Executing Contracts (Alone)
4. Contribution 2 :
   - Modeling Hardware with 42
   - Executing Embedded Software on Hardware Models
5. Conclusion and Perspectives

Contents

1 Introduction & Motivations

2 Overview of the 42 Component Model

3 Contribution 1 :
- Control Contracts
- Executing Contracts (Alone)

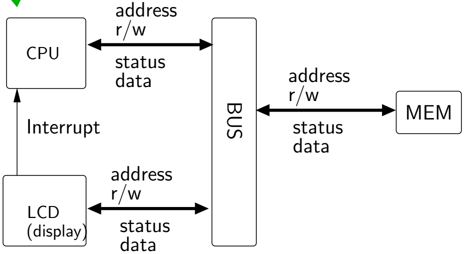
4 Contribution 2 :
- Modeling Hardware with 42
- Executing Embedded Software on Hardware Models

5 Conclusion and Perspectives

Modeling Hardware with 42: Case Study

Embedded Software

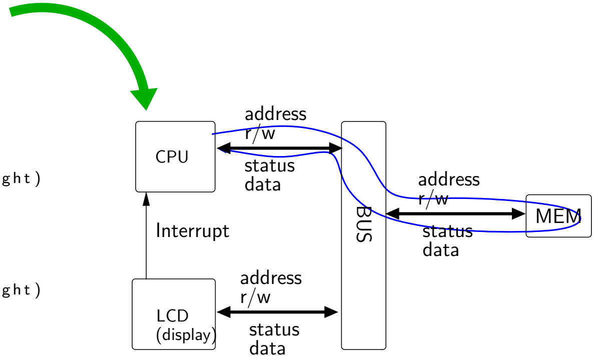
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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

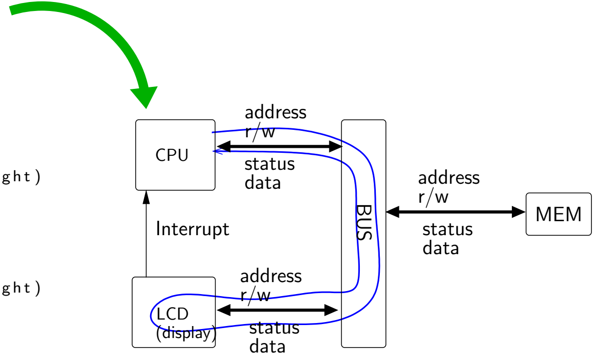
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
      write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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();
}
}
```



Modeling Hardware with 42: Case Study

Embedded Software

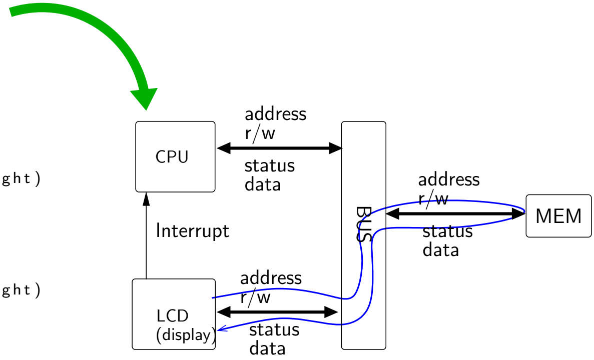
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
       write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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

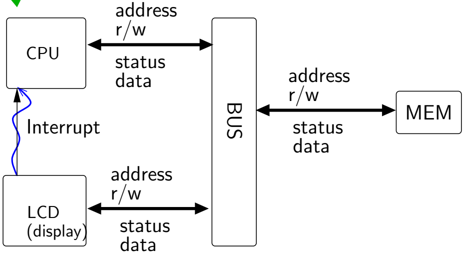
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
      write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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();
}
}
```



Modeling Hardware with 42: Case Study

Embedded Software

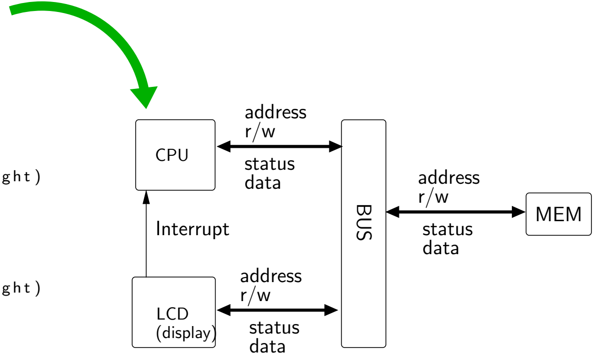
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
      write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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();
}
}
```

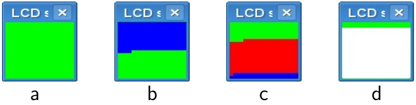


Modeling Hardware with 42: Case Study

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();
}
}
```

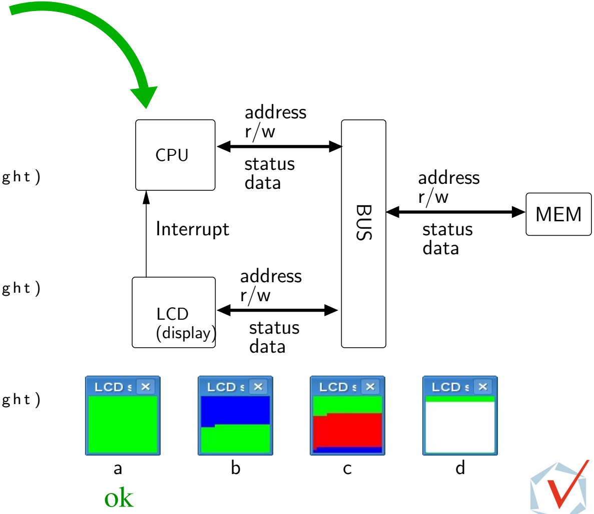




Modeling Hardware with 42: Case Study

Embedded Software

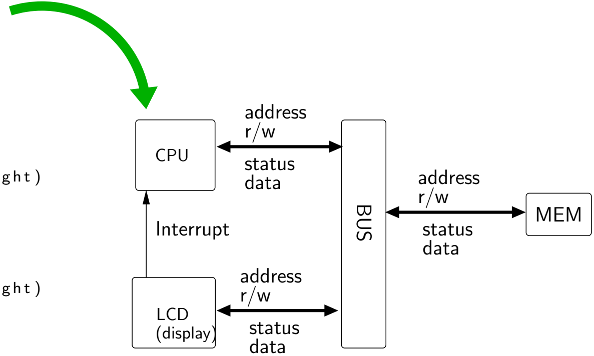
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
      write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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();
}
}
```

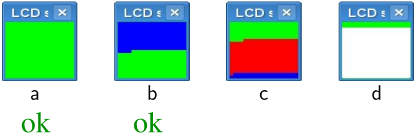


Modeling Hardware with 42: Case Study

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();
}
}
```

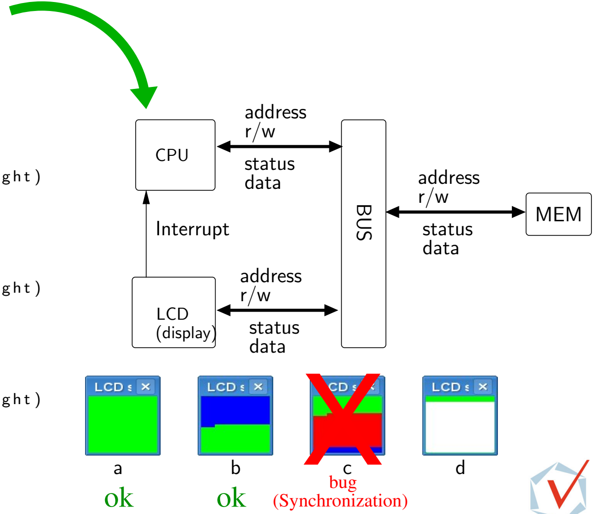




Modeling Hardware with 42: Case Study

Embedded Software

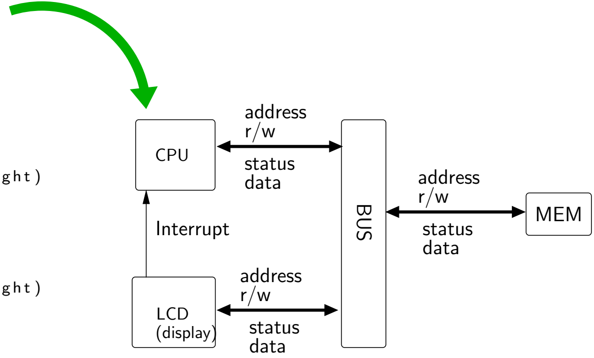
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
      write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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();
}
}
```

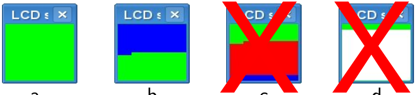


Modeling Hardware with 42: Case Study

Embedded Software

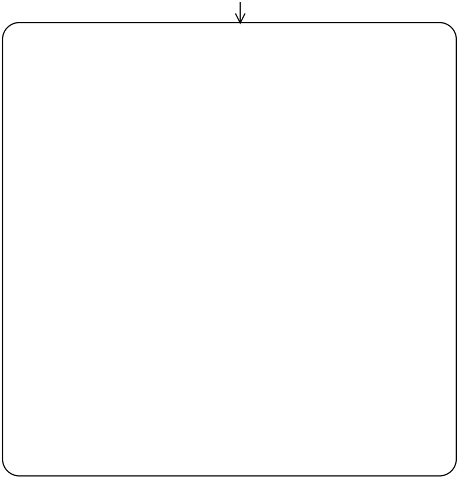
```
#define width 240
#define height 240
#define green 0x0000AABB
...
int main() {
while(1) {
//*writing the green image*/
   for(int x=0; x<width * height)
      write_mem(green);
   write_lcd(0x01,0x1) ;
   wait_interrupt();

//*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();
}
}
```





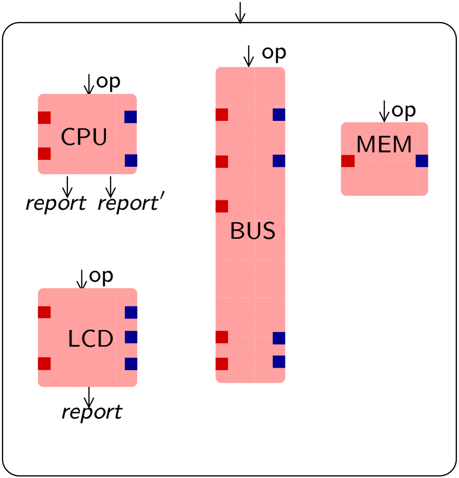
a ok | b ok | c bug (Synchronization) | d bug (Data)

Modeling Hardware with 42: The 42 Model



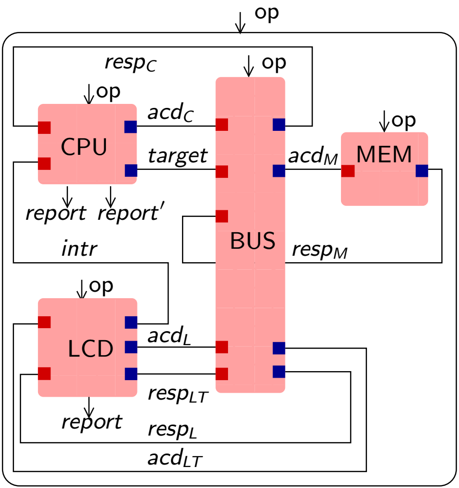
- The Chip is modeled as a 42 component

Modeling Hardware with 42: The 42 Model



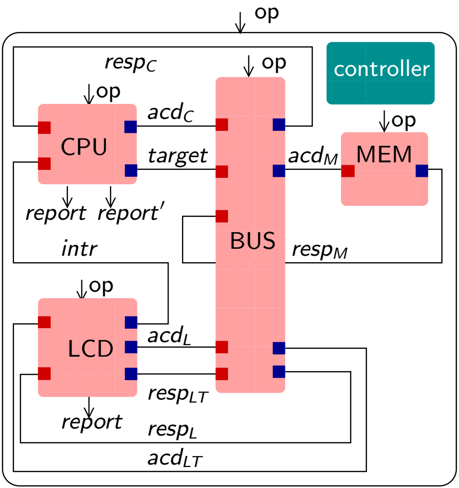
- The Chip is modeled as a 42 component
- Each HW component too

Modeling Hardware with 42: The 42 Model



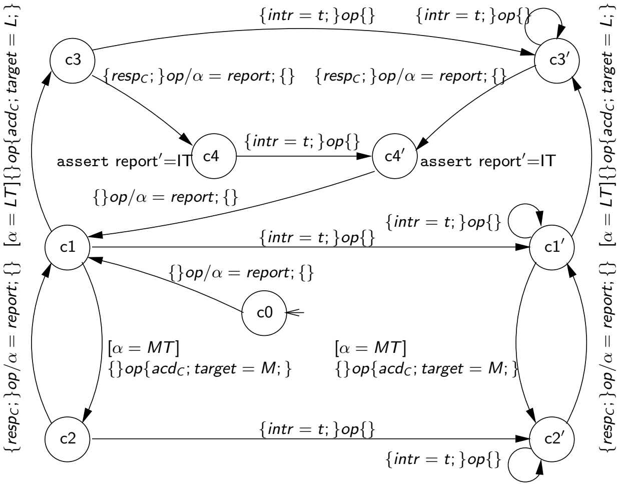
- The Chip is modeled as a 42 component
- Each HW component too
- Connections correspond to HW communication wires

Modeling Hardware with 42: The 42 Model



- The Chip is modeled as a 42 component
- Each HW component too
- Connections correspond to HW communication wires
- The controller is a contract interpreter

Modeling Hardware with 42: The contracts (CPU)



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

Detecting Synchronization Bugs by Executing contracts

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

- Example Bug: The software doesn't wait for interrupts
- Consequence:

- Detection: Deadlock during contracts interpretation

Contents

1 Introduction & Motivations

2 Overview of the 42 Component Model

3 Contribution 1 :
- Control Contracts
- Executing Contracts (Alone)

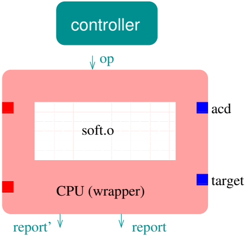
4 Contribution 2 :
- Modeling Hardware with 42
- Executing Embedded Software on Hardware Models

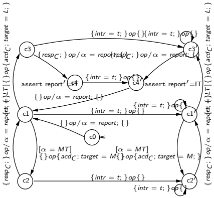
5 Conclusion and Perspectives

Executing the Real Software on HW Models

software wrapper controller

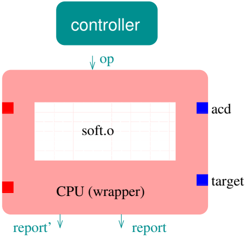
```
(a) int main(){
      while (true){
        int x = ....
        while(x>0){
          x - -;
          ...
(b)       write_mem(adr, data);
        }
        ...
        ...
(d)     write_lcd(adr2, data2);
        if(y!=0)
(e)     write_mem(adr, data)
        ...
      }
    }
```

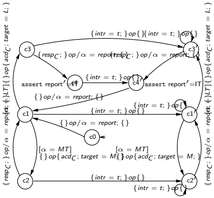




Executing the Real Software on HW Models

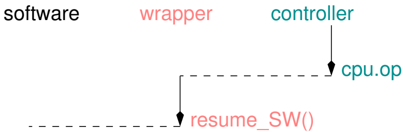
```
(a) int main(){
      while (true){
        int x = ....
        while(x>0){
          x - -;
          ...
(b)       write_mem(adr, data);
        }
        ...
        ...
(d)     write_lcd(adr2, data2);
        if(y!=0)
(e)     write_mem(adr, data)
        ...
      }
    }
```

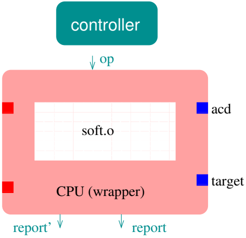


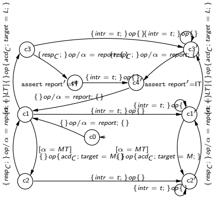


Executing the Real Software on HW Models

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



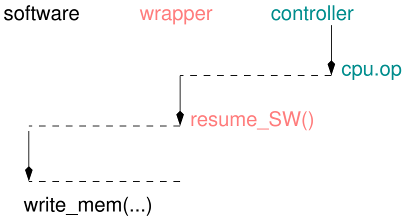


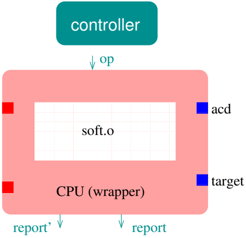


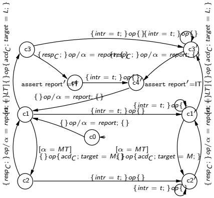
Executing the Real Software on HW Models

atomic

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



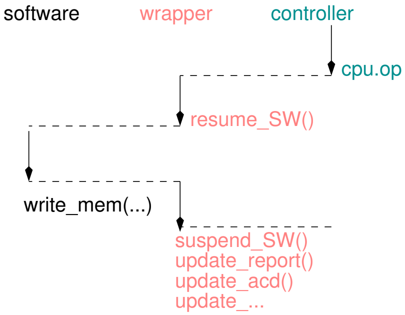


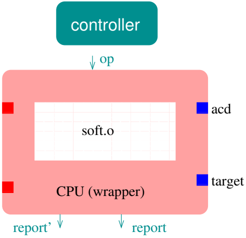


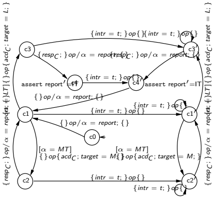
Executing the Real Software on HW Models

atomic

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



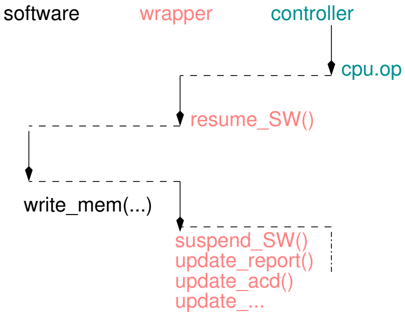


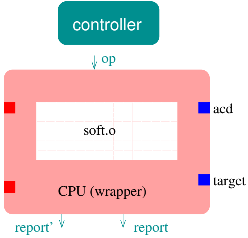


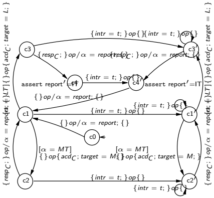
Executing the Real Software on HW Models

atomic

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







Detecting Data Bugs

The real software is executed with the contracts together with the rest of the components' implementations.

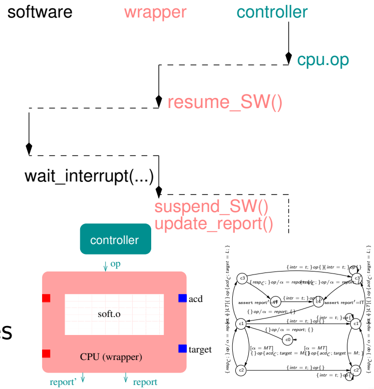
- Example bug: Display dimensions are not correct
- Consequence:

- Detection: Visualizations

Compatibility of the Software with the Contracts

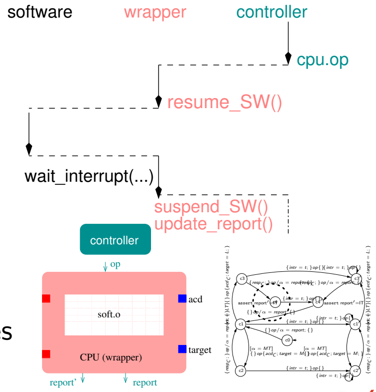
We use the contracts as monitors, the wrapper reports on the activity of the software through control output.

- Example bug: The software doesn't call wait_interrupt() when expected
- Consequence: An unexpected value of output control values
- Detection: Assertions on output control ports values associated with contracts states



Compatibility of the Software with the Contracts

We use the contracts as monitors, the wrapper reports on the activity of the software through control output.

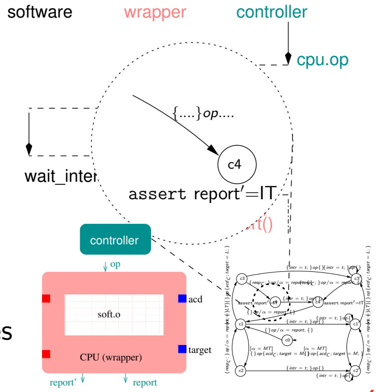


- Example bug: The software doesn't call wait_interrupt() when expected
- Consequence: An unexpected value of output control values
- Detection: Assertions on output control ports values associated with contracts states

Compatibility of the Software with the Contracts

We use the contracts as monitors, the wrapper reports on the activity of the software through control output.

- Example bug: The software doesn't call wait_interrupt() when expected
- Consequence: An unexpected value of output control values
- Detection: Assertions on output control ports values associated with contracts states



Contents

1. Introduction & Motivations
2. Overview of the 42 Component Model
3. Contribution 1 :
   - Control Contracts
   - Executing Contracts (Alone)
4. Contribution 2 :
   - Modeling Hardware with 42
   - Executing Embedded Software on Hardware Models
5. Conclusion and Perspectives

Conclusion

We defined executable contracts for the modeling of hardware components.
It is SystemC-TLM like approach but formal and language-independent.

Typical uses of the approach:

- Modeling the architecture with 42 components
- Executing the contracts alone to detect synchronization bugs
- Implementing the software
- Executing SW+contracts to check compatibility
- Debug and detect data-related bugs

Perspectives

- Adapting the approach to the SystemC-TLM (Submitted to EmSoft09)
- Mixing 42 Contracts and SystemC components
- Extensions to hierarchical contracts interpreters

Questions?