Formal support for Untimed SystemC Specifications: Application to High-Level Synthesis

Eugenio Villar
Fernando Herrera
Víctor Fernández
Index

• Motivation
• Introduction
• Formal Framework
  – Application to High-Level Synthesis Verification
• Conclusions
Motivation

- SystemC applications

System specification
  Functional validation

SystemC Specification
  Architectural design

Design verification
  Performance analysis
  Design-Space exploration
  Behavioral synthesis

Legacy
  SW functionality

Executive Platform
  SW compilation

HW functionality
  HW behavioral synthesis

Target binary
  Synthesizable HW

Executive Platform

DE-TLM

DT
Motivation

• Timing transformations
  – Source of design errors
  – Classical problem in concurrent programming
  – Not enough addressed in system (HW/SW) design

System simulation
Motivation

• Timing transformations
  – Source of design errors
  – Classical problem in concurrent programming
  – Not enough addressed in system (HW/SW) design

TLM simulation
Motivation

- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design

Cycle simulation
Motivation

• Timing transformations
  – Source of design errors
  – Classical problem in concurrent programming
  – Not enough addressed in system (HW/SW) design

![System simulation diagram]
Motivation

- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in SystemC (HW/SW) design

System simulation

TLM simulation
Motivation

• Timing transformations
  – Source of design errors
  – Classical problem in concurrent programming
  – Not enough addressed in system (HW/SW) design

System simulation
TLM simulation
Motivation

- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design

System simulation
TLM simulation
Motivation

• Timing transformations
  – Source of design errors
  – Classical problem in concurrent programming
  – Not enough addressed in system (HW/SW) design

System simulation
TLM simulation
Cycle simulation
Motivation

• Need for formal support
  – SystemC-based design
  – Timing transformations
    • From Untimed Specifications
    • Down to TLM and Cycle-accurate implementations
Introduction

- ForSyDe
  - Formal System Design meta-model from KTH
  - Formal definition of
    - Untimed MoCs
    - Synchronous MoCs
    - Timed MoCs
Formal Framework

- Extraction of the SystemC C&C model
  - Removal of hierarchical facilities
  - Flat model of communicating concurrent processes
Formal Framework

- Abstraction of the formal model

![Diagram showing the abstraction process from SystemC concurrency and communication model to ForSyDE formal model, finally to SystemC untimed specification.]
SC_MODULE(Simple_Example) {
  . Port declarations
  . Channel/submodule instances
  Word16 lsp[10], a[11];
  int initial_position=0;
  Word32 f1[6], f2[6];
  sc_event start_Get_lsp_pol; start_Lsp_Az;
  SC_CTOR(Simple_Example) {
    . Connectivity
    SC_THREAD(Get_lsp_pol);
    SC_THREAD(Lsp_Az);
  }

  void Get_lsp_pol() {
    while (true) {
      wait(start_Get_lsp_pol);
      . Get_lsp_pol computes f1 from the odd positions
      . or f2 from the even positions of 'lsp' depending
      . on the value of 'initial_position'
      notify (start_Lsp_Az);
    }
  }

  void Lsp_Az() {
    while (true) {
      wait(start_Lsp_Az);
      initial_position = 1;
      notify(start_Get_lsp_pol);
      for (i=5;i>0;i--) f1[i]=L_add(f1[i],f1[i-1]);
      wait(start_Lsp_Az);
      initial_position = 0;
      for (i=5;i>0;i--) f2[i]=L_sub(f2[i],f2[i-1]);
      . a is computed from f1 and f2
    }
  }
}
Get_lsp_pol = mapU(1, ν\text{lsp_odd}, ν\text{lsp_even}, f, f)

Get_lsp_pol(initial_position, lsp_odd, lsp_even) = \langle f_1, f_2 \rangle

//an 'initial_position' value is always taken
π(ν\text{initial_position}, initial_position) = \langle \text{initial_position} \rangle

ν\text{initial_position}(i) = 1

If (initial_position = 0) then

f_1 = f(lsp_odd)

//five 'lsp_odd' data are taken but no 'lsp_even'
π(ν\text{lsp_odd}, lsp_odd) = \langle lsp_odd \rangle

ν\text{lsp_odd}(i) = 5

π(ν\text{lsp_even}, lsp_even) = \langle lsp_even \rangle

ν\text{lsp_even}(i) = 0

//six 'f1' data are generated but no 'f2'
π(ν\text{f1}, f_1) = \langle f_1 \rangle

ν\text{f1}(i) = 6

π(ν\text{f2}, f_2) = \langle f_2 \rangle

ν\text{f2}(i) = 0

else

f_2 = f(lsp_even)

//five 'lsp_even' data are taken but no 'lsp_odd'
π(ν\text{lsp_even}, lsp_even) = \langle lsp_even \rangle

ν\text{lsp_even}(i) = 0

π(ν\text{lsp_odd}, lsp_odd) = \langle lsp_odd \rangle

ν\text{lsp_odd}(i) = 5

//six 'f2' data are generated but no 'f1'
π(ν\text{f1}, f_1) = \langle f_1 \rangle

ν\text{f1}(i) = 0

π(ν\text{f2}, f_2) = \langle f_2 \rangle

ν\text{f2}(i) = 6

void Get_lsp_pol() {
    while (true) {
        wait(start_Get_lsp_pol);
        Get_lsp_pol computes f1 from the odd positions
        or f2 from the even positions of 'lsp' depending
        on the value of 'initial_position'
        notify (start_Lsp_Az);
    }
}

void Lsp_Az() {
    while (true) {
        wait(start_Lsp_Az);
        initial_position = 1;
        notify(start_Get_lsp_pol);
        for (i=5;i>0;i--) f1[i]=L_add(f1[i],f1[i-1]);
        wait(start_Lsp_Az);
        initial_position = 0;
        for (i=5;i>0;i--) f2[i]=L_sub(f2[i],f2[i-1]);
        // a is computed from f1 and f2
    }
}
Formal Framework: Simple Example

Az_Lsp = mealyU(ν
f1, νf2,g,fi_p,fa,ω0)

Az_Lsp(f1,f2) = <<initial_position><a>>

If (state=ω0) then

initial_position=fi_p(state)=1
//six 'f1' data are taken but no 'f2'
π(ν1, f1) = <f11>
ν1(i) = 6
π(ν2, f2) = <f21>
ν2(i) = 0

else

initial_position=fi_p(state)=0
//six 'f2' data are taken but no 'f1' although 'f1' is used
π(ν1, f1) = <f11>
ν1(i) = 0
π(ν2, f2) = <f21>
ν2(i) = 6

statei=ω0

a is computed from f1 and f2

void Get_lsp_pol() {
while (true) {
wait(start_Get_lsp_pol);
. Get_lsp_pol computes f1 from the odd positions
. or f2 from the even positions of 'lsp' depending
. on the value of 'initial_position'
notify (start_Lsp_Az);
}
}

void Lsp_Az() {
while (true) {
wait(start_Lsp_Az);
initial_position = 1;
notify(start_Get_lsp_pol);
for (i=5;i>0;i--) f1[i]=L_add(f1[i],f1[i-1]);
wait(start_Lsp_Az);
initial_position = 0;
for (i=5;i>0;i--) f2[i]=L_sub(f2[i],f2[i-1]);
. a is computed from f1 and f2
}
CSV1. Every data token written by the producer process is read by the consumer process.

CSV2. Every data token written by the producer process is read only once by the consumer process.

\[\text{-- psl property } \text{CSV1}_2\text{ initial position is always} \]
\[\text{(initial\_position.write\_CALL() \rightarrow)}\]
\[\text{next(initial\_position.read\_CALL()) \rightarrow before initial\_position.write\_CALL())};\]
\[\text{-- psl assert CSV1}_2\text{ initial position} \]
CSV3. If a consumer uses a shared variable as local memory, no new data can be written by the producer until the last access as local memory by the consumer, that is, during the local memory lifetime of the shared variable.
Application to High-Level Synthesis

- Synthesis results (after Gaut)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Get_lsp_pol</th>
<th>Az_Lsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. Area</td>
<td>1,740</td>
<td>810</td>
</tr>
<tr>
<td>Min. Latency</td>
<td>960</td>
<td>370</td>
</tr>
</tbody>
</table>

- Verification results
  - System verification testbench

<table>
<thead>
<tr>
<th>Design</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV1_2_initial_position</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>CSV1_2_f1</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>CSV1_2_f2</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>CSV3_f1</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>CSV3_f2</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>Functional correctness</td>
<td>OK</td>
<td>OK</td>
<td>Error</td>
<td>Error</td>
</tr>
</tbody>
</table>
Conclusions

• ForSyDe has been shown as a formal meta-model for SystemC
  – Untimed models
  – Specification semantics to be preserved
  – Application to high-level synthesis
Future Work

• Extended SystemC specification
  – More complex communication mechanisms

• Other applications
  – Architectural design
  – HW/SW mappings
Thanks and Questions

• Thank you for your attention

• Funding