Process Algebraic Verification of SystemC Codes

Hossein Hojjat
University of Tehran, IPM, Tehran, Iran
TU/Eindhoven, Eindhoven, The Netherlands

Prose (Process Seminar), TU/Eindhoven
January 21st 2008
Outline

1 SystemC
   - Motivation
   - Quick Introduction

2 mCRL2

3 From SystemC to mCRL2
Design Complexity

Increasing design complexity

- Heterogeneity of modern embedded systems
- Variation of demands in applications
- New standards, New architectures
Design Complexity

Increasing design complexity
- Heterogeneity of modern embedded systems
- Variation of demands in applications
- New standards, New architectures

Higher level abstraction
- Specification using an abstraction of hardware and software
- System-level Design
  - More abstract than RTL (Register Transfer Level)
  - Can specify both software and hardware
A System-level Language

Different languages

- **Software Engineers**: C/C++, Java, ···
- **Hardware Engineers**: VHDL, Verilog, ···
- **Competition between SystemC and SystemVerilog**
  - Both are widely used today
A System-level Language

Different languages

- Software Engineers: C/C++, Java, ···
- Hardware Engineers: VHDL, Verilog, ···
- Competition between SystemC and SystemVerilog
  - Both are widely used today

SystemC

- Language to describe hardware and software at multiple levels of abstractions
- A library based on C++
Highlights

Open SystemC Initiative (OSCI)
Organization dedicated to supporting and advancing SystemC
Growing market of commercial products
EDA tools
IP products
Broad range of applications
(version 1.x) Register-transfer level
(version 2.x) System-level modeling
(version 3.x) Real-time software development
(version 4.x) Analog, mixed-signal design
Open SystemC Initiative (OSCI)
Organization dedicated to supporting and advancing SystemC
Highlights

- Open SystemC Initiative (OSCI)
  - Organization dedicated to supporting and advancing SystemC
- Growing market of commercial products
  - EDA tools
  - IP products
SystemC
Motivation

Highlights

- Open SystemC Initiative (OSCI)
  - Organization dedicated to supporting and advancing SystemC
- Growing market of commercial products
  - EDA tools
  - IP products

Broad range of applications
Highlights

- Open SystemC Initiative (OSCI)
  - Organization dedicated to supporting and advancing SystemC
- Growing market of commercial products
  - EDA tools
  - IP products

Broad range of applications

- (version 1.x) Register-transfer level
Highlights

- Open SystemC Initiative (OSCI)
  - Organization dedicated to supporting and advancing SystemC
- Growing market of commercial products
  - EDA tools
  - IP products

Broad range of applications

- (version 1.x) Register-transfer level
- (version 2.x) System-level modeling
Highlights

- Open SystemC Initiative (OSCI)
  - Organization dedicated to supporting and advancing SystemC
- Growing market of commercial products
  - EDA tools
  - IP products

Broad range of applications

- (version 1.x) Register-transfer level
- (version 2.x) System-level modeling
- (version 3.x) Real-time software development
Highlights

- Open SystemC Initiative (OSCI)
  - Organization dedicated to supporting and advancing SystemC
- Growing market of commercial products
  - EDA tools
  - IP products

Broad range of applications

- (version 1.x) Register-transfer level
- (version 2.x) System-level modeling
- (version 3.x) Real-time software development
- (version 4.x) Analog, mixed-signal design
Highlights

- Open SystemC Initiative (OSCI)
  - Organization dedicated to supporting and advancing SystemC
- Growing market of commercial products
  - EDA tools
  - IP products

Broad range of applications

- (version 1.x) Register-transfer level
- (version 2.x) System-level modeling
- (version 3.x) Real-time software development (Future Plan)
- (version 4.x) Analog, mixed-signal design (Future Plan)
SystemC Library Structure

Channels for MoCs
Kahn process networks, SDF

Methodology-specific Channels
Master/Slave library

Primitive Channels
signal, mutex, semaphore, FIFO

Core Language Elements
modules, ports, processes, events,
Interfaces, channels.
Event-driven simulation kernel

Date Types
4-valued logic, 4-valued logic vector, bits and bit vectors,
arbitrary precision integers, fixed points, C++ user defined types

C++ Language Standard
**Fundamental entity**

Module (SC_MODULE)
- The building block of models
- May contain other modules
- Hierarchical composition
- May contain SystemC processes
- Concurrency

Ports (sc_in, sc_out)
- Connect modules to channels
- sc_signal, sc_fifo

Ports have types

Hossein Hojjat (H.Hojjat@tue.nl)
Fundamental entity

- Module (SC_MODULE)
  - The building block of models
- May contain other modules
  - Hierarchical composition
- May contain SystemC processes
  - Concurrency
Fundamental entity

- Module (SC_MODULE)
  - The building block of models
  - May contain other modules
  - Hierarchical composition
  - May contain SystemC processes
    - Concurrency

- Ports (sc_in, sc_out)
  - Connect modules to channels
    - sc_signal, sc_fifo
  - Ports have types
Connecting modules together

Module1

Module2

Module3

signal

channel

port
Processes

Processes: Describing behaviors

The inside code is sequential

Processes run in parallel

SC METHOD
SC THREAD
SC CTHREAD

Module
in1
in2
out1
out2

Process
Process

SC METHOD
Executes its body from the beginning to end
Cannot be interrupted
Does not save internal state

SC THREAD
Can be suspended by calling wait()
Implicitly keep internal state
Processes

- Processes: Describing behaviors
- The inside code is sequential
- Processes run in parallel
  - SC_METHOD
  - SC_THREAD
  - SC_CTHREAD
Processes

- Processes: Describing behaviors
- The inside code is sequential
- Processes run in parallel
  - `SC_METHOD`
  - `SC_THREAD`
  - `SC_CTHREAD`

- **SC_METHOD**
  - Executes its body from the beginning to end
  - Cannot be interrupted
  - Does not save internal state

- **SC_THREAD**
  - Can be suspended by calling `wait()`
  - Implicitly keep internal state
Events

Low-level synchronization primitives

- Notification of an event schedules the sensitive processes to be triggered
- Rising edge of a clock
- Change in the value of a signal
- Explicit call (en.notify())

Three kinds of notification:
- Immediate
- Delta
- Timed
Events

- Low-level synchronization primitives
Events

- Low-level synchronization primitives
- *Notification* of an event schedules the sensitive processes to be triggered
  - Rising edge of a clock
  - Change in the value of a signal
  - Explicit call (`en.notify()`)
Events

- Low-level synchronization primitives

*Notification* of an event schedules the sensitive processes to be triggered
- Rising edge of a clock
- Change in the value of a signal
- Explicit call (en.notify())

- Three kinds of notification
  - Immediate
  - Delta
  - Timed
Example (D flip-flop)

```c
SC_MODULE(dff)
{
    sc_in<bool> din;
    sc_in<bool> clk;
    sc_out<bool> dout;
    void get_input()
    {
        dout = din;
    }
    SCCTOR(dff)
    {
        SC_METHOD(get_input);
        sensitive << clk.pos();
    }
};
```
SystemC Simulation Kernel

- Elaborate
- Initialize
- Evaluate
- Update
- Advance Time
- Cleanup

Timeline:
- sc_start()
- While processes ready
  - immediate notify()
  - timed notify(SC_ZERO_TIME)
  - timed notify(t)
- Delta Cycle
- Update
- Advance Time
In spite of intensive recent activity in the development of formal-verification techniques for software, extending such techniques to SystemC is a formidable challenge.

The difficulty stems from both the object-oriented nature of SystemC, which is fundamental to its modeling philosophy, and its sophisticated event-driven simulation semantics.
Outline

1 SystemC
   • Motivation
   • Quick Introduction

2 mCRL2

3 From SystemC to mCRL2
Process expressions

Basic operators

Alternative composition
\[ p + q \]

Sequential composition
\[ p . q \]

Conditional operator
\[ c \rightarrow p \diamond q \]

Summation operator
\[ \sum x : D \]

Parallel composition
\[ p \parallel q \]

Synchronization operator
\[ p | q \]

Allow operator
\[ \nabla V (p) \]

Blocking operator
\[ \partial B (p) \]

Communication operator
\[ \Gamma C (p) \]
Process expressions

Basic operators

- Alternative composition $p + q$
Process expressions

Basic operators

- Alternative composition \( p + q \)
- Sequential composition \( p.q \)
Process expressions

Basic operators

- Alternative composition $p + q$
- Sequential composition $p.q$
- Conditional operator $c \rightarrow p \diamond q$
Process expressions

Basic operators

- Alternative composition $p + q$
- Sequential composition $p.q$
- Conditional operator $c \rightarrow p \diamond q$
- Summation operator $\sum_{x:D} p$
Process expressions

Basic operators

- Alternative composition $p + q$
- Sequential composition $p.q$
- Conditional operator $c \rightarrow p \diamond q$
- Summation operator $\sum_{x:D} p$
- Parallel composition $p \parallel q$
Basic operators

- Alternative composition $p + q$
- Sequential composition $p.q$
- Conditional operator $c \rightarrow p \diamond q$
- Summation operator $\sum_{x:D} p$
- Parallel composition $p \parallel q$
- Synchronization operator $p | q$
Process expressions

Basic operators

- Alternative composition $p + q$
- Sequential composition $p.q$
- Conditional operator $c \rightarrow p \diamond q$
- Summation operator $\sum_{x:D} p$
- Parallel composition $p \parallel q$
- Synchronization operator $p | q$
- Allow operator $\nabla_V(p)$
Process expressions

### Basic operators

- Alternative composition $p + q$
- Sequential composition $p \cdot q$
- Conditional operator $c \rightarrow p \diamond q$
- Summation operator $\sum_{x:D} p$
- Parallel composition $p \parallel q$
- Synchronization operator $p \mid q$
- Allow operator $\nabla_V(p)$
- Blocking operator $\partial_B(p)$
Process expressions

Basic operators

- Alternative composition $p + q$
- Sequential composition $p.q$
- Conditional operator $c \rightarrow p \diamond q$
- Summation operator $\sum_{x:D} p$
- Parallel composition $p \parallel q$
- Synchronization operator $p \mid q$
- Allow operator $\nabla_V(p)$
- Blocking operator $\partial_B(p)$
- Communication operator $\Gamma_C(p)$
Outline

1 SystemC
   - Motivation
   - Quick Introduction

2 mCRL2

3 From SystemC to mCRL2
General Idea

mCRL2 Processes

Map each SystemC process to an mCRL2 process

Some processes for the state of the program

Variables

Signals

A queue for the process scheduling information

ID

Process state: ready, suspended, finish

Sensitivity list

Scheduler

Implemented Compiler

Automatically translates SystemC codes to mCRL2

Java, Antlr

Available from

http://www.win.tue.nl/~mousavi/sysc08/
General Idea

mCRL2 Processes

- Map each SystemC process to an mCRL2 process
General Idea

mCRL2 Processes

- Map each SystemC process to an mCRL2 process
- Some processes for the state of the program
  - Variables
  - Signals
From SystemC to mCRL2

General Idea

mCRL2 Processes

- Map each SystemC process to an mCRL2 process
- Some processes for the state of the program
  - Variables
  - Signals
- A queue for the process scheduling information
  - ID
  - Process state: ready, suspended, finish
  - Sensitivity list
General Idea

mCRL2 Processes

- Map each SystemC process to an mCRL2 process
- Some processes for the state of the program
  - Variables
  - Signals
- A queue for the process scheduling information
  - ID
  - Process state: ready, suspended, finish
  - Sensitivity list
- Scheduler
General Idea

mCRL2 Processes

- Map each SystemC process to an mCRL2 process
- Some processes for the state of the program
  - Variables
  - Signals
- A queue for the process scheduling information
  - ID
  - Process state: ready, suspended, finish
  - Sensitivity list
- Scheduler

Implemented Compiler

- Automatically translates SystemC codes to mCRL2
  - Java, Antlr
  - Available from http://www.win.tue.nl/~mousavi/sysc08/
Some problems
Some problems

- Bit length for variables
Some problems

- Bit length for variables
- Unsigned and signed interpretation of numbers
Some problems

- Bit length for variables
- Unsigned and signed interpretation of numbers
- Bitwise operations
  - 2's complement
  - 1’s complement
  - Bitwise and
  - Bitwise or
  - Bitwise xor
  - Shift left
  - Shift right
  - Range
  - Range assignment
Message Sequence

Scheduler

SystemC Processes

Evaluate
SysProc1
SysProc2

State Holders

Process Queue
Variables
Signals

get first ready
Message Sequence
Message Sequence

Scheduler

SystemC Processes

State Holders

Evaluate

SysProc1

SysProc2

Process Queue

Variables

Signals

get first ready

SysProc1

start
Message Sequence

Scheduler

SystemC Processes

State Holders

Evaluate

SysProc1

SysProc2

Process Queue

Variables

Signals

get first ready

SysProc1

start

change signal x
Message Sequence

Scheduler  SystemC Processes  State Holders

Evaluate  SysProc1  SysProc2  Process Queue  Variables  Signals

get first ready

SysProc1

start

change signal x

change my status suspended
From SystemC to mCRL2

Message Sequence

Scheduler

SystemC Processes

State Holders

Evaluate

SysProc1

SysProc2

Process Queue

Variables

Signals

get first ready

SysProc1

start

change signal x

change my status suspended

finish

Hossein Hojjat (H.Hojjat@tue.nl)
Message Sequence

Scheduler
- Evaluate

SystemC Processes
- SysProc1
- SysProc2

State Holders
- Process Queue
- Variables
- Signals

SysProc1
- get first ready
- start
- change signal x
- change my status suspended
- finish
- get first ready

SysProc2
- get first ready

Hossein Hojjat (H.Hojjat@tue.nl)
Message Sequence

Scheduler  SystemC Processes  State Holders

Evaluate  SysProc1  SysProc2  Process Queue  Variables  Signals

get first ready  SysProc1  start  change signal x

change my status suspended

finish

get first ready

No one

Hossein Hojjat (H.Hojjat@tue.nl)
Message Sequence

Scheduler

Update

SysProc1

SysProc2

SystemC Processes

get first ready

SysProc1

start

change signal x

change my status suspended

finish

get first ready

No one

State Holders

Process Queue

Variables

Signals

SysProc1

SysProc2

Variables

Signals

Hossein Hojjat (H.Hojjat@tue.nl) 19/32
Message Sequence

Scheduler

SystemC Processes

State Holders

Update

SysProc1

SysProc2

Process Queue

Variables

Signals

get first ready

SysProc1

start

change signal x

change my status suspended

finish

get first ready

No one

update signal x

Hossein Hojjat (H.Hojjat@tue.nl)
Message Sequence

Scheduler

Update
SysProc1
SysProc2

SystemC Processes

SysProc1

SystemC Processes

get first ready
start
change signal x
change my status suspended
finish
get first ready
No one
update signal x
notify changes

State Holders

Process Queue
Variables
Signals

Hossein Hojjat (H.Hojjat@tue.nl)
A Dining Philosophers Circuit
A Dining Philosophers Circuit
From SystemC to mCRL2

Translation

SC_MODULE(philosopher) {
    sc_out<bool> req_l_out;
    sc_out<bool> req_r_out;
    sc_in<bool> res_l_in;
    sc_in<bool> res_r_in;
    void get_forks() {
        req_l_out.write(true);
        req_r_out.write(true);
    }
    void response_l() {
        if (res_r.in.read() == true &&
            res_l.in.read() == true) {
            req_l.out.write(false);
            req_r.out.write(false);
        }
    }
    void response_r() {
        if (res_r.in.read() == true &&
            res_l.in.read() == true) {
            req_l.out.write(false);
            req_r.out.write(false);
        }
    }
    SCCTOR(philosopher) {
        SC_METHOD(get_forks);
        SC_METHOD(response_l);
        sensitive << res_l_in;
        SC_METHOD(response_r);
        sensitive << res_r_in;
    }
};

proc get_forks( pID : ID, container : ModuleIns) =
∑ro : Rnd . r_start( pID, ro).
       s_changeSig( signalConnect( req_l_out, container),
                    boolean(true)).
       s_changeSig( signalConnect( req_r_out, container),
                    boolean(true)).
       s_changeStat( pID, suspended( event( NullEvn,
                                              container)) ).
       s_finish(pID).get_forks( pID, container);

proc response_l( pID : ID, container : ModuleIns) =
∑ro : Rnd . r_start( pID, ro).
               (∑sl : SigList . s_getSigList( sl).
                (getValBool( findCurSigVal( sl, signalConnect( res_r.in, container))) == true ∧
                 getValBool( findCurSigVal( sl, signalConnect( res_l.in, container))) == true) →
                 (s_changeSig(signalConnect( req_l_out, container),
                              boolean(false)).
                 s_changeSig( signalConnect( req_r_out, container), boolean(false))).)
         s_changeStat( pID, suspended( event( NullEvn,container)))).
         s_finish(pID).response_l( pID, container);
The State Space

- 1149639 states
- 1362861 transitions
Synthesizable hardware cannot have nondeterminism
Deterministic simulation
Scheduling processes in a FIFO manner

How to avoid race conditions?
Applying $\tau$-confluence?
Synthesizable hardware cannot have nondeterminism

Deterministic simulation

Scheduling processes in a FIFO manner

How to avoid race conditions?
  - Applying $\tau$-confluence?
mMIPS processor

Designed at the Electronic Systems group of TU/e

mini MIPS: simplified version of the MIPS processor

Used in MiniNoC

Network-on-Chip (NOC) based multi-processor System-on-Chip (SoC)

We investigated both single cycled and pipelined

About 2500 lines of SystemC code (pipelined design)

Hossein Hojjat (H.Hojjat@tue.nl)
mMIPS processor

- Designed at the Electronic Systems group of TU/e
mMIPS processor

- Designed at the Electronic Systems group of TU/e
- mini MIPS: simplified version of the MIPS processor
mMIPS processor

- Designed at the Electronic Systems group of TU/e
- mini MIPS: simplified version of the MIPS processor
- Used in MiniNoC
  - Network-on-Chip (NOC) based multi-processor System-on-Chip (SoC)
mMIPS processor

- Designed at the Electronic Systems group of TU/e
- mini MIPS: simplified version of the MIPS processor
- Used in MiniNoC
  - Network-on-Chip (NOC) based multi-processor System-on-Chip (SoC)
- We investigated both single cycled and pipelined
mMIPS processor

- Designed at the Electronic Systems group of TU/e
- mini MIPS: simplified version of the MIPS processor
- Used in MiniNoC
  - Network-on-Chip (NOC) based multi-processor System-on-Chip (SoC)
- We investigated both single cycled and pipelined
- About 2500 lines of SystemC code (pipelined design)
From SystemC to mCRL2

Burch & Dill Notion of Correctness

\[
\begin{align*}
M A_0 & \xrightarrow{F_{Impl}} M A_n \\
ISA_0 & \xrightarrow{F_{Spec}} ISA_m
\end{align*}
\]
General Observation

- Using $\tau$-confluence reduction to omit unrelated actions
- Proving the data semantics of each instruction
- Control related issues
  - Found a bug in arithmetic shift instructions
  - Has been reported to the designer
  - Fixed in the new releases
General Observation

- Using $\tau$-confluence reduction to omit unrelated actions
General Observation

- Using $\tau$-confluence reduction to omit unrelated actions
- Proving the data semantics of each instruction
General Observation

- Using \( \tau \)-confluence reduction to omit unrelated actions
- Proving the data semantics of each instruction
- Control related issues
General Observation

- Using $\tau$-confluence reduction to omit unrelated actions
- Proving the data semantics of each instruction
- Control related issues
- Found a bug in arithmetic shift instructions
  - Has been reported to the designer
  - Fixed in the new releases
Detected Bugs

Single Cycled design

Variable for PC was a signed integer
Wrap around in the middle
According to designer this is not important!

Pipelined design
Problem in the arithmetic shift right operations
Pushing zero instead of the sign bit
Results to incorrect programs
Detected Bugs

Single Cycled design

- Divide to components
  - Use model checking

Variable for PC was a signed integer
Wrap around in the middle
According to designer this is not important!

Pipelined design
Problem in the arithmetic shift right operations
Pushing zero instead of the sign bit
Results to incorrect programs
Detected Bugs

Single Cycled design

- Divide to components
  - Use model checking
- Variable for PC was a signed integer
  - Wrap around in the middle

According to the designer, this is not important!
Detected Bugs

### Single Cycled design
- Divide to components
  - Use model checking
- Variable for PC was a signed integer
  - Wrap around in the middle
- According to designer this is not important!

Hossein Hojjat (H.Hojjat@tue.nl)
Detected Bugs

Single Cycled design

- Divide to components
  - Use model checking
- Variable for PC was a signed integer
  - Wrap around in the middle
- According to designer this is not important!

Pipelined design

- Problem in the arithmetic shift right operations
- Pushing zero instead of the sign bit
- Results to incorrect programs
Future Work

- Controlling problems
- Quantitative time
- Extend our work to TL (Transaction Level) models
- Proving an upper bound for number of instructions
Future Work

- Controlling problems
Future Work

- Controlling problems
- Quantitative time
Future Work

- Controlling problems
- Quantitative time
- Extend our work to TL (Transaction Level) models
Future Work

- Controlling problems
- Quantitative time
- Extend our work to TL (Transaction Level) models
- Proving an upper bound for number of instructions
Acknowledgements

- Sander Stuijk
  - Providing the source codes of mMIPS
  - Useful discussions

- Jan Friso Groote
  - Helps in verifying the models
  - Valuable comments
Vragen?