Center for Embedded and Cyber-physical Systems
University of California, Irvine

RISC Compiler and Simulator, Alpha Release V0.2.1: Out-of-Order Parallel Simulatable SystemC Subset

Guantao Liu, Tim Schmidt, and Rainer Dömer

Technical Report CECS-15-02
October 30, 2015

Center for Embedded and Cyber-physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
+1 (949) 824-8919

{guantaol,schmidtt,doemer}@uci.edu
http://www.cecs.uci.edu/~doemer/risc.html

RISC Compiler and Simulator, Alpha Release V0.2.1: Out-of-Order Parallel Simulatable SystemC Subset

Guantao Liu, Tim Schmidt, and Rainer Dömer

Technical Report CECS-15-02
October 30, 2015

Center for Embedded and Cyber-physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
+1 (949) 824-8919

{guantaol,schmidtt,doemer}@uci.edu
http://www.cecs.uci.edu/~doemer/risc.html

Abstract

SystemC is widely used in industry and academia to specify and simulate Electronic System Level (ESL) models. Despite the wide availability of multi-core processor hosts, however, the reference SystemC simulator is still based on sequential Discrete Event Simulation (DES) and executes only a single thread at any time.

In recent years parallel SystemC simulators were proposed which run multiple threads in parallel based on synchronous Parallel Discrete Event Simulation (PDES) semantics. Synchronous PDES, however, limits parallel execution to threads that run at the same time and delta cycle.

In this report, we describe the advanced Recoding Infrastructure for SystemC (RISC) approach where a dedicated SystemC compiler and advanced parallel simulator implement Out-of-Order Parallel Discrete Event Simulation (OoO PDES) for SystemC. OoO PDES can execute threads in parallel and out-of-order (ahead of time) and thus achieves fastest simulation speed but nevertheless maintains the classic SystemC modeling semantics.

This report describes the RISC Compiler and Simulator and details the SystemC subset supported by the RISC Alpha Release V0.2.1, as of October 30, 2015.

Contents

List of Figures

List of Tables

RISC Compiler and Simulator, Alpha Release V0.2.1: Out-of-Order Parallel Simulatable SystemC Subset

Guantao Liu, Tim Schmidt, and Rainer Dömer
Center for Embedded and Cyber-physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
{guantaol,schmidtt,doemer}@uci.edu
http://www.cecs.uci.edu/~doemer/risc.html

Abstract

SystemC is widely used in industry and academia to specify and simulate Electronic System Level (ESL) models. Despite the wide availability of multi-core processor hosts, however, the reference SystemC simulator is still based on sequential Discrete Event Simulation (DES) and executes only a single thread at any time.

In recent years parallel SystemC simulators were proposed which run multiple threads in parallel based on synchronous Parallel Discrete Event Simulation (PDES) semantics. Synchronous PDES, however, limits parallel execution to threads that run at the same time and delta cycle.

In this report, we describe the advanced Recoding Infrastructure for SystemC (RISC) approach where a dedicated SystemC compiler and advanced parallel simulator implement Out-of-Order Parallel Discrete Event Simulation (OoO PDES) for SystemC. OoO PDES can execute threads in parallel and out-of-order (ahead of time) and thus achieves fastest simulation speed but nevertheless maintains the classic SystemC modeling semantics.

This report describes the RISC Compiler and Simulator and details the SystemC subset supported by the RISC Alpha Release V0.2.1, as of October 30, 2015.

1 Introduction

As an IEEE standard [1], the SystemC System Level Description Language (SLDL) is widely used for the specification, modeling, validation and evaluation of Electronic System Level (ESL) models. Under the Accellera Systems Initiative [2], the SystemC Language Working Group [3] maintains not only the official SystemC language definition, but also provides an open source proof-of-concept library [4] that can be used to simulate SystemC design models. However, implementing the classic scheme of Discrete Event Simulation (DES), this reference simulator runs sequentially and cannot utilize the parallel computing resources available on multi-core (or many-core) processor hosts. This severely limits the execution speed of SystemC simulation.

In order to provide faster simulation, Parallel Discrete Event Simulation (PDES) [5] has recently gained again significant attraction (examples include [6], [7], [8], [9], [10], and [11]). The PDES approach issues multiple threads (i.e. `SC_METHOD`, `SC_THREAD` and `SC_CTHREAD`) concurrently and runs them on the available processor cores in parallel. In turn, the simulation speed increases significantly.

Regular PDES is synchronous, however. That is, time advances globally and all threads execute in lock-step fashion. Here, the total order of time imposed by synchronous PDES still limits the opportunities for parallel execution. When a thread completes its evaluation phase, it has to wait until all other threads finish their evaluation phases as well. Earlier completed threads must stop at the simulation cycle barrier and available processor cores are left idle until all runnable threads reach the cycle barrier.

In order to overcome this problem, we have developed a novel technique called Out-of-Order Parallel Discrete Event Simulation (OoO PDES) [12, 13, 14, 15]. By localizing the simulation time to individual threads and carefully handling events at different times, the simulation kernel can issue threads in parallel and ahead of time, following a partial order of time without loss of accuracy. Thus, Ooo PDES significantly reduces the idle time of available parallel processor cores and results in maximum simulation speed, while maintaining the traditional language and modeling semantics.

The OoO PDES technique was originally implemented based on the SpecC language [16, 17, 18, 19]. In this report, we document our efforts to apply OoO PDES to the SystemC SLDL [20, 21, 1] which is both the de-facto and official standard for ESL design today. In particular, we describe our Recoding Infrastructure for SystemC (RISC) [22] which consists of a dedicated SystemC compiler and corresponding out-of-order parallel simulator and implements OoO PDES for SystemC.

The remainder of this report is organized as follows: After a brief description of the simulator scheduling algorithms used for DES, PDES and OoO PDES in Section 2, we describe the RISC Compiler and Simulator proof-of-concept prototype in Section 3. In Section 4, we then list in detail the SystemC subset that is supported by the current RISC Alpha Release V0.2.1 (2015-10-30) and finally conclude this report in Section 5.

2 Out-of-Order Parallel Simulation

In this section, we briefly outline the scheduling algorithm used in out-of-order parallel simulation. We do this incrementally, starting from the traditional Discrete Event Simulation (DES) scheduler, then describe the synchronous Parallel DES (PDES) extension, and finally define the Out-of-Order PDES (OoO PDES) scheduling algorithm.

2.1 Notations

To formally describe the discrete event scheduling algorithms, the following notations are introduced.

1. Each SystemC thread (`SC_METHOD`, `SC_THREAD` and `SC_CTHREAD`) is assigned a localized time stamp (t_{th}, δ_{th}).

2. Each event (`sc_event`) is assigned a notification time stamp (t_e, δ_e), where $EVENTS = \cup EVENTS_{t,\delta}$.

3. Threads are grouped into different queues. Specifically,

   (a) $QUEUES = \{READY, RUN, WAIT, WAITTIME\}$.

   (b) $READY = \cup th_{t,\delta}$ where Thread th is ready to run at time (t, δ).

   (c) $RUN = \cup th_{t,\delta}$ where Thread th is running at time (t, δ).

   (d) $WAIT = \cup th_{t,\delta}$ where Thread th is waiting since time (t, δ).

   (e) $WAITTIME = \cup th_{t,0}$ where Thread th is waiting for simulation time advance to $(t, 0)$.

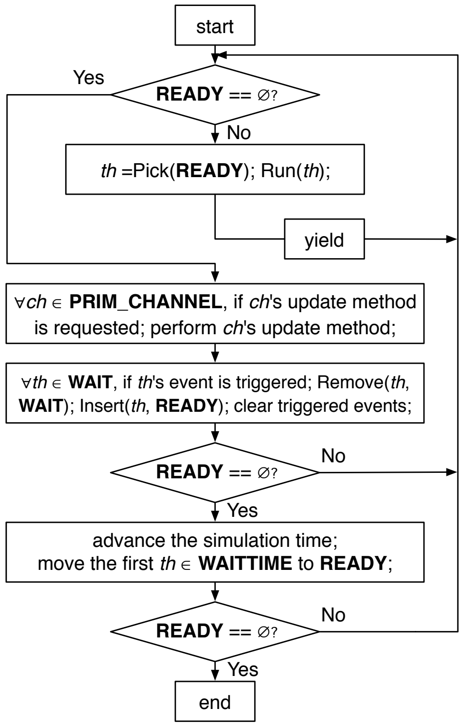
start
READY == ∅?
Yes
No
th =Pick(READY); Run(th);
yield
∀ch ∈ PRIM_CHANNEL, if ch's update method is requested; perform ch's update method;
∀th ∈ WAIT, if th's event is triggered; Remove(th, WAIT); Insert(th, READY); clear triggered events;
READY == ∅?
No
Yes
advance the simulation time; move the first th ∈ WAITTIME to READY;
READY == ∅?
No
Yes
end

Figure 1: Traditional Discrete Event Simulation (DES) scheduler for SystemC.

2.2 Discrete Event Scheduler

The Accellera reference simulation library of SystemC [4] is based on DES. Figure 1 depicts such a traditional DES scheduling algorithm. In DES, a single thread is running at all times. When all threads in the *READY* and *RUN* queues complete their current delta cycle, the root thread resumes and performs the update and notification phase. Then threads are woken up and moved from the *WAIT* queue back into the *READY* queue. A new delta cycle begins.

If no threads are ready after the update and notification phase, the current time cycle finishes. The simulation kernel advances the simulation time and processes the earliest timed event from the *WAITTIME* queue. A new cycle begins for the updated simulated time.

Finally, when both the *WAITTIME* and *READY* queues are empty, the simulation terminates.

2.3 Parallel Discrete Event Scheduler

In comparison to DES, regular synchronous PDES issues multiple threads (`SC_METHOD`, `SC_THREAD` and `SC_CTHREAD`) concurrently in a delta cycle. These threads can then execute truly in parallel on the multiple available processor cores of the host.

Figure 2 shows the regular synchronous PDES scheduling algorithm. In the evaluation phase, as long as the *READY* queue is not empty and an idle core is available, the PDES scheduler will issue a new thread from the *READY* queue. If a thread finishes earlier than other threads in the same cycle, a new ready thread is assigned to the idle processor core, unless there is no thread available in the *READY* queue, in which case the core is keept idle until the next delta cycle.

It should be emphasized that synchronous PDES implies an absolute barrier at the end of each delta and time cycle. All threads need to wait at the barrier until all other runnable threads finish their current evaluation phase.

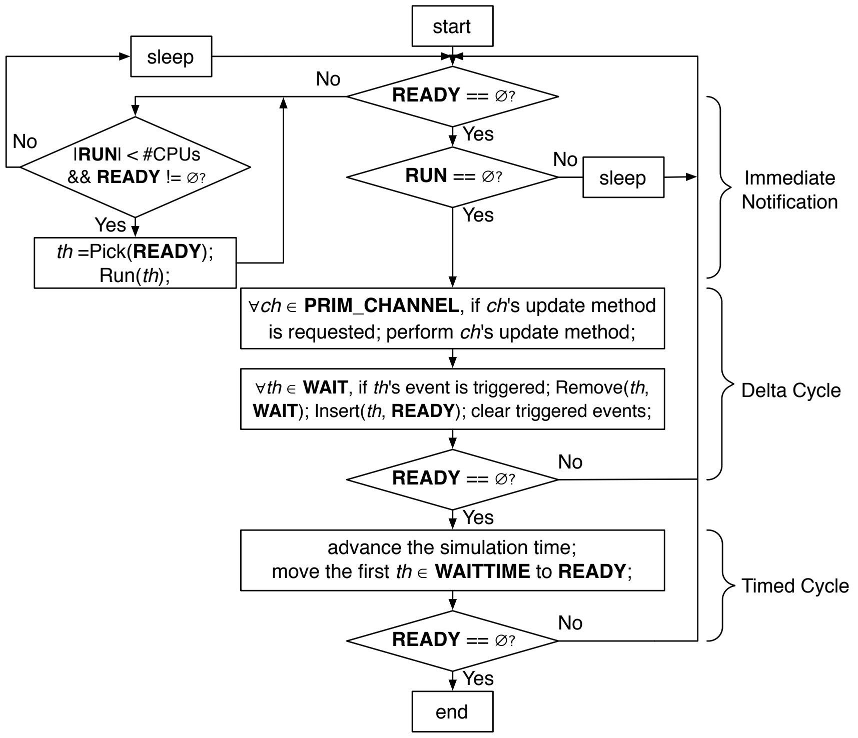
start
sleep
No
READY == ∅?
No
|RUN| < #CPUs && READY != ∅?
Yes
th =Pick(READY); Run(th);
Yes
RUN == ∅?
No
sleep
Yes
Immediate Notification
∀ch ∈ PRIM_CHANNEL, if ch's update method is requested; perform ch's update method;
∀th ∈ WAIT, if th's event is triggered; Remove(th, WAIT); Insert(th, READY); clear triggered events;
Delta Cycle
READY == ∅?
No
Yes
advance the simulation time; move the first th ∈ WAITTIME to READY;
Timed Cycle
READY == ∅?
No
Yes
end

Figure 2: Synchronous Parallel Discrete Event Simulation (PDES) scheduler for SystemC.

Only then the synchronous PDES scheduler resumes and performs the update and notification phases, and finally advances to the next delta or time cycle.

For the SystemC language in particular, there is a very important aspect to consider when applying PDES. For semantics-compliant SystemC simulation, complex inter-dependency analysis over all variables in the system model is a prerequisite to parallel simulation [23].

The Standard SystemC Language Reference Manual (LRM) [1] clearly states that *"process instances execute without interruption"*. This requirement is also known as cooperative (or co-routine) multitasking which is assumed by the SystemC execution semantics. As detailed in [23], the particular problem of parallel simulation is specifically addressed in the SystemC LRM [1]:

> *"An implementation running on a machine that provides hardware support for concurrent processes may permit two or more processes to run concurrently, provided that the behavior appears identical to the co-routine semantics defined [...]. In other words, the implementation would be obliged to analyze any dependencies between processes and constrain their execution to match the co-routine semantics."*

We will describe the required dependency analysis in more detail below (in Section 3.2), as it is also needed for out-of-order PDES.

2.4 Out-of-Order Parallel Discrete Event Scheduler

In OoO PDES, we break the strict order of time (the synchronous barrier) by localizing time stamps to each thread. Figure 3 shows the out-of-order parallel DES scheduling algorithm. Since each thread has its own time stamp, the OoO PDES scheduler relaxes the event and simulation time updates, allowing more threads (at different simulation cycles!) to run in parallel and ahead of time. This results in a higher degree of parallelism and thus higher simulation speed.

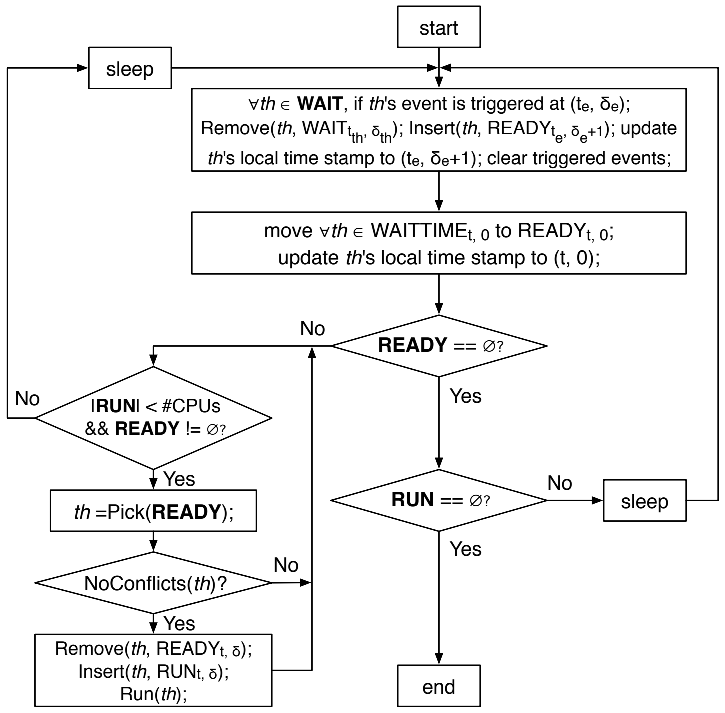
start
sleep
∀th ∈ WAIT, if th's event is triggered at (te, δe);
Remove(th, WAITtth, δth); Insert(th, READYte, δe+1); update
th's local time stamp to (te, δe+1); clear triggered events;
move ∀th ∈ WAITTIMEt, 0 to READYt, 0;
update th's local time stamp to (t, 0);
No
READY == ∅?
Yes
No
|RUN| < #CPUs
&& READY != ∅?
Yes
th =Pick(READY);
NoConflicts(th)?
No
Yes
Remove(th, READYt, δ);
Insert(th, RUNt, δ);
Run(th);
RUN == ∅?
No
sleep
Yes
end

Figure 3: Out-of-Order Parallel Discrete Event Simulation (OoO PDES) scheduler for SystemC.

Note the $NoConflicts(th)$ condition shown in Figure 3. As already mentioned above for the synchronous PDES, detailed dependency analysis is needed to avoid data or event conflicts for any shared variables among the parallel threads. Only if $NoConflicts(th)$ is true, a new thread is issued for parallel execution (moved from the *READY* to the *RUN* queue).

We will be using advanced static compile-time analysis to identify all such potential conflicts. Based on this information (a simple table lookup will suffice!), the OoO PDES scheduler can then at runtime quickly decide whether or not a set of threads has any conflicts with each other.

3 RISC Compiler and Simulator

To realize the OoO PDES approach for the SystemC language, we present now our Recoding Infrastructure for SystemC (RISC) and describe the overall RISC Compiler and Simulator proof-of-concept prototype (Alpha Release V0.2.1 as of 2015-10-30). The RISC software is available as open source and can be downloaded freely from the following web site [22]: http://www.cecs.uci.edu/~doemer/risc.html

To perform semantics-compliant SystemC simulation with maximum parallelism, we introduce a dedicated SystemC compiler. This is in contrast to the traditional SystemC simulation where a regular SystemC-agnostic C++ compiler includes the SystemC headers and links the input model directly against the SystemC library.

As shown in Figure 4, our RISC compiler acts as a frontend that processes the input SystemC model and generates an intermediate model with special instrumention for OoO PDES. The instrumented parallel model is then linked against the extended RISC SystemC library by the target compiler (a regular C++ compiler) to

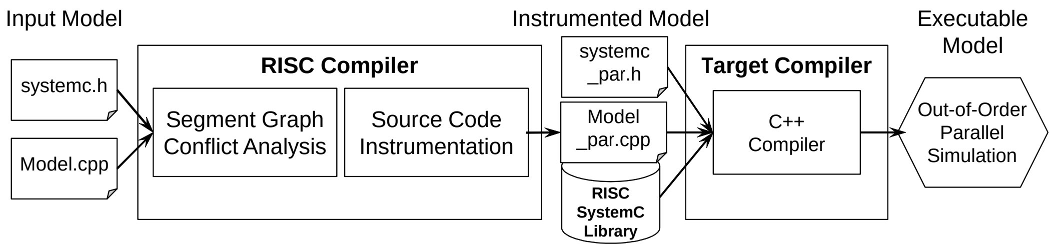

Figure 4: RISC Compiler and Simulator for Out-of-Order PDES of SystemC.

produce the final executable output model. OoO PDES is then performed simply by running the generated executable model.

From the user perspective, we essentially replace the regular SystemC-agnostic C++ compiler with the SystemC-aware RISC compiler (which in turn calls the underlying C++ compiler). Otherwise, the overall SystemC validation flow remains the same as before. It's just faster due to the parallel simulation.

For reference, the detailed Linux manual page of the RISC compiler and simulator is included in Appendix A.1 of this report.

Internally, the RISC compiler performs three major tasks, namely Segment Graph construction, conflict analysis, and source code instrumentation.

3.1 Segment Graph

The first task of the RISC compiler is to parse the SystemC input model into an abstract syntax tree (AST) and then create a SystemC structural representation from the AST which reflects the SystemC module and channel hierarchy, connectivity, and other SystemC-specific relations, similar to the SystemC-clang representation [24, 25]. For details on this part of the RISC application programming interface (API), please refer to the Doxygen-generated documentation [26].

On top of this, the RISC compiler then builds a Segment Graph data structure for the model. A Segment Graph (SG) [12] is a directed graph that represents the code segments executed during the simulation between scheduling steps. That is, every segment is associated with a scheduler entry point, i.e. a `wait` statement in SystemC.

At run time, threads switch back and forth between the states of *running* (threads in *READY* and *RUN* queues) and *waiting* (threads in *WAIT* and *WAITTIME* queues). When *running*, they execute specific segments of their code. These code segments make up the nodes in the Segment Graph, whereas edges in the graph indicate the possible transitions from one segment to another (an abstraction of the model's control flow).

For a formal description of the Segment Graph and it's construction algorithm, the interested reader may refer to [15]. For details on the RISC API, please refer to the Doxygen-generated documentation [26].

3.2 Conflict Analysis

The Segment Graph data structure serves as the foundation for static (compile-time) conflict analysis. As outlined earlier, the OoO PDES scheduler must ensure that every parallel thread to be issued has no conflicts with any other threads currently in the *READY* and *RUN* queues. Here, we utilize the RISC compiler to detect any possible conflicts already at compile time.

Potential conflicts in SystemC include data hazards, event hazards, and timing hazards, all of which may exist among the segments executed by the threads considered for parallel execution. Please refer to [15] for a detailed discussion of these hazards and their conservative identification in the model.

As a result of the conflict analysis, the RISC compiler generates several conflict tables that describe all possible conflicts between threads in any two segments. Using this conservative information, the simulator can then at run-time quickly determine by a simple table look-up whether or not it is safe to issue any given thread in parallel or ahead of time.

3.3 Source Code Instrumentation

As shown above in Figure 4, the RISC compiler and simulator work closely together. The compiler performs conservative static analysis and passes the analysis results to the simulator which then can make safe scheduling decisions quickly.

To pass information from the compiler to the simulator, we use automatic model instrumentation. That is, the intermediate model generated by the compiler contains instrumented (automatically generated) source code which the simulator then can rely on. At the same time, the RISC compiler also instruments user-defined SystemC channels with automatic protection against race conditions among communicating threads.

In total, the RISC source code instrumentation includes four major components:

1. Segment and instance IDs: Individual threads are uniquely identified by a creator instance ID and their current code location (segment ID). Both IDs are passed into the simulator kernel as additional arguments to scheduler entry functions, including `wait` and thread creation.

2. Data and event conflict tables: Segment concurrency hazards due to potential data conflicts, event conflicts, or timing conflicts are provided to the simulator as two-dimensional tables indexed by a segment ID and instance ID pair. For efficiency, these table entries are filtered for scope, instance path, and reference and port mappings.

3. Current and next time advance tables: The simulator can make better scheduling decisions by looking ahead in time if it can predict the possible future thread states. This optimization is discussed in detail in [14].

4. User-defined channel protection: SystemC allows the user to design channels for custom inter-thread communication. To ensure such communication is safe also in the OoO PDES situation where threads execute truly in parallel, the RISC compiler automatically inserts locks (binary semaphores) into these channels so that mutually-exclusive execution of the channel methods is guaranteed. Otherwise, race conditions could exist when communicating threads exchange data.

After this automatic source code instrumentation, the RISC compiler passes the generated intermediate model to the underlying regular C++ compiler which produces the final simulator executable by linking the instrumented code against the RISC extended SystemC library.

4 Out-of-Order Parallel Simulatable SystemC Subset

Over more than a decade, the SystemC language [21], which technically is a C++ application programming interface (API) with a corresponding simulation library, has evolved from basic constructs for modeling parallel modules connected by signals and channels to a highly complex set of macros, types, classes, templates, and functions for very advanced modeling (i.e. Transaction Level Modeling (TLM) 2.0 [27, 28]) and highly optimized simulation of SystemC models. Usually these optimizations have aimed at higher simulation speed, i.e. by minimizing context switches in the simulator, or at higher levels of abstraction due to purposely relaxed timing. Often, the uninterrupted (sequential) execution semantics on a single processor host have been assumed or are explicitly required.

In contrast, RISC now aims for truly parallel execution on multi- or many-core hosts. Changing these fundamental assumptions about SystemC simulator execution consequently may affect some constructs and APIs which need to be revisited and evaluated anew. The goal of this section is to start this process and enable fruitful discussions.

Below, we describe and list the out-of-order parallel simulatable SystemC subset supported by the current RISC Compiler and Simulator, Alpha Release V0.2.1. In particular, Table 1 through Table 8 list for each SystemC construct whether or not it is supported at this time. If applicable, an explanation note is provided that briefly outlines the status and/or the plans for the given feature.

Overall, our current RISC proof-of-concept prototype supports the classic SystemC constructs for hierarchical modeling and multi-threaded execution, but many advanced features are not supported yet or left undecided at this stage. The status "undecided" in particular indicates that further study is needed to decide whether or not the given construct can be supported in efficient and reasonable manner by RISC and its OoO PDES approach.

4.1 SystemC Hierarchical Structure of Modules and Channels

RISC supports the regular hierarchical and structural composition of the SystemC design model. This includes the SystemC program start (`sc_main`, `sc_start`) and the general composition (`SC_CTOR`) of modules (`sc_module`, `SC_MODULE`, `sc_behavior`) and channels (`sc_channel`, `sc_prim_channel`).

Connectivity and communication of the instantiated components is supported through ports (`sc_port`, `sc_in`, `sc_inout`, `sc_out`) and interfaces (`sc_interface`).

In contrast to the traditional Accellera library, which only provides a type definition `sc_channel` to `sc_module`, the RISC header files clearly distinguish channels from modules. Here, a separate `sc_channel` class is inherited from `sc_module`, providing the same functionality, but making the two classes explicit.

Most of the SystemC predefined primitive channels (such as `sc_signal`) are supported for OoO PDES, except `sc_prim_channel::update()` and `sc_fifo::operator=` which are not supported in the current release. For more details, please refer to the Doxygen-generated documentation [29].

4.2 SystemC Threads

The explicit and statically analyzable multi-threading of a SystemC design model is naturally supported in RISC OoO PDES. This includes SystemC processes (`SC_HAS_PROCESS`, `sc_process_handle`, `sc_cthread_process`, `sc_method_process`, `sc_thread_process`) and the corresponding threads and methods (`SC_CTHREAD`, `SC_METHOD`, `SC_THREAD`). For basic inter-thread synchronization, SystemC event notifications (`sc_event.notify`) and waiting for events or simulation time advance (`sc_wait`) are supported.

However, dynamic SystemC thread creation and deletion (`sc_spawn`, `SC_FORK`, `SC_JOIN`) is not supported at this time.

While the application programming interface (API) for these constructs remains unmodified from the SystemC user perspective, the RISC SystemC kernel internally supports extra parameters or arguments for these constructs which are utilized after the automatic source code instrumentation by the RISC compiler (see Section 3.3 above). In particular, segment and instance identifiers are supplied with each of these function calls so that the simulator kernel is aware of the exact thread state upon every scheduler entry. This includes in particular the thread creation constructs (`SC_CTHREAD`, `SC_METHOD`, `SC_THREAD`). and wait (`sc_wait`) statements.

4.3 SystemC Transaction Level Modeling (TLM)

While transaction level modeling in general is a natural feature supported by OoO PDES [15], the modeling and implementation choices made by SystemC TLM 2.0 [28] create significant problems for supporting it efficiently

Table 1: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset

Name	**Type**	**Supported or not**	**Notes**
sc_abs	function	Undecided	This function may not work with some arithmetic SystemC datatypes.
sc_actions	typedef	Supported	typedef unsigned sc_actions
sc_argc	function	Supported	
sc_argv	function	Supported	
sc_assemble_vector	function	Undecided	Work on this function in the future
sc_assert	macro	Undecided	Work on this macro in the future
sc_attr_base	class	Undecided	Work on this class in the future
sc_attr_cltn	class	Undecided	Work on this class in the future
sc_attribute	class	Undecided	Work on this class in the future
sc_behavior	typedef	Supported	typedef sc_module sc_behavior
sc_bigint	class template	Supported	
sc_biguint	class template	Supported	
sc_bind_proxy	class	Supported	
sc_bind	macro	Undecided	Work on this macro in the future
sc_bit	type (deprecated)	Undecided	Work on this type in the future
sc_bitref_r	class template	Undecided	Work on this class template in the future
sc_bitref	class template	Undecided	Work on this class template in the future
sc_buffer	class	Supported	
sc_bv_base	class	Undecided	Work on this class in the future
sc_bv	class template	Undecided	Work on this class template in the future
sc_channel	class	Supported	
sc_clock	class	Not Supported Now	sc_clock::before_end_of_elaboration() calls sc_spawn().
sc_close_vcd_trace_file	function	Undecided	Work on this function in the future
sc_concatref	class	Undecided	Work on this class in the future
sc_concref_r	class template	Undecided	Work on this class template in the future
sc_context_begin	enumeration	Supported	
sc_copyright	function	Supported	
sc_cor	class	Supported	
sc_cor_pkg	class	Supported	
sc_cor_pthread	class	Supported	
sc_cor_pkg_pthread	class	Supported	
sc_create_vcd_trace_file	function	Undecided	Work on this function in the future
sc_cref	macro	Undecided	Work on this macro in the future
sc_cthread_process	class	Supported	
SC_CTHREAD	macro	Supported	The risc compiler can generate the segment graph for SC_CTHREAD, however, it cannot handle the clock.
SC_CTOR	macro	Supported	

Table 2: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset (continued)

Name	**Type**	**Supported or not**	**Notes**
sc_cycle	function (deprecated)	Not Supported Now	sc_cycle() calls sc_simcontext::cycle(), which is not supported in the out-of-order simulation in the current release.
sc_delta_count	function	Supported	This function returns the local delta count of the running process.
sc_elab_and_sim	function	Supported	
sc_end_of_simulation_invoked	function	Undecided	Work on this function in the future
sc_event_and_expr	class	Not Supported Now	Work on this class in the future
sc_event_and_list	class	Not Supported Now	Work on this class in the future
sc_event_finder_t	class template	Undecided	Work on this class template in the future
sc_event_finder	class	Undecided	Work on this class in the future
sc_event_or_expr	class	Not Supported Now	Work on this class in the future
sc_event_or_list	class	Not Supported Now	Work on this class in the future
sc_event_queue_if	class	Supported	
sc_event_queue	class	Not Supported Now	The constructor function is not supported by the out-of-order simulation in the current release.
sc_event	class	Supported	The immediate notification is not supported by the out-of-order simulation in the current release.
sc_exception	typedef	Undecided	Work on this typedef in the future
sc_export_base	class	Not Supported Now	No port following in compiler analysis
sc_export	class	Not Supported Now	No port following in compiler analysis
sc_fifo_blocking_in_if	class	Supported	
sc_fifo_in_if	class	Supported	
sc_fifo_in	class	Supported	
sc_fifo_nonblocking_in_if	class	Supported	
sc_fifo_out_if	class	Supported	
sc_fifo_out	class	Supported	
sc_fifo	class	Not Supported Now	sc_fifo::trace() and sc_fifo::operator = are not supported by the out-of-order simulation in the current release.
sc_find_event	function	Undecided	Work on this function in the future
sc_find_object	function	Undecided	Work on this function in the future
sc_fix_fast	class	Undecided	Work on this class in the future
sc_fix	class	Supported	
sc_fixed_fast	class template	Undecided	Work on this class template in the future
sc_fixed	class template	Supported	

Table 3: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset (continued)

Name	Type	Supported or not	Notes
SC_FORK	macro	Undecided	Work on this macro in the future
sc_fxcast_context	class	Undecided	Work on this class in the future
sc_fxcast_switch	class	Undecided	Work on this class in the future
sc_fxnum_bitref	class	Undecided	Work on this class in the future
sc_fxnum_fast_bitref	class	Undecided	Work on this class in the future
sc_fxnum_fast_subref	class	Undecided	Work on this class in the future
sc_fxnum_fast	class	Undecided	Work on this class in the future
sc_fxnum_subref	class	Undecided	Work on this class in the future
sc_fxnum	class	Supported	
sc_fxtype_context	class	Undecided	Work on this class in the future
sc_fxtype_params	class	Undecided	Work on this class in the future
sc_fxval_fast	class	Undecided	Work on this class in the future
sc_fxval	class	Undecided	Work on this class in the future
sc_gen_unique_name	function	Undecided	Work on this function in the future
sc_generic_base	class	Undecided	Work on this class in the future
sc_get_curr_process_handle	function (deprecated)	Supported	
sc_get_current_process_handle	function	Supported	
sc_get_default_time_unit	function (deprecated)	Supported	
sc_get_status	function	Supported	
sc_get_stop_mode	function	Supported	
sc_get_time_resolution	function	Supported	
sc_get_top_level_events	function	Undecided	Work on this function in the future
sc_get_top_level_objects	function	Undecided	Work on this function in the future
SC_HAS_PROCESS	macro	Supported	
sc_hierarchical_name_exists	function	Undecided	Work on this function in the future
sc_in_clk	typedef	Supported	
sc_in_resolved	class	Supported	
sc_in_rv	class	Supported	
sc_in	class	Supported	sc_in::add_trace() and other tracing functions are not supported by the out-of-order simulation in the current release.
sc_in<bool>	class	Supported	sc_in<bool>::add_trace() and other tracing functions are not supported by the out-of-order simulation in the current release.
sc_in<sc_dt::sc_logic>	class	Supported	sc_in<sc_dt::sc_logic>::add_trace() and other tracing functions are not supported by the out-of-order simulation in the current release.

Table 4: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset (continued)

Name	Type	Supported or not	Notes
sc_initialize	function (deprecated)	Supported	
sc_inout_clk	type (deprecated)	Supported	
sc_inout_resolved	class	Supported	
sc_inout_rv	class	Supported	
sc_inout	class	Supported	
sc_int_base	class	Supported	
sc_int_bitref_r	class	Undecided	Work on this class in the future
sc_int_bitref	class	Undecided	Work on this class in the future
sc_int	class template	Supported	
sc_interface	class	Supported	
sc_interrupt_here	function	Undecided	Work on this function in the future
sc_is_prerelease	function	Undecided	Work on this function in the future
SC_IS_PRERELEASE	macro	Supported	
sc_is_running	function	Supported	
sc_is_unwinding	function	Supported	
SC_JOIN	macro	Undecided	Work on this macro in the future
sc_length_context	class	Undecided	Work on this class in the future
sc_length_param	class	Undecided	Work on this class in the future
sc_logic	class	Undecided	Work on this class in the future
sc_lv_base	class	Undecided	Work on this class in the future
sc_lv	class template	Undecided	Work on this class template in the future
sc_main	function	Supported	
sc_max_time	function	Not Supported Now	This function is not supported by the out-of-order simulation in the current release.
sc_max	function	Supported	
sc_method_process	class	Supported	
SC_METHOD	macro	Supported	
sc_min	function	Supported	
sc_module_name	class	Supported	
sc_module	class	Supported	
SC_MODULE	macro	Supported	
sc_mutex_if	class	Not Supported Now	This class is not supported by the risc compiler in the current release.
sc_mutex	class	Not Supported Now	This class is not supported by the risc compiler in the current release.
sc_object	class	Supported	
sc_out_clk	type (deprecated)	Supported	

Table 5: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset (continued)

Name	**Type**	**Supported or not**	**Notes**
sc_out_resolved	class	Supported	
sc_out_rv	class	Supported	
sc_out	class	Supported	
sc_pause	function	Undecided	Work on this function in the future
sc_pending_activity_at_current_time	function	Undecided	Work on this function in the future
sc_pending_activity_at_future_time	function	Undecided	Work on this function in the future
sc_pending_activity	function	Undecided	Work on this function in the future
sc_phash	class (deprecated)	Undecided	Work on this class in the future
sc_plist	class (deprecated)	Undecided	Work on this class in the future
sc_port	class	Supported	
sc_port_base	class	Supported	
sc_ppq	class (deprecated)	Undecided	Work on this class in the future
sc_prim_channel	class	Supported	sc_prim_channel::update() is not supported by the out-of-order simulation in the current release.
sc_process_b	type (deprecated)	Supported	
sc_process_handle	class	Supported	
sc_pvector	class (deprecated)	Undecided	Work on this class in the future
sc_ref	macro	Undecided	Work on this macro in the future
sc_release	function	Supported	
sc_report_handler_proc	typedef	Undecided	Work on this typedef in the future
sc_report_handler	class	Undecided	Work on this class in the future
sc_report	class	Undecided	Work on this class in the future
sc_semaphore_if	class	Not Supported Now	This class is not supported by the risc compiler in the current release.
sc_semaphore	class	Not Supported Now	This class is not supported by the risc compiler in the current release.
sc_sensitive_neg	class (deprecated)	Not Supported Now	This class is not supported by the risc compiler in the current release.
sc_sensitive_pos	class (deprecated)	Not Supported Now	This class is not supported by the risc compiler in the current release.
sc_sensitive	class	Not Supported Now	This class is not supported by the risc compiler in the current release.
sc_set_default_time_unit	function (deprecated)	Supported	
sc_set_stop_mode	function	Undecided	Work on this function in the future

Table 6: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset (continued)

Name	**Type**	**Supported or not**	**Notes**
sc_set_time_resolution	function	Supported	
sc_set_vcd_time_unit	member function (deprecated)	Undecided	Work on this function in the future
sc_signal_in_if	class	Supported	
sc_signal_in_if<bool>	class	Supported	
sc_signal_in_if<sc_logic>	class	Supported	
sc_signal_inout_if	class	Supported	
sc_signal_out_if	type (deprecated)	Supported	
sc_signal_resolved	class	Supported	
sc_signal_rv	class	Supported	
sc_signal_write_if	class	Supported	
sc_signal	class	Supported	sc_signal::trace() is not supported by the out-of-order simulation in the current release.
sc_signal<bool>	class	Supported	sc_signal<bool>::trace() is not supported by the out-of-order simulation in the current release.
sc_signal<sc_logic>	class	Supported	sc_signal<sc_logic>::trace() is not supported by the out-of-order simulation in the current release.
sc_signed_bitref_r	class	Undecided	Work on this class in the future
sc_signed_bitref	class	Undecided	Work on this class in the future
sc_signed_subref_r	class	Undecided	Work on this class in the future
sc_signed_subref	class	Undecided	Work on this class in the future
sc_signed	class	Supported	
sc_simcontext	class (deprecated)	Supported	sc_simcontext::initial_crunch(), cycle() and other functions are partially supported by the out-of-order simulation in the current release.
sc_simulation_time	function (deprecated)	Supported	
sc_spawn_options	class	Supported	
sc_spawn	function	Not Supported Now	sc_spawn() is not supported by the out-of-order simulation in the current release.
sc_start_of_simulation_invoked	function	Undecided	Work on this function in the future
sc_start	function	Supported	
sc_start(double)	function	Not Supported Now	This function is not supported by the out-of-order simulation in the current release.
sc_status	enumeration	Supported	

Table 7: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset (continued)

Name	Type	Supported or not	Notes
sc_stop_here	function	Undecided	Work on this function in the future
sc_stop	function	Undecided	Work on this function in the future
sc_string	class (deprecated)	Undecided	Work on this class in the future
sc_subref_r	class template	Undecided	Work on this class template in the future
sc_subref	class	Undecided	Work on this class in the future
sc_switch	enumeration	Supported	
sc_thread_process	class	Supported	
SC_THREAD	macro	Supported	
sc_time	class	Supported	
sc_time_stamp	function	Supported	
sc_time_to_pending_activity	function	Undecided	Work on this function in the future
sc_trace_delta_cycles	function (deprecated)	Undecided	Work on this function in the future
sc_trace_file	class	Undecided	Work on this class in the future
sc_trace	function	Undecided	Work on this function in the future
sc_ufix_fast	class	Undecided	Work on this class in the future
sc_ufix	class	Supported	
sc_ufixed_fast	class template	Undecided	Work on this class template in the future
sc_ufixed	class template	Supported	
sc_uint_base	class	Supported	
sc_uint_bitref_r	class	Undecided	Work on this class in the future
sc_uint_bitref	class	Undecided	Work on this class in the future
sc_uint_subref_r	class	Undecided	Work on this class in the future
sc_uint_subref	class	Undecided	Work on this class in the future
sc_uint	class template	Supported	
sc_unsigned_bitref_r	class	Undecided	Work on this class in the future
sc_unsigned_bitref	class	Undecided	Work on this class in the future
sc_unsigned_subref_r	class	Undecided	Work on this class in the future
sc_unsigned_subref	class	Undecided	Work on this class in the future
sc_unsigned	class	Supported	
sc_unwind_exception	class	Undecided	Work on this class in the future
sc_value_base	class	Undecided	Work on this class in the future
sc_vector_assembly	class	Undecided	Work on this class in the future
sc_vector_base	class	Undecided	Work on this class in the future
sc_vector	class	Undecided	Work on this class in the future
sc_version_major	function	Supported	
sc_version_minor	function	Supported	
sc_version_originator	function	Supported	
sc_version_patch	function	Supported	

Table 8: RISC V0.2.1 Out-of-Order Parallel Simulatable SystemC Subset (continued)

Name	Type	Supported or not	Notes
sc_version_prerelease	function	Supported	
sc_version_release_date	function	Supported	
sc_version_string	function	Supported	
sc_version	function	Supported	
wait	function	Supported	wait(sc_event_and_list), wait(sc_event_or_list), wait(void) are not supported by the risc compiler in the current release.
next_trigger	function	Not Supported Now	This function is not supported by the risc compiler in the current release.
halt	function	Not Supported Now	This function is not supported by the risc compiler in the current release.

in RISC. The root problem here lies in the elimination of explicit channels, which were a key contribution in the early days of research on system-level design [16, 17, 18]. As most researchers agreed, the concept of separation of concerns was of highest importance, and for system-level design in particular, this meant the clear separation of computation (in behaviors or modules) and communication (in channels).

Regrettably, SystemC TLM 2.0 chose to implement communication interfaces directly as sockets in modules [30] and this indifference between channels and modules thus breaks the assumption of communication being safely encapsulated in channels. Without such channels, there is very little opportunity for safe parallel execution.

At this point, it is unclear how this situation can be worked around or corrected. Thus, SystemC TLM 2.0 can currently not be supported by RISC.

4.4 SystemC Datatypes

A large part of the SystemC language covers special data types designed for bit-accurate hardware modeling, simulation time representation, and other ESL specifics. These SystemC data types include `sc_bigint`, `sc_biguint`, `sc_bit`, `sc_bv`, `sc_fix`, `sc_ufix`, `sc_fixed`, `sc_ufixed`, `sc_int`, `sc_uint`, `sc_logic`, and `sc_lv`.

While all these SystemC data types are available in RISC, only a few of them have been validated and tested for being safe in a truly parallel multi-threading context. At this point, RISC supports `sc_int`, `sc_uint`, `sc_fixed`, and `sc_ufixed` (which are MT-safe). All other data types are so far untested and may or may not be safely used in OoO PDES.

4.5 SystemC Utilities and Other Constructs

As listed in Table 1 through Table 8, there is a plethora of other SystemC APIs available. Some of these are easily supported in RISC (such as `sc_copyright`, `sc_version_major`, `sc_version_minor`, `sc_version_patch`, `sc_version`), others are not supported at this time, such as the SystemC built-in tracing features (`sc_trace`, `sc_trace_file`) and the end of simulation due to `sc_stop`.

At this point, there is also a large number of special SystemC constructs for which it is unclear whether or not these can be supported in an OoO PDES context with reasonable effort and efficiency. An example of such constructs are those functions which involve or allow to inspect the simulator state at run-time, such as `sc_find_event`, `sc_find_object`, `sc_get_current_process_handle`, `sc_get_status`, `sc_get_time_resolution`, `sc_get_top_level_events`, `sc_get_top_level_objects`, `sc_hierarchical_name_exists`, `sc_is_running`, `sc_is_unwinding`, `sc_simcontext`, and `sc_status`.

On the other hand, access to the current simulated time (`sc_time`, `sc_simulation_time`, `sc_delta_count`), an essential part of every SystemC model evaluation, is supported by RISC OoO PDES.

5 Conclusion

While SystemC is the de-facto and official standard language for ESL design, SystemC simulation largely is still performed sequentially following classic DES semantics. Thus, SystemC simulation cannot utilize the parallel processing capabilities available on today's multi- and many-core host computers.

In this report, we have described the Recoding Infrastructure for SystemC (RISC), an agressive simulation approach beyond traditional parallel DES, where a dedicated SystemC compiler and advanced parallel simulator implement Out-of-Order Parallel Discrete Event Simulation (OoO PDES) for SystemC. This approach promises to exploit parallel computing resources to the maximum extend and thus fastest simulation speed. At the same time, OoO PDES maintains the traditional SystemC modeling semantics.

At this time, this technical report documents the RISC Compiler and Simulator and details the SystemC subset supported by the RISC Alpha Release V0.2.1.

As we move on in the project, we will update this report and in particular the supported subset tables accordingly.

Acknowledgements

This work has been supported in part by substantial funding from Intel Corporation under an initial seed grant and a following three year grant for the project titled *"Out-of-Order Parallel Simulation of SystemC Virtual Platforms on Many-Core Architectures"*. The authors thank Intel Corporation for the valuable support and express special gratitude to Abhijit Davare, Ajit Dingankar and Desmond Kirkpatrick for fruitful discussions, productive feedback and invaluable insights.

References

[1] IEEE Computer Society. *IEEE Standard 1666-2011 for Standard SystemC Language Reference Manual*. IEEE, New York, USA, 2011.

[2] Accellera Systems Initiative. http://www.accellera.org.

[3] SystemC Language Working Group (LWG). http://accellera.org/activities/working-groups/systemc-language.

[4] SystemC Language Working Group. SystemC 2.3.1, Core SystemC Language and Examples. http://accellera.org/downloads/standards/systemc.

[5] Richard Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):30–53, Oct 1990.

[6] Christoph Schumacher, Rainer Leupers, Dietmar Petras, and Andreas Hoffmann. parSC: Synchronous Parallel SystemC Simulation on Multi-Core Host Architectures. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 241–246, 2010.

[7] Dukyoung Yun, Jinwoo Kim, Sungchan Kim, and Soonhoi Ha. Simulation Environment Configuration for Parallel Simulation of Multicore Embedded Systems. In *Proceedings of the Design Automation Conference (DAC)*, pages 345–350, 2011.

[8] Ezudheen P, Priya Chandran, Joy Chandra, Biju Puthur Simon, and Deepak Ravi. Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machines. In *PADS '09: Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 80–87, 2009.

[9] Rohit Sinha, Aayush Prakash, and Hiren D. Patel. Parallel simulation of mixed-abstraction SystemC models on GPUs and multicore CPUs. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 2012.

[10] Weiwei Chen, Xu Han, and Rainer Dömer. Multi-Core Simulation of Transaction Level Models using the System-on-Chip Environment. *IEEE Design and Test of Computers*, 28(3):20–31, May/June 2011.

[11] J.H. Weinstock, C. Schumacher, R. Leupers, G. Ascheid, and L. Tosoratto. Time-decoupled parallel systemc simulation. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Dresden, Germany, March 2014.

[12] Weiwei Chen, Xu Han, and Rainer Dömer. Out-of-Order Parallel Simulation for ESL Design. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, March 2012.

[13] Weiwei Chen and Rainer Dömer. An Optimizing Compiler for Out-of-Order Parallel ESL Simulation Exploiting Instance Isolation. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 461–466, February 2012.

[14] Weiwei Chen and Rainer Dömer. Optimized Out-of-Order Parallel Discrete Event Simulation using Predictions. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, March 2013.

[15] Weiwei Chen, Xu Han, Che-Wei Chang, Guantao Liu, and Rainer Dömer. Out-of-Order Parallel Discrete Event Simulation for Transaction Level Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 33(12):1859–1872, December 2014.

[16] Jianwen Zhu, Rainer Dömer, and Daniel D. Gajski. Syntax and semantics of the SpecC language. In *Proceedings of the International Symposium on System Synthesis*, Osaka, Japan, December 1997.

[17] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao. *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[18] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[19] Rainer Dömer, Andreas Gerstlauer, and Daniel Gajski. *SpecC Language Reference Manual, Version 2.0*. SpecC Technology Open Consortium, http://www.specc.org, December 2002.

[20] Open SystemC Initiative, http://www.systemc.org. *Functional Specification for SystemC 2.0*, 2000.

[21] Thorsten Grötker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[22] Guantao Liu, Tim Schmidt, and Rainer Doemer. Recoding Infrastructure for SystemC (RISC) Compiler and Simulator. http://www.cecs.uci.edu/~doemer/risc.html.

[23] Rainer Dömer, Weiwei Chen, Xu Han, and Andreas Gerstlauer. Multi-Core Parallel Simulation of System-Level Description Languages. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 311–316, January 2011.

[24] Anirudh Kaushik and Hiren D. Patel. SystemC-clang: An Open-source Framework for Analyzing Mixed-abstraction SystemC Models. In *Proceedings of the Forum on Specification and Design Languages (FDL)*, Paris, France, September 2013.

[25] Hiren Patel. ”SystemC-clang: SystemC parser using the clang front-end”. https://github.com/hdpatel/systemcclang.

[26] Tim Schmidt. Recoding Infrastructure for SystemC (RISC) API. http://www.cecs.uci.edu/~doemer/risc/html_risc_021/index.html.

[27] Frank Ghenassia. *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, 2005.

[28] Open SystemC Initiative (OSCI). *OSCI TLM-2.0 Language Reference Manual*. OSCI, July 2009.

[29] Guantao Liu. Out-of-Order Parallel SystemC API. http://www.cecs.uci.edu/~doemer/risc/html_oopsc_021/index.html.

[30] David C. Black. The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard. Tutorial at Design Automation Conference, San Francisco, California, June 2015.

A Appendix

A.1 Manual Page of the RISC Compiler and Simulator

NAME

risc – Recoding Infrastructure for SystemC (RISC) Compiler and Simulator

SYNOPSIS

risc [*options*] *design* [*options*]

DESCRIPTION

risc is a dedicated compiler for the SystemC language. The purpose of **risc** is to parse, analyze, instrument, and compile a SystemC source program into an executable program for out-of-order parallel simulation. **risc** is a frontend source-to-source compiler for SystemC built on top of the ROSE compiler infrastructure with GNU C++ as the backend target compiler. As such, **risc** relies on and supports also most of the ROSE and GNU compiler options.

Using the command syntax shown in the synopsis above, the specified *design* is compiled. By default, **risc** reads the SystemC source file, performs preprocessing and builds an internal representation (abstract syntax tree) and a Segment Graph (SG) of the model. Next, static conflict analysis is performed and the design model is instrumented for Out-of-Order Parallel Discrete Event Simulation (OoO PDES). Finally, instrumented C++ code is generated, compiled, and linked into an executable file that can be run for fast parallel simulation.

On successful completion, the exit value 0 is returned. In case of errors during processing, an error code with a brief diagnostic message is written to the standard error stream and the compilation is aborted with an exit value greater than zero (i.e. 10).

For preprocessing and C++ compilation, **risc** relies on the availability of an external C++ compiler which is used automatically in the background. By default, the GNU C++ compiler **g++** is used.

ARGUMENTS

design specifies the file name of the input SystemC design model; by default, the base name of *design* is used as base name for all intermediate and output files;

OPTIONS

–h | *—-help* print the compiler version and a brief usage information message to standard output and quit;

–v | *—-verbose* increment the verbosity level so that all tasks performed are logged to standard error (default: be silent); at level 1, high-level messages about the tasks performed are displayed; at level 2, additional details such as input and output file names are listed; at level 3, very detailed information about each executed task is printed;

–w | *—-warnings* increment the warning level so that compiler warning messages are enabled (default: warnings are disabled); four levels are supported ranging from only important warnings (level 1) to pedantic warnings (level 4); for most cases, warning level 2 is recommended (–w –w);

–g | *–g level*	add a symbol table suitable for debugging (e.g. gdb) to the generated simulation executable (default: no debugging symbols);
–O | *–O level*	optimize the generated simulation executable for higher execution speed and/or less memory usage (default: no optimization);
–Idir	add the specified *dir* to the include path (extend the list of directories to be searched for including source files); include directories are searched in the order of their specification; the standard include path ($SYSTEMC_LW_HOME/include or $SYSTEMC_OOP_HOME/include) is automatically appended to this list; by default, only the standard include directories are searched;
–Ldir	add the specified *dir* to the library path (extend the list of directories to be searched for linker libraries); the library path is searched in the specified order; the standard library path ($SYSTEMC_OOP_HOME/lib) is automatically appended to this list; by default, only the standard library path is searched;
–llib	add the specified *lib* to the list of libraries for the linker so that the executable is linked against *lib;* libraries are linked in the specified order; the standard libraries (i.e. -lsystemc) are automatically appended to this list; by default, only standard libraries are used;
–c	perform only the preprocessing, analysis, instrumentation, and compilation tasks; skip the final linking stage so that only an object file is created (default: perform all tasks including linking);
–o output file	specify the name of the final output file explicitly (default: a.out);
–rose:option	pass this option through to the underlying ROSE compiler (default: none);
–GNU option	pass this option through to the underlying GNU compiler (default: none);

ENVIRONMENT

RISC is used to determine the installation directory of the RISC compiler and simulator where the RISC system components are located.

SYSTEMC_LW_HOME is used to find the RISC light-weight SystemC header files (in directory $SYSTEMC_LW_HOME/include).

SYSTEMC_OOP_HOME is used to find the RISC OoO PDES SystemC header files (in directory $SYSTEMC_OOP_HOME/include) and the RISC OoO PDES SystemC library (in directory $SYSTEMC_OOP_HOME/lib).

VERSION

The RISC compiler and simulator is alpha release version 0.2.1.

AUTHORS

Tim Schmidt <schmidtt@uci.edu>, Guantao Liu <guantaol@uci.edu>, and Rainer Doemer <doemer@uci.edu>.

COPYRIGHT

LICENSE

BUGS, LIMITATIONS

Probably many, since this is an alpha release of a proof-of-concept prototype implementation.