Tackling an Abstraction Gap: Co-simulating SystemC DE with Bluespec ESL

Hiren D. Patel Sandeep K. Shukla
Virginia Polytechnic Institute and State University
Bradley Department of Electrical and Computer Engineering.

E-mail: hiren@vt.edu shukla@vt.edu

Abstract

The growing SystemC community for system level design exploration is a result of SystemC's capability of modeling at RTL and above RTL abstraction levels. However, managing shared state concurrency using multi-threading in large SystemC models is error prone. A recent extension of SystemC called Bluespec-SystemC (BS-ESL) counters this difficulty with its model of computation employing atomic rule-based specifications. However, for simulating a model that is partly designed in SystemC and partly using BS-ESL, an interoperability semantics and implementation of such a semantics is required. This paper views the interoperability problem as an abstraction gap closure problem. To illustrate the problem, we formalize the simulation semantics of BS-ESL and discrete-event simulation of RTL SystemC and provide a solution based on this formalization.

1. Introduction

Raising the abstraction level helps designers in modeling and simulating large, complex and highly concurrent designs. However, the lack of matured high-level synthesis tools make it difficult for system level design languages and frameworks (SLDLs) to cleanly fit into industry design flows going from modeling and simulation to one that leads to implementation. For synthesis, some industry design flows incorporate refinement and transformation methodologies but, some require manual rewriting of the high-level designs in RTL using traditional hardware description languages (HDL)s.

There are some industry tools such as Forte's Cynthesizer [1] and Mentor's Catapult [2] that support high-level synthesis from C/C++, SystemC RTL and some transaction-level (TL) constructs. However, synthesis from SystemC at TL is still at its infancy making designers use SystemC mainly for modeling and simulation at TL and not for synthesizable designs. Even though SystemC is commonly used at TL modeling and simulation, its use for RTL models is also common. This is exemplified by industry focus on synthesis tools such as Forte's Cynthesizer and Mentor's Catapult [2]. In addition, modeling in SystemC is not much different than using traditional HDLs at the RTL abstraction. Hence, modeling SystemC RTL has the same difficulty in expressing correct concurrent behaviors as traditional HDLs. This is usually done by multi-threading and synchronization constructs.

On the contrary, Bluespec-SystemC (BS-ESL) extension provides designers with a model of computation (MoC) that simplifies this task of describing concurrent behaviors based on the notion of "atomic" rules in the rule-based MoC of Bluespec [3, 4, 5, 6]. Moreover, Bluespec has synthesis from its Bluespec-SystemVerilog descriptions to Verilog [4, 5, 7] and a synthesis path from BS-ESL to Verilog. Both SystemC and BS-ESL can effectively model RTL. However, the manner in which designs are simulated are different. SystemC employs delta cycles, delta-events, and events for simulating its designs. BS-ESL creates a topologically sorted schedule of its rules for simulating its designs. The simulation semantics of the two languages are different because even when BS-ESL is modeling RTL, it maintains an operational abstraction over HDL based RTL. This gives rise to an impedance mismatch or "abstraction gap" if two models, one in BS-ESL and one in SystemC are connected directly. We refer to this abstraction gap as the difference in the modeling paradigms where SystemC requires designers to follow the discrete-event semantics and BS-ESL follows the rule-based semantics. We show that this mismatch leads to incorrect simulation behavior when a BS-ESL model is combinationally connected to a model using an HDL RTL such as SystemC. We use HDL RTL to mean any discrete-event (DE) simulation based HDL such as VHDL, Verilog or SystemC.

We isolate the problem in composing SystemC and BS-ESL models and propose techniques for resolving it. We do not alter the SystemC or BS-ESL kernels since the former is a standard and the latter an industry product. Instead, we propose solutions that designers can make in their models.

The examples in this paper are very simple to illustrate the basic issues, but our interoperability semantics has been employed in BS-ESL and works for much larger models. To our knowledge, the interoperability between Bluespec's rule-based MoC and SystemC's DE MoC has not been resolved before this. Although described in the context of SystemC RTL, our interoperability semantics holds for simulating traditional HDLs against Bluespec as well. For this reason, we restrict our analysis to only RTL models. However, these techniques are applicable to a certain subset of TL models (TLM) and not all, but this is beyond the scope of this paper.

Bluespec-SystemC (BS-ESL)

BS-ESL's MoC [8, 9] requires describing hardware using guarded "atomic" rules. The notion of "atomic" rules allows encapsulation of concurrent transactions into atomic units, thereby solving a number of concurrency synchronization issues present in thread-

978-3-9810801-2-4/DATE07 © 2007 EDAA

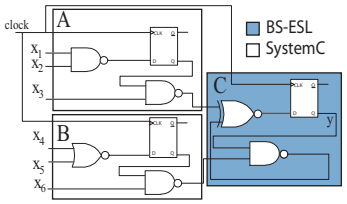


Figure 1. Example of SystemC & BS-ESL

based concurrent modeling such as SystemC. BS-ESL modules may contain multiple rules describing its behaviors and export interfaces by which other modules may query and interact with elements of the modules via methods. In addition, the basic language flavor of Bluespec SystemVerilog with the concepts of methods, interfaces, nested interfaces and a runtime executor are present in the BS-ESL extension [4, 7, 5, 9]. The runtime executor has two phases: the elaboration phase and the execution phase. During the elaboration phase, information about register accesses are used to compute *the order in* which rules may be triggered. For example, multiple rules whose guards evaluate to true, writing to the same state element must be scheduled in separate cycles. This in turn yields a topologically sorted schedule of rule execution. The execution phase fires the rules according to the schedule whereby if the guards of the rules evaluate to true, then their respective actions are triggered. At the end of one iteration of the schedule, all state elements such as registers are updated. Listing 1 describes a GCD module in BS-ESL. As an example, rule `rule_swap` swaps the register values of `x` and `y` if the value of `x` is greater than `y` and `y` is not zero. Similarly the `gcd` module has `rule_decr` and methods `start` and `result`. The authors of [9] provide more details on the constructs and syntax.

Listing 1. Module for GCD

```
1  BS_ESL_MODULE (gcd, gcd_if) {
2    bs_esl_reg <int> x, y;
3    bs_esl_reg <bool> notdone;
4    BS_ESL_RULE (rule_swap, (x > y) && (y != 0)) {
5      int tmp = x; x = y; y = tmp;
6    }
7    BS_ESL_RULE (rule_decr, (x <= y) && (y != 0)) {
8      y = y - x;
9    }
10   BS_ESL_METHOD_ACTION (start, ((y == 0) && !notdone), int num1, int
         num2) {
11     assert ((num1 > 0) && (num2 > 0));
12     x = num1; y = num2; notdone = true;
13   }
14   BS_ESL_METHOD_ACTIONVALUE (result, ((y == 0) && notdone), int) {
15     notdone = false;
16     return x;
17   }
18   BS_ESL_CTOR (gcd) {
19     x.bind      (new bs_esl_mkReg<int> (0));
20     y.bind      (new bs_esl_mkReg<int> (0));
21     notdone.bind (new  bs_esl_mkReg<bool> (false));
22     ESL_END_CTOR;
23   }
24 };
```

Interoperability Issues at RTL

Note that SystemC follows discrete-event (DE) [10] semantics, which does not mandate any specified order of execution of the SystemC processes. As already mentioned, BS-ESL follows the rule-based semantics with a topologically sorted order of rules for execution. So, two designs at RTL will simulate differently. To provide a brief overview of this, take Figure 1 as an example of a simple circuit. For illustration purposes, suppose that all three components were modeled as SystemC processes. The order in which any of the processes are triggered does not impact the final result `y` because *delta-events* [10] are used for changes that must be processed within a cycle. Therefore, the state elements in Figure 1 receive the correct values. Now, suppose that component `C` is modeled using BS-ESL and the firing order of the processes is `A`, `C` then `B`. `A` generates one input which causes `C` to fire. This invokes BS-ESL's runtime executor that fires the rules in the topologically sorted order and updates the state element at the end of the iteration. The result in the state element contains incorrect values because the correct input from `B` is yet to arrive. So, when `B` executes, `C` will refire using the incorrect value in the state element. This occurs because delta-events are not employed in BS-ESL's semantics. This mismatch is a result of the *abstraction gap* between SystemC and BS-ESL.

Interoperability Issues at TL

We present another example of a `GCD` calculator that illustrates the issue of mismatched simulation semantics with TLM. Note that only the `GCD` and `Display` components are clocked and the transfer between the components is via transactions; hence modeled at the cycle-accurate TL abstraction as per the OSCI TLM [10] standards. Figure 2 shows two input generators that act as external data entry points arriving at any time. The `GCD` calculator contains three internal registers to store the two inputs and the result. It performs an iteration of BS-ESL's runtime executor on every clock edge and writes the result when an appropriate condition is met. This register is read by the `Display` component. If the input generators produce new values within the same clock cycle after `GCD` has already been triggered then the computation must be redone with the "latest" input values. Otherwise the registers will hold incorrect values. Furthermore, BS-ESL does not have a notion or interface for delta-events so a direct composition is not possible. We follow this informal description of the issues with a formal presentation in the remainder of the paper focusing only on the RTL scenario.

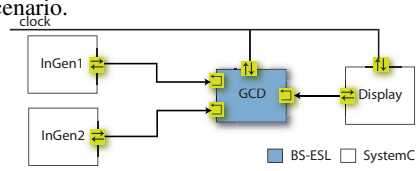


Figure 2. GCD example at TL

2. Related work & Background

Interoperability in Ptolemy II

A pioneer in heterogeneous modeling and simulation for embedded software synthesis is the Ptolemy II project [11]. Ptolemy II supports heterogeneous hierarchy where components following different MoCs can interoperate. Their implementation of the interaction semantics (synonymous to interoperability semantics) describes their approach in mixing MoCs such as synchronous data flow with DE, finite state machine with DE, synchronous reactive with DE and various other combinations [12, 13]. There are other studies that investigate interoperability amongst MoCs but few that match Ptolemy II's history with heterogeneity.

Even though Ptolemy II has done extensive studies in interoperability between MoCs, it is specific to their MoC implementations. For example, algorithms used for the DE MoCs in Ptolemy II [11] and SystemC [10] simulation are different. Ptolemy II's DE performs a topological sort on the actors and verifies that no zero-delay loop exists without a delay actor. This sort is also used in deciding the order in which actors are fired when simultaneous events are received. SystemC on the other hand does no topological sort of its processes and handles simultaneous events as delta-events. This is because SystemC needs to be compiled with standard C++ compilers without any changes and hence static analysis and reordering is not possible. Due to the distinct design of the DE MoCs, the interoperability semantics with other MoCs are expected to be different. Further, Ptolemy II's interaction semantics are not absolute and there can be other possibilities as well. This is acknowledged by the authors in [13] where they suggest that their interaction semantics are often the ones that seem "sensible" for their purpose, but there can be other different interoperability semantics between MoCs.

Interoperability with SystemC

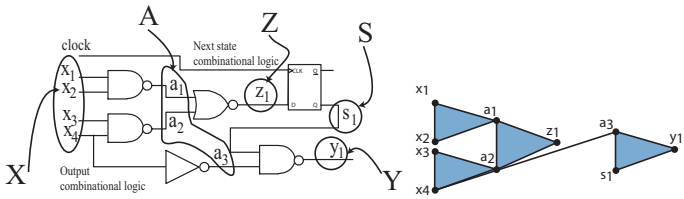
The OSCI TL modeling standard for SystemC is commonly used for prescribing interoperability between models at different levels of abstractions [10, 14]. The TLM standard provides a set of channels and interfaces to allow easier TL modeling. It also provides some common structures such as routers and arbiters. The abstraction levels are generalized to timed and untimed [14] after which clocked transactors are used between timed and untimed models. The transactors convert timed signals to transactions and according to the timing requirements of the timed model, decipher the transactions into pin-accurate signals. Other uses of TLM for interoperability also implicitly use *transactors* to convert hardware signals to events for software models. OSCI's master-slave library built on the TLM standard can also be used for co-simulating models at mixed abstractions [15].

The authors in [16, 17] support heterogeneity in SystemC via kernel-level extensions for MoCs that interoperate with SystemC. Their approach involves wrapping the model representing a different MoC with a SystemC process such that all SystemC communication channels are connected to this toplevel process. The data from these channels is converted to the respective MoC's input and the MoC's kernel is triggered. We use some of the techniques from this approach such as the wrapping of the BS-ESL components for interoperability.

3. Problem Description & Solution

Now that we have informally discussed the problem of interoperability between SystemC and BS-ESL. We present a formal treatment of the problem and solution. We omit the formal proofs for brevity. We begin by formalizing the notion of a simulation behavior, which is the snapshot of register values at the beginning of each clock cycle (Definition 3.1).

DEFINITION 3.1. *Simulation behavior S_b of a model M at RTL level is defined as a function as follows: $S_b : N \rightarrow S \rightarrow V$, where $N = \{0, 1, 2, ...\}$ denotes the sequence of clock cycles, S is the set of registers (states) and V is the domain of values from which the registers take values. We use S_b^M to denote the simulation behavior of a model M.*



(a) Circuit (b) Hypergraph

Figure 3. Example circuit

DE Simulation Semantics for RTL

This section formalizes an RTL circuit description and how current DE simulation semantics employ the use of delta cycles to compute correct outputs. Note that the DE simulation semantics considers a combinational cycle as an incorrect description. This is because combinational cycles in hardware imply an infinite execution, which in simulation may result in errors. Therefore, our formal treatment disallows combinational cycles. We also formally present Bluespec's BS-ESL MoC and show how direct composition of the two may not result in the expected simulation behavior. During our formal presentation we slightly abuse notation and write $\hat{x} \in X$ to mean $\hat{x} \in X^k$ for some k and $\hat{u} \in VARS \backslash Y$ to mean $\hat{u}$ is a vector of variables where each element belongs to $VARS \backslash Y$.

Let

$X = \{x_1, x_2, ..., x_k\}$	be the set of input signals
$Y = \{y_1, y_2, ..., y_l\}$	be the set of output signals
$A = \{a_1, a_2, ..., a_p\}$	be the set of intermediate combinational signals
$Z = \{z_1, z_2, ..., z_n\}$	be the set of internal combinational signals input to the state elements
$S = \{s_1, s_2, ..., s_n\}$	be the set of signals representing state element values
$VARS = X \cup A \cup Z \cup Y \cup S$	be the set of all variables

These sets are shown for the circuit in Figure 3(a). F_c represents the combinational logic of the circuit as a set of equations where

$$\begin{aligned} F_c = \{ &< z_m = f_m^z(\hat{u}) > \mid m = 1, 2, ..., n\} \\ &\cup \{< y_j = f_j^y(\hat{v}) > \mid j = 1, 2, ..., l\} \\ &\cup \{< a_i = f_i^a(\hat{w}) > \mid i = 1, 2, ..., p \\ &\qquad \text{and } \hat{w} \text{ does not contain } a_i\} \end{aligned}$$

and $\hat{u}$, $\hat{v}$, $\hat{w}$ are vectors of variables over the set $(VARS \backslash Y) \backslash Z$. The next state assignments $F_s = \{< s_k \Leftarrow z_k > \mid k = 1, 2, ..., n\}$ are described by delayed assignments. Delayed assignments update the value in the state elements at the beginning of the next clock cycle or at the end of the current cycle.

In hardware, signals propagate from input and current state through the combinational logic within one clock cycle and produce the outputs and latch the next state onto the state elements. Figure 3(a) describes an RTL circuit. However, simulating the combinational part amounts to a number of function computations, which are not necessarily described in their topological order in the actual circuit. Therefore, most DE simulators need to reevaluate

the functions several times using delta steps before all the combinational values are correctly computed.

We formalize any RTL design as a tuple $M =< X, Y, A, Z, S, F_s, F_c >$. F_c syntactically represents all the computation that happens during one clock cycle in the combinational part of the circuit. F_s captures the change of state at the end of the combinational computation and signal propagation. One can capture the execution semantics of a static circuit description $< X, Y, A, Z, S, F_s, F_c >$ using an acyclic hypergraph model. A hypergraph is a graph in which generalized edges may connect more than two vertices. So a hypergraph $G = (V, E)$ has $E \subseteq \bigcup_{k=1}^{|V|} V^k$, which means an edge may connect up to $|V|$ vertices.

The way to visualize the execution semantics in Figure 3(b) of one cycle worth of combinational computation of an RTL design is as follows: consider a directed hypergraph where a hyperedge of the form $< a, b, c, d, e >$ would mean the nodes a, b, c, d together are all connected to node e. So $G_H = (V, E)$ where $V = X \cup Y \cup A \cup Z \cup S$ (variables from S can only be used as inputs) and

$$< \hat{u}, z_m > \in E \quad \text{iff } < z_m = f_m^z(\hat{u}) > \in F_c$$
$$< \hat{v}, y_j > \in E \quad \text{iff } < y_j = f_j^y(\hat{v}) > \in F_c$$
$$< \hat{w}, a_i > \in E \quad \text{iff } < a_i = f_i^a(\hat{w}) > \in F_c$$

In G_H, each hyperedge is marked with the equational assignment in F_c that gave rise to the hyperedge. The following observations are obvious:

OBSERVATION 3.2. *(1) G_H is an acyclic hypergraph for combinational loop-free RTL designs, (2) G_H can be topologically sorted, and (3) for simulation purposes, if we evaluate the variables in their topological sorted order in G_H, we obtain the correct values without having to reevaluate any of the nodes.*

Figure 3(b) shows a hypergraph representation of the circuit description in Figure 3(a). Notice that $s1$ is used only as an input to the output $y1$. From Figure 3(b) it is obvious that a circuit such as this can be topologically sorted. However, most RTL simulators, including SystemC do not topologically sort the design and therefore, notions of delta cycles and delta-events are used to get the same effect. However, they achieve the same effect as can be shown by proving the following theorem (omitted for less of space):

THEOREM 3.3. *Let a model M_1 simulated using delta cycle semantics have the simulation behavior $S_b^{M_1}$. Let the same model evaluated by topologically ordering the hypergraph be M_2 and has the simulation behavior $S_b^{M_2}$. Then, $\forall s_j \in S, \forall i \in N$, $(S_b^{M_1} i)s_j = (S_b^{M_2} i)s_j$.*

Even though DE simulators have delta cycles, they do not go ad infinitum when the combinational logic is correctly designed because delta cycles impose a fixed point computation. To show this formally, consider $< x_1, x_2, ..., x_k, s_1, s_2, ...s_n >$ as a vector of variables that are immutable during simulation of a single cycle. Let $< a_1, a_2, ...a_p, z_1, z_2, ..., z_t, y_1, y_2, ...y_l >$ be variables which can possibly change several times during successive delta cycles. Let their initial values during the beginning of the evaluation process be *don't cares*, which means their values are arbitrary.

Now reconsider graph G_H with the variables (nodes) in the graph being annotated with their order in the topological sort. So variables in $(VARS \backslash X) \backslash S$ can be sorted in the topological order and we rename them as

$$w_1, w_2, ..., w_p, w_{p+1}, ..., w_{p+t}, w_{p+t+1}, ..., w_{p+t+l}$$

The following observation easily follows:

OBSERVATION 3.4. *If topologically ordered before simulation then, (1) once the equation of the form $w_1 = f_1(\hat{u})$ is evaluated, it no longer needs reevaluation and (2) once $w_j = f_j(\hat{u})$ is evaluated and $\forall i < j$ and $w_i = f_i(\hat{u})$ has already been evaluated, w_j does not need reevaluation.*

By Theorem 3.3 and Observation 3.4 it follows that even with arbitrary order of evaluation, the process of delta cycle terminates when no w_k is needed to be reevaluated. At this point, the delayed assignments in F_s can be evaluated and that ends one simulation cycle. This shows that delta cycles actually amount to a fix point computation for $< a_1, a_2, ..., a_p, z_1, z_2, ..., z_t, y_1, y_2, ...y_l >$ where applying one equation of F_c changes only one variable. This change in variable leads us to a new valuation, which is better (in an information order) than the old valuation or the same if it is the final correct valuation.

The use of delta-events is important for correctly simulating hardware in DE-based simulators. However, BS-ESL's rule-based MoC semantics simulate their designs differently, without the need of delta-events. We describe this next.

BS-ESL's Rule-based Semantics

A BS-ESL model is described as a 2-tuple $M = < V, R >$ of variables where V is the set of variables and R a set of rules. The rules are guarded commands of the form $g(\Gamma) \rightarrow b(\Lambda)$ where $\Gamma, \Lambda \subseteq V$, $g(\Gamma)$ is a Boolean function on the variables of Γ and $b(\Lambda)$ is a set of equations of the form $x_i = f_i(\Lambda)$ such that x_i is assigned the result of computing the value based on current values of variables in Λ. Using the definitions presented earlier, the variable set can be partitioned into four disjoint sets $V = X \cup Y \cup A \cup S$. Now, each rule $r_i \in R$ computes $b_i(\Gamma)$ and computes a number of functions $\{v_{ij} = f_{ij}(\hat{w}) \mid v_{ij} \in \Lambda \ \ and \ \hat{w} \in (VARS \backslash Y) \backslash Z\}$ where v_{ij} could be any variable. Now, we denote a syntactic model M_E equivalent to M, where $M_E =< X, Y, A, S, F_c >$. This means there is no distinction between Z and S and there is no F_s. This is because Bluespec's MoC topologically sorts the rules based on usage and assignment of variables, and since a correct RTL level Bluespec design cannot have combinational loops, the variable dependency graph is acyclic; hence topologically sortable. As a result, the state variables are assigned once and only once, and there is no delta cycle based reassignment of these. Therefore, the simulation semantics of Bluespec's rule-based MoC is different with respect to HDL RTL simulation semantics.

Composing HDL RTL models

Let M_1 and M_2 be two RTL models defined as

$$M_1 = \ < X_1, Y_1, A_1, Z_1, S_1, F_S^1, F_c^1 >$$
$$M_2 = \ < X_2, Y_2, A_2, Z_2, S_2, F_S^2, F_c^2 >$$

Note that M_1 and M_2 are both HDL RTL.

We compose them as $M_1 \oplus M_2$ in the simulation model such that $M_1 \oplus M_2 =< X, Y, A, Z, S, F_s, F_c >$. The way the composition works is that a subset of input variables of one module are fed by some of the output variables of the other module and vice versa. So $X_1 \cap Y_2$ represents the set of all output variables of M_2 that are fed into input of M_1 and $X_2 \cap Y_1$ represents the set of all inputs in M_2 fed by outputs of Y_1. Of course, to do this correctly, this should not result in any combinational loop. Also, $S_1 \cap X_2$ represents the set of state elements from M_1 feeding into M_2's inputs and likewise $S_2 \cap X_1$ feeds the inputs for M_1. So,

$$
\begin{aligned}
X = \;& (X_1 \cup X_2) - ((X_1 \cap Y_2) \cup (X_2 \cap Y_1) \\
& \cup (S_1 \cap X_2) \cup (S_2 \cap X_1)) \\
Y = \;& (Y_1 \cup Y_2) - ((X_1 \cap Y_2) \cup (X_2 \cap Y_1)) \\
Z = \;& Z_1 \cup Z_2 \\
A = \;& A_1 \cup A_2 \cup ((X_1 \cap Y_2) \cup (X_2 \cap Y_1)) \\
S = \;& S_1 \cup S_2,\ F_s = F_s^1 \cup F_s^2,\ F_c = F_c^1 \cup F_c^2
\end{aligned}
$$

Note the following:

OBSERVATION 3.5. *A correct composition $M_1 \oplus M_2$ circuit has no combinational loops.*

Let M_1, M_2 both be RTL models or M_1, M_2 both be BS-ESL models. There are two cases covered in the next theorem.

THEOREM 3.6. *Let the composed model $M = M_1 \oplus M_2$ be (1) simulated using delta cycle simulators and the simulation behavior is $S_{b\Delta}^{M_1 \oplus M_2} : N \rightarrow S_1 \cup S_2 \rightarrow V$. (2) simulated using BS-ESL then $S_{b*}^{M_1 \oplus M_2} : N \rightarrow S_1 \cup S_2 \rightarrow V$. Then $\forall s_j \in S_1 \cup S_2, \forall i \in N, \forall \tau \in \{\Delta, *\}$*

$$
(S_{b\tau}^{M_1 \oplus M_2} i)s_j = \begin{cases} (S_{b\tau}^{M_1} i)s_j & \text{if } s_j \in S_1 \\ (S_{b\tau}^{M_2} i)s_j & \text{if } s_j \in S_2 \\ (S_{b\tau}^{M_1} i)s_j = (S_{b\tau}^{M_2} i)s_j & \text{if } s_j \in S_1 \cap S_2 \end{cases}
$$

Theorem 3.6 states that the simulation of M is correct when M_1 and M_2 are both modeled using delta cycle semantics and simulated using a delta cycle simulator. Likewise, the simulation of M is correct if both models are modeled using rule-based semantics and simulated using the rule-based MoC.

Composing HDL RTL & BS-ESL Models

Now we consider composing two models where M_1 is in HDL RTL and M_2 in BS-ESL. We show that the inherent MoCs of these two SLDLs do not allow for direct composition of the models.

$$
\begin{aligned}
M_1 = \;& < X_1, Y_1, A_1, Z_1, S_1, F_S^1, F_c^1 > \\
M_2 = \;& < V_2, R_2 > \\
& \text{such that } V_2 = X_2 \cup Y_2 \cup A_2 \cup S_2.
\end{aligned}
$$

Suppose we create a composition $M = M_1 \oplus M_2$ and ensure there are no combinational cycles as shown in the simple example in Figure 1. Then, M_2 is simulated using BS-ESL simulation semantics by topologically sorting its rules, and M_1 is simulated using any HDL's simulation semantics in DE with delta cycles. So, we claim that the result of simulation for M may differ from the correct result of simulating them if M_1 was first topologically sorted and then composed.

THEOREM 3.7. *If M_1 and M_2 has combinational path between them and M_1 is simulated using delta cycles, M_2 with rule-based execution then it may not be the case that $\forall i \in N, \forall s_j \in S_2$, $(S_b^{M_1 \oplus M_2} i)s_j = (S_b^{M_2} i)s_j$.*

As stated before, since $M_2 =< V_2, R_2 >$ corresponds to an execution model $M_{2E} =< X_2, Y_2, A_2, S_2, F_c^2 >$ where F_c^2 contains all assignments to variables including those in S_2. So, $M_1 \oplus M_2$ composition and topologically sorting the corresponding hypergraph will lead to $G_{H(M_1 \oplus M_{2E})}$. Figure 1 shows an example of a composed model. Unfortunately, now we can no longer prove that delta cycle based evaluation of $M_1 \oplus M_2$ will have the same values as hypergraph based evaluation. The reason is simple to see. If there is combinational path from M_1 and M_2, and during simulation M_2 is evaluated first based on rule-based semantics, the values of its state variables will change permanently for that cycle. This would mean that during reevaluation of M_1, if delta-events trigger recomputation of M_2, then the state variables would contribute their new values and not the values from the previous cycles.

Table 1. Execution trace for RNG

Sim. Time	HDL RTL	RTL & BS-ESL
t	$< 0, 0, 0, 0, 79 >$	$< 0, 0, 0, 0, 79 >$
$t + \Delta$	$< 263, 2, 0, 79 >$	$< 263, 2, 0, 0, 0 >$
$t + 2 * \Delta$	$< 263, 71, 0, 0, 79 >$	$< 263, 71, 263, 2, 2 >$
$t + 1$	$< 263, 71, 263, 71, 48 >$	$< 263, 71, 263, 71, 97 >$
$t + 2$	$< 263, 71, 263, 71, 95 >$	$< 263, 71, 263, 71, 82 >$
$t + 3$	$< 263, 71, 263, 71, 56 >$	$< 263, 71, 263, 71, 37 >$

We use Table 1 to show the execution trace of a composition of RTL and BS-ESL models for a random number generator (RNG) example. The vector describes values for $< p_1, p_2, P1_{Reg}, P2_{Reg}, result_{Reg} >$, where p_1 and p_2 are input signals, variables subscripted with Reg are registers internal to the component, $k*\Delta$ represents the k^{th} delta-event and $t + 1$ represents the first cycle after time t. The seed for the RNG is specified as the initial value for $result_{Reg}$ as 79 and the random numbers are computed using $result_{Reg} := (P1_{Reg} * result_{Reg}) + P2_{Reg}$. For the HDL RTL case, the inputs arrive at different delta-events that are stored in their respective registers in the following clock cycle. From there on, the RNG computation iterates and stores the result. Similarly, the inputs arrive in delta cycles to the composed situation except that due to BS-ESL's ordered schedule and lack of delta-events, registers are updated at the end of the iteration. Hence, the $result_{Reg}$ contains an incorrect value which is continuously used for subsequent random numbers. This is incorrect behavior for a RNG because for the same seed and constants $P1$ and $P2$, the sequence of random numbers generated must be the same.

Our Interoperability Technique

Having identified that a simple minded composition of HDL RTL (M_1) and BS-ESL (M_2) is not correct, we use a technique that enables correct composition.

HDL RTL with BS-ESL RTL

Suppose M_1 and M_2 where $M_1 =< X_1, Y_1, A_1, Z_1, S_1, F_S^1, F_c^1 >$ is an HDL RTL model and $M_2 =< V_2, R_2 >$ such that V_2 is described by $< X_2, Y_2, A_2, S_2, F_c^2 >$ is an BS-ESL model. Now, we must introduce a wrapper such that the composition $M_1 \oplus w(M_2)$ is possible. The wrapper is defined as $w(X_2, Y_2, A_2, S_2, F_c^2) =< X^w, Y^w, A^w, Z^w, S^w, F_s^w, F_c^w >$ where

$$
\begin{aligned}
X^w &= X_2,\ Y^w = Y_2,\ A^w = A_2,\ S^w = S_2 \\
Z^w &= \{z_i | s_i \in S_2\} \\
F_s^w &= \{< s_i \Leftarrow z_i | z_i \in Z^w >\} \\
F_c^w &= \text{ with each occurrence of } s_i \text{ replaced by } z_i
\end{aligned}
$$

The wrapper introduces intermediate combinational signals Z^w that take inputs from A^w and serve as inputs to the state elements S^w. This is indicative of adding temporary variables in M_2 whose inputs are signals from Z^w and the outputs are to the state elements. At every execution of M_2 the result of the combinational

circuit is assigned to the temporary variables. The state elements are then assigned values of z_i as per delayed assignment semantics. In addition, when there is a combinational path between the models $(X_1 \cap Y^w) \cup (X^w \cap Y_1)$, the updates on the signals must occur in delta cycles. Once again, since M_2 does not have the notion of delta cycles, we execute M_2 at every value change on its combinational inputs. The simulation is correct because only the temporary variables are updated with these executions and not the state elements. The state elements are then only updated at the beginning of the next clock cycle so as to retain correct simulation behavior.

The simulation behavior for M_1 is $S_b^{M_1}$ and M_2 is $S_b^{M_2}$ as per Definition 3.1. Then, using our wrapper we can avoid the problem presented in Theorem 3.7.

THEOREM 3.8. $\forall i \in N, \forall s_j \in S_1 \cup S_2$

$$(S_b^{M_1 \oplus w(M_2)} i)s_j = \begin{cases} (S_b^{M_1} i)s_j & \text{if } s_j \in S_1 \\ (S_b^{M_2} i)s_j & \text{if } s_j \in S_2 \\ (S_b^{M_1} i)s_j = (S_b^{M_2} i)s_j & \text{if } s_j \in S_1 \cap S_2 \end{cases}$$

Our implementation uses a SystemC process as the wrapper process. This wrapper process interacts with other external SystemC processes and handles conversions from SystemC channels to the BS-ESL interfaces. For the wrapper process to trigger on every delta-event, we introduce Δ-channels. These channels internally employ the sc_event_queue in SystemC to notify events, but we allow for transferring data along with every event generated as well. We make the wrapper process sensitive to these Δ-channels such that whenever there is an event on the channels, the wrapper process is scheduled for execution within the same clock cycle.

3.1 Examples

We implement the GCD and RNG examples in SystemC and in the BS-ESL co-simulation environment (labeled as composed). Our experiments showed that the correct co-simulation of composed SystemC and BS-ESL models resulted in an approximately 40% simulation performance degradation over pure SystemC simulation. This is an inevitable price for the advantage of correct by construction concurrency management in BS-ESL. The overhead in the composed models is the scheduling, data-structure construction and firing. Furthermore, the implementation in SystemC amounts to a single function call whereas in BS-ESL multiple rules are used. The overhead caused by our solution (duplicating state elements) described in this paper is a fractional amount, approximately 1% of the total overhead (results omitted for space).

Table 2. Simulation times for examples

	GCD (s)		RNG (s)	
Samples	SystemC	Composed	SystemC	Composed
100000	8.0	13.2	8.62	14.7
200000	15.8	26.5	17.4	29.6
300000	24.2	39.8	26.0	43.9

4. Conclusion

This work formally presents the DE simulation semantics as implemented in any HDL including SystemC, followed by BS-ESL's rule-based semantics. Then, we argue that direct composition of the two MoCs does not provide correct simulation because of their inherently distinct MoCs. Traditional HDLs employ delta-events and delta cycles for recomputing certain functions in the designs whereas the rule-based MoC executes its rules in an ordered manner without requiring recomputation. Using the analysis, we identify the problem with state updates during co-simulation and then present one technique that allows designers to correctly co-simulate models using the two MoCs. Our solution proposes adding a wrapper process and duplicating state elements in the BS-ESL model such that all computation is performed using the temporary state elements and the actual state elements are updated at the beginning of the next clock cycle. One of the main thrusts of this paper is to show how wrapper generation should be based in formal model based reasoning as opposed to ad-hoc ideas, as is often done for co-simulation.

5. References

[1] FORTE, "Forte Design Systems," Website: http://www.forteds.com/.

[2] Mentor Graphics, "Catapult C Synthesis," Website:http://www.mentor.com/.

[3] D. Rosenband and Arvind, "Hardware synthesis from guarded atomic actions with performance specifications," *Proceedings of ICCAD'05*, pp. 784–791, November 2005.

[4] ——, "Modular Scheduling of Guarded Atomic Actions," *Proceedings of the Design Automation Conference*, June 2004.

[5] J. C. Hoe and Arvind, "Hardware Synthesis from Term Rewriting Systems," *Proceeding of VLSI'99 Lisbon, Portugal*, December 1999.

[6] M. Pellauer, M. Lis, D. Baltus, and R. Nikhil, "Synthesis of Synchronous Assertions with Guarded Atomic Actions," in *In Formal Methods and Models for Codesign*, 2005.

[7] N. Dave, M. C. Ng, and Arvind, "Automatic Synthesis of Cache-Coherence Protocol Processors Using Bluespec," *Proceedings of Formal Methods and Models for Codesign*, July 2005.

[8] Bluespec, "Bluespec-SystemC Release," Website:http://bluespec.com/products/ESLSynthesisExtensions.htm.

[9] H. D. Patel, S. K. Shukla, E. Mednick, and R. Nikhil, "A rule-based model of computation for systemc: Integrating systemc and bluespec for co-design," in *Proceedings of MEMOCODE*, 2006.

[10] OSCI, "SystemC," Website: http://www.systemc.org.

[11] E. Lee et al, "Heterogeneous Concurrent Modeling and Design in Java: Introduction to Ptolemy II," Memorandum UCB/ERL M03/27, July 2003.

[12] A. Girault and B. Lee and and E. A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, 1999, pp. 742–760.

[13] W. Chang and S. Ha and E. A. Lee, "Heterogeneous Simulation - Mixing DE Models with Dataflow," in *Journal of VLSI Signal Processing*, 1997, pp. 127–144.

[14] F. Ghenassia, *Transaction-Level Modeling with SystemC*. Springer, 2005.

[15] A. Sayinta, G. Canverdi, M. Pauwels, A. Alshawa, and W. Dehaene, "A Mixed Abstraction Level Co-Simulation Case Study Using SystemC for System on Chip Verification ," in *Proceedings of DATE*, 2003.

[16] H. D. Patel and S. K. Shukla, "Towards a heterogeneous simulation kernel for system level models: A systemc kernel for synchronous data flow models," in *IEEE Transactions in Computer-Aided Design*, vol. 24, August 2005.

[17] ——, "Heterogeneous Behavioral Hierarchy for System Level Designs," in *to appear in IEEE Transaction on CAD*, 2006.