Design with Race-free Hardware Semantics

Patrick Schaumont[1], Sandeep Shukla[1], Ingrid Verbauwhede[2]
[1]Electrical and Computer Engineering Department, Virginia Tech
[2]ESAT, Katholieke Universiteit Leuven, Belgium

Abstract

Most hardware description languages do not enforce determinacy, meaning that they may yield races. Race conditions pose a problem for the implementation, verification, and validation of hardware. Enforcing determinacy at the modeling level provides a solution to this problem. In this paper, we consider a common model of computation for hardware modeling — a network of cycle-true finite-state-machines with datapaths (FSMDs) — and we identify the conditions under which such models are guaranteed to be race-free. We base our analysis on the Kahn Principle and a formal framework to represent FSMD semantics. We present our conclusions as four simple and easy to enforce modeling rules. A hardware designer that applies those four modeling rules, will thus obtain race-free hardware.

1 Motivation

In traditional HDL semantics (VHDL, Verilog or SystemC), race conditions occur when a single global variable is concurrently assigned by different processes. The value stored in the global variable is indeterminate. This will show up as 'X' in a four-state simulator, but often the result is simply simulator dependent.

While non-determinism by itself can be useful as a specification mechanism at higher abstraction level [1], in HDL-based design, non-determinism often sneaks in as a side effect. For example, an HDL designer can create race conditions without being aware of it [2]. This poses a challenge to the implementation, verification and comprehension of the RTL code [3]. Therefore we feel that irrespective of the language of choice for RTL coding (SystemC, Verilog or VHDL), the desired model of computation for such coding should avoid race conditions by design.

We will consider a hardware model based on a network of finite-state-machine-datapaths (FSMDs), which execute in a cycle-true fashion. Such a model can be easily constructed in various description languages, including SystemC. Our contribution will be to enumerate the conditions under which this network becomes race-free. The proof involves two observations. First, we show that individual hardware modules modeled as a single FSMD are race-free.

Table 1: Motivating race-free hardware semantics.

Software Design	Hardware Design	
Actual Property (using C)	Actual Property (using HDL)	Desired Property (this paper)
Determinate (for ANSI-C)	Non-determinate	Determinate (race-free)
Regular-time (instruction-driven)	Irregular-time (event-driven)	Regular-time (cycle-true)
Implementation-oriented	Simulation-oriented	Implementation-oriented

And second, we also show that the composition of those modules in a network remains race-free. The latter property corresponds to the Kahn Principle. This system-level determinism is not a trivial consequence of FSMD-level determinism. For example, in SHIM [4], system-level determinacy is a specific objective for developing a new system design semantics.

There is also a second, broader motivation for developing a race-free hardware design mechanism. Currently there is a broad interest from academia and industry to bring the design practice of hardware and software closer together [5]. In addition, contemporary 'software' can take many forms, going from C for cores to RTL for Field Programmable Gate Arrays (FPGA). It is common sense to insist on a uniform value concept (e.g. 2-state) for all programmable elements in a design.

However, the cultural differences in hardware and software design have led to basically different modeling approaches. Table 1 compares contemporary practice in hardware design (using HDL semantics) and software design (using ANSI C). Three key differences pointed out in Table 1 are: HDL models may be non-determinate, they model progression of time in an irregular fashion using events, and they model a simulation rather than an implementation. We are therefore interested in hardware models (a) that will always be determinate, (b) that model time in regular increments and (c) that are implementation-oriented. The FSMD model that we are using has each of these three properties, even though this paper focuses on the deterministic property.

1.1 Paper Organization

After a brief introduction to the Kahn Principle in section 2, we present a formal model of the components and interactions in an closed FSMD network in Section 3. Such a closed network does not interact with the environment. In Section 4, we express FSMD networks in a formal framework known as input-output automata (IOA) [6]. We show that an individual FSMD corresponds to an individual IOA, and that FSMD networks map into an equivalent composition of IOA. While IOA in general are non-determinate, it has been shown that under certain restrictions they become determinate, and that they satisfy the Kahn Principle. These restrictions are used in Section 5 to derive four conditions that will make FSMD networks determinate and race-free. In Section 6, we present our results and in Section 7 we review related work in this area.

2 The Kahn Principle

If we consider Kahn Process Networks, i.e. dataflow-like networks of deterministic sequential processes connected through infinite FIFO's, the Kahn Principle states that such networks are deterministic [7]. Such a network will deliver the same output under the same input, regardless of the scheduling order of individual processes. Moreover, the network can be substituted by a single function equal to the smallest fixed point of a suitable operator derived from the network specification. The Kahn Principle was shown to be applicable to several process semantics besides Kahn Process Networks. These include (a restricted form of) input-output automata (IOA) [8], synchronous programs [9], and deterministic receptive processes [4]. Especially the result for IOA is relevant for this paper, as we will show that FSMD precisely implement such restricted IOA. Note also that our FSMD communicate synchronously, and thus cannot be modeled as classic Kahn Processes with infinite-FIFO communication. In the next section, we present a formal basis for FSMD networks.

3 FSMD networks

This section provides a definition of a FSMD and a FSMD network.

3.1 Definition of a FSMD network

Definition 1. *A FSMD M is a pair <D,C> where D is a datapath with internal states, and C is a controller state machine as defined below. A datapath D can be described as a 4-tuple* $D = \langle I, O, V, F \rangle$ *where I is the set of input signals, O is the set of output signals, V are names of state elements and F is a set of functions. Any function* $f \in F$ *takes as input a subset of the current values of the input signals and state elements. It produces as outputs a subset of the output signals and a subset of the next-state of the state variables.*

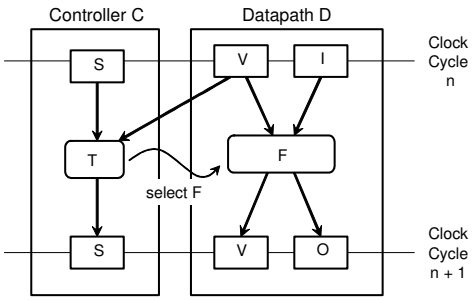

Figure 1: A single cycle of FSMD operation.

To formally define the nature of the set of datapath function set F, we need to define a few notations. Let $\delta(a)$ denote the set of values that an entity $a \in A$ can take. For example, if we consider an input signal $i \in I$, then $\delta(i)$ may be the set of all 16-bit integers, or the set of Booleans, etc. Now, consider a vector

$$\langle x_1, x_2, \ldots, x_n \rangle \in A^n$$

The set of values that this vector can take is

$$\delta(x_1) \times \delta(x_2) \times \ldots \times \delta(x_n)$$

For notational convenience, we denote by $\delta(A)$: the set of values that a vector of any size constructed from elements of A can take. In other words,

$$\delta(A) = \bigcup_{1 \le i \le |A|} \prod_{1 \le j \le i} \delta(x_j)$$

where $x_j \in A$. Further, let A_k denote a subset of k elements of A. Then $\delta(A_k)$ is a subset of $\delta(A)$ where all vectors are of length k. We can now define the function set F as follows.

Definition 2. *F is a set of functions where each function* $f \in F$ *only depends on a subset of I and a subset of V, and produces values to be assigned to a subset of signals in O and subset of state elements in V.*

$$F = \{ f \mid f: \delta(I_k) \times \delta(V_l) \rightarrow \delta(O_m) \times \delta(V_n), \\ 0 \le k \le |I|, 0 \le l \le |V|, 0 \le n \le |O|, 0 \le m \le |V| \}$$

Since the execution semantics of a datapath is related to the controller associated with it, we will define the execution semantics of a controller and datapath together once we have formally defined the controller.

Definition 3. *A controller C is a 4-tuple* $C = \langle D, S, \alpha, T \rangle$ *where* $D = \langle I, O, V, F \rangle$ *is a datapath as described above that is associated with the controller, S is a set of state ele-*

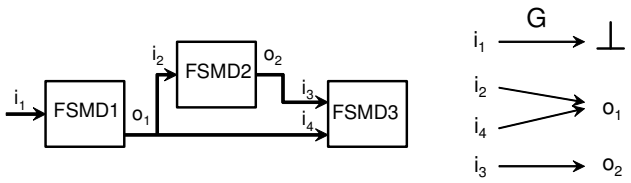

Figure 2: FSMD network and connection graph.

ments, α is the initial state and T is the transition function defined as

$$T{:}\delta(V) \times S \rightarrow F \times S$$

From this definition one can see that the controller's transitions are dependent on the current values of the datapath state elements, and its own state. When the transition takes place, it selects a function $f \in F$, and executes it while changing its own state to a new state. This is symbolically represented in Figure 1.

The execution semantics of FSMD *M* will be defined in terms of an execution sequence.

Definition 4. *An execution sequence is a sequence of pairs from $\delta(V) \times S$. V is the set of state elements in the datapath and S is the set of states in the controller. The first element in that sequence is the initial state, which is equal to* $s_\alpha = \langle 0^{|V|}, \alpha \rangle$.

Thus, initially all datapath registers are zero, and the controller assumes the initial state α.

Given an element of the execution sequence, and the values at each input, one can uniquely determine what the corresponding values of the output signals would be. If the values on all input signals during that cycle are expressed as *i*, and the execution state is $\langle v, s \rangle$, then one can determine the output as follows. Let $T(v, s) = \langle f, s' \rangle$ where $f \in F$. Then the output *o* will be given by $f(i, v) = \langle o, v' \rangle$. Here, the pair $\langle v', s' \rangle$ represents the new execution state, that will be used to evaluate the output of the next clock cycle.

A network of FSMD can now be defined as a set of FSMD M_i and a bipartite graph *G*, called the connection graph. Let I_A be the set of all FSMD input signals and O_A the set of all outputs:

$$I_A = \bigcup_i I_i, \quad O_A = \bigcup_i O_i$$

The nodes of the connection graph *G* are given by $I_A \cup O_A \cup \{\perp\}$. The edges *E* of *G* are such that if $(x, y) \in E$, then $x \in I_A$ and $y \in O_A \cup \{\perp\}$. The special value $\perp$ is used to represent unconnected inputs with an edge $(x, \perp)$ in the connection graph. The meaning of an edge in *E* is that the value of input *x* is instantaneously defined by the value of output *y*.

We further constrain the connection graph such that any input can connect only to a single output, while a single output may connect to multiple inputs (Figure 2). Thus if $(x, y) \in E$ and $(x, y') \in E$, then *y* and y' must be identical.

The definitions of FSMD and FSMD network lead to the following conclusion:

Theorem 1: *Given a closed FSMD network consisting of a set of FSMDs and a connection graph G, in the absence of combinatorial loops, any output $o \in O_A$ at any point in the execution of this network is only dependent on the state of the execution sequence up to that point.*

Proof. According to the execution semantics, the values of the outputs *o* of a FSMD are defined by the execution state $\langle v, s \rangle$ of that FSMD, and the values *i* at the inputs of the FSMD. The values of the inputs *i* are defined, through the connection graph, by some other FSMD outputs or $\perp$. We can thus find the values of those inputs using a recursive argument. Note that $\perp$ evaluates to itself: $f(\perp) = \perp$. During this recursion, we must encounter a datapath function with empty *I*, for which the outputs *o* only depend on the execution state. If not, the recursive expansion will never end and we have a combinatorial loop. If on the other hand we do find an empty *I*, then it follows that the outputs of all FSMD depend only on the set of execution states in the network, and possibly $\perp$. ■

In the next section, we will map the FSMD model on an underlying formalism called input-output automata. This will allow us to derive formally the conditions under which a FSMD is determinate, and thus under which a FSMD network can satisfy the Kahn Principle.

4 Mapping FSMD into I/O automata

In this section we map the FSMD model into input-output automata, a model proposed by Lynch and Tuttle that captures concurrent, distributed discrete-event systems [6].

4.1 FSMD = FSM + FSM

The first thing to realize is that a FSMD actually consists of two synchronous FSM, arranged on top of each other. One FSM implements the datapath, and the other implements the controller. Each of these FSM has its own state-space and state transition functions, but they are able to observe each others' state. In addition, the inputs and outputs are restricted to the datapath FSM.

The composition of a datapath FSM and a controller FSM therefore corresponds to a FSM. This joint FSM implements a state transition function *H* over the combined state space of the datapath and the controller:

$$H{:}\delta(I_k) \times \delta(V_l) \times S \rightarrow \delta(O_m) \times \delta(V_n) \times S$$

The partitioning between datapath and controller is, in fact, driven by pragmatic concerns rather than formal ones. Indeed, datapaths are typically designed using expressions, while controllers are designed with state transition diagrams. Datapath state-transition logic tends to be regular, while controller state-transition logic is irregular. The state in a datapath is present already at the highest abstraction level of a specification, such as for example in the taps of an FIR filter. The state in a controller on the other hand is the result of the scheduling of an algorithm onto a time axis.

Davio has shown that a FSMD can always be transformed by moving logic from the controller (FSM) part to the datapath (D) part, and vice versa [10].

4.2 Synchronous FSM as input-output automata

An input/output automaton is a formal model for a discrete event system of concurrent components [6]. An input/output automaton (IOA) consists of four components (We leave the equivalence relation on non-input actions out of consideration).

1. an action signature *acts(A)*, containing input actions, output actions and internal actions.
2. a set of states *states(A)*.
3. a set of start states $start(A) \in states(A)$.
4. A transition relation *steps(A)*

$$steps(A) \subseteq states(A) \times acts(A) \times states(A)$$

An action *a* consist of a signature and a set of operations. The signature of an action forms the interface with the environment. When an action executes, its signature is communicated with the environment, and the operations associated with that action signature execute. The response of an IOA to an input action signature is instantaneous. An output action signature of an IOA transmits instantaneously to all IOA that have the same action included in their input action signature.

An action *a* needs to be enabled before it can execute. Action *a* is enabled in state *q* when there exists a state transition *(q,a,r)* in *steps(A)*. Because of nature of input actions, the transition relation must be such that for each input action *a* and each state *q*, there exists a *transition(q,a,r)* in *steps(A)*.

Lynch uses this framework to define port-automata [8]. These are IOA in which the input and output action signatures have the special form $P \times N$, with *P* a set of ports and *N* a set of values. Communication then proceeds as output actions *(p,n)* of an IOA generate a corresponding input action *(p,n)* on another IOA.

The FSM model defined above can be expressed as a port automaton. This is done by modeling the clock as an input action, and by creating port-input and port-output actions for each FSM port.

5 Determinate FSMD networks

5.1 Properties of determinate IOA

Lynch derives the conditions under which networks of port automata become determinate. She uses the notion of an input/output relation on a port automaton, which is a set of pairs *(Hist(P_{in}), Hist(P_{out}))*. *Hist* corresponds to the history of values on a port. She goes on to show that, if the input/output relation of each port automaton in the network is single-valued, then the network of port automata is determinate. A single-valued input/output relation means that for each possible input-port history, there is only a single possible output-port history. In the next subsection, we consider the consequence of single-valued input/output relations to the FSMD model.

5.2 Determinate FSMD networks

In a FSM model, the history of an input port reflects onto the histories of the output port and the state variables by means of the state transition function *H*. In a FSM network, an input port history will correspond to the output port history of the output connected to this input. From Theorem 1, we know that a loop-free network only depends on the execution state, and thus that the single-valuedness of the outputs only depends on the single-valuedness of the execution state. This leads to the definition of *proper* FSMD network.

Definition 5. *A proper FSMD network is an FSMD network for which the execution sequence of all FSMD in the network is single valued, excluding the value* $\perp$.

Proper FSMD networks are determinate and race-free. We now can define four rules under which a FSMD network will have single-valued execution sequences. In the following, we will assume that the datapath function *f* in a particular clock cycle is composed as a subset of functions, expressed as g_1, g_2,..., g_K. Each of the functions g_i have their own set of inputs and outputs, formed by datapath inputs and outputs, as well as an internal pool of signals *L*. For example, the signals that hold intermediate values of datapath expressions are part of *L*. The functions g_1, g_2,..., g_K can be expressed with the same δ notation introduced for *f*, provided that we take this extra pool of signals *L* into account:

$$g_i : \delta(I_k) \times \delta(V_l) \times \delta(L_p) \rightarrow \delta(O_m) \times \delta(V_n) \times \delta(L_q),$$

A FSMD in a proper FSMD network must have the following four properties:

Single-assignment. A signal or register can be assigned only once during a clock cycle. Datapath behaviors are not

allowed to execute together when they would violate this condition. Let the set of output signals and registers of g_i be given by

$$O(g_i) = O_{m_i} \cup V_{n_i} \cup L_{q_i}$$

The single-assignment condition requires that

$$\forall g_i, g_j \in f, j \neq i \Rightarrow O(g_i) \cap O(g_j) = \phi$$

This condition implies that instructions cannot execute together if they would imply multiple assignments on a single variable. This rule prevents races to originate in the datapath of individual FSMD.

No undefined operands. At any particular clock cycle, all operands of an expression inside of any g_i must be defined. In particular, dangling signals in L are not allowed. All signals from the signal pool L that are consumed (in L_p) must also be produced (in L_q). Thus, if

$$f = \bigcup_K g_i$$

then

$$\bigcup_K L_{p_i} = \bigcup_K L_{q_i}$$

Another way of stating this is that all signals created in L during a clock cycle must also be consumed; they do not have an existence outside of f nor outside of that clock cycle. This rule prevents undefined values to originate in the datapath of individual FSMD.

No combinational loops. Cyclic data precedences are not allowed for any signals except state variables. For example if, during a particular clock cycle, signal `a` is used to define signal `b`, then `b` cannot be used to define signal `a`. This condition holds only for signals from I, O, and L, and not for state variables in V. This rule prevents false state in the FSMD models. All state variables will be explicitly visible and modeled as registers.

No undefined output signals. All outputs of a FSMD must be defined at all clock cycles. This ensures that the value $\perp$ will not show up in the port history of a connected FSMD. As stated, this condition is slightly too strict, since it is possible that in a given FSMD network, an output will not be read by the input port it is connected to.

When a FSMD satisfies the above four properties, it is part of a proper FSMD because it will never generate unknown signal values and it will never create internal races. While these conditions are sufficient to obtain proper FSMD network, we have not shown that all of these are also necessary. Thus, these four rules represent one possible solution to obtaining a proper FSMD network.

The four rules can be statically checked on the FSMD model, by extracting the signal sets I, O, V, and L from the FSMD datapath and by intersecting them according to schedule given in the FSMD controller. The meaning of a proper FSMD network can be understood as follows. In an HDL simulation, an indeterminate result in the simulation shows up as an unknown ('X'). In simulations with proper FSMD networks, this 'X' does not occur. In addition, because the initial value of registers is defined to be zero, undefined values ('U') won't occur either.

6 Results

We have implemented FSMD semantics as described above in a design language called GEZEL [11], and we have also build a VHDL code generator that converts GEZEL code into an implementation. The resulting VHDL code is race-free, obviously because of the semantic properties of the initial model. Presently, this design environment is in active use within a design group for digital design in the area of embedded security, targeting ASIC and FPGA. Several demonstrators have been completed that have relied on GEZEL-generated VHDL code [12]. A second area where the language has been useful is in education, for undergraduate [13] as well as graduate teaching.

We also emphasize that the formal model developed in this contribution is language-independent, and can as well be applied to Verilog, VHDL, and SystemC. In that case one will need to work with modeling conventions and/or build tools that will check the source code for 'proper FSMD'.

7 Related Work

Most contemporary HDL do not obey the Kahn Principle, and races are a fact of life for HDL designers. Many of them rely on guidelines such as [14] that list a set of conventions leading to circuits with desirable characteristics (including freedom of races). Those conventions translate to similar conditions as the four rules we enumerated for FSMD networks, with one important difference: in our model, they can be automatically enforced rather than being a simple convention.

SHIM [4] targets a unified design environment for deterministic hardware/software systems, and is based on Kahn Processes with rendez-vous communication. By making use of rendez-vous communication, the requirement for infinite-storage on the communication links of the Kahn Process Network is removed. SHIM processes are strictly sequential, meaning that communication links for individual processes will operate sequentially. In contrast, a proper FSMD network allows the communication links on individual FSMD to operate in parallel. A second difference is that SHIM allows processes to operate asynchronously. The proper FSMD model that we proposed is synchronous.

Despite these differences, there is a shared fundamental design consideration in both SHIM and proper FSMD, which is that design using deterministic semantics is more convenient. SHIM approaches this design consideration in a top-down fashion. Proper FSMDs, on the other hand, do this bottom-up.

Bluespec [15] organizes state variables inside of modules and then employs a rule-based coding style for the behavior of the module. Concurrent execution of rules is allowed provided that the result is equivalent to sequential execution of single rules. By enforcing this constraint of equivalence to a sequential execution, and by checking race-free properties within the context of a single rule, race-free behavior for the entire circuit is obtained. This is similar to the FSMD model, where only a single state transition can be taken per clock cycle, and where race-free properties are checked in the context of a single FSMD state transition. In fact, by encoding the FSM state variable and any state transition conditions as Bluespec rule conditions, the FSMD model could be expressed as a rule model.

Some other languages use correct-by-construction rules or compiler techniques to guarantee hardware determinism. In the case of Handel-C for example, concurrent access to global variables is restricted [16].

Finally, we should point to an interesting feature of SystemVerilog, one of the new proposals in the ESL language space. SystemVerilog supports 2-state data types [17]. Such a model only captures 0 or 1 (and no longer deals with 'X' or 'Z'). This new data type is proposed to support 2-state design methodologies. Indeed, for complex design the advantage of an 'X' as a don't care in logic synthesis no longer weighs up to the added complexity in verification. In fact, the 'X' is only a crude representation for the possible values in the set (1,0). Consider the following Verilog example by Turpin [3].

```
assign b = a & ~a;
```

This expression will always assign the value 0 to b. When `a` contains 'X' however, the resulting simulated value of b will still be 'X'. In this case, the simulator looses information on the actual value because it cannot distinguish between 'X' and its complement..

8 Conclusions

We have presented a race-free hardware design model, based on FSMD semantics. Building on the Kahn Principle and the properties of determinate Input-Output Automata, we have defined four rules that will help the designer in the modeling of race-free FSMD. The absence of race conditions eases the validation. We have implemented the race-free FSMD model in the GEZEL design environment, which we use in hardware design and validation, typically in the context of cosimulation. We are working on additional hardware language mappings out of this model, by means of code generators that translate GEZEL code into another model. Besides VHDL, we expect to be able to generate Verilog and SystemC.

9 Acknowledgements

This research has been supported through SRC grant SRC-JH-1116.

10 References

[1] S. Edwards, L. Lavagno, E. Lee, A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis," Proceedings of the IEEE, 85(3):366—390, 1997.

[2] OSCI, "Non-determinism in SystemC", Ch. 5.6 in "Functional Specification for SystemC 2.0", v. 2.0-Q, [Online]. Available: http://www.systemc.org

[3] M. Turpin, "The dangers of living with an X", presented at Synopsys Users Group Meeting, Boston, 2003.

[4] S. Edwards, O. Tardieu, "SHIM: A Deterministic Model for Heterogeneous Embedded Systems," IEEE Proc. EMSOFT 2005.

[5] A. Jerraya, W. Wolf, "Hardware/software interface codesign for embedded systems," IEEE Computer, 38(2):63—69, Feb 2005.

[6] N. Lynch, E. Tuttle, "An Introduction to Input/Output automata," CWI-Quarterly, 2(3):219—246, September 1989.

[7] G. Kahn, "The semantics of a simple language for parallel programming," Information Processing 74, 1974.

[8] N. Lynch, K. Stark, "A proof of the Kahn principle for input/output automata," Information and Computation archive, 82(1):81—92, 1989.

[9] D. Potop, "The Kahn principle for networks of synchronous endochronous," Proc. of FMGALS 2003.

[10] M. Davio, L. Deschamps, M. Thayse, "Digital Systems with Algortihm Implementation," Wiley and Sons, 1983.

[11] P. Schaumont, D. Ching, I. Verbauwhede, "An Interactive Codesign Environment for Domain-specific Coprocessors," ACM Transactions on Design Automation of Electronic Systems, to appear.

[12] D. Hwang, K. Tiri, A. Hodjat, B. Lai, S. Yang, P. Schaumont, I. Verbauwhede, "AES-Based Cryptographic and Biometric Security Coprocessor IC in 0.18-um CMOS Resistant to Side-Channel Power Analysis Attacks," 2005 Symposium on VLSI Circuits, Kyoto.

[13] J. Steensgaard-Madsen, J. Madsen, "Introduction to Codesign", [Online]. Available: http://www.imm.dtu.dk/courses/02130/02130/web/

[14] C. Cummings, "Verilog Nonblocking Assignments With Delays, Myths & Mysteries," Synopsys User Group Meeting (SNUG), Boston, 2002.

[15] Arvind, R. Nikhil, D. Rosenband, N. Dave, "High-level Synthesis: An Essential Ingredient for Designing Complex ASICs," Proc. International Conference on Computer-Aided Design (ICCAD), 2005.

[16] Celoxica, "Handel-C," [Online]. Available: http://www.celoxoca.com

[17] C. Cummings, L. Bening, "SystemVerilog 2-State Simulation Performance and Verification Advantages", Synopsus User Group Meeting (SNUG), Boston, 2004.