On the Verification of a WiMax Design Using Symbolic Simulation

Salim Ismail Al-Akhras, Sofiène Tahar
ECE Department, Concordia University, Montreal, QC, Canada
{s_alakhr,tahar}@encs.concordia.ca

Gabriela Nicolescu
CSE Department, Ecole Polytechnique de Montreal, Montreal, QC, Canada
gabriela.nicolescu@polymtl.ca

Michel Langevin, Pierre Paulin
STMicroelectronics Inc., Ottawa, ON, Canada
{michel.langevin,pierre.paulin}@st.com

In top-down multi-level design methodologies, design descriptions at higher levels of abstraction are incrementally refined to the final realizations. Simulation based techniques have traditionally been used to verify that such model refinements do not change the design functionality. Unfortunately, with computer simulations it is not possible to completely check that a design transformation is correct in a reasonable amount of time, as the number of test patterns required to do so increase exponentially with the number of system state variables. In this paper, we propose a methodology for the verification of conformance of models generated at higher levels of abstraction in the design process to the design specifications. We model the system behavior using sequence of recurrence equations. We then use symbolic simulation together with equivalence checking and property checking techniques for design verification. Using our proposed method, we have verified the equivalence of three WiMax system models at different levels of design abstraction, and the correctness of various system properties on those models. Our symbolic modeling and verification experiments show that the proposed verification methodology provides performance advantage over its numerical counterpart.

1 Introduction

System verification at earlier design stages is extremely important as the cost of fixing bugs at later design stages is usually very high. At higher levels of abstraction, the function of state-of-the-art SoCs and embedded systems is described using programming languages such as C/C++. High level design support is still in its infancy and there is a real need for the development of efficient verification techniques at these higher levels. Traditional simulation based techniques compare simulated design outputs against expected outputs and determine if the design functionality is correct. Since, the number of test patterns required to exhaustively check a design increases exponentially with the number of system state variables, it is infeasible to verify an overall design using exhaustive simulations. Even with carefully selected test vectors designed to cover typical and corner cases, it is often impossible to rule out the presence of design bugs using simulations.

In this paper, we propose an efficient semi-formal verification method for the verification of designs at higher levels of abstraction. We use sequence of recurrence equations (SRE) [3] to mathematically model the system at various levels of abstraction. Then, we use symbolic simulations for the evaluation of design behavior in which multiple input values are encoded as symbols. Finally, we use the results of symbolic simulation on SRE system models along with an equivalence checking technique and an assertion based verification technique to address the verification problem. The use of formal methods, which are exhaustive by nature, guarantees 100% functional coverage of the design, and symbolic simulations

Adel Bouhoula, Tetsuo Ida and Fairouz Kamareddine (Eds.):
Symbolic Computation in Software Science 2012 (SCSS2012)
EPTCS 122, 2013, pp. 23–37, doi:10.4204/EPTCS.122.3

provides scalability. We apply this methodology to an industrial design - STMicroelectronics WiMax modem.

The rest of the paper is organized as follow. We first present related work in Section 2. We then present a brief introduction to Symbolic Simulation and Sequence of Recurrence Equations in Section 3. We describe our modeling and verification methodology in Section 4. Using our proposed methodology, we then verify the STMicroelectronics WiMax modem models in Section 4. We also present experimental results in this section. Finally, Section 5 concludes the paper and discusses possible directions for future work.

2 Related Work

Complex telecommunication hardware is being designed using top down multilevel design approaches. In [8], Deb *et al* propose a Transaction Level Modeling (TLM) based design methodology for refining C and MATLAB functional models of a DSP system into a realistic implementation. In [11], Fujita *et al* present a similar multi-level system design methodology which uses C/C++, SpecC [12] and SystemC [13] for describing the system. [5] describes a framework for refining functional descriptions to FPGA implementations and presents a physical layer implementation of a WiMax modem. In [15] and [17] STMicroelectronics presents a framework for DSP system design using a set of design refinements applied to C/C++ descriptions for a target system. In this paper, we use the WiMax modem implementation described in [15, 17]. [18] presents a WiMax design and verification kit for WiMax certified products. Chiang *et al* used SystemVerilog to validate the physical access layer of WiMax systems [7]. Both [7, 18] use computer simulations for the verification and suffer from numerical inaccuracies. Moreover, the absence of bugs cannot be guaranteed unless exhaustive simulations are performed. The verification of the designs at high levels of abstraction, such as the WiMax model, is still an open problem. In this paper, we focus on the verification of WiMax design at the function and the architecture level.

Matsumoto *et al* [14] present an equivalence checking method for two C descriptions using symbolic simulation and prove the equivalence of all variables in the descriptions. We use a similar concept in our proposed methodology to improve the comparison performance of the high level descriptions of hardware designs. In [1], Abdi *et al* describe a verification method based on model algebra. Systems are described as model algebra expressions. Equivalence of models is checked by proving correctness of model transformation based on a set of predefined rules. If these rules are not used in model transformation, then the correctness of model transformations cannot be proven. Moreover, this work focuses on the correctness of the transitions rather than the functional correctness of the transformed models themselves. In this paper, we propose a higher level symbolic simulation based technique that uses sequence of recurrence equations (SRE) and pattern matching. In [3], the notion of recurrence equation is extended to describe digital systems for formal verification purposes including support for mathematical reasoning based on symbolic algebra and recurrence equations.

In [4] and [20], Zaki *et al* used symbolic simulation and SRE to verify properties of continuous Analog and Mixed Signal (AMS) systems. They show that the speed of verification and the coverage of the verification can be enhanced using their method. This work in the AMS field is the inspiration of our work in the system level verification of digital systems.

3 Preliminaries

3.1 Sequence of Recurrence Equations (SRE)

A recurrence equation or a difference equation is the discrete version of an analogue differential equation. In conventional system analysis, recurrence equations are used in the definition of relations between consecutive elements of a sequence. In [3], the notion of recurrence equation is extended to describe digital circuits using the normal form: *generalized If-formula*.

Definition: Generalized If-formula In the context of symbolic expressions, the generalized If-formula is a class of expressions that extend recurrence equations to describe digital systems. Let K be a numerical domain $(\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{B})$, a generalized If-formula is one of the following:

- A variable $X_i(n)$ or a constant $C \in K$
- Any arithmetical operation $\alpha \in \{+, -, \times, \div\}$ between variables $X_i(n) \in K$
- A logical formula: any expression constructed using a set of variables $X_i(n) \in B$ and logical operators: *not, and, or, xor, nor . . .* etc.
- A comparison formula: any expression constructed using a set of $X_i(n) \in K$ and comparison operator $\alpha \in \{=, <>, <, =, >, =\}$.
- An expression *IF(X, Y, Z)*, where X is a logical formula or a comparison formula and *Y, Z* are any generalized If-formula. Here, *IF(x, y, z)*: $B \times K \times K \rightarrow K$ satisfies the axioms:

  IF(True , X, Y) = X

  IF(False, X, Y) = Y

Definition: A System of Recurrence Equations (SRE) Consider a set of variables $X_i(n) \in K, i \in V = 1...k, n \in Z$, an SRE is a system of the form:

$$X_i(n) = f_i(X_j(n-\gamma)), (j,\gamma) \in \varepsilon_i, \forall n \in Z$$

where $f_i(X_j(n-\gamma))$ is a generalized If-formula. The set ε_i is a finite non empty subset of $1...k \times N$. The integer γ is called the delay.

3.2 Symbolic Simulation

Symbolic simulation is a form of simulation where many possible executions of a system are considered simultaneously. The symbolic simulation described in this section relies on rewriting rules based on the algorithms developed in [3] for digital systems. In the context of functional programming and symbolic expressions, we define the following functions.

Definition: Substitution. Let u and t be two distinct terms, and x a variable. We call $x \rightarrow t$ a substitution rule. We use $Replace(u, x \rightarrow t)$, read replace in u any occurrence of x by t, to apply the rule $x \rightarrow t$ on the expression u. The function *Replace* can be generalized to include a list of rules. *ReplaceList* takes as arguments an expression *expr* and a list of substitution rules $\Re = \{\Re_1, \Re_2, ..., \Re_n\}$. It applies each rule sequentially on the expression. The symbolic simulation function $ReplaceRepeated(Expr; \Re)$ shown in the definition below is based on rewriting by repetitive substitution, which applies recursively a set of

rewriting rules $\Re$ on an expression $Expr$ until a fixpoint is reached.

Definition: Repetitive Substitution. $ReplaceRepeated(expr;\Re)$ applies a set of rules $\Re$ on an expression *expr* until a fixpoint is reached as shown in the next definition.

Definition: Substitution Fixpoint. A substitution fixpoint $FP(expr;\Re)$ is obtained, if: $Replace(expr,\Re) \equiv Replace(Replace(expr,\Re),\Re)$
Depending on the type of expressions, we distinguish the following kinds of rewriting rules:

- *Polynomial Symbolic Expressions* R_{Math}: are rules intended for the simplification of polynomial expressions $(\mathbb{R}^n[x])$.
- *Logical Symbolic Expressions* R_{Logic}: are rules intended for the simplification of Boolean expressions and to eliminate obvious ones like $(and(a,a) \rightarrow a)$ and $(not(not(a)) \rightarrow a)$.
- *If-formula Expressions* R_{IF}: are rules intended for the simplification of computations over If-formulae. The definition and properties of the *IF* function, like reduction and distribution, are defined as follows:

  IF Reduction: $IF(x;y;y) \rightarrow y$

  IF Distribution: $f(A1,...,IF(x,y,z),...,An) \rightarrow$

  $IF(x, f(A1,...,y,...,An), f(A1,...,z,...,An))$

3.2.1 Symbolic Simulation Algorithm.

The symbolic simulation algorithm used in the symbolic trace computation step is based on rewriting by substitution. The idea is to compute the symbolic execution trace of the SRE model after *n* simulation cycles. During each cycle, the symbolic expressions of each design object are computed using a set of simplification rules. This algorithm is based on repeated substitutions as defined in Algorithm 1. The algorithm repeatedly applies a set of substitution rules *R*, until a fixed point is reached.

Algorithm 1 Repetitive Substitution

1: $ReplaceRepeated(Expr;\Re)$;
2: *Begin*
3: **repeat**
4: $expr_1 = ReplaceList(expr,\Re)$
5: $expr = expr_1$
6: **until** $FP(expr_1,\Re)$
7: *End*

Three kinds of symbolic expressions are considered: Algebraic, Logical and If-formula expressions. Each kind is associated with a set of rewriting rules: R_{Math}, R_{Logic} and R_{IF}.

- **Algebraic expressions** R_{Math}**:** are Mathematica built-in rules intended for the simplification of polynomial expressions $(R^n[x])$.
- **Logical symbolic expressions** R_{Logic}**:** are rules intended for the simplification of Boolean expressions and to eliminate obvious ones like (and(a,a)$\rightarrow$ a) and (not(not(a)) $\rightarrow$ a).

- **If-formula expressions R_{IF} :** are rules intended for the simplification of computations over If-formulas. The definition and properties of the IF function, like reduction and distribution, are used.

We add to these rules the trace of the equation at time *n-1* that we consider as rewriting rules of the time (*n-1*) see [3] for more details.

3.2.2 Verification of Symbolic Traces

The result of the symbolic simulation is a set of expressions that represent the symbolic trace of the system after *n* cycles. The comparison of expressions is achieved using: *Pattern Matching* [10] and *Equational Theorem Proving* [6]. Pattern matching is used to check that expressions have the desired structure, to find relevant structure, and to substitute the matching part with other expressions. In Mathematica, it is presented as of a regular expression language (Mathematica pattern language) and a set of matching functions. The designer writes properties of the form: $P = verify\ (U_i, S_i(t_n))$ where U_i is a regular expression that describes the expected symbolic expression of a simulated object. $S_i(t_n)$ is the symbolic simulation result of the element S_i after t_n simulation cycles.

4 Proposed Verification Methodology

Figure 1 shows the proposed methodology. First, key system specifications (properties) and the design model at each level of abstraction is translated into a Sequence of Recurrence Equations (SRE)s. Then, two complementary verification processes are applied to these models. The first process uses symbolic simulation to verify the conformance of SRE models to key features of the system. The second process proves the functional equivalence of SRE models. These two processes are incrementally used to verify the correctness of design refinements, and thereby guaranteeing the verification coverage of the design refinements at each level of abstraction and at corner specification points in the design.

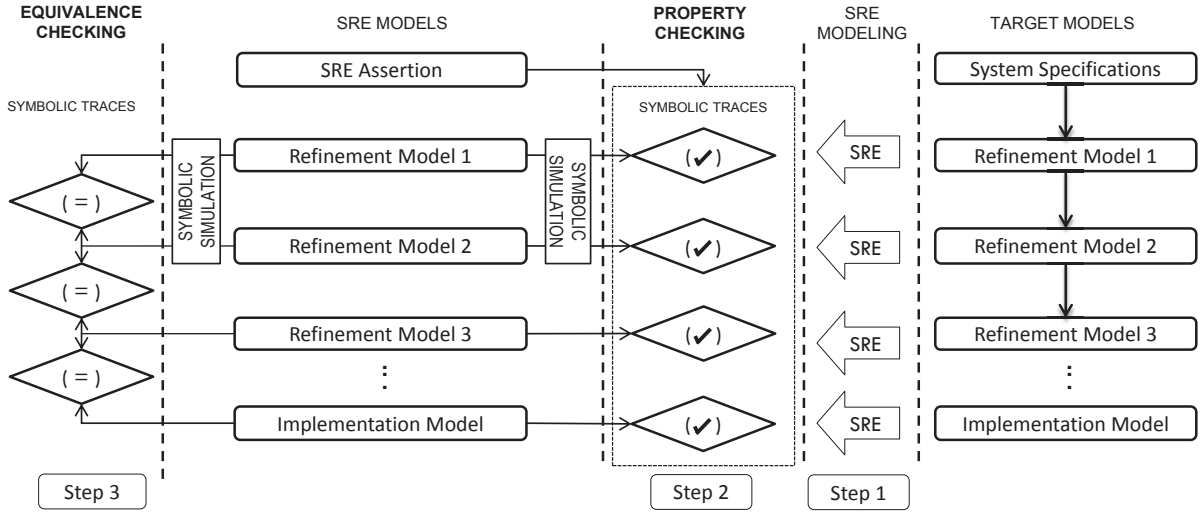

Figure 1: Proposed Verification Methodology Framework.

Our methodology aims to prove that a system description satisfies a set of properties using pattern matching and equation solving in Symbolic Algebra. This is achieved via several steps as shown in

Figure 1. The system is described using recurrence equations. The properties (or assertions) are algebraic relations between signals of the system. The system description and properties are input to a symbolic simulator that performs a set of transformations by using rewriting rules in order to obtain the symbolic traces. The next step is to use Pattern Matching and Equation solving in Symbolic Algebra to prove the conformance of the properties with the specific system description under test. If the proof is obtained, then the property is verified. Otherwise, we provide counterexamples for the non-proved properties.

4.1 Computational Equivalence Checking Algorithm

Algorithm 2 presents our Computational Equivalence checking algorithm, which we explain in the sequel.

Algorithm 2 Computational Equivalence Checking

1: t = t_0;
2: $\phi(t_0) = \{Spec_j(t_0)\}\ 0 < j \leq m$;
3: **while** $t \leq K_{Spec}$ **do**
4: $\phi(t) = SymSim\_Step(\phi)$
5: If NoDeltaCycle then t = t+1
6: **end while**
7: SPEC = $\phi(t_0 + K_{Spec})$
8: t = t_0;
9: $\varphi(t_0) = \{Imp_i(t_0)\}\ 0 < i \leq m$;
10: **while** $t \leq K_{Imp}$ **do**
11: $\varphi(t) = SymSim\_Step(\varphi)$
12: If NoDeltaCycle then t = t+1
13: **end while**
14: IMPL = ReplaceRepeated ($\varphi(t_0 + K_{Imp}), R_{Abst}$)
15: MatchQ $(\varphi(T), \phi(T))$; // T = t_0 + k

4.1.1 Computing the Trace of the SPEC

(Lines 1-7): Line 1 first initializes the simulation time t to t_0 (equal to zero in most cases). The purpose of line 2 is to store the initial SRE of the SPEC model in the variable $\phi(t_0)$. Lines 3-6 repeatedly execute a symbolic simulation for K_{Spec} steps using the symbolic simulation algorithm; the time is advanced only if no more delta cycles are needed. K_{Spec} is determined by the verifier and it depends on the temporal complexity of the SRE description of the system. For the WiMax application we set K_{Spec} to 1 because the SRE describing the system is of first order. The variable SPEC stores the computed expressions in line 7. This is equivalent to a new SRE where the time variable is changed to $T = t_0 + K_{Spec}$. This traced SRE will be used to compare the traces in line 15.

4.1.2 Computing the Trace of the IMPL

(Lines 8-14): In the same way, the trace of the IMPL model is computed using a symbolic simulation for K_{Imp} steps (same as K_{Spec}). K_{Spec} and K_{Imp} are the number of times (steps), that the symbolic simulation algorithm is repeated on the specification and implementation system descriptions, respectively,

to generate the symbolic traces. The verifier should choose K_{Spec} and K_{Imp} depending on the temporal complexity of the SRE equations used to describe the system at those levels. So, in general, K_{Spec} may differ from K_{Imp}, however, in the case of the WiMax modem, we chose K_{Spec} and K_{Imp} to be 1, because, the SREs used in the experiment are of first order. The new SRE where the time variable is changed to $T = t_0 + K_{Imp}$ is stored to be used to compare the trace of the IMPL model in line 15. In fact, as the IMPL model is more detailed, the direct comparison is not correct. Thus, we need to add some abstraction rules to refine the computed expressions before comparing the results with SPEC. These rewriting rules R_{Abst} are intended to eliminate calls for functions that convert to integers and rename signals in the IMPL model by their correspondent in SPEC. In line 14, these abstracted expressions are stored in the variable IMPL.

4.1.3 Comparing Both Traces

(Line 15): Using pattern matching and algebraic verification, we verify that symbolic expressions in SPEC can be substituted by variables computed in IMPL. The traced symbolic expressions are put in a normal form, and then verified using the function *MatchQ*. This is a built-in function in the computer algebra system, Mathematica 6.0 [19] and it implements the *Pattern Matching* and the *Equational Theorem Proving*. *k* in (line 15) is the maximum of K_{Spec} and K_{Imp} and is used as an input to the pattern matching function. For the WiMax experiment, we set *k* = 1. If the verification returns true, then computational equivalence is checked. Otherwise, the pattern matcher gives the non equivalent patterns.

4.2 Property Checking Algorithm

Algorithm 3 presents our proposed property checking Algorithm, which we describe in the following.

Algorithm 3 Property Checking

1: PROP = { Prop(IMPL) };
2: t = t_0;
3: $\varphi(t_0) = \{Imp_i(t_0)\}\ 0 < i \leq m$;
4: **while** $t \leq K_{Imp}$ **do**
5: $\varphi(t) = SymSim\_Step(\varphi)$
6: If NoDeltaCycle then t = t+1
7: **end while**
8: IMPL = ReplaceRepeated ($\varphi(t_0 + K_{Imp}), R_{Abst}$)
9: MatchQ (IMPL, PROP) // T = t_0 + k

4.2.1 Storing System Properties

(Line 1): *Prop (IMPL)* is the set of properties of the system that we want to verify. Those properties are written manually as a system of recurrence equations (SRE).

4.2.2 Computing the Trace of IMPL

(Lines 2-8): Similar to what we have done in the equivalence checking part, the trace of the IMPL model is computed using a symbolic simulation for K_{Imp} steps. In line 7 the new SRE where the time variable is changed to $T = t_0 + K_{Imp}$ is stored in IMPL to be used for property checking later. In fact, as the

IMPL model is more detailed, the direct property checking is not correct. Thus, we need to add some abstraction rules to refine the computed expressions before comparing the results with PROP. These rewriting rules R_{Abst} are intended to eliminate calls for functions that convert to integers and to rename signals in the IMPL model by their correspondent ones in PROP. In line 8, these abstracted expressions are stored in the variable IMPL.

4.2.3 Comparing PROP and IMPL

(Line 9): Using pattern matching and algebraic verification, we verify that symbolic expressions in PROP can be substituted by variables computed in IMPL. The traced symbolic expressions are put in a normal form, and then verified using the function *MatchQ*. If the verification returns True, then properties are checked. Otherwise, the pattern matcher gives a counterexample.

5 Modeling and Verification of a WiMax Modem

5.1 ST WiMax Modem Models

STMicroelectronics provided us with three different C/C++ models of their proposed design each at a different level of abstraction. They are:

- Model 1: Functional Level Model.
- Model 2: FIFO Based Process Transfer Model.
- Model 3: FIFO and Scheduler Based Process Transfer Model.

Figure 2 shows the three models. The functional model consists of serially connected functional blocks without any additional communication components. These components include the Input block, the Randomizer, the Convolutional coder, the Puncturing block, the Interleaving block, the Repetition block, the Modulator and the Output block. These functional blocks implement various functions specified in the WiMAX standard [9]. In the FIFO based process transfer model, a FIFO is used between each of the functional blocks of the system. It allows handling of different timing requirements of the system blocks. Moreover, in this model, each functional block is mapped to a separate processing unit. All SRE system component models including the FIFO model were extracted from the corresponding C/C++ models provided by STMicroelectronics. Finally, in the FIFO and Schedular based process transfer model, the functionality of more than one functional block were mapped to a single processing unit. In our SRE model, we used a generic scheduler model provided by STMicroelectronics. The scheduler scans the functional blocks in a round-robin manner with a predefined order of the blocks, implementation details can be found in [2]

We verified the functional equivalence between Model 1 and Model 2, and between Model 2 and Model 3. We first wrote the SRE description of each model using Mathematica (see Appendix A for a sample C++ code and its equivalent SRE description).Then, we validated the correctness of their basic functionality using sample numerical simulation. Next, we generated symbolic traces for SRE models using symbolic simulation in Mathematica. Finally, we used Pattern Matching on those symbolic traces to verify the functional equivalence of all SRE models of the system (Algorithm 2). This equivalence implies the equivalence of the corresponding C/C++ models.

We also verified the conformance of these models to important properties of the WiMax transmitter. We wrote those properties as SREs. Then, we used Pattern Matching and Equation Solving Functions

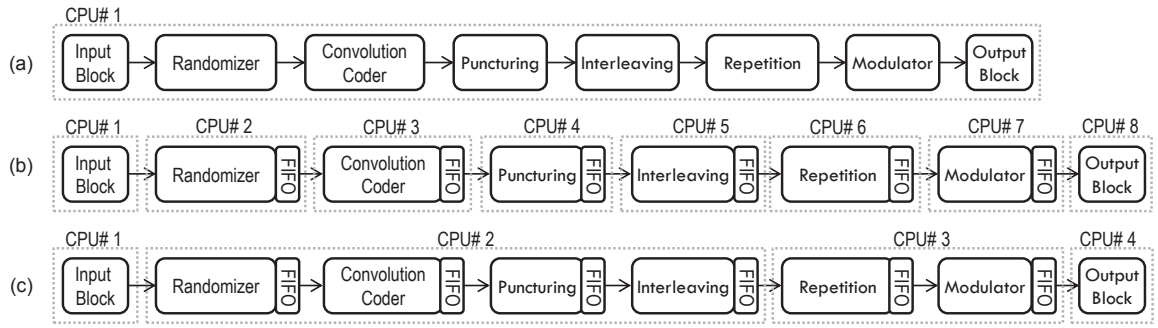

Figure 2: (a) Functional Level Model (FL), (b) FIFO Based Process Transfer Model (PTL-8), (c) FIFO and Scheduler Based Process Transfer Model (PTL-4).

from Mathematica to verify these properties (Algorithm 3). The conformance of SRE models to these properties implies the conformance of the corresponding C/C++ models to the same properties.

5.1.1 Single and Multiple Control Scenarios.

The physical layer implementation of the ST WiMax modem has various control signals. These control signals define the behavior of the internal blocks of the WiMax physical layer. Predefined combinations of these control signals are called modes of operation. The WiMax system supports 52 different modes of operation. In a single control scenario simulation, we perform symbolic simulation assuming that the control signals have predefined value of mode_0. Whereas, in the multiple control scenario, we perform symbolic simulation seven times once for each mandatory mode of operation supported by the physical layer implementation of the ST WiMax modem [16, 17].

We validated the correctness of the basic functionality of our SRE models using numerical simulation and compared the simulation results with the corresponding ST functional models. We simulated each of the models for 100 test vectors with a random selection of operation modes and verified that all SRE model outputs were identical to the original ST models. We also generated symbolic traces for the three SRE models of the WiMax modem for seven operation modes. We call this "mixed simulation mode" because it used both symbolic and numerical simulation to generate the symbolic traces. Table 1 shows the time and memory utilization of these experiments, which shows the superiority of symbolic simulation over numerical simulation in terms of time requirements.

Table 1: Symbolic Simulation Results.

Model	Single Control				Multiple Control			
	Symbolic Sim.		Numerical Sim.		Symbolic Sim.		Numerical Sim.	
	Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)
C/C++ Functional Model	N.A.	N.A.	3.21	2.30	N.A.	N.A.	3.21	2.30
SRE Functional Level (FL)	0.32	13.60	10.00	13.45	2.05	11.96	10.10	10.11
SRE Proc. Trans. Level (PLT-8)	5.17	19.59	252.4	16.54	33.70	12.91	211.3	10.00
SRE Proc. Trans. Level (PLT-4)	5.26	28.31	266.4	25.32	34.18	13.37	222.3	11.44

5.1.2 Equivalence Checking.

We verified the computational equivalence between SRE models at different levels of abstraction. We applied the Pattern Matching techniques on the symbolic traces calculated by symbolic simulation. We used Mathematica Pattern Matching built-in function to compare symbolic traces as described in Algorithm 2.

5.1.3 Verified Properties.

We conducted four equivalence checking experiments to prove the following relations:

- Equivalence of Functional Model and FIFO based Process Transfer Model in the Single Control scenario.
- Equivalence of FIFO based Process Transfer Model and FIFO and Scheduler based Process Transfer Model in the Single Control scenario.
- Equivalence of Functional Model and FIFO based Process Transfer Model in the Multiple Control scenario.
- Equivalence of FIFO based Process Transfer Model and FIFO and Scheduler based Process Transfer Model in the Multiple Control scenario.

The results show that the SRE models at different levels of abstraction are functionally equivalent. In order to guarantee the basic functionality of our equivalence checking algorithm, we injected one bug in one of the SRE models and re-ran the symbolic simulation and equivalence checking experiment. The bug was injected in the SRE FIFO based Process Transfer Model. We changed the functional description of the mapping block. Then, we ran the equivalence experiments again. Now, the results showed non equivalence between the models and returned the nonequivalent symbols from the model's symbolic trace. By inspecting those symbols we found that they were generated only at three modes of operation (0, 1, or 2). From these results, we concluded that the bug was injected in the mapping block implementation only when its puncturing value equals 1/2. More details are provided in [2].

Table 2 summarizes the performed equivalence and non equivalence experiments along with their time and memory utilization results. The computation time and memory results in Table 2 include the time and memory utilization for both symbolic trace computation and pattern matching. From the results in Table 2, we conclude the following:

- The run time of the experiments is linearly proportional to the number of control scenarios. This is interesting because other techniques have exponential increase in time requirements when we increase the execution paths.
- Memory requirements of various experiments are comparable to each other.
- Since our verification technique depends on pattern matching, we obtained interesting results in the case of non-equivalence. Both time and memory requirements stayed at the same rank as in the equivalence experiments.

5.2 Property Checking.

To verify the conformance of the WiMax models to important properties of the system and to generate counterexamples for properties that turn out to be false, we use the Property Checking algorithm (Algorithm 3), described in Section 4.

Table 2: Equivalence Checking Results

Models	Design	Single Control		Multiple Control		Result
		Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)	
FL vs. PTL-8	Original	5.17	19.59	33.70	12.91	Equivalent
PTL-8 vs. TPL-4		5.26	28.31	34.18	13.37	Equivalent
FL vs. PTL-8	Injected Bugs	4.96	20.88	32.10	14.51	Not Equivalent
PTL-8 vs. TPL-4		6.21	27.21	30.26	15.67	Not Equivalent

5.2.1 Verified Properties.

We divided the properties with respect to their scope into three categories:

1. Global Properties: specify a functionality of the whole system.
2. Local Properties: specify a functionality of a single block.
3. Control Properties: specify a functionality of a single control configuration (Code Type in the WiMax case)

We wrote several properties of each of these main categories. Here we describe three of them.

- P1: Eventually all Input Data Bits will be transmitted
- P2: Eventually all Input Data Bits with the positions specified by the randomizer bit list will be flipped.
- P3: Eventually the Appropriate Puncturing Function will be applied to all Convolution Coded Data Bits in the same order.

Next, we translated those properties into SRE (see Appendix B for a sample property written in a form of SRE). After that we applied our proposed Property Checking algorithm to verify their correctness according to the symbolic traces calculated in the symbolic simulation of the model under test. The results of our experiments show that all tested properties are verified to be true under the three models in both single and multiple control scenarios. This shows the conformance of the corresponding C/C++ models from STMicroelectronics to the verified properties.

We also repeated our experiments after injecting the following bugs into all the SRE models.

1. Cut one of the data lines between two of the internal blocks.
2. Changed the randomizer reference array.
3. Changed the condition checker at the mapping block that specifies the block behavior when the control scenario changes.

The results of the simulation detected all bugs and returned counterexamples that specified the failed property and print the wrong signal value. Table 3 shows the property checking experiments results, together with their time and memory requirements. The results in Table 3 include the time and memory utilization of both symbolic trace computation and pattern matching used in the property checking process. By looking at these results we conclude the following:

1. The run time of the experiments is linearly proportional to the number of control scenarios.
2. Memory requirements of various experiments are close to each other.
3. Both time and memory requirements stayed at the same rank in both cases, verified true and verified false cases.

Table 3: Property Checking Results (Single Control Scenario)

Model	Property	Original				Result	Injected Bugs				Result
		Single Control		Mult. Control			Single Control		Mult. Control		
		Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)		Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)	
FL	1	3.25	22.05	25.70	20.15	True	3.00	22.23	30.71	22.81	False
	2	3.10	23.10	24.31	21.23	True	3.58	21.25	25.22	20.60	False
	3	2.92	22.65	24.10	23.10	True	3.10	20.36	26.10	13.80	False
PTL-8	1	10.52	22.81	75.20	26.20	True	10.12	20.15	76.90	22.05	False
	2	11.10	20.60	80.12	23.28	True	10.22	21.23	78.15	23.10	False
	3	10.30	13.80	78.58	16.55	True	11.11	23.10	79.80	22.65	False
PTL-4	1	10.16	23.84	79.95	22.23	True	10.16	26.23	80.94	22.81	False
	2	10.23	21.02	78.55	21.25	True	11.22	23.28	97.78	20.60	False
	3	10.98	22.36	72.00	20.36	True	11.94	16.55	71.02	13.80	False

6 Conclusion

In this paper, we proposed a semi-formal verification methodology that uses sequence of recurrence equations as a formalism for modeling and property specification. We used symbolic simulation traces and two proposed algorithms for equivalence checking and property checking for design verification.

We illustrated the effectiveness of our methodology by verifying STMicroelectronics WiMax system designs at three different levels of abstraction, one functional level model and two architectural level models. Our experimental results show that the three models are functionally equivalent and the design refinements are correct. In addition, the results show that all models do conform to the specified properties. We detected manually injected bugs in design models and successfully generated counterexamples leading back to the bugs in the design. The performance measurements show that the proposed symbolic simulation based verification method is more efficient than numerical simulations.

We are currently investigating efficient techniques to reduce symbolic simulation time with multiple control signals. In order to automate the modeling part of our methodology, we plan to define transition rules to translate system descriptions from standard programming languages such as C or C++ to SRE. Finally, we plan to apply this semi-formal verification methodology to other more complex system designs.

References

[1] S. Abdi & D. Gajski (2006): *Verification of System Level Model Transformations*. *International Journal of Parallel Programming* 34(1), pp. 29–59, doi:10.1007/s10766-005-0001-y.

[2] S. Al-Akhras (2012): *On the Verification of a WiMax Design using Symbolic Simulation*. Master's thesis, Concordia University. Available at http://spectrum.library.concordia.ca/974451/.

[3] G. Al-Sammane (2005): *Simulation Symbolique des Circuits Decrits au Niveau Algorithmique*. Ph.D. thesis, Universite Joseph Fourier Grenoble. Available at http://tel.archives-ouvertes.fr/docs/00/04/82/66/PDF/tel-00009776.pdf.

[4] G. Al-Sammane, M. H. Zaki & S. Tahar (2007): *A Symbolic Methodology for the Verification of Analog and Mixed Signal Designs*. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, EDA Consortium, San Jose, CA, USA, pp. 249–254, doi:10.1109/DATE.2007.364599.

[5] Altera (2009): *DSP Builder*. Available at http://www.altera.com/products/software/products/dsp/dsp-builder.html.

[6] L. Bachmair & H. Ganzinger (1994): *Rewrite-based Equational Theorem Proving with Selection and Simplification* 4(3), pp. 217–247. doi:10.1093/logcom/4.3.217.

[7] A. Chiang, H. Wei-Hua & B. Kapoor (2009): *Validating Physical Access Layer of WiMAX using Systemverilog*. In: *Proceedings of the 2009 10th International Symposium on Quality of Electronic Design*, IEEE Computer Society, Washington, DC, USA, pp. 356–359, doi:10.1109/ISQED.2009.4810320.

[8] A. Deb, A. Jantsch & J. Öberg (2004): *System Design for DSP Applications in Transaction Level Modeling Paradigm*. In: *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, ACM, New York, NY, USA, pp. 466–471, doi:10.1145/996566.996698.

[9] C. Eklund, K. L. Stanwood, S. Wang & R. B. Marks (2006): *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1*. *IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005 (Amendment and Corrigendum to IEEE Std 802.16-2004)*, pp. 1–822, doi:10.1109/IEEESTD.2006.99107.

[10] F. Franek, C. G. Jennings & W. F. Smyth (2007): *A Simple Fast Hybrid Pattern-Matching Algorithm*. *J. of Discrete Algorithms* 5(4), pp. 682–695, doi:10.1016/j.jda.2006.11.004.

[11] M. Fujita, I. Ghosh & M. Prasad (2008): *Verification Techniques for System-Level Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[12] D. D. Gajski, J. Zhuand, R. Dömer, A. Gerstlauerand & S. Zhao (2000): *SpecC: Specification Language and Methodology*. Springer, Massachusetts: Kluwer Academic Publishers, doi:10.1145/266021.266037.

[13] S. Liao, S. Tjiang & R. Gupta (1997): *An Efficient Implementation of Reactivity for Modeling Hardware in the Scenic Design Environment*. In: *Proceedings of the 34th Annual Design Automation Conference*, ACM, New York, NY, USA, pp. 70–75, doi:10.1145/266021.266037.

[14] T. Matsumoto, H. Saito & M. Fujita (2005): *An Equivalence Checking Method for C Descriptions Based on Symbolic Simulation with Textual Differences*. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E88-A(12), pp. 3315–3323, doi:10.1093/ietfec/e88-a.12.3315.

[15] STMicroelectronics (2007): *Model-Based Mapping to Parallel Architecture of 802.16a (WiMax)*.

[16] STMicroelectronics (2007): *Multiflex WiMax OFDMA Library 2007 - Code Package*.

[17] STMicroelectronics (2007): *WiMAX Wireless-MAN OFDMA Functional Refinement*.

[18] Agilent Technologies (2009): *Move Forward to What's Possible in WiMAX*. Available at http://cp.literature.agilent.com/litweb/pdf/5989-5914EN.pdf.

[19] S. Wolfram (1991): *Mathematica: A System for Doing Mathematics by Computer, Second Edition*. Addison-Wesley Longman Publishing Co. Available at http://library.wolfram.com/infocenter/Books/3617/.

[20] M. H. Zaki, G. Al-Sammane & S. Tahar (2007): *Formal Verification of Analog and Mixed Signal Designs in Mathematica*. In: *Computational Science*, Springer-Verlag, Berlin, Heidelberg, pp. 263–267, doi:10.1007/978-3-540-72586-2_37.

A Appendix: Sample C++ Code described in SRE

An example of writing recurrence equations description of a C++ code is given in this section. The following C++ is just a sample code that has been written to help illustrating the idea of C++ to SRE conversion.

```
1  int main(int argc, char *argv[])
2  {
3    class Intermediate_Signal
4    {
5      int[32] input, output;
6    };
7
8    class Randomizer_Block
9    {
10     Intermediat_Signal blockInput , blockOutput, blockControl;
11
12     void randomize()
13     {
14       if (blockControl == MODE_0)
15       {
16         blockOutput = blockInput;
17       }
18       else if (blockControl == MODE_1)
19       {
20         blockOutput = randFunc_01(blockInput);
21       }
22       else if (blockControl == MODE_2)
23       {
24         blockOutput = randFunc_02(blockInput);
25       }
26       else
27       {
28         blockOutput = INVALID_DATA;
29       }
30     }
31   };
32
33   // The actual main
34   ... Randomizer_Block WimaxRand = new Randomizer_Block;
35   WimaxRandomizer->blockControl = MODE_1;
36   WimaxRandomizer->blockInput = PREVIOUS_BLOCK_OUTPUT;
37   WimaxRandomizer->blockOutput = NEXT_BLOCK_INPUT;
38   WimaxRandomizer.randomize();    ...
39 }
```

The following is the SRE representation of the above C++ code.

```
RAND_OUT =
IF [ RAND_CTRL = MODE_0, RAND_IN,
   (IF [ RAND_CTRL = MODE_1, randFunc_01(RAND_IN),
       (IF [ RAND_CTRL = MODE_2, randFunc_02(RAND_IN),
           INVALID_DATA]) ]) ];
```

B Appendix: Sample Property Code in SRE

An example of writing system properties using recurrence equations in Mathematica is given in the following:

"**Property P3:** Eventually the appropriate Puncturing Function will be applied to all Convolution Coded Data Bits in the same order"

```
If[ CodeRate == WMRATE23,

For [i = 0, i < CycleCounter, i++,

  If[ PunctOutput[[1, i*3 + 1]] == CCOutput[[1, i*4 + 1]],
  PuncturedSymbols ++,
  Print["Symbol Not Punctured Properly"];,
  Print["Symbol Not Punctured Properly"];
  ];

  If[ PunctOutput[[1, i*3 + 2]] == CCOutput[[1, i*4 + 2]],
  PuncturedSymbols ++,
  Print["Symbol Not Punctured Properly"];,
  Print["Symbol Not Punctured Properly"];
  ];

  If[ PunctOutput[[1, i*3 + 3]] == CCOutput[[1, i*4 + 4]],
  PuncturedSymbols ++,
  Print["Symbol Not Punctured Properly"];,
  Print["Symbol Not Punctured Properly"];
  ];
];,

If[ CodeRate == WMRATE46, ... , ...];
];
```