An Efficient System-Level to RTL Verification Framework for Computation-Intensive Applications

Nikolaos D. Liveris*
Northwestern University
Evanston IL 60208

Hai Zhou*
Northwestern University
Evanston IL 60208

Prithviraj Banerjee†
University of Illinois
Chicago IL 60607

Abstract

In this paper a new framework for formal verification is presented. The new framework called EVRM (Efficient VeRification based on Mathematica [1]) can be used for the property verification of a Register Transfer Level implementation using a System Level description as the golden model. EVRM is based on word level techniques and uses the Mathematica tool for the satisfiability procedure. Results show that it can be orders of magnitude faster than CBMC [2] in proving property correctness or providing a counterexample for computation-intensive applications. For certain applications CBMC requires more than 5 hours to provide an answer, while EVRM provides an answer in less than 10 minutes.

1 Introduction

It is known that almost two thirds of the design cycle of a digital integrated circuit is spent on verifying its functionality [14]. As designs become more complicated and time-to-market periods decrease, the needs for efficient verification frameworks increase.

For verifying digital integrated circuits several approaches exist. One is the simulation based verification, whose drawbacks are the long execution time and the inability to assure correctness of the design. On the other hand, formal verification can be an efficient alternative to prove that specific properties of the design hold.

Formal verification methods include Symbolic Model Checking and Theorem Proving. In Model checking [4], the temporal logic specification is used to check system properties where the system is modeled as a finite state machine. Symbolic Model Checking, with boolean encoding of the finite state machine as ordered binary decision diagrams (BDDs) can handle more than 10^{20} states [5]. On the other hand, Bounded Model Checking (BMC) for linear temporal logic (LTL) can be reduced to propositional satisfiability in polynomial time where bound is the maximal length of a counterexample and solved using SAT solvers [6].

For Theorem provers both the system and its desired properties are expressed as formulas in some mathematical logic and the theorem prover finds a proof from axioms of the system. SVC [8] and its enhanced version CVC [9] are automatic theorem provers for first order logic. PVS [7] combines decision procedures and model checking with interactive proof. Theorem provers in contrast to model checkers can handle infinite state spaces but generally require manual intervention and are hard to use.

It is a common practice to write a System Level functional description of the digital IC in the first design stages. This description after verification can be used as the golden model for the Register Transfer Level implementation of the circuit. One approach for this kind of verification is CBMC [2], which uses a C specification of the circuit to verify the RTL model. The techniques to capture the model are the same as in BMC approaches and a bit-level SAT solver [10] is used to produce a counterexample or to prove the correctness of the assertions.

In this paper we describe an alternative approach to CBMC for verifying properties of an RTL description using its System Level specification. The approach is orders of magnitude faster than CBMC for computational intensive applications by sacrificing bit-level accuracy, which may not be needed during the early stages of the verification process. The back-end tool used in the framework is Mathematica, a well known commercial symbolic analysis tool. In the next section of this paper we describe the motivation behind this approach. In Section 3 we formulate the problem that needs to be solve, while in Sections 4 and 5 we explain the reasons behind using Mathematica and the proposed framework in more detail. Finally, Sections 6 and 7 present our results and conclusions.

2 Motivation

In this section we discuss the motivation behind the usage of word level techniques and Mathematica. As men-

*{nikos,haizhou}@ece.northwestern.edu
†prith@uic.edu

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

tioned before, CBMC is the only bounded model checking approach for verifying properties of an RTL implementation based on a System Level description. CBMC uses a SAT solver as the back-end tool for proving the assertion or providing a counterexample. The whole program needs to be converted in a Conjuntive Normal Form (CNF), which will be the input to the SAT solver [10].

A major bottleneck for the SAT solver is the memory requirements of a CNF. If the required memory exceeds the available physical memory, the swap file will be employed. Therefore, all non-local accesses for the formula will involve the disk and become very expensive. There are two factors that will determine the size of a CNF, the number of clauses and the number of literals.

The number of clauses depends on the logic that the circuit will implement. Arithmetic operations like addition, or multiplication produce a large number of clauses [3]. A behavioral System Level description of a computational intensive application is expected to include mainly arithmetic operations. Therefore, for computational intensive applications the number of generated clauses can make the usage of a SAT solver inefficient.

Besides the number of clauses, the number of literals of a CNF will impact the running time and depends again on the logic of the circuit. The arithmetic operations affect that number depending on the type of operators and the size of input operands. Moreover, the number of literals increases with the number of cycles for which the model checking is performed. However, given the area constraints in the synthesis of digital circuits and that significant properties require a large amount of cycles to be proved, the bound in the number of cycles cannot be small for realistic designs and that can lead to an explosion in the memory requirements. As an example in [2] for a DRAM Arbiter, which is not expected to have many arithmetic operations, the memory usage was 2GB for sufficiently large bounds.

The above discussion reveals the need for a higher level of a abstraction for the System Level and the RTL models used for verification. Especially, for computational intensive applications verification techniques at the word level would be extremely useful for the reasons mentioned above. By representing the two models using a conjunction of word level formulas many important properties of a computational intensive description can be verified, as will be shown in the next sections. The cycle accuracy of the RTL description can be maintained, however the bit accuracy will be lost. The bit accurate implementation of the design can verified later in the design cycle when word level design faults have been eliminated.

Approaches that have already used word level techniques for verification problems include [7],[8], [9]. None of these approaches can handle multiplication of two variables in an efficient way. In [11] the authors used SVC [8] to verify assembly routines used in DSP software. Because SVC uses uninterpreted functions to handle all operations except addition and subtraction, the authors had to support user-defined properties, which would specify for example the multiplication is a commutative operator. Moreover, heuristics were written for operand reordering and expression normalization.

Since computational intensive applications are expected to have a large amount of complicated arithmetic expressions, the designer would need to define all the important properties for multiplication and division. Then the properties that combine two or more operators also have to be specified, like $a(b+c) = ab+ac$. Furthermore, the heuristics for simplifying or reducing complex arithmetic expressions should be as complete as possible for these applications. Obviously, the manual effort for verifying computational intensive applications using these word level tools is large and makes their usage for this kind of verification problems inappropriate.

In conclusion, there is a need for another approach to functional verification of computation intensive applications. This approach should be able to prove functional correctness by bringing variables to the infinite domain, so that the verification procedure will be fast and should use word-level provers that will not require manual effort from the user for defining the properties of commonly used arithmetic operators like non-linear multiplication.

3 Problem Formulation

In this section we formulate the problem that we try to solve. Given a System Level and a synchronous, single-clocked RTL description, with a mapping of corresponding input variables, a bound in terms of clock cycles, and an assertion inserted in the code, we try either to prove that the assertion is true for that specific bound and for all valid input values, or find a counterexample that can make the assertion false. We try to solve the problem by bringing all variables to the infinite-precision domain.

4 Mathematica

Mathematica [1] is a technical computing package with a very rich library for symbolic computation. It includes solvers for sets of formulas in different domains (Boolean, Integer, Real) and has built-in the properties of the most commonly used arithmetic operators of each domain. The advantage of Mathematica is that is a generic package and, therefore, the user will not have to spend manual effort to express the properties of word level operations. Moreover, it is a powerful symbolic analysis tools and supports solvers at the integer and real domain, avoiding the translation of

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

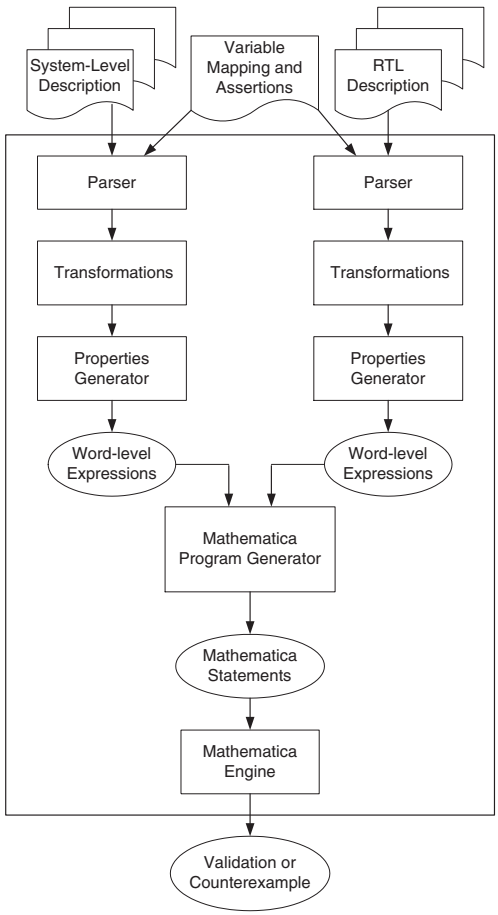

Figure 1. The EVRM framework

arithmetic expressions to binary numbers. Finally, arithmetic expression manipulation with Mathematica has been used efficiently in the past for several compiler problems [12], [13].

5 Verification Framework

In this section we will present the EVRM (Efficient VeRification based on Mathematica) framework. A schematic representation of EVRM is shown in Figure 1. As it can be seen from this figure the framework can be broken into three parts. The first and second parts implement the transformation of the System-Level and RTL descriptions to word level expressions. The third part generates the Mathematica statements from these expressions. Then these statements are going to be the input to the Mathematica Kernel, which is going to either prove the validity of the asserted property or provide a counterexample, for which the property is invalid. In the next sections we describe each part of the framework in more detail.

5.1 System-Level description to Word-Level Expressions

The SL description is transformed to word level expressions using the techniques presented in [3]. Currently, the input SL description should be in ANSI C. The first step is to unwind all loops in the code and to tranform switch statements to if statements. Then the code is converted in a Static Single Assignment (SSA) form. At the end of the transformation phase the code consists of assignment statements and if statements.

5.2 Register Transfer Level description to Word-Level Expressions

For this process we assume that the RTL description represents a Finite State Machine of a single clock design. The process starts by finding the registers of the circuit. Then the program is transformed to a set of assignment statements and if statements by modifying the control structures of the description. This process is described in detail in [3].

5.3 Word-Level Equations

After transforming the two descriptions in sequential programs that contain only if statements and assignments, the generation of guarded word-level equations starts.

A guarded word-level equation has the form:

$$(cond)\ ?\ var = Expr1\ :\ var = Expr2$$

where *cond* is a binary variable that represents the conjuction of all conditions that should be true, in order for the assignment to be executed. These are the conditions of the nested if-statements, in the blocks of which the assignment is placed. The conditions are represented by:

$$ExprLeft\ \ Operator\ ExprRight$$

where $Operator$ can be any of the $>, <, ==, !=, >=, <=$ and $ExprLeft$, $ExprRight$ are arithmetic expressions consisting of constants, variables, parenthesized subexpressions, and arithmetic operators, like $+, -, *, /$. The same applies for $Expr1$ and $Expr2$ of the guarded word level equation. In the guarded word level equation *var* is the name of the variable that is assigned.

All guarded word-level expressions are later transformed to Mathematica statements. The above generic guarded word level expression will be translated to:

$$\{cond \,\&\&\, var == Expr1\} \,||\, \{!(cond) \,\&\&\, var == Expr2\}$$

The variables of the above expressions can have several types. Most commonly used for DSP applications are Integers and Reals. Variables may also have compound types and indexed multi-dimensional arrays can be used in all arithmetic expressions.

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

5.3.1 FindInstance statement

After all expressions, which represent the constraints imposed by the two descriptions, are transformed to equivalent Mathematica expressions, the asserted expression will also be transformed. Then the conjunction of the all constraints and the negation of the assertion will be the expression that if it is satisfiable then the asserted property does not hold. Satisfiability of this expression means that there are inputs that satisfy all constraints imposed by the descriptions of the specification and the implementation and satisfy the negated property as well. Therefore, the property is not valid for all inputs.

Mathematica version 5.0 has a statement that allows the user to find an instance of a set of variables that makes an expression valid. The syntax of the generated statement will be:

$$FindInstance\,[\,\underbrace{E_1 \&\& E_2 \&\& ... \&\& !A}_{expression}\,,\,\overbrace{\{a,b,c,...\}}^{variables}\,,\,D\,]$$

The first argument is the expression which may become true for an instance of the set of variables given as a second argument. The third argument D is the domain to which the variables belong. The domain can be Integers, Reals, or Booleans in this case. As explained previously the generated statement by EVRM sets as first argument the conjunction of the constraints $(E_1, E_2, ...)$ and the negated assertion A. The set of variables given as the second argument are the set of all inputs and variables of the description.

The *FindInstance* exits either by reporting the empty set, in which case it has proven that there are no values for the variables that can make the expression hold, or by reporting values for the variables that will make the expression hold.

5.3.2 Optimizations

In the CBMC approach the SL and RTL descriptions were translated to a CNF. Bit vector arithmetic operations in this case were transformed to CNF by using actual circuit gate representation. Since these circuits normally work on two input variables, there was no way to reduce the problem instance by replacing variables with their definition. However, in our case replacing a variable with its actual definition can reduce the size of the input expression and the cardinality of the variables set for the *FindInstance* statement.

So, if $a == b + c$ is an input word level expression with $guard == TRUE$, then it was transformed to a statement $a = b + c$, which was executed before *FindInstance*. Moreover, $a == b + c$ could be removed from the set of constraint expressions and did not need to be part of the first argument of *FindInstance*, since it was forced. Added to that, a was removed from the set of variables, as Mathematica would replace every instance of a with $b + c$. This optimization was applied to non-guarded expressions and reduced the problem instances.

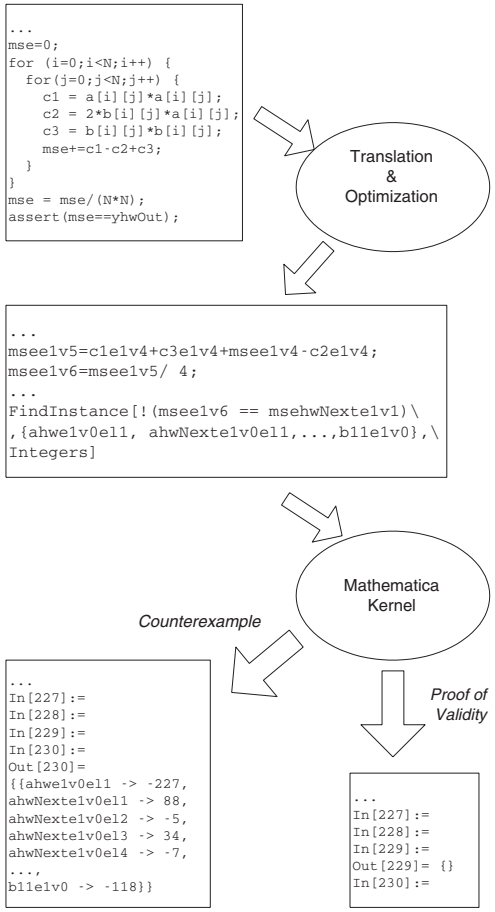

Figure 2. Example for the Generated Code

Figure 2 shows some example instances of the generated statements and the output of Mathematica for a part of a C program.

6 Experimental Results

In this section the experimental results for 5 benchmarks will be presented. The characteristics of each benchmark are summarized in Table 1. The system-level specification of each application was written in C. The RTL implementation was described in RTL C. All results were taken on SUN Ultrasparc machines, with OS Solaris 9, and 256MB of physical memory. As explained above, the results are compared to CBMC as existing word-level solvers,to the best of our knowledge, cannot handle frequently used arithmetic operators without manual effort for the specification of the operators' properties.

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

Table 1. Applications Characteristics

Application	Matrix Multiplication	FIR filter	Laplace Transform	Sobel Transform	MSE
System-level lines	27	15	19	24	19
RTL lines	185	106	129	146	97
Chaff input clauses	1,034,271	145,635	8,964	28,885	339,010
Chaff input literals	308,158	44,024	4,547	10,332	101,206
Mathematica input expressions	3,701	1,037	294	303	222
Mathematica input variables	99	51	165	170	33

Table 2. Time to Prove Validity of the Assertion

Application	Matrix Multiplication	FIR filter	Laplace Transform	Sobel Transform	MSE
time with CBMC	> 5 h	> 5 h	> 5 h	> 5 h	> 5 h
time with EVRM	5:06 min	1:14 min	10:27 min	44:40 min	0:41 min

Matrix Multiplication This application implemented the multiplication of two 3x3 matrices. The assertion was checking whether the element of the last row and last column of the resulting matrix was the same for the two descriptions. For the system-level description the input values were stored in 2 dimensional matrices, whereas in the RTL implementation they were stored in single dimensional register arrays. Moreover, the elements of the resulting matrix were computed in a column-wise manner in system-level and in a row-wise in the RTL description. Added to that, the structure of the code was completely different for the two descriptions. In the system-level specification a 3-deep loop nest was implementing the multiplication after the initialization stage, compared to an 18-state RTL implementation, which produced the multiplication result after 140 cycles.

As shown in Table 2 for a correct version of the RTL implementation, it took almost 5 minutes for the EVRM framework to prove correctness for any two input matrices. This time is broken into two parts, the time to generate the statements and expressions from the two descriptions and the time it takes for Mathematica to read these statements and expressions and produce a result. The generation part took 4 minutes and the Mathematica part took 1 minute. With CBMC after 5 hours no result could be produced.

In the second experiment done with this application a bug was inserted in the RTL implementation. The bug was in the initialization of one of the variables. As shown in Table 3 it takes almost 5 minutes for EVRM to produce a counterexample, while for CBMC the time is close to 15 minutes. In case the matrices were larger, like 4x4 or 5x5 the difference could be larger as the memory requirements of CBMC would increase.

FIR filter The next application used was a 4-tap FIR filter. The assertion was checking for the equivalence of the last element of the output array. The RTL description was implemented as an FSM of 12 states and produced the result after 65 cycles.

EVRM proved the equivalence of the two descriptions in less than 2 minutes, out of which 1 minute was spent in the generation of the statements and expressions for Mathematica.

In the second expreriment for this application a bug was added for the output result. This time the bug would affect only a subset of the possible output values. The bug forced the output values for the RTL implementation not to exceed the value 8192, whereas for the system-level specification the result could be any integer value. EVRM found a counterexample after 1 minute and 17 seconds. In both cases the running time with CBMC was more than 5 hours.

Laplace Transform This application implemented one step of the Laplace transformation. The values were stored in an two dimensional array in the System Level description, whereas in the RTL they were in a one dimensional array of registers. Moreover, the order of the computations was different in the two models.

EVRM proved that the output value of the two descriptions was equal for all inputs in almost 10 minutes. For the faulty version of the same algorithm a counterexample was found in less than a minute. The fault was the removal of the condition that did not allow the array elements to exceed the value 255 in the RTL description. For CBMC the bound of 5 hours was reached before an answer was given by the program.

Sobel Transform This application was implementing the Sobel Transform. The assertion checked to whether the computation of the new value based on its eight neighbor points was correct for the RTL implementation. It took almost 45 min for EVRM to prove the validity of the property.

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

Table 3. Time to Produce Counterexample

Application	Matrix Multiplication	FIR filter	Laplace Transform	Sobel Transform	MSE
time with CBMC	14:52 min	> 5 h	> 5 h	> 5 h	40:27 min
time with EVRM	5:23 min	1:17 min	0:37 min	7:07 min	0:40 min

The second experiment was done with a faulty implementation. The original RTL implementation was implementing the abs function by checking if the difference for the horizontal sobel transform was negative and in that case multiplying by -1. This was deleted in the faulty implementation allowing negative results of the difference. Again this bug will not be visible for all inputs as many will produce positive values for the horizontal difference. It took about 7 minutes for EVRM to find a counterexample that invalidated the assertion.

In both cases after 5 hours CBMC did not produce any result.

Mean Squared Error Computation The last application was implementing mean squared error computation. The input to the algorithm were two 2x2 matrices for which the mse value was computed by the SL and RTL descriptions. EVRM could prove correctness for this property in time less than a minute.

The second experiment with this application involved the addition of bug in the RTL description. The inserted bug affected the initialization of a variable. Again EVRM found a counterexample after 40 secs. In this case CBMC provides a counterexample for the assertion after 40 minutes.

7 Conclusions

In this paper we have presented a new approach for the verification of specific properties of an RTL implementation based on an executable System Level specification. The new framework is intended for computational intensive applications and is based on word level techniques and uses Mathematica for the satisfiability procedure. The current approaches for the same problem are based on bit level SAT solvers.The results show orders of magnitude of performance improvement compared to CBMC, a tool used for RTL to SL verification. Our future work will include extension to the Mathematica expressions, so that more programming constructs can be processed, like pointers. Moreover, we plan to develop more optimizations to reduce the problem instance for the Mathematica kernel. With those optimizations we will try to reduce the running time even more. Finally, we will try to tackle some aspects of the fault coverage issues at the word level.

References

[1] Mathematica Package, www.wolfram.com

[2] E. Clarke, D. Kroening, K. Yorav; "Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking", DAC, June 2003

[3] E. Clarke,D. Kroening,K. Yorav; "Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking"; Technical Report CMU-CS-03-126, Carnegie Mellon University, School of Computer Science 2003

[4] E. Clarke, E. A. Emerson; "Design and synthesis of synchronization skeletons using branching time temporal logic"; In Proceedings of the IBM Workshop on Logics of Programs, Springer-Verlag, 1981.

[5] K. L. McMillan; "Symbolic Model Checking: An Approach to the State Explosion Problem"; Kluwer Academic Publishers, 1993.

[6] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, Y. Zhu; "Symbolic model checking using SAT procedures instead of BDDs"; DAC 1999

[7] Owre, S., Rushby, J., Shankar, N.;"PVS: A prototype verification system"; Lecture Notes in Computer Science, Vol. 607 (1992), Springer-Verlag.

[8] C. W. Barrett, D. L. Dill, A. Stump. "A Framework for Cooperating Decision Procedures"; In David McAllester, editor, 17th International Conference on Computer Aided Deduction, volume 1831 of LNAI, pages 79-97, Springer-Verlag, 2000.

[9] A. Stump, C. W. Barrett, D. L. Dill; "CVC: a Cooperating Validity Checker"; In Proceedings of CAV 2002, Copenhagen, Denmark, July 2002

[10] M.W. Moskewicz, C.F.Madigan, Y.Zhao, L.Zhang, S.Malik; "Engineering an efficient SAT solver"; DAC 2001

[11] D. W. Currie, A. J. Hu, S. Rajan, M. Fujita; "Automatic Formal Verification of DSP Software"; DAC 2000

[12] P.Joisha, P. Banerjee; "The MAGICA Type Inference Engine for MATLAB"; Proceedings of the 12th International Conference on Compiler Construction, April 2003

[13] P. Joisha, P. Banerjee; "PARADIGM (version 2.0): A New HPF Compilation System"; International Parallel Processing Symposium (IPPS'99), April 99

[14] T. Kropf; "Introduction to Formal Hardware Verification"; Springer-Verlag, 1999

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE