A Hybrid Verification Approach : Getting Deep into the Design

Scott Hazelhurst
School of Computer Science
University of the Witwatersrand
Johannesburg, South Africa
scott@cs.wits.ac.za

Osnat Weissberg
Logic and Validation Technology
Intel Corporation
Haifa, Israel
osnat.weissberg@intel.com

Gila Kamhi
Logic and Validation Technology
Intel Corporation, Haifa, Israel
gila.kamhi@intel.com

Limor Fix
Logic and Validation Technology
Intel Corporation, Haifa, Israel
limor.fix@intel.com

ABSTRACT

One method of handling the computational complexity of the verification process is to combine the strengths of different approaches. We propose a hybrid verification technology combining symbolic trajectory evaluation with either symbolic model checking or SAT-based model checking. This reduces significantly the cost (both human and computing) of verifying circuits with complex initialisation, as well as simplifying proof development by enhancing verification productivity. The approach has been tested on current Intel designs.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*verification*; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: mechanical verification

General Terms

Verification, Theory

Keywords

symbolic model checking, symbolic trajectory evaluation, hybrid verification

1. INTRODUCTION

The need for and benefits of formal verification (FV) have been accepted for some time. However, *symbolic model checking* (SMC) [11], one of the more automated and therefore more popular FV techniques, while a very valuable method for verifying commercial sequential designs, is still limited with respect to the size of the verifiable designs. This capacity problem also affects productivity: not only is effort needed to decompose proofs into simpler proof obligations on the modules of the design, the abstractions and assumptions needed increase the chance of getting false failure reports. Another reason for false failure reports is the difficulty of initialising an abstracted design. This results in spurious failures, which in turn causes the verifier to get into a loop of model checking and specification modification, which consequently hinders productivity.

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.

1.1 MIST: The Hybrid Approach

This paper introduces a hybrid model checking approach (which we call MIST) that makes use of a complementary FV technique, *Symbolic Trajectory Evaluation* (STE) [7] to empower symbolic model checking-based verification.

In the hybrid approach, MIST, the user provides the design under test and the specification just as if using a classic model checker. Additionally, s/he explicitly specifies the initialising behaviour of the design under test or the desired initial state/states for the model checker. The verification flow consists of two phases:

1. STE performs initialising computation and calculates the state (or set of states) which the design would be in at the end of this initialisation process.
2. Using the set of states in the previous step as the starting point, a SAT/BDD-based model checker completes the verification.

1.2 Contributions of research

MIST provides more control over verification and simplifies specification. The resulting benefits are not only improvement in performance, but also productivity advantages due to finding errors more quickly and reducing the chances of spurious counter-examples. The different aspects of this are detailed below:–

Improving the specification process: SMC, though an automatic method, requires significant human intervention in modelling. We need to abstract or prune to bring the design within the capacity range of model checkers, and part of this abstraction requires building abstract models of our pruned model's environment. MIST uses STE's ability to treat large state spaces directly to provide a more accurate and cheaper process of abstraction.

Finding bugs quickly: Generating counter-examples is faster with MIST since it does not require the reconstruction of the initialisation sequence. By making use of previous counter-example

traces, the verifier can find multiple failures fast and consequently back-trace to the root cause more efficiently. Experience with real-life designs shows that the use of STE to initialise the design and bring it to error-prone areas by making use of previous counter-examples can reduce the overall verification time and help in debugging failures efficiently. Precise initialisation of the design by STE also reduces the causes of spurious failures and enables the verifier to move quickly from the specification debugging to the model debugging stage.

Improving capacity and performance of automatic verification: Since the expensive image computations in initialisation can be handled by STE, especially for designs that require a long initialisation sequence, the overall cost of verification can be reduced and we may be able to verify larger systems.

Enhancing SAT-based model checking: A unique contribution is the application of MIST to SAT-based bounded model checking (BMC). SAT-based BMC methods (e.g. [2]) have recently been introduced as a complementary technique to BDD-based symbolic model checking. The cost of BMC is very dependent on the bound used. By allowing fast penetration into the design by STE, MIST reduces the bound needed for the problem.

1.3 Structure of paper

Section 2 provides background and summarises related work. In Section 3, we provide an overview of our hybrid approach to verification. Section 4 presents experimental results illustrating the benefits of the hybrid model checking over classic model checking using current Intel designs. Section 5 concludes.

2. RELATED WORK

2.1 Symbolic Model Checking

Symbolic model checking [11] takes a state-transition graph of the model under test and a property specified in an expressive temporal logic such as CTL or LTL and determines which states satisfy the property. Verification can be performed by computing all states reachable from the initial states and checking that these states satisfy the property. This is done by traversing the state transition graph, representing sets of states symbolically as BDDs, which provide a *characteristic* representation of a state space. SMC's primary limitation is that the BDDs encountered at each iteration can grow very large leading to a blow-up in memory or to a verification time-out. It may be impossible to perform image computation because of the BDDs involved in the intermediate computations. In our experiments we have used Forecast [5], Intel's model checker, a state-of-the art tool that has many optimisations and features that allow it to be used on industrial problems. Nevertheless, a severe capacity problem is witnessed when applied on verification sessions that require many image computation steps.

An important contribution of MIST is its application to bounded model checking. Thunder, our SAT-based model checker [5], turns a symbolic model checking problem to a bounded model checking problem. BMC finds counterexamples of limited length k, and thus it targets falsification and partial verification. To fully verify a property we increase the bound until there are no counterexamples less than the diameter of the FSM. The diameter is very large in some examples, and there's no easy way to compute it in advance. In practice the choice of the bound is critical: too small may fail to find a counter-example, while too large may make the verification problem intractable. Performance is very dependent on bound length. MIST helps because STE can be used for a number of cycles, before switching to BMC, thereby reducing the bound needed for BMC. Consequently, harder BMC problems can be solved.

2.2 Symbolic Trajectory Evaluation

While STE [7] also uses a temporal logic for specification and an FSM for the representation of the circuit, there are important distinctions in relation to SMC. The heart of the STE algorithm is symbolic simulation. Specification is given by assertions of the form $\langle A \Longrightarrow\!\!\gg C \rangle$ where A and C are two temporal logic formulas. A, the antecedent, describes (symbolically) input or stimulus to the circuit while C, the consequent, gives the desired resultant behaviour.

The STE algorithm uses the antecedent to initialise the circuit. STE computes a representative, symbolic sequence of states which satisfies the antecedent. Using the values of this sequence as initialising values, STE symbolically executes the circuit and then checks to see that the consequent is also satisfied.

What makes STE efficient is its novel state representation: the use of X or don't care values induces a lattice structure on the state space. STE also uses a *parametric* representation of the state space rather than the characteristic representation (see [8] for a discussion of the difference between these methods). Together, this enables STE to deal with much larger circuits than SMC.

STE complements SMC. STE is relatively insensitive to circuit size. So, directly representing a system with tens of thousands of latches and simulating for thousands of steps can often be done with relatively low cost. However, the temporal logic which STE supports is much weaker. Recent work on generalised symbolic trajectory evaluation (GSTE) [14] has significantly extended the power of STE-based algorithms to support richer specifications (to all ω-regular properties). We have taken a different approach, where we empower SMC capacity and performance by making use of STE technology and make use of this hybrid verification flow to address shortcomings of debugging and initialisation specification in an SMC-based FV environment.

2.3 Hybrid approaches

Since verification is at least NP-hard, there is no one way that can solve all verification problems efficiently. Our approach, MIST, tackles the verification problem by combining different approaches so as to complement strengths and weaknesses. There has recently been some work in this direction.

Ho *et al.* [9] combined the use of scalar and symbolic simulation together with BMC for improving automatic test generation and unreachability analysis. Later work [13] uses SMC to verify abstractions of their circuits, construct hard counter-examples using ATPG and then use simulation-based techniques to check whether counter-examples are traces of the underlying models.

MIST shares some ideas in common with the idea of using *hints* [3]. Hints, typically constraints on the model's primary inputs, are used to improve the performance of CTL model checking by guiding state space search. MIST differs in purpose (we use the primary inputs to model environment behaviour) and technology.

Technically, the use of both characteristic and parametric representations and the conversion between them is critical for MIST (see [8] for a discussion). [10] developed a general method for creating parametric vectors for parametric representation, but focussed on methods for common predicates used in verification. More recently, Aagaard *et al.* [1] used a method based on Shannon's expansion to partition a verification problem into separate parts. Each part was characterised by a predicate (in characteristic representation) before being converted to parametric form for STE.

MIST converts the other way – from parametric to characteristic – to facilitate STE and SMC hand-shake. Conversion in this direction is easier (but can still be very expensive in general). Because of the nature of computation that STE is doing here, the conversion is cheap in MIST.

Yuan *et al.* [15] propose hybrid approaches that they call saturated simulation that performs a partial traversal of the state space while covering controller interactions. At each step they use symbolic model checking to compute the full set of states reachable in one step from the current step. Although this work has a resemblance to ours, the aim there is falsification and partial traversal of the state space where our approach encompasses the full traversal and addresses the initialisation problem.

Comparison with MIST: MIST introduces a novel integration of STE and SMC that resolves the initialisation problem in SMC based formal verification and simplifies proof development. To our knowledge, there has not been significant work that tackles the initialisation problem in formal property verification, though this topic has been focus for research for the field of formal equivalence verification [12]. Moreover, MIST enhances both the capacity and performance of SAT and BDD-based SMC and helps the debugging process by letting the verifier focus on critical, error-prone areas.

3. MIST: A HYBRID SCHEME

This section presents MIST, the hybrid approach that combines SMC and STE to exploit their respective strengths. STE is used for computations where the focus of the work is computing sets of states, whereas SMC is used for that part of the computation where we are focussed on proving properties.

The scenario envisaged for the use of the hybrid approach is as follows. The user has a model, M, and some property ϕ to be proved. STE cannot be used because its logic is not expressive enough for ϕ. SMC cannot be used because of capacity constraints; dealing with these capacity constraints has not only the obvious computational costs, but more importantly the human costs of modelling abstractions are expensive and reduce productivity.

3.1 Existing flow

To verify M using SMC alone we do the following:

- Build a pruned model M' of M (a relatively automated step)
- Build environment E' (a model of M''s environment and that part of M not modelled in M'). Commonly, initialising behaviour is modelled as part of E', and E' contains relatively detailed knowledge of the circuit behaviour.
- Use SMC to verify the property ϕ of the pruned model M' composed with the environment E'.

The disadvantages of the existing flow are the capacity and modelling constraints discussed in previous sections.

3.2 Proposed flow

The proposed flow works as follows

- Build M', a pruned model of M (the same step as above). The initialising behaviour and inputs of M are given by an STE antecedent A.
- Use STE to exercise the *unpruned* model M under the influence of A. STE's ability to deal with the large unpruned model easily provides the key benefits of enhancements of performance and simplification of modelling.
- The run computes a symbolic set of states which gives, S, the set of states of the machine after initialisation.
- Prove ϕ of M' using SMC/BMC starting from the state set S.

The MIST flow addresses the problems discussed in Section 3.1: construction of initialising behaviour is likely to be more efficient and less error-prone; and STE's computation being significantly faster, we are able to make large performance gains.

MIST Correctness: In principle, STE's computation to find the set of states after initialisation completes is the same computation that SMC would do. There are two provisos. (1) The property ϕ should say nothing about the initialisation phase (since STE cannot capture the path semantics of SMC's temporal logic in this phase of the computation). Provided we just need to know the set of states after initialisation completes (and not the paths required to get there), there is no difference between the two flows in what is being computed, just in how it is being computed. (2) Correctness relies on the correctness of the antecedent A. Of course, in the traditional flow correctness relies on the correct modelling of environmental constraints, and we argue that it is easier to do this in MIST than in the traditional flow.

3.3 Generating the initialising behaviour

MIST requires the initialising sequence of the model M. This sequence may be the circuit's reset behaviour, or any other behaviour that the user wishes (so 'initialises' may be from the circuit's or the user's perspective). An example of the former is the external stimulus to the circuit on physical resetting. An example of the latter is a sequence of actions to bring the circuit into a state or set of states from which the verifier is particularly interested in exploring.

Specifying external stimulus for initialising: This mode is very useful since we can reduce the cost of modelling the environment, which is a lengthy and error-prone process. In our approach, computation can be done by STE on the (large) original model as STE is able to cope with large state spaces directly. In the standard flow, the verifier has to build a model that describes how the pruned-off parts of the model behave. Therefore the hybrid approach reduces the work required by the user. The user can also have much higher confidence in the result as the validity of the result will not be affected by the modelling of the environment.

Providing external stimulus is particularly useful when the circuit has a relatively long reset behaviour. Here significant reduction in computation times will be seen too, since the computation of the reset behaviour by STE is extremely efficient compared to SMC. The longer the reset sequence, the greater the savings.

While we anticipate that describing the reset sequence of the circuit will be the most common way in which external stimulus will be used, MIST is not restricted to this use. Any sequence of external stimuli can be used (and we emphasise that the stimuli can be symbolic). This enables a verifier to drive deep into the state space during the more focused, earlier debugging phases of verification.

Example: For an example circuit, the reset sequence takes 50 cycles. The *reset* line is high for the first 50 cycles and low thereafter. A line called *netInit* toggles up and down a few times, and the mode lines of the circuit (collectively called *configFlag*) is given a set of values. This could be expressed more formally by a temporal logic formula such as (note the use of symbolic values for input):

```
clock_ticks &
always [0-49] reset = 1 &
always [49-99] reset = 0 &
always [0-9,20-29,40-49] netInit=0&
always [10-19,30-39,50-99] netInit=1&
always [0-99] configFlags=[a,b,c,d]
```

Providing an initialising sequence: An alternative approach, useful in the debugging phase where it is important to find counter-examples, is to use a prefix of a trace of states for initialising. Here we have as a starting point a known sequence of states that the circuit goes through: this sequence of states may be symbolic or partial (we may only have a partial state description for each instant). This can be used to drive our machine M into an interesting

starting point.

This is very useful in specification debugging (an unfortunate fact of life) where we can use the same computation several times to find specification errors.

A typical use of providing initialisation sequences is finding multiple counter-examples (MCEs). When a failed verification produces a counter-example, it is often useful to have many counter-examples, but finding MCEs is expensive [4, 6]. The set of MCEs often forms a tree-like structure: i.e., they share a long common prefix (because you need to get to an 'interesting' stage before any failure happens). So we can skip the first part of the counter-example by replaying it with STE to get to the interesting part, before switching to SMC/BMC-based approaches to find MCEs. In BMC, the counter-example found depends on the bound chosen; so by choosing different bounds in the second phase, we can find MCEs. SMC always finds the shortest counter-example, so replaying the prefix will always lead to the same counter-example. However, we can introduce symbolic values in the prefix or use MCE-finding techniques (e.g. [6]) from the end point of the prefix to find MCEs. This improves the performance of these techniques which are very sensitive to counter-example length.

We can also re-use the result of one STE run in many SMC verifications — useful in the debugging phase, and when a number of properties share the same common initialisation.

3.4 Prototype tool

We have built a prototype using this hybrid approach based on Forecast and Thunder, state-of-the-art industrial-strength tools that support both STE and SMC (both BDD-based and SAT-based model checking algorithms). The user provides

- the model description in RTL format
- pruning directives which indicate which part of the RTL model should be pruned for SMC
- the initialisation information
- the properties to be proved.

When verification starts, STE runs on the original model. When it terminates, we have the initial states for SMC. This is converted from the parametric to the characteristic representation (this conversion takes into about the Xs). The SMC tool is then invoked: first the large model is pruned automatically using the pruning directives. The resultant model is then model-checked taking into account the starting state. This is shown in Figure 1. Note, that in conventional model checking, the STE component is not there. Although in MIST we must provide some additional information, the benefit is the reduced cost in modelling the environment and the performance improvements.

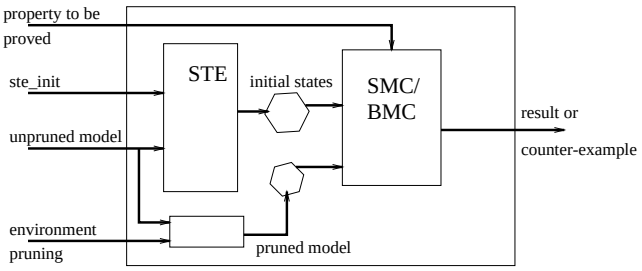

Figure 1: Overview of MIST Prototype System

4. EXPERIMENTS

We evaluated MIST prototype flow against classic SMC and SAT-based BMC flows using Forecast and Thunder, Intel's symbolic and bounded model checkers. The potential benefits of MIST (enhanced specification, capacity and performance boost over classic SMC and BMC, and enhanced debugging) drove our experiments. We assessed each benefit by making use of real verification and falsification sessions on two different current Intel designs.

Properties proved include liveness and safety properties, and focus on proving the correctness of the control-path. The typical format of a property is

```
assuming x, y, z: always (a ==> b).
```

where x, y, and z are LTL liveness or safety properties and a and b are LTL safety properties. (These are examples that occur in real designs — our method doesn't rely on formulas of this structure.)

Section 4.1 compares the MIST and SMC/BMC flows. 64 verification sessions on a current Intel design were used for evaluation. The key benefit demonstrated is improvement in the specification process. Initialisation is given using the full, unpruned model in MIST, but in the classical model checking flow, additional assumptions are needed on the pruned model. We also measure a number of performance parameters (CPU time and number of variables) to show the improvement of capacity and performance.

Section 4.2 assesses the benefit of the MIST flow in improving the debugging phase of circuit verification. We show, by running a set of experiments comparing MIST to both classic SMC and BMC flows, how MIST helps in this phase (when verification sessions mainly fail), and how MIST can be used to improve the finding of counter-examples.

4.1 Verification of a FIFO queue

The first case study was an example from the Intel Pentium III microprocessor. This RTL describes the control of a pseudo-FIFO queue (a FIFO queue with several tail pointers). Verification deals with the correctness of relationships between these pointers. Large parts of the circuit share similar initialisation sequences and thus this case study can be easily extended to new specifications. The circuit had been verified using standard techniques. It contains 15500 variables in the full model, and the verification has been broken into 64 separate model checking tasks. Different verification sessions are used to hierarchically compare different subsets of the reference model and implementation. Properties are in the style: *if the RTL and the reference model agree so far, then they will agree in the next clock cycle.*

Specifying the initialisation sequence: In the original 'standard' proof, reset behaviour is modelled by assumptions, FSMs that observe and restrict the behaviour of the RTL model. Some operations are especially expensive in this working model, particularly the counting of time. For example, the specification 'reset should be high during the first 5 cycles' is modelled by a 3 bit counter that counts the time since startup and makes sure reset is true. In STE, the assignments are coded using an antecedent, similar to test vectors in simulation. Such specification is straight forward. You simply drive reset appropriately during the first 5 cycles.

In general, initialisation sequences lend themselves well to STE-style specification, which can be done in a straight forward way. In this case the initialisation sequence can be broken into the following lemmas:

- clocks are active, and clock enables are high, and reset is active for 5 cycles and then inactive forever;
- No outside events are allowed in the second part of reset, and for some time after reset becomes inactive;
- Testability features are disabled;

- Configuration flags are stable (do not change over a sampling period)–achieved via symbolic values.

The full initialisation sequence costs 2 variables in STE versus 20 in SMC specification (needed for the building the abstraction of the environment, in particular the counters needed to describe the reset input behaviour).

A benefit of the STE style of initialisation is that it is done on the full RTL model (before pruning and abstraction) and thus is easier to specify in terms of RTL primary inputs. On the other hand, the SMC specification must take into account the abstractions/pruning done and thus is lower-level and deals with the model's internal signals. Usually initialisation sequences are well-defined and documented over the primary inputs, but not as well-defined or justifiable over the internal signals.

Debugging the initialisation sequence: Once the initialisation sequence specification is written, one needs to debug to ensure its correctness (as with any formal specification). The main problem in initialisation sequence specification is that it may be partial, i.e. some relevant signals are not being initialised. In STE, this is debugged by running the sequence and checking for Xs (undefined values). Once all relevant Xs are eliminated, and the key signals have the correct value, the initialisation is ready for use as input to the model checker.

Problems in the initialisation sequence in SMC manifest as spurious counter-examples, usually because some of the initialisation in the RTL has been abstracted out. These are hard to reconstruct and solve. Debugging the SMC initialisation took weeks. Switching to STE style took a few days. Of course, we had some benefit of experience with the model, but on the other hand we had no experience with the MIST methodology before this example.

Experimental results of performance improvement: We ran 64 real test cases using Forecast in the traditional flow and in the MIST flow. Figures 2 shows how the two methods compared by showing for each experiment, how long the verification took. Table 1 summarises this data and shows the relative performance of the MIST flow to the standard flow, by showing how many experiments showed what improvement (or degradation) of performance. 84% of the test cases led to improvement – in 61% of cases MIST was at least twice as fast.

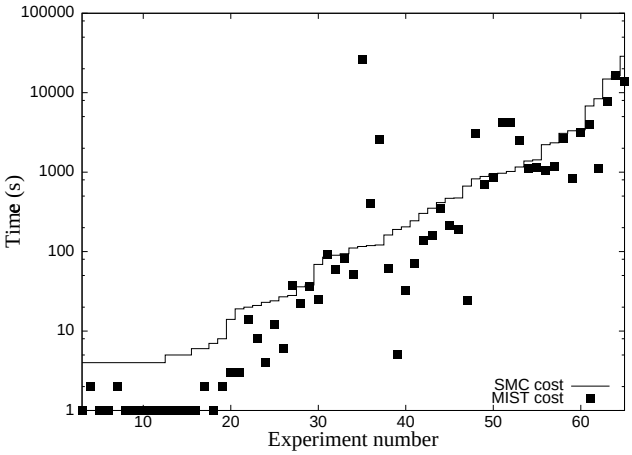

Figure 2: ***MIST versus SMC performance: Time is measured in seconds and shown on a log scale.***

Other metrics also show improvement under MIST. For example, the number of variables needed in verification is reduced on average by 22% (the average number of variables used in the standard flow is 120, with the smallest case being 69 and the largest case being 185). There are some cases where MIST leads to longer results. These are likely caused by the different initialisations which sometimes has a negative effect on performance.

r	< 0.5	$[0.5, 1)$	$[1, 2)$	$[2, 3)$	$3 \leq$
	7	3	15	13	26

Table 1: ***Relative performance of MIST to standard flow: r gives the MIST speedup; entries in the table show number of cases with speed-up in that range.***

We have not done a comparison using SAT-based technologies: in all these cases, the verifications are true and we have no reasonable way of choosing bounds for BMC.

4.2 Internal communication circuit

In this experiment we looked at the verification of a bus protocol from an Intel Pentium III mobile microprocessor. The circuit had been verified using standard techniques. However, the process was extremely expensive, especially in the debugging phase. Some verification runs took days to complete. The circuit has extensive initialisation (124 steps). Taking this circuit, we verified using both the standard and MIST flow five properties, which we call A, B, C, D and E. These are large problems with between 330 and 540 variables needed for model checking.

For all properties we used Forecast. The table below shows the results, the time taken for verification. The cost of the STE component was very small, in the order of 10s. There is a small reduction (about 4%) of the number of variables needed in the MIST flow.

Case	MIST Flow Time (s)	Forecast Flow Time (s)
A	75294	129230
B	7588	24989
C	54045	54725
D	642	66654
E	2643	68779

We also examined test case E using Thunder, comparing the standard flow to MIST. Figure 3 shows the results. We did the verification using MIST and the standard flow with different bounds in order to understand the effect of the choice of bound. A verifier using MIST would adjust the bound set downwards: if they would have used a bound of length b in the standard flow, they would use a bound of length $b - i$ in the MIST flow, where i is the length of the initialising sequence. In this experiment $i = 125$. Thus we compare the case where the standard flow has bound 225 and the MIST flow has bound 100, the case where the MIST flow has bound 120 and the standard flow has bound 245 and so on. In the figure, the x-axis is labelled by the bounds (MIST/Standard) and the y-axis by time.

These results also show that using MIST we can explore deeper into the state space. For example, in less than the time required by the standard flow to get to step 225, using MIST we can go to step 325. This shows that the verifier can guide the system into an interesting state and continue exploration from there.

Finding multiple counter-examples: We also assessed the effectiveness of the MIST flow on finding multiple counter examples. We used multiple counter example finding techniques [6] on a test case using both MIST and standard flows. Using MIST, the time required to generate multiple counter examples dropped from 16409s to 1368s.

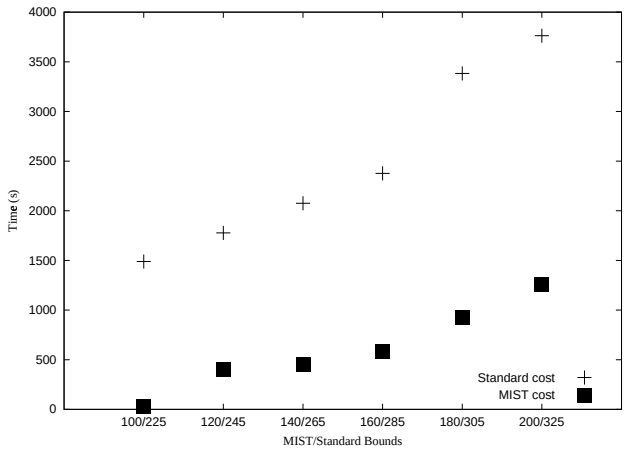

Figure 3: ***MIST versus BMC performance***

5. CONCLUSION AND FUTURE WORK

MIST is a hybrid method of verification using STE, BMC, and SMC. We are able to use the power of STE to deal with large circuits directly to improve ease of specification, performance, and productivity.

We tested this method with current Intel tools and designs. The experiments show that the modelling of initialisation can be simplified and made more precise, thereby increasing productivity and the confidence in the meaningfulness of the verification result, as well as the chances of getting spurious counter-examples. We can use the unpruned model efficiently to extract out useful information in the initialisation phase that can be used in the later phase of verification.

The application of our hybrid approach on real-life industrial test cases shows that MIST can significantly boost the performance and capacity of SAT/BDD-based symbolic model checking. Furthermore, our methodology enables the verification engineer to have much more control over the verification process, facilitating a better debugging environment.

Future work: First, we would like to look at how the cooperation between STE and SMC can be generalised further (not just during the initialisation phase). Can we use the complementary strengths of the two techniques together to extend performance and simplify the modelling phase? Second, the experimentation showed that the performance is very sensitive to the initial states. We believe there is more scope in leveraging even better performance here, and more work needs to be done to understand the best way of automating the flow. Finally, the insight that initialisation is a very important part of the verification of many systems may be helpful in other flows and verification methodologies. This is a point which we wish to take further.

Acknowledgements: We warmly thank the following for their generous help: Evguenia Belfer for her help in applying MIST on real-life Intel designs; Ranan Fraer, Moshe Y. Vardi for their review, useful comments and suggestions; Shlomit Ozer helped in the development of MIST; and Robert Beers and Tom Schubert originally suggested the use of STE for initialising symbolic model checking.

6. REFERENCES

[1] M.D. Aagaard, R.B. Jones, and C.-J.H. Seger. Formal verification using parametric representations of boolean constraints. In *Proc. 36th ACM/IEEE Design Automation Conf.*, pp. 402–407, 1999.

[2] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. 36th ACM/IEEE Design Automation Conf.*, pp. 317–320, 1999.

[3] R. Bloem, K. Ravi, and F. Somenzi. Symbolic Guided Search for CTL Model Checking. In *Proc. 37th. ACM/IEEE Design Automation Conference*, pp. 29–34, 2000.

[4] E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. of 32nd ACM/IEEE Design Automation Conf.*, 1995.

[5] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *Proc. 13th Int. Conf. on Computer-Aided Verification*, Lecture Notes in Computer Science 2102, pp. 436–453. Springer-Verlag, 2001.

[6] F. Copty, A. Irron, O. Weissberg, N. Kropp, and G. Kamhi. Efficient Debugging in a Formal Verification Environment. In *Conf. on Correct Hardware Design and Verification Methods (CHARME 2001)*, pp. 275–292, September 2001.

[7] S. Hazelhurst and C.-J.H. Seger. Symbolic Trajectory Evaluation. In Kropf, T (Ed). *Formal Hardware Verification: Methods and Systems in Comparison.* LNCS 1287. Springer-Verlag, Berlin, 1997, pp. 3–79.

[8] S. Hazelhurst. On Parametric and Characteristic Representations of State Spaces. *Technical Report 2002-1*, School of Computer Science, University of the Witwatersrand, March 2002. Available as ftp://ftp.cs.wits.ac.za/pub/research/reports/TR-Wits-CS-2002-1.ps.gz.

[9] P.-H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long. Smart Simulation using Collaborative Formal and Simulation Engines. In *Proc. IEEE/ACM International Conf. on Computer Aided Design*, pp. 120–126, November 2000.

[10] P. Jain and G. Gopalakrishnan. Efficient symbolic simulation-based verification using the parametric form of boolean expressions. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 13(8):1005–1015, August 1994.

[11] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academics, 1993.

[12] C. Pixley, S. W. Jeong, and G. D. Hachtel. Exact Calculation of Synchronization Sequences Based on Binary Decision Diagrams. In *Proc. ACM/IEEE Design Automation Conf.*, pp. 620–623, June 1992.

[13] D. Wang, P.-H. Ho, J. Long, J. Kukula, Y. Zhu, T. Ma, and R. Damiano. Formal Property Verification by Abstraction Refinement with Formal, Simulation and Hybrid Engines. In *Proc. 38th ACM/IEEE Design Automation Conf.*, pp. 35–40, 2001.

[14] J. Yang and C.-J.H. Seger. Introduction to generalized symbolic trajectory evaluation. In *Proc. of ICCD*, September 2001.

[15] J. Yuan, J. Shen, J. Abraham, and A. Aziz. On Combining Formal and Informal Verification. In *Proc. CAV '97*, pp. 376–387, 1997.