

Demonstration of Hardware-Accelerated Formal Verification

Hiroaki Yoshida^{1,2} Satoshi Morishita³ Masahiro Fujita^{1,2}

1. VLSI Design and Education Center (VDEC), University of Tokyo

2. CREST, Japan Science and Technology Agency

3. Dept. of Electronic Engineering, University of Tokyo

Abstract—A semi-formal verification technique, which performs a brute-force compiled simulation with a sophisticated search space pruning, has been proposed and shown to be competitive with the state-of-the-art SAT-based verification techniques [1]. This paper presents a novel approach for accelerating the semi-formal verification by utilizing hardware/software co-execution. To maximize the gain from hardware acceleration, we propose two novel techniques such as hardwired conflict analysis for learning and speculative input pattern generation. We demonstrate that our FPGA-based prototype system achieves about 7x speedup compared against the software implementation of the semi-formal verifier.

I. INTRODUCTION

With the increasing importance and substantial cost, efficient formal verification of a large-scale design has become a major challenge. One of the most promising approaches for overcoming this challenge is the use of hardware accelerator. A number of studies on hardware acceleration of SAT solver, a core engine of formal verification, have been available [2], [3]. However, most hardware-accelerated SAT solvers have not integrated advanced techniques used in the state-of-the-art SAT solvers, such as the conflict learning and non-chronological backtracking. Recently a semi-formal verification technique, which performs a brute-force compiled simulation with a sophisticated search space pruning, has been proposed and shown to be competitive with the state-of-the-art SAT-based verification techniques.

This paper presents a novel approach for accelerating the semi-formal verification by utilizing hardware/software co-execution. To maximize the gain from hardware acceleration, we propose several novel techniques such as hardwired conflict analysis and speculative input pattern generation. Experimental results demonstrate that our FPGA-based prototype system achieved about 7x speedup compared against the software-only implementation of the semi-formal verifier.

II. SEMI-FORMAL VERIFICATION TECHNIQUE

Bingham and Hu [1] proposed a semi-formal bounded model checking method¹ based on compiled simulation, as a replacement of SAT-based model checkers. The main disadvantage of a simulation-based approach is the exponential number of possible input vectors. The semi-formal method

¹We omit the details of bounded model checking here since it is irrelevant to the main argument. For details please refer to [4]

overcomes this disadvantage by pruning the search space analogously to the advanced techniques used in the state-of-the-art SAT solvers.

Procedure 1 Semi-Formal Verification

```

1:  $V \leftarrow \emptyset$ 
2: repeat
3:    $v \leftarrow$  A random minterm in  $\bar{V}$ 
4:    $(f, c_{skip}) \leftarrow \text{simulate\_circuit}(v)$ 
5:   if  $f = \text{TRUE}$  then
6:     return SATISFIABLE
7:   end if
8:    $V \leftarrow V \cup \{c_{skip}\}$ 
9: until  $V = 1$ 
10: return UNSATISFIABLE

```

In the remainder of this section, we will explain the details of the semi-formal verification method. Procedure 1 describes the procedure of the semi-formal verification. In the procedure, V maintains the set of input vectors which have been verified. First, a new input vector which has not been verified is picked up as a random minterm from \bar{V} and a compiled simulation for the input vector is performed by calling a function `simulate_circuit`.

In addition to the compiled simulation, the function `simulate_circuit` computes the skip cube of the primary output (verification target). A skip cube is a subset of the input values which determines the value of the output independent of the other input values. The skip cube $A_v(w)$ of a wire w with respect to an input vector v is defined as follows:

Definition 1 The skip cube $A_v(w)$ of a wire w with respect to an input vector v is defined depending on the type:

- 1) If w is the output of a two-input gate with controlling value μ (e.g. $\mu = 0$ for AND gates) and input wires a and b ,

$$A_v(w) = \begin{cases} A_v(a) \cap A_v(b) & \text{if } a_v = \bar{\mu} \wedge b_v = \bar{\mu} \\ A_v(b) & \text{if } a_v = \bar{\mu} \wedge b_v = \mu \\ A_v(a) & \text{if } a_v = \mu \wedge b_v = \bar{\mu} \\ \max(A_v(a), A_v(b)) & \text{if } a_v = \mu \wedge b_v = \mu \end{cases}$$

where `max()` returns the set of greater cardinality.

- 2) If w is the output of an inverter with input a , $A_v(w) = A_v(a)$.

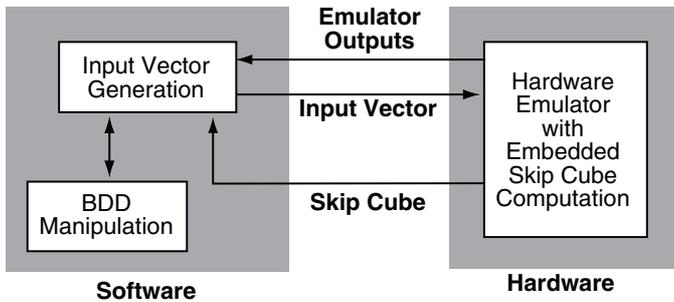


Fig. 1. System organization.

- 3) If w is an input x_i , $A_v(w) = (B_1, \dots, B_N)$ where $B_j = w_v$ if $j = i$, or $B_j = \text{---}$ (don't care) otherwise.
- 4) If w is a two-input gate without a controlling value (i.e. exclusive-OR or exclusive-NOR) with input wires a and b , $A_v(w) = A_v(a) \cap A_v(b)$.

The skip cube of a gate with more than 2 inputs can be defined similarly. In the procedure, the skip cube $A_v(f)$ of the primary output f is computed for each input vector v in the function `simulate_circuit` as well as the simulation of the target circuit. Since any input vector contained in the skip cube leads to the same output, we can skip all input vectors in the skip cube from the next iterations by adding the skip cube to V . Thus, the number of input vectors which are actually simulated is significantly small.

III. HARDWARE/SOFTWARE CO-EXECUTION SYSTEM FOR FORMAL VERIFICATION

A. System Overview

In this paper, we assume that a given verification problem can be transformed into a problem of checking the satisfiability of a circuit. For instance, suppose a given problem is the equivalence checking between two circuits. First a new circuit is generated by adding a gate which computes the exclusive-OR of the primary outputs of two circuits, then the equivalence is proved by checking that the output of the exclusive-OR gate is false for all possible input vectors. Similarly, a property checking problem can be transformed by adding an extra circuit which checks the property where the output of the extra circuit is true *iff* the property is true. There are several methods proposed for generating such an extra circuit [5]. Thus, under the assumption that a target circuit has one and only primary output f which is the verification target, our verification system focuses only on the problem of checking if f is true for all input vectors.

Initially, we implemented a software semi-formal verifier described in Procedure 1 and performed a runtime profiling analysis using an example circuit used in Section IV. The profiling result, as presented in Table I, shows that the circuit simulation and the skip cube computation spent most of the execution time. Based on the analysis, we designed a system which performs Procedure 1 in a hardware/software co-execution fashion, as illustrated in Figure 1. The circuit simulation and skip cube computation in the procedure are executed using hardware. We first describe how to efficiently

TABLE I
RUNTIME PROFILING RESULT OF SOFTWARE SEMI-FORMAL VERIFIER.

Process	Percentage of Execution Time
Circuit simulation	49.2%
Skip cube computation	36.0%
Update of input vector set (BDD)	13.1%
Input vector generation (BDD)	1.7%

implement the skip cube computation. Next, to resolve the communication bottleneck, we propose a speculative input pattern generation technique.

B. Hardware Emulation with Embedded Skip Cube Computation

In our verification system, we accelerate Procedure 1 by executing the circuit simulation and the skip cube computation (i.e. `simulate_circuit`) using hardware. Obviously, the circuit simulation can be easily implemented in hardware. If the skip cube computation is implemented using hardware just the way described in Def. 1, it may require N -bit Boolean operations for each gate in the circuit where N is the number of primary inputs. The size of skip cube computation circuit increases $O(G \cdot N)$ where G is the number of gates, and critically limits the capacity of a target design. To avoid this large overhead, we propose a novel circuit where every gate embeds a fixed skip cube computation logic. The size of our circuit is proportional to the original circuit size regardless of the number of inputs.

First, we slightly modify the definition of the skip cube for efficient hardware implementation as follows:

Definition 2 The skip cube $A_v(w)$ of a wire w with respect to an input vector v is defined depending on the type:

- 1) If w is the output of a two-input gate with controlling value μ (e.g. $\mu = 0$ for AND gates) and input wires a and b ,

$$A_v(w) = \begin{cases} A_v(a) \cap A_v(b) & \text{if } a_v = \bar{\mu} \wedge b_v = \bar{\mu} \\ A_v(b) & \text{if } a_v = \bar{\mu} \wedge b_v = \mu \\ A_v(a) & \text{if } a_v = \mu \wedge b_v = \bar{\mu} \\ A_v(a) & \text{if } a_v = \mu \wedge b_v = \mu \end{cases}$$

- 2) 3. 4. Equivalent to Definition 1.

Next, we define a propagation flag:

Definition 3 The propagation flag (p-flag) $w_v^p \in \{0, 1\}$ of a wire w with respect to an input vector v is 1 if the skip cube $A_v(w)$ of the wire w is propagated to the primary output, and 0 otherwise.

In the proposed circuit, we replace each logic gate with a logic block which not only performs the original logic operation but also computes the p-flags. Each wire in the original circuit is replaced with two wires, one is for the signal and the other is for the p-flag. The direction of the p-flag computation is opposite to the logic operation, that is, the p-flags of the gate inputs are computed from the p-flag of the gate output.

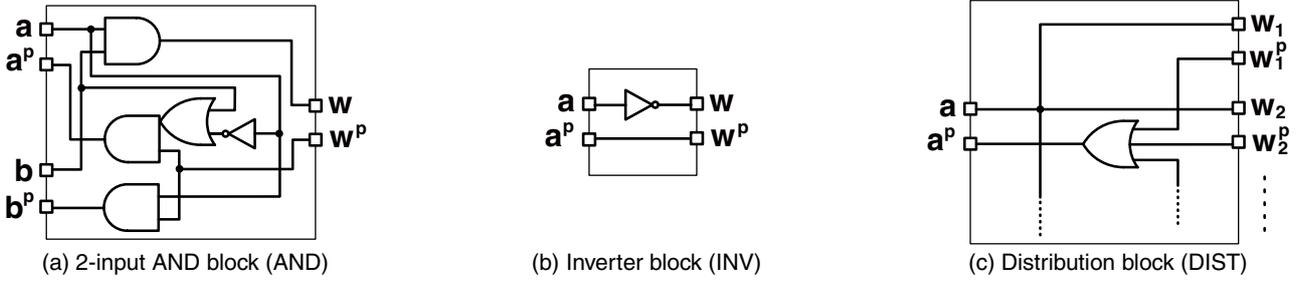


Fig. 2. Basic logic gate blocks for embedded skip cube computation.

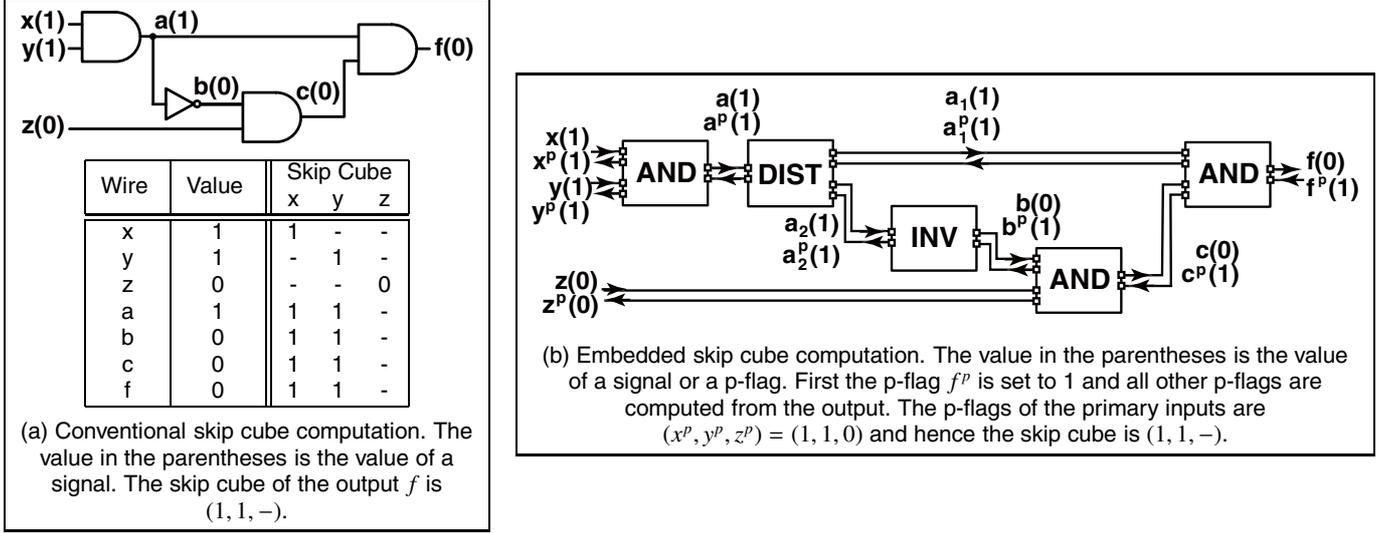


Fig. 3. Examples of two skip cube computation methods. In this example, an input vector $(x, y, z) = (1, 1, 0)$ is simulated and the computed skip cube $A_v(f)$ is $(1, 1, -)$. Thus, an input vector $(1, 1, 1)$ can be skipped from the simulation.

After setting the p-flag of the primary output to 1, the p-flag of each wire is computed in a topological order from the primary output to the primary inputs. Figure 2 (a)-(c) show basic logic gate blocks with embedded p-flag computation. For ease of explanation, it is assumed that a given circuit consists only of 2-input AND gates and inverters. An extension to more general circuits should be straightforward. A circuit for hardware emulation with embedded skip cube computation can be obtained by simply replacing each 2-input AND gate with the 2-input AND block in Figure 2 (a), each inverter with the inverter block in Figure 2 (b) and each multiple fanout point with the distribution block in Figure 2 (c). Figure 3 illustrates the example of the skip cube computation by the conventional method (Figure 3 (a)) and the proposed hardwired method (Figure 3 (b)). The circuit in Figure 3 (a) is transformed into the circuit in Figure 3 (b) by applying the rules described above.

After computing the p-flags of all wires, we can compute the skip cube $A_v(f)$ of the primary output f by the following theorem:

Theorem 1 Let C be a cube (B_1, \dots, B_N) such that

$$B_i = \begin{cases} v_i & \text{if } P_i = 1 \\ \text{"-"} & \text{if } P_i = 0 \end{cases}$$

where v_i is the value of the i -th input and P_i is the p-flag of i -th primary input. Then, C is equivalent to the skip cube $A_v(f)$ of the primary output f .

Proof We omit the proof due to the limit of space.

Thus, we can easily compute the skip cube from the p-flags of the primary inputs.

C. Speculative Input Vector Generation

In a typical hardware/software co-execution system, the communication between hardware and software could be a major performance bottleneck. Every step in Procedure 1 must be executed sequentially because an input vector for an iteration depends upon a skip cube of the previous iteration and vice versa. Therefore, every iteration in the procedure requires at least one distinct hardware/software communication. Since our system uses a generic communication method like UDP/IP via Ethernet, there is a large overhead in every communication. To resolve such a communication overhead, the number of communications is reduced by applying a slight modification to the original procedure. Instead of generating an input vector for each iteration, a certain number of input vectors are randomly picked up from \bar{V} . Then, the set of input vectors are transmitted to the hardware at once. The simulation is performed for all input vectors and the set of the outputs and

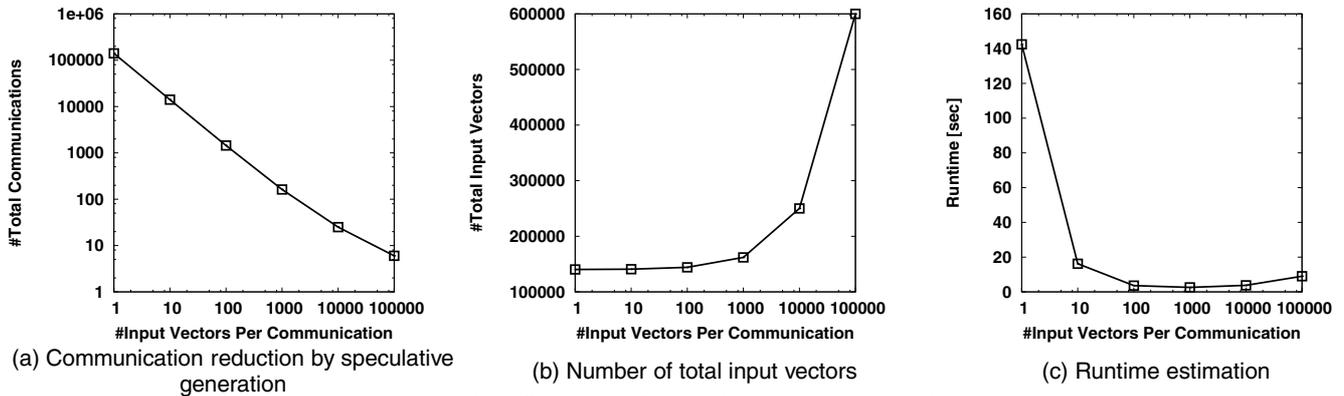


Fig. 4. Effectiveness of speculative input vector generation.

skip cubes are returned. The set of the skip cubes are added into V .

A potential drawback of this speculative vector generation is that many input vectors could be redundant because they are contained in the skip cubes computed for the other input vectors. By using an example, we will show the effectiveness of the speculative generation. As an example, the equivalence of the lower 8 bits of the outputs of two 16×16 bit multipliers is checked. Figure 4 (a) presents the number N_C of total communications when a certain number N_S of input vectors are generated speculatively for each communication. This graph clearly shows that the number of communications can be reduced dramatically by using the speculative generation. Figure 4 (b) presents the number of total input vectors N_V generated for the full verification, *i.e.* $N_C \times N_S$. As can be seen easily, the speculative generation is effective up to $N_S = 1000$. Figure 4 (c) presents the estimated runtime under the practical assumption that the computation for each iteration requires $25\mu s$ and the single communication requires $1ms$. In this example, we found that $N_S = 1000$ leads to the best. This example demonstrates that the speculative generation can dramatically reduce the communication overhead.

IV. EXPERIMENTAL RESULTS

We implemented the proposed hardware/software co-execution verification system described in Section III. The software portion is implemented on Linux PC system on Core 2 Duo 2.0GHz with 2 GB main memory, and the hardware portion is implemented using a Xilinx Virtex-II Pro FPGA with embedded PowerPC processor. For the communication between the PC and the FPGA, the UDP/IP via Ethernet is used and the embedded processor in the FPGA controls the UDP/IP communication.

We used a 16×16 bit multiplier as an example circuit, and our system is used for checking the equivalence of the lower n -bits of the outputs. We also implemented the semi-formal model checker explained in Section II and performed the same equivalence checking as a reference. Table II compares the runtime of software-only execution and hardware/software co-execution, and also shows the speedup factor. The results show

TABLE II
SPEEDUP BY HARDWARE ACCELERATION.

Name	Software Only [sec.]	HW/SW Co-Execution [sec.]	Speedup Factor
MULT16-8	5.7	1.0	5.7
MULT16-9	25.9	3.7	7.0
MULT16-10	122.5	16.5	7.4
MULT16-11	575.3	88.0	6.5
Average			6.7

that our system was about 7x faster than the state-of-the-art semi-formal model checker.

V. CONCLUSIONS

In this paper, we proposed a novel hardware/software co-execution approach for accelerating the semi-formal verification. To maximize the gain from hardware acceleration, we proposed several novel techniques such as hardwired conflict analysis and speculative input pattern generation. Despite the recent progress in formal verification, conventional simulation is still widely used for functional verification. From another perspective, our approach can be regarded as a technique for enhancing the coverage of simulation-based verification. Experimental results demonstrated that our FPGA-based prototype system achieved about 7x speedup compared against the software implementation of the semi-formal verifier.

REFERENCES

- [1] J. D. Bingham and A. J. Hu, "Semi-formal bounded model checking," in *Proceedings of International Conference on Computer Aided Verification*, 2002, pp. 280–294.
- [2] P. Zhong, M. Martonosi, P. Ashar, and S. Malik, "Solving boolean satisfiability with dynamic hardware configurations," *Lecture Notes In Computer Science*, vol. 1482, pp. 326–335, 1998.
- [3] M. Abramovici, J. T. de Sousa, and D. Saab, "A massively-parallel easily-scalable satisfiability solver using reconfigurable hardware," in *Proceedings of Design Automation Conference*, 1999, pp. 684–690.
- [4] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," *Lecture Notes in Computer Science*, vol. 1579, pp. 193–207, 1999.
- [5] M. Boule and Z. Zilic, "Incorporating efficient assertion checkers into hardware emulation," in *Proceedings of IEEE International Conference on Computer Design*, 2005, pp. 221–228.