Hardware/Software Co-Design/Execution Approach to Silicon Debug and Diagnosis

Masahiro Fujita* **Hiroaki Yoshida*** **Satoshi Morishita****
*VLSI Design and Education Center, University of Tokyo
**Department of Electronic Engineering, University of Tokyo

Extended Abstract

This paper introduces new approaches to silicon debug and diagnosis by utilizing hardware/software co-design methodologies which realizes semi-formal verification/debug/diagnosis on chips. The approach implement them as hardware/software co-design/execution on the chips. In general large VLSI, so called SoC (System on a Chip), has one or more microprocessors and/or DSPs which can be utilized to run software parts of the semi-formal verification/debug/diagnosis, and extra programmable devices, such as sorts of FPGAs, are added to run the hardware parts as shown in Figure 1.

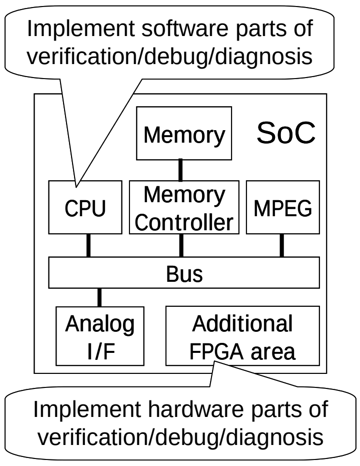
Implement software parts of verification/debug/diagnosis
Memory
SoC
CPU
Memory Controller
MPEG
Bus
Analog I/F
Additional FPGA area
Implement hardware parts of verification/debug/diagnosis

Figure 1. Basic ideas of the proposed method

Techniques for formal verification/debug/diagnosis are complicated programs which can be run on microprocessors on SoC in order to realize silicon debug and diagnosis. Although the techniques are typically implemented as software, its performance can be significantly improved if performance critical portions are implemented as hardware instead of software. This is a typical approach in C-based hardware/software co-design methodologies as shown in Figure 2. Figure of all the original software-only implementation as evaluated with same design data in order to identify which portions are relatively time-consuming. These are currently the bottleneck in software implementation and the target for hardware implementation. Of course even if some portions are time consuming in software, they may not be good for hardware implementation due to their complicated control flow and/or ways of computations. In such cases, the original software could re rewritten so that their hardware implementations become easier..

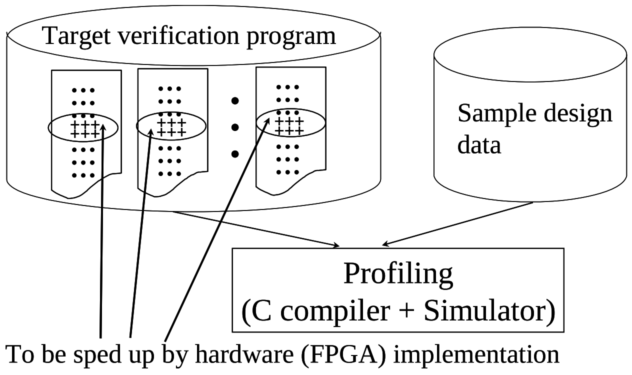
Target verification program
Sample design data
Profiling (C compiler + Simulator)
To be sped up by hardware (FPGA) implementation

Figure 2. A typical way to identify portions to be implemented as hardware

Sometimes this modification of the original software or even changes of the partial algorithms used are the keys for the success of hardware/software co-design approaches. We have explored such modifications targeting semi-formal verification techniques, such as bounded model checking and model checking with some sorts of random walks. Please note that this formal methods can be the bases for silicon debugging and diagnosis, since they can identify error/fault location through state space search possibly with extra observability and controllability through extra test/observation points inserted into the silicon.

As mentioned before, although every formal method can be run on microprocessors in SoC, their performance can be significantly improved by using extra programmable areas (such as the ones implemented like FPGAs) as shown later. Based on this understanding, we have explored possibilities to implement well known formal analysis techniques such as SAT, ATPG, BDD, and others efficiently and effectively in FPGAs as hardware. If this can be made, the architecture shown in Figure 1 works well, resulting in realization of higher performance on formal verification/debug/diagnosis. In SAT procedures, there are several intelligent ways to prune dramatically search space and also very sophisticated ways to implement the algorithms in software. The first question we try to resolve is on how to implement them in hardware. For example, non-chronological backtracking dramatically improves the performance.

There are several ways to implement it efficiently in hardware.

Here we briefly show one such example. We start with the semi-formal verification method shown in [1] and apply hardware/software co-design approach shown in Figure 2 to implement it. The technique shown in [1] is based on complied simulation which can be easily realized on top of FPGAs. In order to prune search space in exhaustive compiled simulations, it uses so called "skip cube" to cut redundant patterns for simulation. A skip cube represent all equivalence input patterns which result in the same simulation results at outputs. For example, if the circuit is the one shown in Figure 3, the value 0 of the third input completely determines the output value. So all input patterns in {-, -, 0} where "-" means don't care generates the same output value. The basic algorithm shown in [1] is to compute such skip cube when simulating the target circuit with given input patterns. In the case of Figure 3, given input pattern is (1, 1, 0). When computing the value of each gate pin, the algorithm checks if other value combinations for the inputs of the gate give the same value of its output. Computing complete sets of skip cubes corresponds to obtaining the complete set of satisfiability don't cares for the gate and could be very time consuming process. Therefore in [1] only one skip cube is extracted to be friendly integrated with compiled simulations. Skip cubes dramatically reduce the search space for exhaustive simulation as shown in [1].

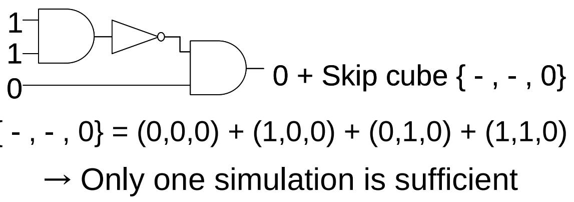
1
1
0
0 + Skip cube { - , - , 0}
{ - , - , 0} = (0,0,0) + (1,0,0) + (0,1,0) + (1,1,0)
→ Only one simulation is sufficient

Figure 3. One way to prune search space for formal verification/debug/diagnosis based on [1]

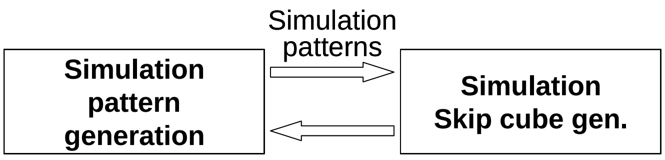

Figure 4. A hardware/software co-design/execution to semi-formal verification

The original implementation in [1] is fully software based one. We try to implement it partially in FPGA as hardware so that it can be efficiently implemented in the architecture shown in Figure 1. The profiling results for the original software implementation (of our own based on [1]) shows the most time consuming portion is the computation of skip cubes. So we decide to try to implement it in hardware which results in the hardware/software partitioning shown in Figure 4. The left part of the figure is implemented as software, and the right part is implemented in hardware.

So the left part of the figure is run on microprocessor in Figure 1, whereas the right part is implemented on FPGA areas in Figure 1.One critical issue is the fact that we have to revise the way to compute skip cubes from the original one so that it can be efficiently and effectively implemented as circuits of FPGA. We come up with a new way to compute skip cubes with additional hardware whose size is (number of inputs) * (the original circuit size) as shown in Figure 5.. This could be too large if the original circuit has many inputs, and so in such cases, we will limit the search space of skip cubes by limiting the numbers of inputs to skip cube computation circuits shown in Figure 5. Also as shown in the figure there are a number of connections from the original circuits to the skip cube computation circuits. These connections can be reduced depending on the requirements from the architecture shown in Figure 1, which also results in less search space in skip cube computations.

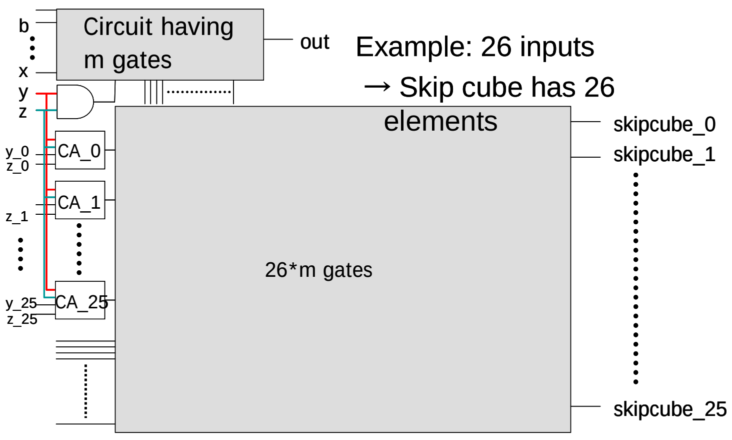

Figure 5. Use of additional FPGA are in Figure 1 to implement hardware portions of proposed method

Our experiment on the semi-formal verification of various multipliers show that several times speed up can be realized with the architecture shown in Figure 1. In the case of 16-bit multipliers, around 7 times speed up is observed in average. The speed up is in comparison with the state-of-the-art Intel fastest processors. Since typical SoC has much slower microprocessors on chip, the speed up may become much larger.

The above experiment is on semi-formal verification of circuits which can be the first step in silicon debug and diagnosis. The method shown above can be utilized effectively in after silicon debugging. The extra FPGA areas in Figure 1 take the most important roles in them. In the full paper, we will demonstrate them by using hardware/software co-design/execution approaches.

References

[1] J. D. Bingham, A. J. Hu, "Semi-Formal Bounded Model Checking," CAV 2002