A Property Checking Approach to Microprocessor Verification using Symbolic Simulation

Prabhat Mishra[†] pmishra@cecs.uci.edu
Narayanan Krishnamurthy[‡] nari@ibmoto.com
Nikil Dutt[†] dutt@cecs.uci.edu
Magdy Abadir[‡] M.Abadir@motorola.com

[†]Architectures & Compilers for Embedded Systems
Center for Embedded Computer Systems
University of California, Irvine, CA 92697

[‡]High Performance Tools and Technology
Somerset Design Center
Motorola Inc., Austin, TX

Abstract

Several bottom-up validation techniques have been proposed to formally verify the implementation of a microprocessor by comparing the pipelined implementation with its Instruction-Set Architecture (ISA) specification model, or by deriving the ISA model from the implementation. We present a top-down validation approach using symbolic simulation. We define a set of properties and verify the correctness of the processor by verifying if the properties are met. We applied our methodology to verify several properties on a Memory Management Unit (MMU) of a microprocessor that is compliant with the PowerPC instruction-set architecture to demonstrate the usefulness of our approach.

1 Introduction

Shrinking time-to-market cycles coupled with short product lifetimes create a critical need to drastically reduce microprocessor design cycle time. Since verification and design analysis are major components of this cycle time, any effort that improves verification effectiveness and design quality is crucial for meeting customer deadlines and requirements. Design validation techniques can be broadly categorized into simulation-based approaches and formal techniques. Due to the complexity of modern designs, validation using only traditional scalar simulation cannot be exhaustive. Formal techniques do an exhaustive analysis of the design but can check only small designs completely.

In current state-of-the-art verification methodology, the architect prepares an informal specification of the microprocessor in the form of an English document. The logic designer implements the modules and verifies them using combination of simulation and formal techniques. Many existing approaches ([2], [3], [6]) employ a bottom-up approach to validation, where the functionality of an existing processor is, in essence, reverse-engineered from its RTL implementation. Hauke et al. [2] compare an extracted Instruction-Set Architecture (ISA) description with the given ISA specification. Ho et al. [1] extract controlled token nets from a logic design to perform efficient model checking. Our verification technique is complementary to these bottom-up approaches: we leverage the system architects knowledge about the behavior of the architecture through properties, thereby allowing a powerful top-down approach to microprocessor validation. For example, the property to verify the implementation of a n input adder would be $output = \sum_{i=1}^{n} input_i$. This property should be satisfied irrespective of the adder implementation such as ripple carry adder, carry lookahead adder etc.

In this paper, we present a top-down validation approach using symbolic simulation. We applied our methodology to verify several properties on a microprocessor that is compliant with the PowerPC instruction-set architecture to demonstrate the usefulness of our approach.

2 Related Work

Several approaches for formal or semi-formal verification of processors has been developed in the past. Theorem proving techniques, for example, have been successfully adapted to verify processors ([8], [11], [12]). However, these approaches require a great deal of user intervention, especially for verifying control intensive designs. Burch and Dill presented a technique for formally verifying processor control circuitry [4]. Their technique verifies the correctness of the implementation model of a pipelined processor against its ISA model based on quantifier-free logic of equality with uninterpreted functions. The technique has been extended to handle more complex pipelined architectures by several researchers [5, 9]. In [10], Levitt and Olukotun presented a verification technique, called unpipelining, which repeatedly merges the last two pipe stages into a single stage, resulting in a sequential version of the processor. All the above techniques attempt to formally verify the implementation of pipelined processors by comparing the pipelined implementation with its sequential (ISA) specification model, or by deriving the sequential model from the implementation. On the other hand, in our verification approach, we are trying to define a set of properties which have to be satisfied for the correct behavior, and verify the correctness of the processor by verifying if the properties are met.

Symbolic simulation has proved to be an efficient technique, bridging the gap between traditional simulation and full-fledged formal verification. Versys2 [14] serves as the mainstream custom-memory verification tool for checking Register Transfer Language (RTL) designs against schematics at Motorola's Somerset Design Center. Beatty [13] verified a switch-level non-pipelined processor description by using Binary Decision Diagrams (BDDs) and symbolic simulation. Bhagwati and Devadas [7] verified a pipelined implementation of the DLX processor architecture using BDDs and symbolic simulation.

3 Our Approach

Figure 1 depicts our verification approach. Logic designers implement the architecture in Verilog RTL. The verification engineers write several properties using the information available in the architecture specification document to ensure that the implementation satisfies the specification. A Boolean model is extracted from the RTL and a state machine is coded in Verilog for the properties to be checked. The Boolean model and the Verilog state machine are fed to the Versys2 symbolic simulator. Versys2 [14] is used to verify that the RTL design satisfies the properties. A counter-example is generated if the RTL design violates the property.

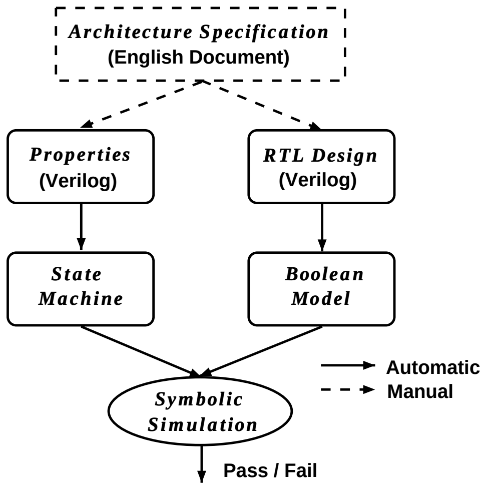

Figure 1. Top-down validation flow

4 Experiments

We applied our methodology to a microprocessor that is compliant with the PowerPC instruction-set architecture. In this section we briefly describe how we verified Memory Management Unit (MMU) of the microprocessor using our approach.

The MMU supports demand-paged virtual memory. It consists of blocks such as *Segment Registers*, *Translation Lookaside Buffers (TLBs)*, and *Block Address Translation (BAT) Arrays*. Each of these memory blocks are composed of sub-blocks. For example, the TLB has three sub-blocks viz., *entry* (data information), *LRU* (least recently used information), and *valid* (information regarding validity of the data) as shown in Figure 2. Each of these sub-blocks is implemented as SRAM. The typical operations in SRAM are read and write. So a natural property to verify is to check read and write for each SRAM cell. The following Verilog code segment shows the read and write properties for an SRAM cell. Similar properties are verified for all the memory blocks.

```
always @ (wrClk or wrEn or dIn or wrAddr)
begin
  if (wrClk & wrEn) ram[wrAddr] <= dIn;
end

assign out = (rdClk & rdEn) ? ram[rdAddr] : 32'b0;
```

The following Verilog code segment verifies the TLB miss detection property using the information available in the architecture specification document. In a similar manner we can write the property for the BAT array miss detection.

```
assign inp =({1'b1,vsid[0:23],ea[4:9],ea[10:13]});
assign out0=({vld0,e0[0:23],e0[24:29],e0[54:57]});
assign out1=({vld1,e1[0:23],e1[24:29],e1[54:57]});
assign hit0=(inp == out0);
assign hit1=(inp == out1);
assign miss=~(hit0 | hit1);
```

The TLB is 2-way set-associative. It would be a simple extension for associativity *n*. Here *vsid* (virtual segment id) and *ea* (effective address) are inputs and *pa* (physical address) is output of the TLB block. The *e* and *vld* variables are output from *entry* and *valid* blocks respectively as shown in Figure 2.

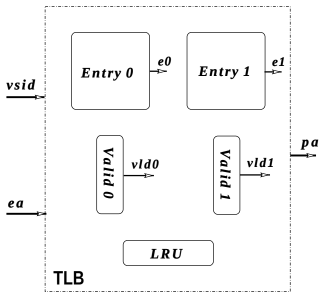

Figure 2. TLB block diagram

There were several issues in verifying these properties. The architecture specification document does not provide the value for the else condition (default value of a signal for example) most of the time. As a result the description of the property does not have the default value for a signal whereas the signal has a definite value in its implementation under all possible conditions. Symbolic simulation produces mismatches in those cases. It is possible to impose certain constraints in Versys2 [14] to avoid the detection of such false negatives. For example, an architecture specification document does not have the default value for the signal *out*, whereas its implementation has a default value (as shown below). To avoid detection of false negatives we can set the condition (*rdClk* & *rdEn*) as true in the Versys2 configuration file.

```
assign out = (rdClk & rdEn) ? ram[rdAddr] : 32'b0;
```

We encountered certain mismatches (not real bugs) due to simulation specific implementation style of certain designs. For example, if a signal is delayed using temporary latches it will generate mismatch during property verification.

5 Summary

Verification is one of the most complex and expensive tasks in the current microprocessor design process. Any effort that improves verification effectiveness and design quality is crucial to meeting customer deadlines and requirements. We presented here a top-down validation approach using symbolic simulation. We wrote several properties using the information available in the architecture specification document and applied them to RTL using symbolic simulation.

We applied our methodology to verify the memory management unit of a microprocessor that is compliant with the PowerPC instruction-set architecture. In the future, we plan to extend this methodology for the verification of the complete microprocessor.

References

[1] P. Ho et al. Formal verification of pipeline control using controlled token nets and abstract interpretation. In *ICCAD*, 1998.

[2] J. Hauke and J. Hayes. Microprocessor design verification using reverse engineering. In *HLDVT*, 1999.

[3] S. Ur and Y. Yadin. Micro architecture coverage directed generation of test programs. In *DAC*, pages 175–180, 1999.

[4] J. Burch and D. Dill. Automatic verification of pipelined microprocessor control. In *CAV*, 1994.

[5] J. Skakkebaek, R. Jones, and D. Dill. Formal verification of out-of-order execution using incremental flushing. In *CAV*, 1998.

[6] R. Ho et al. Architecture validation for processors. In *ISCA*, 1995.

[7] V. Bhagwati and S. Devadas. Automatic verification of pipelined microprocessors. *DAC*, 1994.

[8] M. Srivas and M. Bickford. Formal verification of a pipelined of a pipelined microprocessor. *IEEE Software*, 7(5), Sept. 1990.

[9] M. Velev and R. Bryant. Formal verification of superscalar microprocessors with multicycle functional units, exceptions, and branch prediction. *DAC*, 2000

[10] J. Levitt and K. Olukotun. Verifying correct pipeline implementation for microprocessors. *ICCAD*, 1997.

[11] D. Cyrluk. Microprocessor verification in PVS: A methodology and simple example. TR SRI_CSL9312, 1993.

[12] J. Sawada and J. W.A. Hunt. Trace table based approach for pipelined microprocessor verification. In *CAV*, 1997.

[13] D. L. Beatty. A methodology for formal hardware verification with application to microprocessors. PhD Thesis, School of Computer Science, Carnegie Mellon University, 1993.

[14] N. Krishnamurthy et al. Design and Development Paradigm for Industrial Formal Verification Tools. *IEEE Design & Test of Computers*, July-August 2001.

[15] C-J.H. Seger et al. Formal Verification by Symbolic Evaluation of Partially Ordered Trajectories. *J. Formal Methods in System Design*, vol6, Mar. 1995, pp. 147-189.