EVC: A Validity Checker for the Logic of Equality with Uninterpreted Functions and Memories, Exploiting Positive Equality, and Conservative Transformations*

Miroslav N. Velev[1] and Randal E. Bryant[1,2]

[1] Department of Electrical and Computer Engineering
[2] School of Computer Science
Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
http://www.ece.cmu.edu/~mvelev
http://www.cs.cmu.edu/~bryant
mvelev@ece.cmu.edu, randy.bryant@cs.cmu.edu

Abstract. The property of Positive Equality [2] dramatically speeds up validity checking of formulas in the logic of Equality with Uninterpreted Functions and Memories (EUFM) [4]. The logic expresses correctness of high-level microprocessors. We present EVC (Equality Validity Checker)—a tool that exploits Positive Equality and other optimizations when translating a formula in EUFM to a propositional formula, which can then be evaluated by any Boolean satisfiability (SAT) procedure. EVC has been used for the automatic formal verification of pipelined, superscalar, and VLIW microprocessors.

1 Introduction

Formal verification of microprocessors has historically required extensive manual intervention. Burch and Dill [4] raised the degree of automation by using flushing—feeding the implementation processor with bubbles in order to complete partially executed instructions—to compute a mapping from implementation to specification states. The correctness criterion is that one step of the implementation should be equivalent to 0, or 1, or up to k (for an implementation that can fetch up to k instructions per cycle) steps of a specification single-cycle processor when starting from equivalent states, where equivalency is determined via flushing. However, the verification efficiency has still depended on manually provided case-splitting expressions [4][5] when using the specialized decision procedure SVC [16]. In order to apply the method to complex superscalar processors, Hosabettu [9] and Sawada [15] required months of manual work, using the theorem provers PVS [13] and ACL2 [10], respectively. We present EVC, a validity checker for the logic of EUFM, as an alternative highly efficient tool.

2 Hardware Description Language

In order to be verified with EVC, a high-level implementation processor and its specification must be defined in our Hardware Description Language (HDL). That

* This research was supported by the SRC under contract 00-DC-684.

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 235 - 240, 2001.

HDL is similar to a subset of Verilog [17], except that word-level values do not have dimensions but are represented with a single term-level expression, according to the syntax of EUFM [4]. Hence, nets are required to be declared of type `term` or type `bit`. Additionally, a net can be declared as `input`, e.g., the phase clocks that determine the updating of state or the signals that control the flushing. The HDL has constructs for the definition of memories and latches (see Fig. 2 for the description of two stages of the processor in Fig. 1). Memories and latches can have multiple input and/or output ports—of type `inport` and `outport`, respectively. Latch ports have an enable signal and a list of data signals. Memory ports additionally have an address signal after the enable. Logic gates—`and`, `or`, `not`, = (term-level equality comparator), and `mux` (multiplexor, i.e., ITE operator)—are used for the description of the control path of a processor. Uninterpreted functions and uninterpreted predicates—such as `ALU` in Fig. 2—are used to abstract blocks of combinational logic—the ALU in Fig. 1—as black boxes. Uninterpreted functions and uninterpreted predicates with no arguments are considered as term variables and Boolean variables, respectively, and can be used to abstract constant values that have special semantic meaning, e.g., the data value 0.

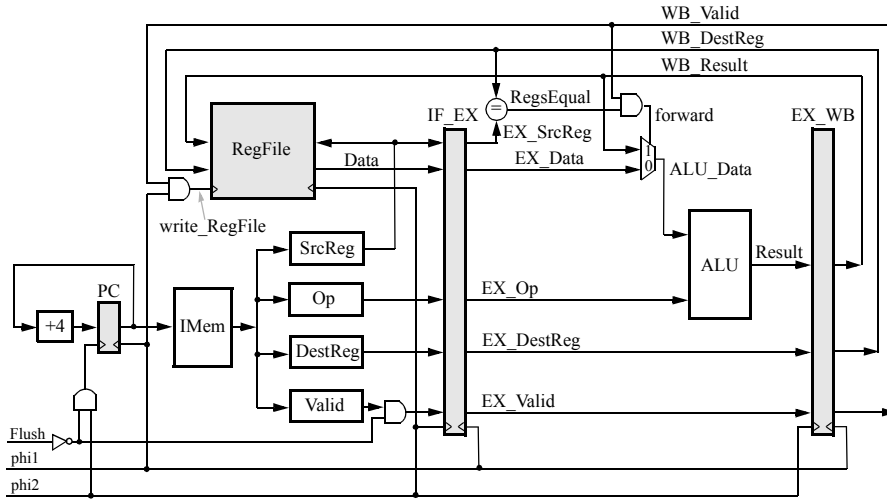

Fig. 1. Block Diagram of a 3-Stage Pipelined Processor.

In order to fully exploit the efficiency of Positive Equality, the designer of high-level microprocessors must follow some simple restrictions. Data operands must not be compared by equality comparators, e.g., in order to determine a branch-on-equal condition. Instead, the equality comparison must be abstracted with the same uninterpreted predicate in both the implementation and the specification processor. Also, a flush signal must be included in the implementation processor, as shown in Fig. 1, in order to turn newly fetched instructions into bubbles during flushing. That extra input will be optimized away by setting it to 0 (the value during normal operation) when translating the high-level processor description to a gate-level synthesizable HDL.

```
Flush_bar = (not Flush)
IF_Valid  = (and Valid Flush_bar)
(latch IF_EX
 (inport  phi2 (SrcReg Data Op DestReg IF_Valid))
 (outport phi1 (EX_SrcReg EX_Data EX_Op EX_DestReg EX_Valid)))
RegsEqual = (=   EX_SrcReg WB_DestReg)
forward   = (and RegsEqual WB_Valid)
ALU_Data  = (mux forward WB_Result EX_Data)
Result    = (ALU EX_Op ALU_Data)
(latch EX_WB
  (inport  phi2 (Result EX_DestReg EX_Valid))
  (outport phi1 (WB_Result WB_DestReg WB_Valid)))
write_RegFile = (and phi1 WB_Valid)
(memory RegFile
  (inport  write_RegFile WB_DestReg (WB_Result))
  (outport phi2 SrcReg (Data)))
```

Fig. 2. Using our HDL to Describe the Execution and Write-Back Stages.

3 Tool Flow

Our term-level symbolic simulator, `TLSim`, takes as input an implementation and a specification processor described in our HDL, as well as a command file that defines simulation sequences by asserting the input signals—phase clocks and flush controls—to binary values. Symbolic initial state for latches and memories is introduced automatically and event-driven symbolic simulation is performed according to the command file. `TLSim` allows for multiple simulation sequences to start from the same initial state, as well as to use the final state reached after symbolically simulating one processor as the initial state for another. States of the same memory or latch, reached after different simulation sequences, can be compared for equality. The resulting formulas can be connected with similar formulas for other memories and latches via Boolean connectives in order to form the EUFM correctness formula. The symbolic simulation and generation of the correctness formula take less than a second even for complex designs. The formula is output in the `SVC` command language [16].

Our second tool, `EVC` (Equality Validity Checker), automatically translates the EUFM correctness formula to an equivalent propositional formula by exploiting Positive Equality [2] and a number of other optimizations [3][18][20][21]. The implementation processor is correct if the propositional formula is a tautology. Otherwise, a falsifying assignment is a counterexample. The propositional formula can be output in a variety of formats, including CNF and ISCAS, allowing the use of many SAT procedures for evaluating it. BDD [6] and BED [23] packages are integrated in `EVC`.

4 Summary of Results

A single-issue 5-stage pipelined DLX processor [8] can be formally verified with EVC in 0.2 seconds on a 336 MHz Sun4. In contrast, SVC [16]—a tool that does not exploit Positive Equality—does not complete the evaluation of the same formula in 24 hours. Furthermore, the theorem proving approach of completion functions [9] could be applied to a similar design after 1 month of manual work by an expert user. Finally, the symbolic simulation tool of Ritter, *et al.* [14] required over 1 hour of CPU time for verification of that processor. A dual-issue superscalar DLX with one complete and one arithmetic pipeline can be formally verified with EVC in 0.8 seconds [21]. A comparable design was verified by Burch [5], who needed 30 minutes of CPU time only after manually identifying 28 case-splitting expressions, and manually decomposing the commutative diagram for the correctness criterion into three diagrams. Moreover, that decomposition was sufficiently subtle to warrant publication of its correctness proof as a separate paper [24]. The theorem proving approach of completion functions [9] required again 1 month of manual work for a comparable dual-issue DLX.

EVC has been used to formally verify processors with exceptions, multicycle functional units, and branch prediction [19]. It can automatically abstract the forwarding logic of memories that interact with stalling logic in a conservative way that results in an order of magnitude speedup with BDDs [21]. A comparative study [22] of 28 SAT-checkers, 2 decision diagrams—BDDs [1][6] and BEDs [23]—and 2 ATPG tools identified the SAT-checker Chaff [11] as the most efficient means for evaluating the Boolean formulas generated by EVC, outperforming the other SAT procedures by orders of magnitude. We also compared the e_{ij} [7] and the small domains [12] encodings for replacing equality comparisons that are both negated and not negated in the correctness EUFM formula. We found the e_{ij} encoding to result in 4 times faster SAT checking when verifying complex correct designs and to consistently perform better for buggy versions. Now a 9-wide VLIW processor that imitates the Intel Itanium in many speculative features such as predicated execution, register remapping, branch prediction, and advanced loads can be formally verified in 12 minutes of CPU time by using Chaff. That design was previously verified in 31.5 hours with BDDs [20]. It can have up to 42 instructions in flight and is far more complex than any other processor formally verified in an automatic way previously. We also found Positive Equality to be the most important factor for our success—without this property the verification times increase exponentially for very simple processors [22], even when using Chaff.

A preliminary version of the tools has been released to the Motorola M•Core Microprocessor Design Center for evaluation.

5 Conclusions and Future Work

EVC is an extremely powerful validity checker for the logic of Equality with Uninterpreted Functions and Memories (EUFM) [4]. Its efficiency is due to exploiting the property of Positive Equality [2] in order to translate a formula in EUFM to a propo-

sitional formula that can be evaluated with SAT procedures, allowing for gains from their improvements. In the future, we will automate the translation of formally verified high-level microprocessors, defined in our HDL and verified with EVC, to synthesizable gate-level Verilog [17]. TLSim and EVC, as well as the benchmarks used for experiments, are available by ftp (http://www.ece.cmu.edu/~mvelev).

References

[1] R.E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," ACM Computing Surveys, Vol. 24, No. 3 (September 1992), pp. 293-318.

[2] R.E. Bryant, S. German, and M.N. Velev, "Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic," ACM Transactions on Computational Logic (TOCL), Vol. 2, No. 1 (January 2001). Available from: http://www.ece.cmu.edu/~mvelev.

[3] R.E. Bryant, and M.N. Velev, "Boolean Satisfiability with Transitivity Constraints, *Computer-Aided Verification (CAV'00)*, E.A. Emerson and A.P. Sistla, *eds.*, LNCS 1855, Springer-Verlag, July 2000, pp. 86-98. Available from: http://www.ece.cmu.edu/~mvelev.

[4] J.R. Burch, and D.L. Dill, "Automated Verification of Pipelined Microprocessor Control," *Computer-Aided Verification* (*CAV'94)*, D.L. Dill, *ed.*, LNCS 818, Springer-Verlag, June 1994, pp. 68-80. http://sprout.stanford.edu/papers.html.

[5] J.R. Burch, "Techniques for Verifying Superscalar Microprocessors," *33rd Design Automation Conference (DAC '96)*, June 1996, pp. 552-557.

[6] CUDD-2.3.0, http://vlsi.colorado.edu/~fabio.

[7] A. Goel, K. Sajid, H. Zhou, A. Aziz, and V. Singhal, "BDD Based Procedures for a Theory of Equality with Uninterpreted Functions," *Computer-Aided Verification (CAV '98)*, A.J. Hu and M.Y. Vardi, *eds.*, LNCS 1427, Springer-Verlag, June 1998, pp. 244-255.

[8] J.L. Hennessy, and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd edition, Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[9] R. Hosabettu, "Systematic Verification of Pipelined Microprocessors," Ph.D. thesis, Department of Computer Science, University of Utah, August 2000.

[10] M. Kaufmann, P. Manolios, J.S. Moore, Computer-Aided Reasoning: ACL2 Case Studies, Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[11] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *38th Design Automation Conference (DAC'01)*, June 2001.

[12] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel, "Deciding Equality Formulas by Small-Domain Instantiations," *Computer-Aided Verification (CAV'99)*, N. Halbwachs and D. Peled, *eds.*, LNCS 1633, Springer-Verlag, June 1999, pp. 455-469.

[13] PVS Specification and Verification System (PVS), http://pvs.csl.sri.com.

[14] G. Ritter, H. Eveking, and H. Hinrichsen, "Formal Verification of Designs with Complex Control by Symbolic Simulation," *Correct Hardware Design and Verification Methods (CHARME'99)*, L. Pierre and T. Kropf, *eds.*, LNCS 1703, Springer-Verlag, September 1999, pp. 234-249.

[15] J. Sawada, "Formal Verification of an Advanced Pipelined Machine," Ph.D. thesis, Department of Computer Science, University of Texas at Austin, December 1999.

[16] Stanford Validity Checker (SVC), http://sprout.stanford.edu.

[17] D.E. Thomas, and P.R. Moorby, The Verilog® Hardware Description Language, 4th edition, Kluwer Academic Publishers, Boston/Dordrecht/London, 1998.

[18] M.N. Velev, and R.E. Bryant, "Superscalar Processor Verification Using Efficient Reductions of the Logic of Equality with Uninterpreted Functions to Propositional

Logic," *Correct Hardware Design and Verification Methods (CHARME'99)*, L. Pierre and T. Kropf, *eds.*, LNCS 1703, Springer-Verlag, September 1999, pp. 37-53. Available from: `http://www.ece.cmu.edu/~mvelev`.

[19] M.N. Velev, and R.E. Bryant, "Formal Verification of Superscalar Microprocessors with Multicycle Functional Units, Exceptions, and Branch Prediction," *37th Design Automation Conference (DAC'00)*, June 2000, pp. 112-117. Available from: `http://www.ece.cmu.edu/~mvelev`.

[20] M.N. Velev, "Formal Verification of VLIW Microprocessors with Speculative Execution," *Computer-Aided Verification (CAV'00)*, E.A. Emerson and A.P. Sistla, *eds.*, LNCS 1855, Springer-Verlag, July 2000, pp. 296-311. Available from: `http://www.ece.cmu.edu/~mvelev`.

[21] M.N. Velev, "Automatic Abstraction of Memories in the Formal Verification of Superscalar Microprocessors," *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, T. Margaria and W. Yi, *eds.*, LNCS, Springer-Verlag, April 2001, pp. 252-267. Available from: `http://www.ece.cmu.edu/~mvelev`.

[22] M.N. Velev, and R.E. Bryant, "Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors," *38th Design Automation Conference (DAC'01)*, June 2001. Available from: http://www.ece.cmu.edu/~mvelev.

[23] P.F. Williams, "Formal Verification Based on Boolean Expression Diagrams," Ph.D. thesis, Department of Information Technology, Technical University of Denmark, Lyngby, Denmark, August 2000.

[24] P.J. Windley, and J.R. Burch, "Mechanically Checking a Lemma Used in an Automatic Verification Tool," *Formal Methods in Computer-Aided Design (FMCAD'96)*, M. Srivas and A. Camilleri, *eds.*, LNCS 1166, Springer-Verlag, November 1996, pp. 362-376.