Equivalence checking hardware multiplier designs

Matti Järvisalo*

Abstract

We generate SAT benchmarks encoding the problem of equivalence checking two different industrial hardware designs for integer multiplication.

1 Problem

Our goal is to generate interesting SAT benchmarks based on real-life hardware designs. We consider the problem of checking whether two different hardware designs for integer multiplication are equivalent in the sense that both produce the same output on all inputs.

The designs we use are the *adder tree* and *Braun* multipliers [1]. For a fixed n, both multipliers take as input two integers $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$ in binary, and output the product $\mathbf{o} = (o_1, \ldots, o_{2n})$. Both designs consist of $\mathcal{O}(n^2)$ gates, using nots and binary ands, ors, and xors. The propagation delays (max max height from inputs to outputs) are $\mathcal{O}(n)$ for Braun, and $\mathcal{O}(\log(n \log n))$ for adder tree. While Braun consists of a grid of full-adders, adder tree applies adders in a tree-like fashion, summing up partial products.

We will construct a Boolean circuit describing an instance of the equivalence checking problem for given n-bit adder tree (output bits $\mathbf{o}^{\mathrm{a}} = (o_1^{\mathrm{a}}, \ldots, o_{2n}^{\mathrm{a}})$) and Braun multipliers (output bits $\mathbf{o}^{\mathrm{b}} = (o_1^{\mathrm{b}}, \ldots, o_{2n}^{\mathrm{b}})$) as follows:

- The inputs of the multipliers are made equivalent by sharing the input gates $a_1, \ldots, a_n, b_1, \ldots, b_n$.
- No other gates are shared between the multiplier circuits.
- Bit-wise equivalence of the outputs $\mathbf{o}^{\mathrm{a}}$ and $\mathbf{o}^{\mathrm{b}}$ is enforced by introducing gates

$$o_i^{\mathrm{eq}} = \mathsf{equiv}(o_i^{\mathrm{a}}, o_i^{\mathrm{b}})$$

  for $i = 1 \ldots 2n$.
- As a single output gate introduce

$$\mathbf{out} = \mathsf{and}(o_1^{\mathrm{eq}}, \ldots, o_{2n}^{\mathrm{eq}}).$$

- Constrain **out** to 0 (false).

Since the multiplier designs produce equivalent results for any two multiplicants, we arrive at an **unsatisfiable** equivalence checking instance.

The Boolean circuit descriptions of the multiplier designs are produced by the genfacbm generator [7] for SAT benchmarks based on integer factoring in the BCSat Boolean circuit format [3], see [6] for details.

2 CNF Encoding

For the CNF encoding, we apply the bc2cnf Boolean circuit simplifier/clausifier [4]. The bc2cnf tool applies structure sharing and simplification (Boolean propagation, cone-of-influence reduction, monotone input gate rule) techniques to the circuit, and translates the simplified circuit into CNF in a standard way *a la* Tseitin [8] (however, nots are interpreted as negated literals). This results in a linear translation with respect to the simplified circuit.

3 Instances

The following set of benchmarks are available at

`http://www.tcs.hut.fi/~mjj/benchmarks/`.

- `eq.atree.braun.[n].unsat.cnf` unsatisfiable instances for multiplicant bit-widths $\mathtt{n} = 7 \ldots 13$

*Address: Helsinki University of Technology, Laboratory for Theoretical Computer Science, P.O. Box 5400, FI-02015 TKK, Finland. Email: matti.jarvisalo@tkk.fi.

4 Some Experiments

The following experiments were run using a Debian Linux based PC with a 2-GHz AMD processor and 2 GB of memory. The running time and number of decisions/branches for the CNF solvers Satz [5] (version 2.15) and Minisat [2] (version 2.0 with preprocessing) are shown in Figures 1 and 2, respectively. Notice that the input bit-widths are rather small; e.g. for $n = 11$ the number of CNF variables and clauses are 1400 and 4732, respectively.

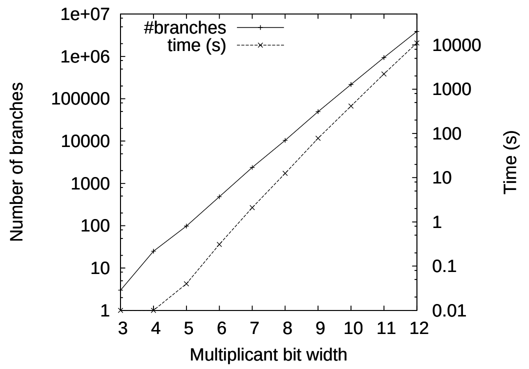

Figure 1: Results for Satz 2.15

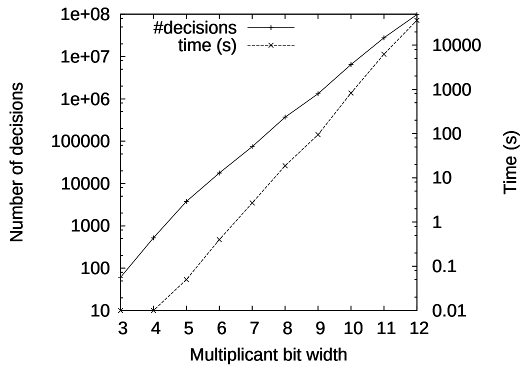

Figure 2: Results for Minisat 2.0 with preprocessing

References

[1] S. Brown and Z. Vranesic. *Fundamentals of digital logic with VHDL design*. McGraw-Hill, 2000.

[2] N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[3] T. Junttila. The BCSat file format, 2001. Available at `http://www.tcs.hut.fi/~tjunttil/bcsat/`.

[4] T. Junttila. The bc2cnf Boolean circuit simplifier/clausifier, 2006. Software, available at `http://www.tcs.hut.fi/~tjunttil/bcsat/` as part of the BC-Tools package.

[5] C.M. Li. A constraint-based approach to narrow search trees for satisfiability. *Information Processing Letters*, 71(2):75–80, 1999.

[6] T. Pyhälä. Factoring benchmarks for SAT-solvers, 2004. Available at `http://www.tcs.hut.fi/Software/genfacbm/`.

[7] T. Pyhälä. genfacbm — a benchmark generator based on factoring for SAT and ASP solvers, 2004. Software, available at `http://www.tcs.hut.fi/Software/genfacbm/`.

[8] G.S. Tseitin. On the complexity of derivation in propositional calculus. In A.O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, volume 8 of *Seminars in Mathematics, V.A. Steklov Mathematical Institute, Leningrad*, pages 115–125. Consultants Bureau, 1969. [Translated from Russian. Reprinted in J. Siekmann and G. Wrightson, e ditors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer, 1983.].