Evaluation of SAT like Proof Techniques for Formal Verification of Word Level Circuits

André Sülflow Ulrich Kühne Robert Wille

Daniel Große Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
Email: {suelflow,ulrichk,rwille,grosse,drechsle}@informatik.uni-bremen.de

Abstract

Word level information on the Register Transfer Level (RTL) offers information for efficient guidance of the proof process in formal verification. Therefore several proof techniques with integrated word level support from other research fields can be applied for formal verification of circuit designs as well.

The focus of this work is to evaluate the proof techniques Boolean Satisfiability (SAT), SAT Modulo Theories (SMT), SWORD and Constraint Satisfaction Problem (CSP) in the context of formal hardware verification. An estimation of the effort to encode standard circuit elements is given and the advantages and disadvantages of the different encodings is studied. In our experiments we consider equivalence checking problems for circuit designs given on bit and word level.

1. Introduction

Meanwhile formal verification is widely applied in industry. However, due to exponential growth of the design sizes the effort for verification has become the major economical issue. Usually circuits are modeled at the *Register Transfer Level* (RTL), i.e. the specification includes word level information. More precisely a design description contains bit level parts (e.g. a flag in a CPU pipeline) as well as word level operators applied to bit-vectors (e.g. addition, multiplication or comparison) [7].

For verification an appropriate model is compiled from two designs in case of equivalence checking or from the design and a property in case of property checking. The granularity of the resulting model depends on the chosen proof technique. In the development of proof techniques for formal verification an enormous progress has been made during the last two decades. Efficient symbolic representations have become possible using *Binary Decision Diagrams* (BDDs) [3]. The next step was reached by dramatic improvements in *Boolean Satisfiability* (SAT). As reported, verification methods based on new SAT solvers (e.g. [12],[10]) handle significantly larger circuits. In parallel to the progress made in SAT the techniques for *Constraint Satisfaction* (CSP) [6] were improved. Satisfiability problems in combination with decidable theory solvers led to the development of *Satisfiability Modulo Theories* (SMT) [18].

In this paper we consider the equivalence checking of two circuit designs given in terms of bit level and word level operations. First, for all proof techniques mentioned above we analyze the effort to encode the verification task into the dedicated problem instance. Thereby, we discuss the advantages and disadvantages of each proof technique with respect to the considered operation. In first experimental results circuit descriptions from different domains are studied. The experiments show that word level provers benefit during the solve process if the fraction of bit level operations is low.

The paper is structured as follows. In Section 2 we briefly review the different proof techniques. In Section 3 it is shown how verification problems on word level circuits can be encoded for the respective techniques. Experimental results are given in Section 4 before the paper is concluded in Section 5.

2. Proof techniques

In this work several techniques are used for the formal verification of circuits. To keep the paper self-contained, the basic concepts of the proof techniques are briefly reviewed in this section. For a more detailed description we refer the reader to the respective publications. The encoding of several circuit elements for the different proof engines is discussed in more detail in the next section.

2.1. Boolean Satisfiability (SAT)

The *Boolean Satisfiability problem* (SAT problem) is defined as the problem to determine whether there exists an assignment to the variables of a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ such that f evaluates to 1 or to prove that no such assignment exists. Thereby, f is given in *Conjunctive Normal Form* (CNF), i.e. a product-of-sum representation. Each CNF is a set of clauses where each clause is a set of literals and each literal is a propositional variable or its negation.

In the past several (backtracking) algorithms (*SAT solvers*) were proposed [5, 13, 15, 10]. Most of them are based on three essential procedures: (1) The decision heuristic assigns values to free variables, (2) the propagation procedure determines implications due to the last assignment(s) and (3) the conflict analysis tries to resolve conflicts that occur during the search by backtracking. Advanced techniques like e.g. *efficient Boolean constraint propagation* [15] or *conflict analysis* [13] are common in state-of-the-art SAT solvers today.

Since a circuit is a representation of a Boolean formula, it can be transformed into CNF. If the circuit consists of logic gates each signal is mapped to a Boolean variable and each gate to a set of clauses. If there are also word level structures within the circuit, additional clauses and auxiliary variables have to be introduced that describe the characteristic function of the word level operation. This mapping can be done in time and space linear in the size of the circuit [19].

2.2. SAT Modulo Theories (SMT)

Due to the tremendous improvements in the performance of provers for Boolean SAT, several researchers investigated the combination of SAT solvers with decision procedures for decidable theories resulting in *SAT Modulo Theories* (SMT) [2, 8]. An SMT solver integrates a Boolean SAT solver with other solvers for specialized theories (e.g. linear arithmetic or bit-vector logic). Here the SAT solver works on an abstract representation (also in CNF) of the problem and guides the overall search process, while each (partial) assignment of this representation has to be validated by the theory solver for the theory constraints. Thus, advanced SAT techniques together with specialized theory solvers can be utilized.

In this work we use the bit-vector logic to represent the circuit as an SMT instance. Similar to the SAT encoding each signal and operation is represented by (bit-vector) variables and constraints, respectively. As constraints *bit-vector operations* (e.g. bit-vector AND) or *arithmetic operations* over bit-vectors (e.g. multiplication) can be used.

Because of this higher level of abstraction more information of the problem instance is available. This can be exploited by e.g. *canonizing* [1] or *bit-blasting* [4]. The first technique tries to represent as many constraints as possible in a canonical form, which are then solved by a specialized solver. During bit-blasting the constraints are normalized and rewritten before transforming them into CNF and passing them to a common SAT solver. This preprocessing step can reduce the size of the problem and may lead to an instance that is easier to solve.

2.3. SWORD

Another proof engine is SWORD (*S*AT-like prover using *word* level information) which was first introduced in [21]. In contrast to the bit-vector logic of SMT, SWORD still uses Boolean variables, but takes care of their connection as a bit-vector. This allows the usage of (pure Boolean) SAT-techniques like e.g. conflict analysis which cannot be adapted to bit-vectors in an easy way.

Furthermore, word level information is given in terms of so called *modules*. Each module defines an operation over bit-vectors and provides a specialized decision and implication strategy for the respective operation. While a main algorithm controls the overall solve process (including backtracking and conflict analysis) these specialized strategies can be used to guide the search.

Thus, in contrast to SAT the heuristics and implication engines of SWORD are more powerful because of the available word level information. Moreover, SWORD uses word level information *within* the search process, not as a preprocessing step like for bit-blasting in the corresponding SMT-logic.

2.4. Constraint Satisfaction Problem (CSP)

Orthogonally to SAT, researchers also developed techniques for the *Constraint Satisfaction Problem* (CSP) [6]. In CSP variables can be assigned with different values according to the domain the variable is associated with. As constraints arithmetic and logic operations over integers as well as relations (i.e. a listing of supporting or conflicting sets of variable assignments) can be used.[1]

This abstract model is utilized in the decision heuristic and the propagation engine. E.g. variables which are supposed to have a high impact on the problem will be preferred by the decision heuristic or implication will be done

[1] In this work only the constraints and relation definitions of [17], which are used for the CSP competitions, are considered.

Table 1. Encodings for the different proof techniques

CIRC. ELEM.	SAT	SWORD	SMT	CSP
Signal (n bit)	n Boolean variables	n Boolean variables constitute a bit-vector	1 bit-vector *vec* of size n	1 integer *int* range $[0:2^n-1]$
extraction ith bit	direct access	direct access	bit = extract[i:$i+1$] vec	eq(bit,mod(div(int,2^{n-i}),2))
vars. in total	–	–	1 additional	1 additional
constrs. in total	–	–	2	3
n-bit and	clauses for n and gates	and module	o = (bvand a b)	for each bit i of a (and b, o resp.) eq(aux_{ai},mod(div(a,2^{n-i}),2)) for each $aux_{ai}, aux_{bi}, aux_{oi}$ eq(aux_{oi},(aux_{ai} and aux_{bi}))
vars. in total	$2\cdot n+n$	$2\cdot n+n$	3	$3+3\cdot n$
constrs. in total	$3\cdot n$	1	2	$3\cdot n+2\cdot n$
n-bit multiplexer	clauses for n 1-bit multiplexer	multiplexer module	o = (ite s d_0 d_1)	and(or(eq(s,1) , eq(o,d_0)) , or(eq(s,0) , eq(o,d_1)))
vars. in total	$2\cdot n+1+n$	$2\cdot n+1+n$	4	4
constrs. in total	$4\cdot n$	1	2	7
n-bit adder	gate representation of 1 HA and $(n-1)$ FA aux. variables needed	adder module	o = (bvadd a b)	eq(o, add(a,b))
vars. in total	$2\cdot n+(n+1)+aux\_vars$	$2\cdot n+(n+1)$	3	3
constrs. in total	$7+(n-1)\cdot 17$	1	2	2
n-bit multiplier	gate representation of $(n-1)$ n-bit adder and n $2n$-bit multiplexer aux. variables needed	multiplier module	o = (bvmul a b)	eq(o,mul(a,b))
vars. in total	$2\cdot n+2\cdot n+aux\_vars$	$2\cdot n+2\cdot n$	3	3
constrs. in total	$(n-1)\cdot(7+(n-1)\cdot 17)+n\cdot(4\cdot n)$	1	2	2

with respect to the particular operation. In contrast to other proof techniques, the description (and thus the solvers as well) do not support Boolean operations until now. However, the high level problem description motivate a more detailed consideration of CSP for word level circuit verification.

3. Application to Equivalence Checking

Formal equivalence checking of high level systems means to compare implementations either on word level or on bit level or a mix of both. Therefore, different requirements for the formal proof technique and the underlying input language are given. The following word level circuit elements should be supported:

- n-bit signals
- Bit access (e.g. bit select, bit slicing)
- n-bit logic operations (e.g. and, or, xor)
- n-bit arithmetic operations (e.g. addition, multiplication)

The encoding of the required circuit elements above have different complexities for the studied proof techniques. In the following we describe the encoding effort followed by a discussion on the capabilities and strengths of each proof technique.

3.1. Encoding Effort

In Table 1 an overview of the estimated effort for the encoding of standard circuit elements is presented, including an n-bit variable declaration, n-bit logic operation and n-bit arithmetic operation. The focus is on the number of variables and constraints that are necessary to declare a circuit element. The number of variables includes inputs, outputs and auxiliary variables.

Of course this estimation method provides only an indication for the effort of the underlying solver. Furthermore, the implementations given in the table are the most common implementations only. The intension was not to find the optimal implementation (see e.g. an *add* operation), but to describe the effort for declaration and definition of the functionality in general.

3.1.1 Declaration of n-bit Signals

The declaration of an n-bit signal is equivalent in SAT and SWORD. For each bit of the signal a separate Boolean vari-

able has to be introduced resulting in n variables. Thus, the connectivity of an n-bit signal, like a 32-bit integer, is lost during declaration. This is different to a signal in SMT and in CSP. SMT and CSP provide bit-vector variables and CSP variables over a domain, respectively. Both types encapsulate the connectivity.

3.1.2 Bit Extraction from a n-bit Signal

Accessing single bits in SAT and SWORD is straight forward because the bits are already given. So no extra overhead is necessary. SMT provides the bit extracting function *extract* for bit vector variables. The definition requires one additional variable (the output bit) and two operations (assignment ($=$) and *extract*). Because CSP does not provide direct bit access, the (potentially expensive) modulo and division have to be used to extract a single bit. Overall it takes three constraints and a declaration of the additional output bit variable.

3.1.3 n-bit Logic Operations

The encoding of typical logic operations like *and*, *or*, *xor* in SAT takes e.g. for an *and* operation, with two n-bit inputs and one n-bit output, $3 \cdot n$ separate clauses. The knowledge of the connectivity of the clauses is lost during encoding. SWORD instead keeps this knowledge by providing an *and* module while using the same number of variables as SAT. This is similar to SMT which provides a *bvand* operation, but the definition takes an additional equality constraint. Because bit access in CSP is complex, the definition of logic operations is complex, too. First, each bit of the inputs has to be assigned to auxiliaries variables using modulo and division. Then these variables have to be connected with an *and* and equality (*eq*) operation to the output bits. So in total $3 \cdot n$ constraints for bit access and $2 \cdot n$ constraints for *eq* and *and* are needed.

3.1.4 n-bit Arithmetic Operations

SAT does not provide arithmetic operations. Thus in order to encode an adder or a multiplier it is necessary to find a suitable representation by clauses. For example, for an n-bit addition it takes 7 clauses for one half adder and $(n-1) \cdot 17$ clauses for the $n-1$ full adders. Thus the encoding effort is very high and the information that a specific clause once belonged to an *add* operation is lost. SWORD, SMT and CSP provide constraints for all word level operations.

3.2. Discussion

The encodings of circuits for the given proof techniques differ in their complexity. Whereas some provide direct support for the needed operations, we have to find a functional equivalent implementation with a given limited set of operators for the other techniques. For example CSP does not provide a multiplexer operation which is a standard operation of SMT.

As SAT is designed for Boolean reasoning, it has advantages for circuits with a high number of logic gates. It provides not only direct bit access, but also an easy encoding for logic gates. Because SAT solvers were significantly improved during the last years, today SAT is the state-of-the-art technique for equivalence checking on pure logic level. The disadvantages of SAT are arithmetic operations which are not directly supported. Instead, a transformation to a number of clauses has to be provided, whereas the knowledge of the original operation is lost. This means that information for potential guidance of the search process are missing while the number of variables and clauses grows. This makes the solve process harder as the circuit becomes more complex. In order to use further information to guide the search process, additinal techniques have to be implemented on top of a SAT solver (see e.g. [20]).

SWORD provides a mix of direct bit access technique in combination with word level operations. Based on the knowledge of the connectivity of n-bit signals to a word level module the SWORD solver can on the one hand efficiently direct the search process. On the other hand the cheap bit access capabilities entails the creation of n Boolean variables for an n-bit signal. Thus, the number of variables are the same as for SAT except for auxiliaries variables, but the number of constraints is far less. This leads to smaller instance sizes and the search process can still be improved. However, the modules for each operation have to be supported by the solver.

The bit-vector theory of SMT allows to use one variable for an n-bit signal. Therefore, SMT can show a benefit for circuits with a high number of arithmetic modules, because the instance size grows far less in comparison to SAT and SWORD. By its bit extracting technique, SMT provides capabilities for circuits with logic gates, too. However the bit access takes always one more auxiliary signal and two extra operator declarations. The information on the connection of operators via word level variables enables SMT to provide an efficient pre-optimization step, before verifying the simplified instance.

CSP as a word level proof technique is well applicable for arithmetic operations. It uses exactly the same number of variables and constraints to declare an adder and a multiplier as in SMT. Because bit access is expensive, it should not be used frequently for verification of circuits with many logic gates.

In the following section we evaluate the different proof techniques for a number of equivalence checking problems.

Table 2. Results for the equivalence check of a CSA

BITS	MINISAT (SAT)	YICES (SMT)	STP (SMT)	SWORD	ABSCON (CSP)
4	0.004 s	0.024 s	0.012 s	$<$ 0.001 s	1.010 s
8	0.004 s	0.136 s	0.024 s	0.040 s	428.440 s
16	0.048 s	1.384 s	0.084 s	0.300 s	T.O.
32	0.248 s	8.157 s	0.264 s	2.110 s	T.O.
64	1.580 s	147.657 s	0.900 s	12.810 s	T.O.
128	10.945 s	T.O.	3.984 s	76.130 s	T.O.

Table 3. Results for the equivalence check of a multiplier

BITS	MINISAT (SAT)	YICES (SMT)	STP (SMT)	SWORD	ABSCON (CSP)
2	$<$ 0.001 s	0.004 s	$<$ 0.001 s	$<$ 0.001 s	0.450 s
4	0.016 s	0.088 s	0.020 s	0.020 s	1.740 s
6	0.496 s	2.264 s	0.688 s	0.300 s	T.O.
8	22.677 s	69.264 s	27.150 s	6.76 s	T.O.
10	491.251 s	T.O.	1159.990 s	151.79 s	T.O.

4. Experimental Evaluation

In this section we report results for the application on equivalence checking. As representatives for the proof techniques that are described in Section 2 the following solvers have been used: MiniSat [10] (version 2) is an open source state-of-the-art SAT solver. As SMT solvers Yices [8, 9] (version 1.0.8) and STP [11] have been selected. These solvers implement different strategies for the handling of bit vector operations. Yices uses bit-blasting for almost all bit-vector operations while STP performs additional optimizations before the problem is bit-blasted and then given to MiniSat. STP and Yices (version 1.0) have been ranked best in the SMT-COMP'06 in the category of bit vector logic. The SWORD solver has been described in [21]. For CSP we have used the solver Abscon [14] (version 1.09) which achieved a good ranking in the CSP Competition 2006 [16].

As benchmarks three different equivalence checking problems have been selected to demonstrate the different behavior of the various proof techniques:

- A gate level implementation of a conditional sum adder (CSA) vs. a word level specification
- A multiplier implemented on the basis of addition and bit shifting vs. a word level specification
- A binomial $(a+b)^2$ vs. the equivalent formulation $(a^2+b^2+2\cdot a\cdot b)$ using only word level operations

All experiments have been carried out on a AMD Athlon 3700+ (2.2 GHz) with 1GB main memory running Linux. The time limit for each instance was set to 1 hour (denoted by T.O.). The results of all experiments can be found in Table 2, Table 3 and Table 4, respectively. Thereby, column BITS denotes the bitwidth of the respective benchmark while the other columns denote the run time of the respective proof technique in CPU seconds.

In the first experiments the CSA is considered. As can be seen from Table 2, one of the SMT solvers (STP) performs best on these benchmarks, followed by the SAT solver MiniSat. For the equivalence check of the multiplier (see Table 3) SWORD performs best, again followed by MiniSat. For the equivalence check of the binomial finally, SWORD outperformed all other solvers (see Table 4). CSP performs worse for all the benchmarks considered.

However, we have found that the straightforward encoding that is inspired by the translation of circuits to CNF is not the best one for the STP solver. The declaration of auxiliary variables in-between the operations seems to prevent the solver from efficient pre-processing. Thus, instances that could otherwise easily be solved become hard for the solver.

In summary we conclude from this first experiments that SMT and SWORD benefit from word level information, but SAT still has comparable run time. Because of the worse run time CSP is not competitive at the moment. But further improvements can be expected, because the development is still in progress.

Table 4. Results for the equivalence check of a binomial

BITS	MINISAT (SAT)	YICES (SMT)	STP (SMT)	SWORD	ABSCON (CSP)
2	0.004 s	0.004 s	0.004 s	< 0.001 s	0.640 s
4	0.056 s	0.088 s	0.052 s	0.020 s	13.700 s
6	1.524 s	2.068 s	1.648 s	0.340 s	T.O.
8	68.404 s	83.281 s	64.812 s	6.640 s	T.O.
10	T.O.	T.O.	T.O.	122.920 s	T.O.

5. Conclusion and Future Work

In this work a first evaluation of different SAT like proof techniques and their application in formal verification of word level circuits was given. We described and discussed the effort to encode circuit elements into dedicated problem instances.

Based on first experiments SAT was seen as the best proof technique for circuits with a high number of logic gates. The usage of SMT and SWORD show advantages for RTL-circuits. Moreover todays state-of-the-art SAT solvers are competitive. But many new word level approaches are under development and further improvements in this area can be expected.

In future work we plan to study the influence of alternative encodings for the proof engines.

References

[1] C. Barrett, D. Dill, and J. Levitt. A decision procedure for bit-vector arithmetic. In *Design Automation Conf.*, pages 522–527, 1998.

[2] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. The MathSAT 3 System. In *Proceedings of the 20th Int. Conference on Automated Deduction (CADE)*, volume 3632 of *LNCS*, pages 315–321. Springer-Verlag, 2005.

[3] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[4] R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. Brady. Deciding bit-vector arithmetic with abstraction. In *Proceedings of TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 358–372. Springer, 2007.

[5] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

[6] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[7] R. Drechsler. *Formal Verification of Circuits*. Kluwer Academic Publishers, 2000.

[8] B. Dutertre and L. Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Computer Aided Verification*, volume 4114 of *LNCS*, pages 81–94, 2006.

[9] B. Dutertre and L. Moura. The YICES SMT Solver. 2006. Available at http://yices.csl.sri.com/.

[10] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[11] V. Ganesh and D. L. Dill. A decision procedure for bit-vectors and arrays. In *Computer Aided Verification (CAV '07)*, Berlin, Germany, July 2007. Springer-Verlag.

[12] J. Marques-Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *Int'l Conf. on CAD*, pages 220–227, 1996.

[13] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[14] S. Merchez, C. Lecoutre, and F. Boussemart. AbsCon: A Prototype to Solve CSPs with Abstraction. In *Proceedings of the 7th Int. Conference on Principles and Practice of Constraint Programming*, volume 2239 of *LNCS*, pages 730–744. Springer-Verlag, 2001.

[15] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[16] O. C. of the Second International Competition of CSP Solvers. Results of the Second International Competition of CSP and Max-CSP Solvers, 2006. http://www.cril.univ-artois.fr/CPAI06/.

[17] O. C. of the Second International Competition of CSP Solvers. XML Representation of Constraint Networks (Version 2.0), 2006. http://www.cril.univ-artois.fr/ lecoutre/research/tools/format2.pdf.

[18] C. Tinelli. A dpll-based calculus for ground satisfiability modulo theories. In *JELIA*, pages 308–319, 2002.

[19] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968. (Reprinted in: J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, 1983, pp. 466-483.).

[20] M. Wedler, D. Stoffel, and W. Kunz. Arithmetic reasoning in dpll-based sat solving. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 1, pages 30–35Vol.1, 16-20 Feb. 2004.

[21] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler. SWORD: A SAT like Prover Using Word Level Information. In *Int'l Conference on Very Large Scale Integration*, 2007.