PROSYD

Property Based System Design
IST-507219

Deliverable 3.2/3

Exploitation of RT information in static property checking algorithms

Version 1.0
16 June 2004

Infineon Technologies AG

Author:

Raik Brinkmann

Abstract:

This document describes an approach for preserving the RTL structure of designs and properties to be exploited in property checking algorithms. Two approaches, for preprocessing such problems are considered, namely symmetry reduction and date path scaling, and the effects are outlined.

1 Introduction and Background

1.1 Bounded Model Checking

Within the BMC-approach [3, 6, 14], the expected behavior of a design is formally described by a bounded-time temporal specification, which is a set of properties capturing interesting behavior over a short period of time. A property is essentially a constraint on the design's signals over a finite time interval. For the property to be valid means to hold for every observation window of the appropriate length, in every run admitted by the design. To verify a property, a so called 'BMC-problem' is generated from the design and a property to be verified. A BMC-problem is represented by a Boolean function f. In order to solve BMC-problems, it is necessary to check whether this Boolean function is constantly 1 (respectively satisfiable), depending on the property. A tool checks each property and produces a counter-example by dissatisfying f (respectively proves that f is not satisfiable). For blocks in the 30 to 100k gates range this check takes from seconds to a few minutes per property. Even though such properties are not as expressive as general temporal logic formula used in symbolic model checking, they are sufficient for many practical applications.

Although all known procedures for checking f for constantness have exponential worst case complexity, they are practical in many cases. They use clever search heuristics, utilizing specifics of hardware designs, which enables the skipping of certain parts of the search space. DLL-based SAT is especially sophisticated in doing so, therefore, even though other approaches are possible, BMC-problems are usually translated into conjunctive normal form, and a SAT solver is used to check validity.

Recent advances in SAT solver technology [21] have pushed the limits of BMC further. This makes the BMC-approach even more attractive for industrial application [23, 5]. Today the complexity limits of BMC allow verification of typical design blocks of control oriented nature. However, some important classes of hardware designs, including large regular designs, as well as arithmetic designs, cannot yet be handled robustly.

By exploiting the underlying structure of design and property this could be changed. This exploitation depends on the representation of BMC-problems as bitvector terms [11, 22, 2] directly on the register transfer level (RTL), thus preserving the syntactic correlation of single bits in bitvectors and bitvector terms naturally inhabiting HDLs. To receive such a representation, the BMC-approach is lifted to the RTL. In the sequel we will consider the problem of checking whether a Boolean function f is constantly 1 to be a BMC-problem. (The other case, checking whether f is constantly 0 is symmetrical.)

1.2 Problem Construction on the RTL

All formal approaches to verification, including property checking, use some kind of formal representation for a design. Structure, i.e. functional units and their operations, can only be exploited, if it is explicitly available in this formal representation. Hardware description languages, in which the functionality of a design is first formulated, operate on the register transfer level (RTL). On the register transfer level structure is explicit. It is prevalent in data types and functional units. Bitvectors (vectors of bits) are the prevailing data type.

We exploit this structure to reduce the computational effort for solving property checking problems.

However, the standard BMC flow destroys a lot of the structure, because the design is flattened into a bit-level representation. Hence the syntactic correlation of the single bits, available on the register transfer level, is lost. Therefore, the regular structure of a design is not explicit in its formal representation. To be able to exploit the structure of a design it must be preserved. If, in contrast to the standard approach, the design is synthesized into a register transfer level representation, a word-level representation of the property checking problem can be constructed, and the structure is preserved.

For evaluating such ideas, as well as an experimental platform for RTL-based property checking, the program *RtProp* has been developed. It takes a register transfer level state machine (RSM) and a a property. The RSM is a constructive representation of a digital circuit as a Mealy machine on the register transfer level. The property is first translated into a basic form removing syntactic sugar, such that it is an expression over input output and state variables of the design annotated with time points. The output and transition functions of the design as well as property are represented by bitvector terms. Bitvector terms are represented internally by fully collapsed term graphs, maximally sharing common subexpressions. Normalization techniques such as constant propagation are built into the term generation process and are implicitly applied when terms are constructed. From design and property a bitvector term with Boolean output is constructed which is constantly 1 iff the property holds in all states of the design. Bitvector terms can then be translated into vectors of Boolean terms as usual, and Boolean terms can then be checked for constantness using standard techniques. This makes the approach comparable to others. However, we want to exploit structural information gathered in the representation of the BMC-problem as bitvector term.

Structural information can be used in different ways, firstly, to reduce the size of property checking problems before solving them by using reduction or abstraction techniques, and secondly, by exploiting the structure during search [7]. Here we focus on reduction techniques, describing one approach in detail, namely symmetry reduction for regular designs, before outlining another, namely automated data path scaling.

2 Symmetry Reduction

A reduction is of most practical use, if this reduction is fully automatic, and is seamlessly integrated into a tool. In this section we consider a novel form of symmetry reduction which operates on the RTL representation of the BMC-problem. It is most effective for regular designs.

Regular designs form a large and important class of hardware designs. They are usually constructed by repetitive use of some basic building blocks. Often many of these building blocks are treated uniformly. Take for example the cells of a memory, participants in a communication network, or masters on a bus waiting for a slave device to become available. If this uniformity is reflected by the internal structure of the design, it is called 'regular'. Regular designs can contain many instances of similar building blocks, such as memories, bus-systems, multiplexers, and arbiters and thus have many state variables.

For example, even a small memory with 8 address lines and 16 data lines has at least 2^{4096} states. This often leads to a very large search space for the search procedure and to a high degree of symmetry. It can make BMC impractical, as it often leads to exponential runtime behavior, and consequently to inefficiency.

Analysis of the structure of property checking problems for regular designs such as memories, arbiters and bus systems, has shown that symmetry is an important structural feature, that can, and should be exploited to make the bounded model checking approach even more powerful.

2.1 Symmetry in Property Checking Problems

To solve a BMC problem, it has to be checked whether some Boolean function is identical to 1. As described above, this is mainly done using search procedures like SAT, which employ clever search heuristics. However, symmetries in search problems, and in particular in bounded model checking problems, make these search heuristics ineffective. This often leads to exponential behavior and thus to inefficiency, because symmetrical parts of the search space are considered several times. The idea of symmetry reduction is to prevent this.

Basically, there are two kinds of symmetry found in Boolean functions. The first is invariance under permutation of some variables (symmetrical variables), the second is invariance under assignment of different values to the same variable (symmetrical values of a variable).

The intuitive notion of symmetry is that of symmetrical variables. A Boolean function f is symmetrical in some variables x and y if f stays invariant under permutation of x and y. Consider for example the function g:

$$\begin{aligned} g : \{0,1\} \times \{0,1\} &\rightarrow \{0,1\} \\ g(x,y) &\mapsto x \wedge y \end{aligned}$$

In g, x and y are interchangeable since $\wedge$ is a commutative operation. Hence x and y are symmetrical variables in g.

There are several published approaches on how to exploit this kind of symmetry [12, 9, 19], however, for our examples neither of them worked well since either such symmetries were not present, or if they were detected, the reduction did not show much effect.

In contrast, we will focus on symmetrical values, which are a generalization of equivalent values. We will first give a notion of equivalent values and their exploitation and then generalize to symmetrical values.

In general, if $f : D_1 \times \cdots \times D_n \rightarrow D$ is a function in n variables X with discrete domains and codomains, then the values a and b are *equivalent values* for a variable $x \in X$ iff the restrictions of f to $x = a$ and $x = b$ are the same, i.e. iff $f|_{x=a} = f|_{x=b}$. If all values for x are pairwise equivalent then the problem whether $f = 1$ is independent of x.

Let in the following $f : \{0,1\} \times \cdots \times \{0,1\} \rightarrow \{0,1\}$ be an n-ary Boolean function in n variables $X = \{x_1, \ldots, x_n\}$. The restrictions of f to $x = 0$ and $x = 1$ are the cofactors of f w.r.t. x. If the values 0 and 1 are equivalent for $x \in X$ (i.e. if $f_x = f_{\neg x}$ then $f = 1$

is independent of x. So, if we want to check whether $f = 1$ holds, it is sufficient to show either $f_x = 1$ or $f_{\neg x} = 1$. Conversely, if we know that $f_x \neq f_{\neg x}$ then $f = 1$ does not hold.

Now the notion of equivalent values is extended to symmetrical values. First consider the following example.

As an example consider the function f_2 below.

$$\begin{array}{rcl} h : \{0,1\} \times \{0,1\} \times \{0,1\} & \rightarrow & \{0,1\} \\ h(x,y,z) & \mapsto & ite(x, y \rightarrow (y \vee z), z \rightarrow (z \vee y)) \end{array}$$

Then $h|_{x=1} = y \rightarrow (y \vee z)$ and $h|_{x=0} = z \rightarrow (z \vee y)$. It is easy to see that both cofactors are equivalent, since they are both tautological (identical to 1). (However, in general (e.g. when h is a property checking problem) this is not so easy to see, i.e. it is in general a computationally complex task.)

Obviously, if a function is constant, renaming variables does not change this fact, since it is independent of all variables. For a Boolean function f and a permutation π of variables in f we have in particular $f = 1$ iff $\pi(f) = 1$. We can conclude that $f = 1$ holds iff $f|_{x=1} = 1$, and there exists some π fixing x such that $f|_{x=1} = \pi(f|_{x=0})$.

Consider h in the example above. By looking at the cofactors $h|_{x=1} = y \rightarrow (y \vee z)$ and $h|_{x=0} = z \rightarrow (z \vee y)$ it is evident that their representations are very similar. In fact the terms $y \rightarrow (y \vee z)$ and $z \rightarrow (z \vee y)$ have the same structure. They only differ in the names of variables. By permuting y and z they can be mapped onto each other. This means that π permuting y and z (while fixing x) we found such a variable renaming and can conclude: $h = 1$ iff $h|_{x=1} = 1$. Thus we can eliminate x from the problem and consider either cofactor further.

In the following the values 0 and 1 are called *symmetrical values* for x in f, iff the two cofactors of f w.r.t. x are *permutation equivalent* (symmetrical), i.e. if there exists a permutation π of the variables in f fixing x such that $f|_{x=1} = \pi(f|_{x=0})$. Note that if we can prove that there is no such π then f may not be constant and we can conclude that $f = 1$ does not hold. (Note that if π is restricted to be the identity the argument above is still valid, and we are back to the equivalent value case.)

Now let X be the set of all variables in f. Consider the situation where a set of k variables $X_k = \{x_1, \ldots, x_k\} \subset X$ is 'factored out' at the same time. Let $a_1, \ldots, a_k, b_1, \ldots, b_k \in \{0,1\}$ then vectors $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ are called *equivalent value vectors* for the vector of variables $(x_1, \ldots, x_k)$ w.r.t. f, iff

$$f|_{x_1=a_1,\ldots,x_k=a_k} = f|_{x_1=b_1,\ldots,x_k=b_k}$$

(Here the functions $f|_{x_1=a_1,\ldots,x_k=a_k}$ and $f|_{x_1=b_1,\ldots,x_k=b_k}$ are k'th order cofactors of f w.r.t. X_k.) It is easy to see that equivalence of value vectors is an equivalence relation on the 2^k different value vectors, which are partitioned into equivalence classes.

Applying the same extension as in the one variable case the vectors $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ are called *symmetrical value vectors* for the vector of variables $(x_1, \ldots, x_k)$ w.r.t. f, iff

$$\exists \pi \in Sym(X - X_k) : f|_{x_1=a_1,\ldots,x_k=a_k} = \pi(f|_{x_1=b_1,\ldots,x_k=b_k})$$

(Note that π above can be extended trivially to a permutation of X, fixing the elements of X_k.) Symmetry of value vectors is an equivalence relation on the 2^k different value vectors, which are partitioned into equivalence classes. This equivalence relation is called the *symmetry relation* for f w.r.t. X_k.

Now, symmetrical value vectors for vectors of variables in some Boolean function, if known, can be used for preprocessing in property checkers. Let F_{X_k} be the set of all k-th order cofactors of f w.r.t. a set $X_k \subseteq X$ with $|X_k| = k$. Then $f = 1$ holds iff all cofactors in F_{X_k} are constantly 1, or, respectively, if they are all pairwise permutation equivalent and either of them is constantly one.

$$\begin{aligned} & f = 1 \\ \Leftrightarrow \;\; & \forall f', g' \in F_{X_k} \exists \pi \in Sym(X - X_k) : f' = \pi(g') \;\wedge\; \exists h' \in F_{X_k} : h' = 1 \end{aligned}$$

As an example
consider a 256×16 memory and a property describing its write behavior, as shown in Figures 1 and 2. The resulting problem is obviously symmetrical in all values for the bitvector of `ax` and `dx`, respectively. If this can be shown formally by a tool this knowledge can be exploited in such a way that only one 8×16-order cofactor of the respective Boolean function has to be considered. This reduces the search space by a factor of 2^{128}.

In general the symmetry relation is not a single equivalence class. Even if it is, sometimes just an approximation of the symmetry relation might be known. Then still only one cofactor for each equivalence class has to be checked. Let $P = \{P_1, \ldots, P_l\}$ be a partition of 2^k, and let $R = \{r_1, \ldots, r_l\}$ with $r_j = (a_{j1}, \ldots, a_{jk}) \in P_j, 1 \leq j \leq l$ be a set of representatives of each equivalence class. Then the following holds:

$$f = 1 \Leftrightarrow \bigwedge_{1 \leq j \leq l} f|_{x_1 = a_{j1}, \ldots, x_k = a_{jk}} = 1$$

Obviously the quality of the reduction depends on how good the approximation of the symmetry relation is and how easy it can be computed. Approaches for finding good approximate symmetry relations are described below.

2.2 Finding Symmetrical Value Vectors

The problem of finding symmetrical value vectors can be solved by showing permutation equivalence of the respective cofactors. In general, two n-ary Boolean functions f and g over variables $X = \{x_1, \ldots, x_n\}$ are *permutation equivalent* (symmetrical) iff there is a permutation $\pi \in Sym(X)$ of the variables X such that $f = \pi(g)$, i.e. $f(x_1, \ldots, x_n) = g(\pi(x_1), \ldots, \pi(x_n))$ for all $x_1, \ldots, x_n \in \{0, 1\}$. Such a permutation is called a *variable renaming*. The problem of finding π such that for two functions f and g, we have $f = \pi(g)$ is known as the permutation equivalence problem. For Boolean functions it is also known as Boolean isomorphism problem.

As an example consider the functions f and g:

$$\begin{aligned} f : \{0,1\} \times \{0,1\} & \rightarrow \{0,1\} \\ f(x, y) & \mapsto (x \wedge \neg y) \vee (\neg x \wedge \neg y) \\ g : \{0,1\} \times \{0,1\} & \rightarrow \{0,1\} \\ g(x, y) & \mapsto (y \wedge \neg x) \vee (\neg y \wedge \neg x) \end{aligned}$$

```
`define MEM_WIDTH 16
`define MEM_DEPTH 8
`define MEM_CELLS 256

module memory (cs, rw, rs, a, di, do, clk);

   input cs;
   input rw;
   input rs;
   input [`MEM_DEPTH-1:0] a;
   input [`MEM_WIDTH-1:0] di;
   input                 clk;
   output [`MEM_WIDTH-1:0] do;
   reg [`MEM_WIDTH-1:0]    m [`MEM_CELLS-1:0];
   reg [`MEM_WIDTH-1:0]    o;

   integer                 i;

   always @(posedge clk)
      begin
         if (rs)
            begin
               for (i = 0; i<=`MEM_CELLS-1 ; i = i+1)
                  begin
                     m[i]=`MEM_WIDTH'd0;
                  end
               o = `MEM_WIDTH'd0;
            end
         else   // if (clk)
            if (cs)
               begin
                  if (rw)
                     m[a]=di;
                  else
                     o = m[a];
               end
            else
               if (! rw)
                  o = `MEM_WIDTH'd0;
      end // always @ (posedge clk or negedge reset)
      assign do = o;

endmodule // memory
```

Figure 1: Verilog-HDL Code of Memory

```
assert
forall ax in {0, MEM_CELLS-1}
  {(rs == 1'b0) && (cs == 1'b1) && ( rw == 1'b1) && (a == ax)}
  |=>
  m[ax] == prev(di)
```

Figure 2: Write Property for Memory in PSL

They are identical after renaming x with y and y with x in either function, i.e. for $\pi = (x \mapsto y, y \mapsto x)$ we have $f(x,y) = g(\pi(x), \pi(y))$. Note that the functions f' and g' below are also symmetrical by the same π.

$$\begin{array}{rcl} f' : \{0,1\} \times \{0,1\} & \rightarrow & \{0,1\} \\ f'(x,y) & \mapsto & \neg y \\ g' : \{0,1\} \times \{0,1\} & \rightarrow & \{0,1\} \\ g'(x,y) & \mapsto & \neg x \end{array}$$

The functions f and f' are actually the same. Therefore they are also symmetrical. (The same holds for g and g'.)

It is well known that the permutation equivalence problem for Boolean functions is Co-NP-complete [1]. So, the question is why should it be easier to solve the permutation equivalence problem for cofactors than proving their constantness individually.

The answer is twofold. On one hand it is to be expected that, for Boolean functions generated when property checking, self similarities, prevalent in the structure of design and property, are reflected in the representation of this function (as in h). By choosing the set X_k appropriately this should lead to similar representations for the cofactors.

On the other hand we do not really need to decide the permutation equivalence of cofactors. Instead it is often sufficient to employ semi-decision procedures. Using a semi-decision procedure the set of all possible value vectors, and hence the set of all cofactors F_{X_k} w.r.t. X_k is partitioned into equivalence classes of an approximate symmetry relation. In this case we need to prove $f' = 1$ for only one representative f' of each equivalence class.

Such procedures are for example (semi-) decision procedures for Boolean equivalence (where π is the identity), as known from combinational equivalence checking or certain rewrite heuristics. Structural symmetry of cofactors is another criterion implying permutation equivalence. Obviously different approaches can be combined in order to improve a given symmetry relation. It is also possible to compute symmetry relations for two disjoint sets of variables X_k and X_l separately and to join them yielding a symmetry relation for $X_k \cup X_l$.

Since many of these approaches benefit from structural similarities in the function representation our practical implementation lifts the idea to the register-transfer-level, i.e. instead of a Boolean function we consider a bitvector function f' with Boolean codomain. The grouping of Boolean variables to bitvector variables and the representation of f' is naturally derived from the bitvector variables of the RTL representation of design and property. This ensures that structural self similarities are preserved as much as possible by representing the Boolean function as bitvector term (comparable to an HDL expression). It also gives a guideline on how to choose X_k which is done along the bits of bitvector variables.

For computing approximate symmetry relations we developed new methods operating on sets of bitvector functions. They combine different rewrite heuristics for normalization and reduction of complex bitvector term graphs with isomorphism and automorphism procedures [20] for specially labeled directed acyclic graphs. These graphs originate in the term graph representation of the bitvector functions (cofactors) in question. If two graphs for two functions are isomorphic, then the respective term graphs have the same structure modulo permutation of arguments of commutative function symbols. The automorphism procedure allows to check many graphs (and thus many functions) for isomorphism at the same time. A detailed description of the approach is out of our scope here, but some of the ideas are presented in more detail in [8].

2.3 Practical Results

The reduction approach described above has been integrated into the tool RtProp and has been used on industrial designs, in many cases showing considerable performance gains.

To make the approach work practically, different rewrite heuristics for bitvector terms have been developed. Furthermore different degrees of structural symmetry can be discovered by specially adopted graph algorithms working on bitvector terms. Interleaving them leads to fast and reliable preprocessing techniques for property checking, exploiting symmetrical values.

One of our industrial applications is a 256x55-bit 2 way set associative Tag-RAM as they are commonly used in memory caches. It consists of about 500 lines of Verilog source code and could not be scaled down by hand easily. Therefore it was desirable to verify the design as is. It contains a lot of control logic to organize the data transfers and some arithmetic for determining whether a hit of miss occurred. It was verified that the read and write operations work correctly and the content of the tag memory does not change if the memory is not enabled. The properties are parameterizable in the number of memory cells, between 2 and 256. The properties are specified over 8 time steps. Up to 6 variables have been selected for elimination. The advantages of the reduction approach are tremendous for all three properties.

The first property ensures that the memory content does not change unwanted. For the configurations proving this property for 2 to 32 addresses (address input values) and 2 to 32 memory cells all cofactors were collapsed to a single one which was proven almost in no time. In the other cases not all cofactors were identified leaving 64, 128 and 256 small problems. The reduction time increased with the number of cofactors, except the transition from 128 to 256. Here the reduction procedure used a different strategy due to the large number of cofactors (It gave up earlier on some reductions). Proving the property without reduction required more than three hours CPU-time. Using reduction this time was reduced to about 7 minutes.

The situation is comparable for the other properties. Using reduction the last property has been proven for the first time (however, spending 14 hours CPU time). It could not be proven before. Here, the remaining cofactors were often symmetrical for different configurations since just the property was scaled up.

3 Automated Data Path Scaling

RTL descriptions of digital circuts naturally contain structural information how single bits belong to the same signal. Under certain conditions it is possible to scale down a data path to a smaller bit width, such that a property in question holds for the original design if and only if it holds for the scaled down design. Data path scaling is widely used to reduce problem complexity, however up to now it is a manual process, which is time consuming and error prone. In this section we outline an one-to-one abstraction based on automated data path scaling.

Based on data flow descriptions by formal bitvector theories the verification problem is interpreted as SAT problem and is represented as a set E of bitvector equalities and inequalities. Here, RT-Signals correspond to bitvector variables occuring in E. Several decision procedures for satisfiability of bitvector equations have been proposed in the past [2, 4, 10, 11], however, they either lacked the expressiveness or the performance to compete with established SAT and BDD techniques.

Instead of solving an instance E of the bitvector satisfiability problem directly, the approach of [14, 16, 18] is to construct an equivalent but reduced instance E'. The reduction is based on reducing the bit widths on data paths by abstracting from redundant information, while maintaining the satisfiability. Since E' is smaller than E, it is usually easier to solve using standard techniques. Since the scaling process delivers a one-to-one abstraction, E' is satisfiable iff E is satisfaible, i.e. satisfying assignments for E' can be translated into satisfying assignments for E. There are no false positives or false negatives introduced.

The scaling process takes place in two phases. First, the coarsest granularity of each RT-signal (variable of E) is computed. This is done by considering structural data dependencies of E. The computed granularity of variable delivers a partition of its bits into consecutive slices, which denote the largest parts of the corresponding RT-signal computable on the same data path.

In the second phase, for each such slice a minimal width is computed such that the equivalence of E and E' is maintained. Depending on these minimal slices, the minimal width of the RT-signal is computed and a reduced model of the design can be calculated, where the amount of reduction that can be achieved depends on the property to be verified. The scaled down version of the design is then used for verification instead of the original one.

In theory, a linear reduction from n to m bits corresponds to an exponential reduction of the search space of the SAT problem by $2^{-(n-m)}$. This results in significant a speed-up of the verification process, and larger designs can be verified. Experimental results can be found in [14].

References

[1] Agrawal, M. and Thierauf, T. *The Boolean Isomorphism Problem.* IEEE Symposium on Foundations of Computer Science, pp. 422–430, 1996.

[2] Barrett, C. W. and Dill, D. L. and Levitt, J. R., *"A Decision Procedure for Bitvector Arithmetic"*, Proc. DAC, pp. 522–527, 1998

[3] Biere, A. and Cimatti, A. and Clarke, E.M. and Zhu, Y. *Symbolic Model Checking Without BDDs* Proc. of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), number 1579 in LNCS, pp 193–207, 1999.

[4] Bjørner, N. and Pichora, M. C., *"Deciding Fixed and Non-fixed Size Bit-vectors"*, Proc. TACAS, pp. 376–392, 1998

[5] Bormann, J., *"Productivity Figures for Complete Formal Block Verification"*, User Forum, DATE 2003

[6] Brinkmann, R. *Preprocessing for Property Checking on the Register Transfer Level.*, PhD Thesis, University of Kaiserslautern, 2003.

[7] Brinkmann, R. and Drechsler, R. *RTL-Datapath Verification Using Integer Linear Programming.* Proc. of ASP-DAC/VLSI Design 2002, January 07 - 11, 2002 Bangalore, India, pp. 741–746, 2002.

[8] Brinkmann, R. *Using Symmetry for Problem Reduction in Bounded-Model-Checking on the Register-Transfer-Level.* Proc. of SymCon'01 – Symmetry in Constraint Satisfaction Problems, CP'01 Post-Conference Workshop, 2001.

[9] Crawford, J. and Ginsberg, M.L. Eugene Luks, and Amitabha Roy. *"Symmetry-Breaking Predicates for Search Problems."* KR'96: Principles of Knowledge Representation and Reasoning, pp. 148–159. Morgan Kaufmann, 1996.

[10] Cyrluk, D. and Möller, M. O. and Rueß, H., *"An Efficient Decision Procedure for a Theory of Fixed-Sized Bitvectors with Composition and Extraction"*, Ulmer Informatik-Berichte, Fakultät für Informatik, Universität Ulm, 1996

[11] Cyrluk, D. and Möller, M. O. and Rueß, H., *"An Efficient Decision Procedure for the Theory of Fixed-Sized Bit-Vectors"*, Proc. CAV, pp. 60–71, 1997

[12] Markov, I.L. and Sakallah, K.A., and Aloul, F.A. and Ramani, A. *Solving Difficult SAT Instances in the Presence of Symmetry.* Proc. of ACM/IEEE Design Automation Conf., pp. 731–736, 2002.

[13] Ip, C.N. and Dill, D.L. *Better verification through symmetry.* Proc. of the 11th International Conference on Computer Hardware Description Languages and their Applications (CHDL'93), pp. 97–112, 1993.

[14] Johannsen, P., *"Speeding Up Hardware Verification by Automated Data Path Scaling"*, PhD Thesis, Christian-Albrechts-University of Kiel, 2003

[15] Johannsen, P. and Drechsler, R., *"Formal Verification on the RT-Level – Computing One-To-One Design Abstractions by Signal Width Reduction"*, Proc. VLSI, pp. 127–132, 2001

[16] Johannsen, P. and Drechsler, R., *"Speeding Up Verification of RTL Designs by Computing One-To-One Abstractions with Reduced Signal Widths"*, VLSI 2001 Post Conference Book, 2002, Kluwer Academic Publishers

[17] Johannsen, P., *"On Solving Systems of Bitvector Equations – An Efficient Decision Procedure for a Theory of Fixed-Size Bitvectors with Concatenation, Extraction and Negation"*, Siemens Corp., CT SE 4, 1999

[18] Johannsen, P., *"Reducing Bitvector Satisfiability Problems to Scale Down Design Sizes for RTL Property Checking"*, IEEE Proc. HLDVT, pp. 123–128, 2001

[19] Kravets, V.N. and Sakallah, K.A. *"Generalized Symmetries in Boolean Functions."* Proc. of ICCAD 2000, pp. 526–532, 2000.

[20] McKay, B.D. *Practical Graph Isomorphism.* Congressus Numerantium, volume 30, pp. 45–87, 1981.

[21] Moskewicz, M.W. and Madigan, C.F. and Zhao, Y. Zhang, L. and Malik, S. *Chaff: Engineering an Efficient SAT Solver.* Proc. of the 38th Design Automation Conference (DAC'01), pp. 530–535, 2001.

[22] Möller, M. O. and Rueß, H., *"Solving Bit-Vector Equations"*, Proc. FMCAD, pp. 36–48, 1998

[23] Winkelmann, K., Trylus, J., Stoffel, D., Fey, G., *"Cost-Efficient Formal Block Verification for ASIC Design"*, in: Drechsler, R., Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, 6. GI/ITG/GMM-Workshop Modellierung und Verifikation, Shaker-Verlag 2003.