Formal Verification on the RT Level
Computing One-To-One Design Abstractions by Signal Width Reduction

Peer Johannsen
Corporate Technology CT–SE–4,
Siemens AG, 81730 Munich, Germany
peer.johannsen@mchp.siemens.de

Rolf Drechsler
Institute of Computer Science
University of Bremen, 283539 Bremen, Germany
drechsle@informatik.uni-bremen.de

Abstract

Digital circuit designs are usually given as Register-Transfer-Level (RTL) specifications, but most of today's hardware verification tools are based on bit-level methods, using SAT or BDD-based techniques. RTL specifications contain more explicite structural information than bit-level descriptions. This paper presents a new approach to scale down design sizes before verification by exploiting word-level information. We introduce a one-to-one abstraction technique for RTL property checking, which computes a scaled-down abstract model of a design, in which signal widths are reduced with respect to a property. The property holds for the abstract RTL if and only if it holds for the original RTL. If the property fails, counterexamples for the original design are computed from counterexamples found on the reduced model. The verification task is completely carried out on the scaled-down version of the design; false-negatives cannot occur. Linear signal width reductions result in exponentially smaller state spaces and have a significant impact on the runtimes of verification tools. Experimental results on large industrial circuits have demonstrated the applicability and efficiency of our method.

1 Introduction

Verification has become one of the most important steps in digital circuit design. Today's circuit designs often contain up to several million transistors. The test for correct behavior before manufacturing becomes more and more important and a major economical issue. While design sizes are ever increasing, this test grows more complex, time-consuming, and expensive. Formal verification tasks often fail already due to design sizes. Automated abstraction techniques (e.g. [5]) are a promising approach to enhance capabilities of formal verification tools.

Recently, *Bounded Model Checking* (cf. [2]) and *Bounded Property Checking* have gained increased significance in *Electronic Design Automation* (EDA), as recently surveyed in [13]. The majority of today's industrial hardware verification tools uses bit-level decision procedures, like SAT or BDD-based techniques (see e.g. [3, 12]). However, circuit designs are usually given in terms of RTL specifications, for example coded in *Hardware Description Languages* (HDLs), like VHDL or Verilog. RTL specifications of digital circuits contain explicite structural information which is lost in bit-level descriptions. On bit-level, for example in Boolean formulae, all signals are of one-bit width, and all available functional units are Boolean gates. In contrast to that, on RTL, word-level data structures (e.g. bitvectors and busses) as well as high-level operators (e.g. adders, multipliers, and shifters) are still visible. Several approaches to formal circuit verification have been proposed which make use of such high-level information and which are based on word-level verification techniques, like for example *Word-Level Decision Diagrams* (e.g. [8]), formal *Bitvector Theories* (e.g. [1, 6]), *Integer Linear Programming* (e.g. [14]), *Symmetry Reductions* (e.g. [4, 9]) and *Term Rewriting* (e.g. [7]), to survey only a few.

2 Scaling Design Sizes before Verification

This paper presents a new word-level abstraction technique which is used as a preprocess in high-level property checking of digital circuits. The proposed method automatically scales down data-path widths while preserving design properties. We consider the property checking flow shown in Figure 1. Circuit designs are given as HDL specifications, and properties are described in a linear time logic used in *Symbolic Trajectory Evaluation* and specify the intended behavior of the design within a finite bounded interval of time. Typical properties are subdivided into an assumption part implying a commitment part and consist of temporal operators and state expressions, involving relationships among data words. As an example consider:

```
assume:  (during [t, t+4]:  reset = 0) and
         (at t:  request = 1);
prove:   (at t+3:  acknowledge = 1) and
         (at t+4:  data = 11111111);
```

The standard verification flow is indicated by white boxes. Design and property are transformed (*Synthesis, Unrolling*) into an instance of propositional SAT, i.e. a bit-level formula φ. Satisfiability of φ corresponds to invalidity of the property. φ is handed to a property checker, which uses bit-level verification techniques in order to either prove that the property holds, or to return a counterexample. The gray shaded areas in Figure 1 illustrate how the proposed abstraction technique is incorporated into such a flow. Instead of immediately going down to the bit-level, an RTL representation E of φ is generated, consisting of high-level primitives, like word-level signals (variables) and word-level operators (e.g. arithmetic units, comparators, multiplexors and memory elements). Each signal $\boldsymbol{x}$ has a fixed width $n \in \mathbb{N}_+$ and takes bitvectors of respective length as values.

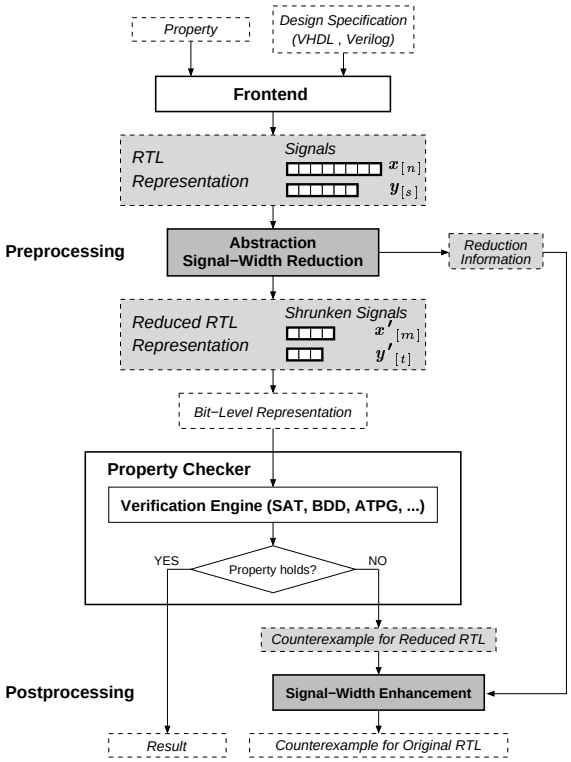

Figure 1. Property Checking Flow

In a preprocessing step, our method takes the RTL representation and computes a second, scaled down RTL model E' by replacing each word-level signal $\boldsymbol{x}$ of E by a corresponding shrunken signal of width $m_{\boldsymbol{x}} \leq n$ (whereby n denotes the original width of $\boldsymbol{x}$). Original and abstract model differ from each other only as far as signal widths are concerned. All other data-flow aspects are preserved. The width of each signal in the abstract RTL is the minimum width which is necessary and sufficient in order to establish a one-to-one abstraction with respect to design, property, and the reduction technique we propose (i.e. by solely changing signal widths):

The property holds for the reduced RTL $\iff$ *The property holds for the original RTL*

The reduced RTL is transformed into a bit-level formula φ' which is given to the property checker instead of φ. Thus, the proposed abstraction technique can be used in combination with a variety of existing (powerful) property checking tools. The bit-level representations φ and φ' contain bit-level variables for each bit of each word-level signal of E and E'. Depending on the degree of reduction of signal widths, φ' can contain significantly less variables than φ. A linear reduction of a signal's width from n bits down to m bits, $m < n$, causes an exponential reduction of the size of the induced state space from 2^n down to 2^m, which can cause a significant speed-up of verification runtimes. If the property does not hold, the property checker returns a counterexample for φ', i.e. for the *reduced* RTL. Our method provides a postprocessing technique which takes such a counterexample and computes values for all inputs of the original design, for which the property fails.

3 Bitvector Equations

Circuit designs can be represented on RTL by systems of bitvector equations such that validity of design properties corresponds to satisfiability of the equations. We define an equational theory $\mathcal{L}(\mathsf{Bv})$ of fixed-size bitvectors, which is an extension of the core theory of bitvectors presented in [6]. Let $\mathbb{B} := \{0, 1\}$. A **bitvector** of width $n \in \mathbb{N}_+$ is a finite vector element $v = \langle v_{n-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}^n$, consisting of n individual bits, which are indexed from right to left, starting with index 0. The set $\mathbb{B}^n$ of bitvectors of width n is denoted by $\mathbb{B}_{[n]}$. A **bitvector variable** of width $n \in \mathbb{N}_+$ is a typed variable $\boldsymbol{x}_{[n]}$, representing fixed-size bitvectors $v \in \mathbb{B}_{[n]}$ of width n. We use bold face characters for bitvector variables, and the width (i.e. the type) is always explicitely denoted. We write $\boldsymbol{x}_{[n]}[i]$ to refer to the i^{th} bit of the value of $\boldsymbol{x}_{[n]}$. The set of well-formed $\mathcal{L}(\mathsf{Bv})$ terms is defined inductively over a finite set of bitvector variables and the set of bitvector operators shown in Table 1.

Operator	Syntax	Example
bitvector variables bitvector constants	$\boldsymbol{x}_{[n]}$ $v_{n-1} \ldots v_1 v_0$	$\boldsymbol{x}_{[8]}$, $\boldsymbol{y}_{[16]}$, $\boldsymbol{z}_{[4]}$, ... 0000, 1111, 00101011, ...
concatenation	$\otimes$	$\boldsymbol{x}_{[16]} = \boldsymbol{y}_{[12]} \otimes \boldsymbol{z}_{[4]}$
extraction	$[j, i]$	$\boldsymbol{x}_{[4]} = \boldsymbol{y}_{[32]}[5, 2]$
bitwise negation	neg	$\boldsymbol{x}_{[8]} = \mathsf{neg}(\boldsymbol{y}_{[8]})$
bitwise Boolean connectives	and, or, xor nand, nor, xnor	$\boldsymbol{x}_{[16]} = \boldsymbol{y}_{[16]}$ and $\boldsymbol{z}_{[16]}$ $\boldsymbol{x}_{[16]} = \boldsymbol{y}_{[16]}$ or $\boldsymbol{z}_{[16]}$
if-then-else	ite	$\boldsymbol{x}_{[8]} = \mathsf{ite}(a_{[4]} = b_{[4]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]})$ $\boldsymbol{x}_{[8]} = \mathsf{ite}(a_{[4]} < b_{[4]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]})$
arithmetic	$\oplus, \ominus, \circledast$	$\boldsymbol{x}_{[32]} = \boldsymbol{y}_{[32]} \oplus \boldsymbol{z}_{[32]}$
memory read memory write	read write	$c_{[4]} = \mathsf{read}(\boldsymbol{m}_{[1024]}, i_{[8]})$ $\boldsymbol{m'}_{[64]} = \mathsf{write}(\boldsymbol{m}_{[64]}, i_{[4]}, v_{[2]})$

Table 1. Bitvector Operators and Equations

Well-formedness of terms implies that variable widths have to comply with operator demands (e.g. index expressions must not exceed the widths of argument terms). Note that additional high-level operators, e.g. shifts and rotations, can already be expressed with the shown set of operators. Within our framework, RTL models consist of a system E of equations of bitvector terms over $\mathcal{L}(\mathsf{Bv})$ such that:

$$E \text{ is satisfiable} \iff \text{The property does } \textbf{not} \text{ hold for the design} \quad (1)$$

A system E is satisfiable if there exists a valuation of the variables of E such that all equations are satisfied. Multi-bit circuit signals directly correspond to bitvector variables of E. A satisfying solution – if existent – yields a counterexample indicating values for all circuit signals such that the property does not hold for these assignments (*falsification, bug hunting*). The proposed abstraction technique generates a second system E' of bitvector equations, which differs from E solely in the manner that variable widths are reduced. The width of each bitvector variable is shrunken to the smallest possible number of bits (with respect to E' differing from E only by reduced variable widths), such that:

$$E' \text{ is satisfiable} \iff E \text{ is satisfiable} \quad (2)$$

E' represents a scaled down version of the original design, and property checking can be done entirely on E'.

4 Signal Width Reduction

Bitvector equations describe data dependencies on word-level. The equations explicitely contain the high-level information, which individual bits belong to the same word-level signal, and how single bits are ordered within multi-bit signals. In the following we show how this information can be used to reduce computational complexity of satisfiability checks of bitvector equations.

If neighboring bits of bitvector variables are computed uniformly according to the same bit-level data flow, then such data dependencies are called a **uniform data flow**.

Example 1 (Uniform Data Dependencies) *Let $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}$ and $\boldsymbol{z}_{[8]}$ be 8-bit signals, and consider the following bitvector equation:*

$$\boldsymbol{x}_{[8]} \text{ and } \boldsymbol{y}_{[8]} \;=\; \boldsymbol{z}_{[8]} \tag{3}$$

Equation (3) specifies functional data dependencies between $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}$ and $\boldsymbol{z}_{[8]}$. Satisfiability of (3) corresponds to satisfiability of the following bit-level representation

$$(x_0 \text{ and } y_0 = z_0) \;\wedge\; \ldots \;\wedge\; (x_7 \text{ and } y_7 = z_7)\,, \tag{4}$$

involving 24 Boolean variables and 8 equations. Obviously it is not necessary to solve all 8 equations of (4) separately, because the data flow for bit-positions 0–7 is computed uniformly. Let $\boldsymbol{x'}_{[1]}$,$\boldsymbol{y'}_{[1]}$,$\boldsymbol{z'}_{[1]}$ denote new signals of width 1, derived from the variables of (3). It is sufficient to check satisfiability of

$$\boldsymbol{x'}_{[1]} \text{ and } \boldsymbol{y'}_{[1]} \;=\; \boldsymbol{z'}_{[1]}\,, \tag{5}$$

because (3) is satisfiable if and only if (5) is satisfiable. A satisfying solution of (3) can be obtained from a solution of (5) by signed extension. For example, $\boldsymbol{x'}_{[1]} = 0$, $\boldsymbol{y'}_{[1]} = 1$, $\boldsymbol{z'}_{[1]} = 0$ yields $\boldsymbol{x}_{[8]} = 00000000$, $\boldsymbol{y}_{[8]} = 11111111$ and $\boldsymbol{z}_{[8]} = 00000000$. □

Uniform data flow is formally characterized by **bitwise bitvector functions**.

Definition 1 (Bitwise Bitvector Functions) *Let $n, k \in \mathbb{N}_+$ and $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitvector function of width n on bitvectors of width n. $\mathcal{F}_{[n]}$ is a **bitwise bitvector function** if there exists a k-ary Boolean function $\mathcal{B}_{[1]} : \mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]} \longrightarrow \mathbb{B}_{[1]}$ such that $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})[i] = \mathcal{B}_{[1]}(\boldsymbol{x}^1_{[n]}[i], \ldots, \boldsymbol{x}^k_{[n]}[i])$ for all $i \in \{0, \ldots, n-1\}$ and all $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$.* ■

$\mathcal{B}_{[1]}$ is called the **characteristic Boolean function** of $\mathcal{F}_{[n]}$. Satisfying solutions of bitvector equations with uniform data dependencies can be characterized by zeros of bitwise bitvector functions.

Definition 2 (Bitwise Bitvector Equations) *Let e be a bitvector equation over $\mathcal{L}(\mathsf{Bv})$, and let $V = \{\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}\}$ be the set of bitvector variables occurring in e. Then e is called a **bitwise bitvector equation** if there exists a bitwise bitvector function $\mathcal{F}_{[n]}$ such that the set of satisfying solutions of e is exactly the set of satisfying solutions of $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) = 0_{[n]}$.* ■

Satisfiability of bitwise bitvector equations for bitvectors of width n can be mapped onto satisfiability of equations for bitvectors of width 1 by simply reducing variable widths, as seen in Example 1 and formalized as follows.

Corollary 1 (Bitwise Bitvector Functions) *Let $n, k \in \mathbb{N}_+$ and $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitwise bitvector function with characteristic Boolean function $\mathcal{B}_{[1]}$. Then $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) = 0_{[n]}$ is satisfiable if and only if $\mathcal{B}_{[1]}(\boldsymbol{x'}^1_{[1]}, \ldots, \boldsymbol{x'}^k_{[1]}) = 0_{[1]}$ is satisfiable.* ■

In general, data flow must be analyzed for all equations of a given system. Reduction depends on structural and dynamical data dependencies imposed by the conjunction of all equations. Thus, even if uniform data flow exists for a specific equation, other equations can be the reason that reduction to only 1-bit width might not preserve satisfiability.

Example 2 (Dynamical Data Dependencies) *Let $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}$ and $\boldsymbol{z}_{[8]}$ be signals of width 8, for which uniform data dependencies exist. Consider a system of bitvector equations, which additionally contains the following expressions:*

$$\begin{array}{lcl} \ldots & = & \ldots\ \mathsf{ite}(\boldsymbol{x}_{[8]}{=}\boldsymbol{y}_{[8]},\ \ldots,\ \ldots)\ \ldots \\ \ldots & = & \ldots\ \mathsf{ite}(\boldsymbol{y}_{[8]}{=}\boldsymbol{z}_{[8]},\ \ldots,\ \ldots)\ \ldots \\ \ldots & = & \ldots\ \mathsf{ite}(\boldsymbol{z}_{[8]}{=}\boldsymbol{x}_{[8]},\ \ldots,\ \ldots)\ \ldots \end{array} \tag{6}$$

A satisfying solution of (6) might require that the values of $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}$ have to be mutually different, i.e.

$$\boldsymbol{x}_{[8]} \neq \boldsymbol{y}_{[8]} \;\wedge\; \boldsymbol{y}_{[8]} \neq \boldsymbol{z}_{[8]} \;\wedge\; \boldsymbol{z}_{[8]} \neq \boldsymbol{x}_{[8]}\,. \tag{7}$$

Then, reduction to only one bit width is not possible, because $\boldsymbol{x'}_{[1]} \neq \boldsymbol{y'}_{[1]} \wedge \boldsymbol{y'}_{[1]} \neq \boldsymbol{z'}_{[1]} \wedge \boldsymbol{z'}_{[1]} \neq \boldsymbol{x'}_{[1]}$ is not satisfiable, while (7) is. Instead the following holds:

$$\boldsymbol{x}_{[m]} \neq \boldsymbol{y}_{[m]} \;\wedge\; \boldsymbol{y}_{[m]} \neq \boldsymbol{z}_{[m]} \;\wedge\; \boldsymbol{z}_{[m]} \neq \boldsymbol{x}_{[m]} \tag{8}$$

is satisfiable for all $m \geq 2$, and at the same time 2 is the minimum value for m, for which

$$(7) \textit{ satisfiable} \iff (8) \textit{ satisfiable}$$

holds. Therefore, satisfiability of (6) can be preserved by choosing reduced bitvectors of width 2. But, even if a solution of (6) maybe only requires $\boldsymbol{x}_{[8]} \neq \boldsymbol{y}_{[8]} \wedge \boldsymbol{y}_{[8]} \neq \boldsymbol{z}_{[8]}$, a reduction to one bit widths still might not be sufficient, although $\boldsymbol{x'}_{[1]} \neq \boldsymbol{y'}_{[1]} \wedge \boldsymbol{y'}_{[1]} \neq \boldsymbol{z'}_{[1]}$ is satisfiable, because the uniform data dependencies between $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}$, as imposed by further equations of (6), could be the following:

$$\begin{array}{lcl} 11111111 & = & (\,\boldsymbol{x}_{[8]} \text{ and } \boldsymbol{y}_{[8]} \text{ and } \mathsf{neg}(\boldsymbol{z}_{[8]})\,) \text{ or} \\ & & (\,\mathsf{neg}(\boldsymbol{x}_{[8]}) \text{ and } \boldsymbol{y}_{[8]} \text{ and } \boldsymbol{z}_{[8]}\,) \end{array}$$

Such conditions are satisfiable for bitvectors of width 8, but the corresponding problem, where variables are reduced to 1-bit width, is unsatisfiable. Instead, satisfiability again is preserved when reduced bitvectors of width 2 are chosen. □

Data dependencies can exist between complete bitvectors or only between certain bits. Typically, different data dependencies exist for different chunks of a variable. Variables can be partitioned into contiguous parts in which all bits are treated uniformly with respect to data dependencies.

Example 3 (Structural Data Dependencies) *Let $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}$ and $\boldsymbol{z}_{[8]}$ be bitvector variables of width 8, and let $\boldsymbol{a}_{[2]}, \boldsymbol{b}_{[2]}$ be bitvector variables of width 2. Consider the following system E of bitvector equations:*

$$E \left\{ \begin{array}{lcl} \boldsymbol{x}_{[8]} \text{ and } \boldsymbol{y}_{[8]} & = & \boldsymbol{z}_{[8]} \\ \boldsymbol{x}_{[8]} & = & \boldsymbol{a}_{[2]} \otimes \boldsymbol{b}_{[6]} \end{array} \right. \quad (9)$$

*The first equation specifies uniform data dependencies for $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}$, but the second one imposes different structural dependencies for the upper two and the lower six bits of $\boldsymbol{x}_{[8]}$. E can be decomposed into two **disjoint** independent systems E_1 and E_2 of bitvector equations,*

$$E_1 \left\{ \begin{array}{lcl} \boldsymbol{x}_{[8]}[7,6] \text{ and } \boldsymbol{y}_{[8]}[7,6] & = & \boldsymbol{z}_{[8]}[7,6] \\ \boldsymbol{x}_{[8]}[7,6] & = & \boldsymbol{a}_{[2]} \end{array} \right.$$
$$E_2 \left\{ \begin{array}{lcl} \boldsymbol{x}_{[8]}[5,0] \text{ and } \boldsymbol{y}_{[8]}[5,0] & = & \boldsymbol{z}_{[8]}[5,0] \\ \boldsymbol{x}_{[8]}[5,0] & = & \boldsymbol{b}_{[6]} \end{array} \right. \quad (10)$$

such that the set of satisfying solutions of E is the same as the set of satisfying solutions of the conjunction of E_1 and E_2, i.e. E is satisfiable if and only if $E_1 \wedge E_2$ is satisfiable. Furthermore, all data dependencies in E_1 and in E_2 are uniform. Satisfiability of (10) then can be reduced to satisfiability of:

$$E_1' \left\{ \begin{array}{lcl} \boldsymbol{x}'_{[1]} \text{ and } \boldsymbol{y}'_{[1]} & = & \boldsymbol{z}'_{[1]} \\ \boldsymbol{x}'_{[1]} & = & \boldsymbol{a}'_{[1]} \end{array} \right.$$
$$E_2' \left\{ \begin{array}{lcl} \boldsymbol{x}''_{[1]} \text{ and } \boldsymbol{y}''_{[1]} & = & \boldsymbol{z}''_{[1]} \\ \boldsymbol{x}''_{[1]} & = & \boldsymbol{b}''_{[1]} \end{array} \right. \quad (11)$$

and from (11) we can recompose

$$E' \left\{ \begin{array}{lcl} \boldsymbol{x}'''_{[2]} \text{ and } \boldsymbol{y}'''_{[2]} & = & \boldsymbol{z}'''_{[2]} \\ \boldsymbol{x}'''_{[2]} & = & \boldsymbol{a}'''_{[1]} \otimes \boldsymbol{b}'''_{[1]} \end{array} \right. \quad (12)$$

with (12) is satisfiable if and only if (9) is satisfiable. $\boldsymbol{a}'''_{[1]}$ relates to $\boldsymbol{a}_{[2]}$, $\boldsymbol{b}'''_{[1]}$ relates to $\boldsymbol{b}_{[6]}$, $\boldsymbol{x}'''_{[2]}[1,1]$ relates to $\boldsymbol{x}_{[8]}[7,6]$, and $\boldsymbol{x}'''_{[2]}[0,0]$ to $\boldsymbol{x}_{[8]}[5,0]$, and so on. To obtain a solution of (9), signed extension is done separately for related chunks according to the prior decomposition, for example $\boldsymbol{a}'''_{[1]} = 0$, $\boldsymbol{b}'''_{[1]} = 1$, $\boldsymbol{x}'''_{[2]} = 01$, $\boldsymbol{y}'''_{[2]} = 11$, $\boldsymbol{z}'''_{[2]} = 01$, yields $\boldsymbol{a}_{[2]} = 00$, $\boldsymbol{b}_{[6]} = 111111$, $\boldsymbol{x}_{[8]} = 00111111$, $\boldsymbol{y}_{[8]} = 11111111$, $\boldsymbol{z}_{[8]} = 00111111$. □

5 Granularity Decomposition

According to Definition 2 and Corollary 1, satisfiability of bitwise bitvector equations can be mapped to satisfiability of corresponding equations over bitvectors of reduced width. This technique can be generalized to scale down signal widths for whole systems of bitvector equations as shown in Example 3.

A **chunk** of a bitvector variable $\boldsymbol{x}_{[n]}$ is a triple $(\boldsymbol{x}_{[n]}, j, i)$ with $0 \leq i \leq j < n$, representing the contiguous part of $\boldsymbol{x}_{[n]}$ which is described by the bitvector term $\boldsymbol{x}_{[n]}[j,i]$. Two chunks $\boldsymbol{x}_{[n]}[j_1, i_1]$ and $\boldsymbol{x}_{[n]}[j_2, i_2]$ are **disjoint** if either $i_1 > j_2$ or $j_1 < i_2$. A finite number of *mutually disjoint* sets $C_1, \ldots, C_q$ of chunks of bitvector variables is called a **granularity**, if $\bigcup C_i$ is a set of *disjoint* chunks.

Definition 3 (Granularity Decomposition) *Let V be a set of bitvector variables and let E be a system of bitvector equations over variables of V. A* granularity decomposition *of E is a decomposition of E into a finite number $E_1, \ldots, E_q$ of* independent *systems of $\mathcal{L}(\mathsf{Bv})$ equations and into sets of chunks $C_1, \ldots, C_q$ and sets $I_1, \ldots, I_q \subseteq V \times V$ such that*

- *$C_1, \ldots, C_q$ is a granularity of V,*
- *each C_i is exactly the set of chunks (variables) occurring in E_i,*
- *all equations of each E_i are bitwise bitvector equations, and*
- *the set of satisfying solutions of E consists exactly of all compositions of satisfying solutions of $E_1, \ldots, E_q$, which additionally satisfy the inequalities specified by $I_1, \ldots, I_q$.* ■

The process of scaling the widths of bitvector variables for a system E of bitvector equations is separated into two subsequent phases. Section 6 describes how first a granularity decomposition of E is computed, which, for each bitvector variable $\boldsymbol{x}_{[n]}$ of E, describes a splitting of $\boldsymbol{x}_{[n]}$ according to uniform data flow, as imposed by structural data dependencies. Then, for each chunk of $\boldsymbol{x}_{[n]}$, the necessary minimum width is computed, which preserves satisfiability as required by dynamical data dependencies (cf. Example 2). This is further explained in Section 7. According to these computed minimum chunk widths, the reduced width for the corresponding shrunken variable of E' is reassembled, as illustrated in Figure 2.

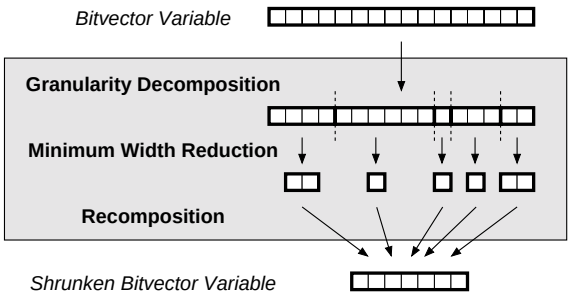

Figure 2. Basic Abstraction Technique

A granularity decomposition of a given system E always exists, because bitvector variables can always be decomposed into single bits and the data dependencies of E can always be described on bit-level. Usually, there exists a variety of possible decompositions. If, for example, uniform data dependencies exist for a chunk $\boldsymbol{x}_{[8]}[5,0]$, then a finer splitting, e.g. into $\boldsymbol{x}_{[8]}[5,4]$ and $\boldsymbol{x}_{[8]}[3,0]$, is also a valid decomposition. The highest amount of reduction can be achieved for bitwise decompositions with chunks of the largest possible width. Finding the coarsest possible decomposition for an arbitrary E, is a problem of detecting uniform data flow. Hence, it is a problem of deciding equality of Boolean functions, and thus is NP-complete.

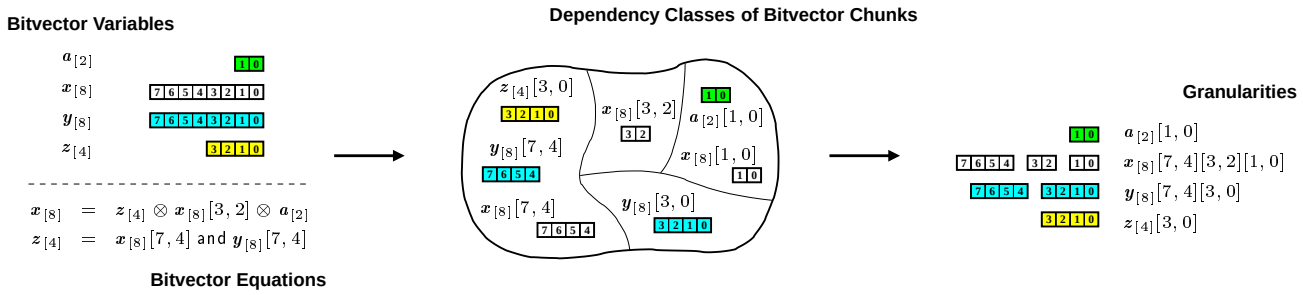

Figure 3. Granularity Decomposition

6 Uniform Data Flow Analysis

Our methods computes a granularity decomposition of a given system E by syntactical analysis of the bitvector equations of E. According to the high-level information about operators and multi-bit signals, which is available within the $\mathcal{L}(\mathsf{Bv})$ formalism, chunks of neighboring bits of bitvector variables are determined, for which uniform data dependencies exist. The granularity decomposition is determined by means of an equivalence class structure, which groups chunks of bitvector variables between which functional dependencies can be described by a bitwise bitvector function.

The equivalence class computation can efficiently be done by employing a union-find algorithm, which, besides the known union() and find() operations, defines a new procedure slice(). Initially, one complete chunk for each bitvector variable $\boldsymbol{x}_{[n]}$ resides in its own singleton equivalence class $\{\boldsymbol{x}_{[n]}[n-1,0]\}$. A call of $\mathsf{find}(\boldsymbol{x}_{[n]}, i)$ yields the (non-ambiguous) equivalence class which includes a chunk of $\boldsymbol{x}_{[n]}$ which contains bit position i; union() performs the usual set union of two classes; and $\mathsf{slice}(\boldsymbol{x}_{[n]}, j, i)$ calls $\mathsf{find}(\boldsymbol{x}_{[n]}, i)$ and $\mathsf{find}(\boldsymbol{x}_{[n]}, j)$ and splits all chunks of the respective classes at the bit positions corresponding to i and j. The originating parts are grouped in two new classes.

Each bitvector equation e of E is sequentially (but in an arbitrary order) analyzed, and the next state of the equivalence class structure is computed by means of the procedure $\mathsf{gran}(e)$, which is outlined in Algorithm 1. The procedure recursively performs a case split according to the top-level operators occurring in the bitvector terms and computes a coarsest-possible[1] granularity decomposition of all variables occurring in e. When all bitvector equations have been processed, the granularity decomposition of E is given by the sets of chunks residing in each equivalence class. An example is illustrated in Figure 3.

Algorithm 1 Granularity Analysis of Bitvector Equations

```
gran(e) {
  switch (e);
  case e ≡ 's[n] = t[q] ⊗ u[r] ':
    gran('s[n][n-1, r] = t[q] '); gran('s[n][r-1, 0] = u[r] ');
  case e ≡ 's[n] = neg(t[n]) ':
    gran('s[n] = t[n] ');
  case e ≡ 's[n] = ite(a[q] = b[q], t[n], u[n]) ':
    gran('a[q] = b[q] '); gran('s[n] = t[n] '); gran('s[n] = u[n] ');
  case e ≡ 's[n] = t[n] and u[n] ':
    gran('s[n] = t[n] '); gran('s[n] = u[n] ');
  ...
  case e ≡ 's[n] = (t[q] ⊗ u[r])[j, i] ':
    if (j < r) {
      gran('s[n] = u[r][j, i] ');
    } else if (i ≥ r) {
      gran('s[n] = t[q][j - r, i - r] ');
    } else {
      gran('s[n] = t[q][j - r, 0] ⊗ u[r][r - 1, i] ');
    }
  case e ≡ 's[n] = (t[q][l, k])[j, i] ':
    gran('s[n] = t[q][k + j, k + i] ');
  case e ≡ 's[n] = ite(a[q] = b[q], t[r], u[r])[j, i] ':
    gran('s[n] = ite(a[q] = b[q], t[r][j, i], u[r][j, i]) ');
  case e ≡ 's[n] = (t[q] and u[q])[j, i] ':
    gran('s[n] = t[q][j, i] and u[q][j, i] ');
  ...
  case e ≡ 'x[n][j, i] = y[q][l, k] ':
    slice(x[n], j, i); slice(y[q], l, k); union(x[n]⟨j, i⟩, y[q]⟨l, k⟩);
}
```

7 Minimum Width Abstraction

The initial satisfiability problem for E is decomposed into a number of independent satisfiability problems E_i as described in Definition 3, which are associated with the computed equivalence classes. The specialty of our method is that these instances E_i do **not** have to be computed or represented explicitely. For the reduction technique, which we propose, it is sufficient to know, that the solutions of these problems can be characterized by satisfiability problems for bitwise bitvector functions and sets of inequalities.

Definition 4 (BvSat) *Let $k, n \in \mathbb{N}_+$ and $V = \{1, \ldots, k\}$. Let $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitvector function of width n on bitvectors of width n. Let $I \subseteq V \times V$. Then $\mathsf{BvSat}(\mathcal{F}_{[n]}, I)$ denotes the problem whether there exist $\boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$, such that $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]}) = 0_{[n]}$ and $\boldsymbol{x}^i_{[n]} \neq \boldsymbol{x}^j_{[n]}$ for all $\{i, j\} \in I$.* ■

BvSat is an NP-complete Problem, and is in detail investigated in [11], where the following fundamental theorem on width reductions for bitwise bitvector functions and inequalities is presented.

[1] Coarsest, in the sense of: as coarse as can be concluded by purely syntactical analysis. Yet in fact, for many practical applications in digital circuit design our technique yields the optimum decomposition.

If two k-ary bitwise Bitvector functions $\mathcal{F}^1_{[n]}$ and $\mathcal{F}^2_{[m]}$ on bitvectors of width n and m operate according to the same characteristic Boolean function, then let this correspondence be denoted by $\mathcal{F}^1_{[n]} \simeq \mathcal{F}^2_{[m]}$.

Theorem 1 (Minimum Width Reduction) *Let $k, n \in \mathbb{N}_+$ and $V = \{1, \ldots, k\}$. Let $I \subseteq V \times V$ and let $p \in \mathbb{N}_+$ be the number of connected components of the undirected graph $G(V, I)$ with vertices V and edges I. Let $m := \min\big(n, \max(1,\ k - p)\big)$. Then m is the minimum value for which the following holds:*

$$\textit{for all } k\textit{-ary bitwise } \mathcal{F}_{[n]} : \ \underset{\textit{satisfiable}}{\mathsf{BvSat}(\mathcal{F}_{[n]}, I)} \iff \underset{\textit{satisfiable}}{\mathsf{BvSat}(\mathcal{F}_{[m]}, I)}$$

whereby for each $\mathcal{F}_{[n]}$, $\mathcal{F}_{[m]}$ denotes the corresponding bitwise bitvector function of width m width $\mathcal{F}_{[n]} \simeq \mathcal{F}_{[m]}$. ■

Note that the reduced width m, which is computed in Theorem 1, depends only on the arity k of the bitwise functions and on the set I of inequalities. Thus, for each equivalence class C_i a reduced width $\varphi(C_i)$ can be computed, which preserves satisfiability of the associated bitwise bitvector equations E_i in a one-to-one fashion. $\varphi(C_i)$ only depends on the size of the equivalence class (i.e. on the number of chunks contained in C_i) and on the number of connected graph components as induced by the inequalities, which are derived from guard expressions of **`if-then-else`** terms which involve variables of C_i.

The computation of the number of connected graph components for each class can efficiently be done by using a union-find algorithm, and moreover, can be embedded within the computation of the equivalence classes during the granularity decomposition.

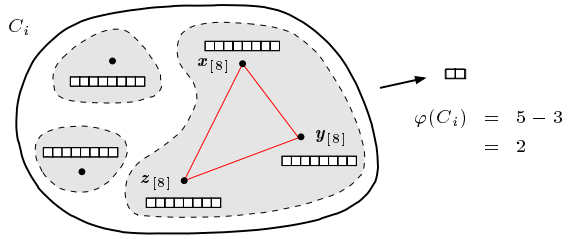

Figure 4. Minimum Width Abstraction

Fig. 4 reconsiders Example 2, showing a sample equivalence class; inequalities are drawn as edges between chunks.

8 Experience and Summary

Reduction of data path widths has always been a classical attempt of minimizing state place explosion for formal model checking methods. Many EDA companies today perform such reductions manually to reduce verification runtimes, often without having the guarantee that the chosen amount of scaling does not falsify verification results.

In this paper we have presented a fully automated one-to-one RTL abstraction technique, which efficiently analyzes word-level data-flow in RTL descriptions with respect to a specified property. Designs are scaled down by reducing signal widths before property checking, while guaranteeing that the property holds for the scaled model if and only if it holds for the original design.

The proposed abstraction technique has been implemented in C++ in a tool called BOOSTER (Boolean String Length Reduction) and can easily be integrated into existing verification flows. BOOSTER was tested in several case studies at the EDA departments of Siemens Corporation in Munich and Infineon Technologies in San Jose, CA. Experiments on large industrial circuits have demonstrated the applicability and efficiency of the presented technique (for details see [10]). Good results have been achieved for specific types of digital designs, which provide a high degree of uniform data-flow, as for example memories, queues, stacks, bridges and interface protocols. In an experiment we considered a design of roughly 3.000 lines of Verilog code with a synthesized netlist of 24.000 gates and 35.000 RAM cells. The width of the data-path could be reduced from 10 bits down to 3 bits. The size of the bit-level model was shrunken to 30% of the original size, and property checker runtimes dropped down to only 5% of the former runtimes.

The proposed technique can furthermore be applied in high-level equivalence checking and high-level simulation.

References

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

[2] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. "Symbolic Model Checking Using SAT Procedures instead of BDDs". In *Proc. DAC*, pages 317–320, 1999.

[3] R. E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[4] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. "Symmetry Reductions in Model Checking". In *Proc. CAV*, pages 147–158, 1998.

[5] E. M. Clarke, O. Grumberg, and D. E. Long. "Model Checking and Abstraction". In *Proc. POPL*, pages 342–354, 1992.

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

[7] N. Dershowitz and J. P. Jouannaud. *"Handbook of Theoretical Computer Science, Formal Models and Semantics"*, chapter "Rewrite Systems", pages 243–320. J.v. Leeuwen, Elsevier, 1990.

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

[9] E. A. Emerson and R. J. Trefler. "From Asymmetry to Full Symmetry: New Techniques for Symmetry Reduction in Model Checking". In *Proc. CHARME*, pages 142–156, 1999.

[10] P. Johannsen. "BOOSTER : Speeding Up RTL Property Checking of Digital Designs by Word-Level Abstraction". In *Proc. CAV'01*, pages 373–377, 2001.

[11] P. Johannsen. "Reducing Bitvector Satisfiability Problems to Scale Down Design Sizes for RTL Property Checking". In *IEEE Proc. HLDVT'01*, 2001.

[12] J. P. M. Silva. *"Search Algorithms for Satisfiability Problems in Combinational Switching Circuits"*. PhD thesis, University of Michigan, 1995.

[13] J. P. M. Silva and K. A. Sakallah. "Boolean satisfiability in electronic design automation". In *Proc. DAC*, pages 675–680, 2000.

[14] Z. Zeng, P. Kalla, and M. Ciesielski. "LPSAT: A Unified Approach to RTL Satisfiability". In *Proc. DATE*, pages 398–402, 2001.