SYNTHESIZING CHECKERS FOR ON-LINE VERIFICATION OF SYSTEM-ON-CHIP DESIGNS

Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
email: drechsle@informatik.uni-bremen.de

ABSTRACT

In modern System-on-Chip (SoC) designs verification becomes the major bottleneck. Since by using state-of-the-art techniques complete designs cannot be fully formally verified, it becomes more and more important to check the correct behaviour during operation. This becomes even more significant in systems that are changed during life-time, like re-configurable systems.

In this paper we present a hardware extension that allows to efficiently synthesize checkers and properties that have been used in the verification process. This allows for an on-line verification of SoC designs. For the verification hardware a regular layout is discussed that can easily be synthesized and has a very low area overhead. The on-line check has (nearly) no effect on the delay of the considered chip.

1. INTRODUCTION

Modern circuits contain up to several million transistors. In the meantime it has been observed that verification becomes the major bottleneck, i.e. up to 80% of the overall design costs are due to verification. This is one of the reasons why recently several methods have been proposed as alternatives to classical simulation, since it cannot guarantee sufficient coverage of the design. E.g. in [2] it has been reported that for the verification of the Pentium IV more than 200 billion cycles have been simulated, but this only corresponds to 2 CPU minutes, if the chip is run with 1 GHz.

As alternatives, formal verification or symbolic simulation have been proposed and in the meantime these have been successfully applied in many projects [6]. But so far, all approaches are based on software solutions and cannot be applied after the chip is fabricated. On-line verification approaches have not been considered. But there is a need for these techniques in at least two problem domains:

- If the properties specified by the designer or verification engineer cannot be proven by the verification tool. This might result from too complex properties or from the difficulty of the circuit considered (e.g. for multipliers).
- If the circuit is re-configured during operation [14,4]. The new programmed hardware has to be checked for correct functional behaviour.

In this paper, we present an approach to synthesize hardware that can check properties that have been applied during the verification process. Often these properties are also available directly from the specification [13]. The approach of adding extra hardware has been very successfully applied in the testing domain for many years (see e.g. [15,9]), while hardware verification is mainly applied on the software level. With new emerging technologies this has to be extended. The synthesized circuits are called *verification hardware* in this paper. Verification hardware allows to check the correct behaviour also later, after the chip production. These techniques become very important in SoC designs that allow parts to be re-configured during operation.

After explaining the underlying principle of our approach, a regular hardware layout is discussed that allows to map properties defined in the verification process directly on the circuit with very low hardware overhead. The extra delay resulting from the verification hardware is small, i.e. only one extra fanout per checked signal.

2. CHECKER STRUCTURES

Even though the formalism to describe properties in verification languages varies a lot (see e.g. [10]), the underlying mechanisms are very similar. In the following we use the notation from the property checker used at Infineon Technologies AG (see e.g. [7,8] for more details).

Notice that it is straightforward to generalize the results to also work for other verification languages.

A property consists of an *assume part* and a *proof part*. If all assumptions hold, the property specified in the proof part must hold.

Example 1: We want to prove a property `test`. The property says that whenever signal `x` becomes `1`, two clock cycles later signal `y` has value `0`. More formally:

```
theorem test is
assume:
  at t: x = 1 ;

prove:
  at t+2: y = 0 ;
end theorem ;
```

In general, each property is of the form that whenever some signals have given values (eventually over several time frames), other – or the same – signals assume specified values. Notice that property languages also allow to argue over time intervals, e.g. a requirement can be that a signal assumes a value within a given time interval, while the concrete time point is not given. It is obvious that each of the properties can easily be transferred to hardware realization based on shift registers and some additional logic (see below).

Remark: This is exactly the method by which efficient property checkers formulate the problem by translating the property to a Boolean network and running Boolean provers, like e.g. SAT and BDD [11]. (For more details on SAT-based property checking see [3].) In contrast to shift registers the solvers "un-roll" the circuit for the maximal number of time frames specified in the property. For our approach shift registers are more adequate, since for simulation only the value at an earlier time frame has to be stored, while the surrounding logic can be ignored.

While property checkers "un-roll" the circuit, we now show by a motivating example a hardware realization:

Example 2: Consider the property from Example 1. Whenever signal `x` is `1` it has to be checked that `y` is `0` two time frames later. This can be easily done by storing the value of `x` for two clock cycles and then computing the result by performing an AND operation of the `x`-signal with the negated value of `y` (`y` has to be negated, since it should have the value 0). The corresponding circuit is shown in Figure 1. If the output of the AND gate is 0, the property is violated.

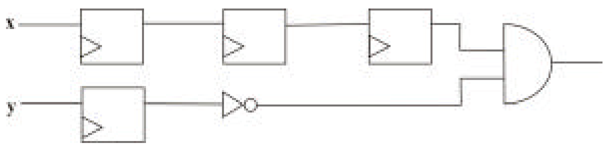

Figure 1. Shift register and logic for Example 1

In this way, it becomes very easy to generate monitors in hardware from given properties. The hardware can be used to check the correct circuit behaviour during operation, i.e. it is applicable for on-line test. This finds application in several scenarios, e.g.:

- "Hard" to proven properties: Some properties turn out to be too difficult to be formally verified. This depends on the property itself and the circuit under consideration. If the design e.g. contains large multipliers, formal verification cannot prove the property. In these cases, simulation is not sufficient (see [2] and the discussion in Section 1).

- In re-configurable computing hardware components are exchanged during normal operation. An "external" verification method based on software cannot be applied in this case. Here verification hardware is a promising alternative. (Similar concepts have been studied in the area of testing for a longer time [15], while verification is so far mainly "software-oriented".)

It is straightforward to also include more complex operations, like reasoning over several time intervals, by simply adding the corresponding signals, i.e. OR-ing the signals.

2.1 Algorithmic Solution Using String Matching

Summing up the observations above, the problem to be solved is a specific type of string matching, i.e. during circuit operation it has to be determined whether a pattern occurred that is defined over a number of signals for the assume and proof part. While exact string matching is a very well studied problem (see [5] for a list of more than 30 algorithms), in our case some further constraints or properties have to be considered:

- Since the realization has to be included in the circuit, we need an efficient hardware description.
- We can make use of parallelism, while "classical" string matching is used in software.
- The signals only assume binary values.

- The layout of the circuit should be regular and scalable.

Due to these reasons, the technique described in the following differs to the software approaches presented earlier, but has some similarities to the *Shift-Or-Algorithm* [1].

We first describe the main idea following Example 2. The signals that are used in the property to be checked are fed into chains of shift registers. The outputs of the flipflops of the chain give the signal values in the corresponding time frame. Then the property can easily be mapped by AND- and OR-gates resulting in the required behaviour.

If several properties are defined over the same set of variables, of course these signals only have to be stored once allowing for an efficient reuse methodology.

3. REGULAR LAYOUT

In this section we show a block diagram, how to realize the concept above using a regular layout. The overall flow is given in Figure 2. Here, the method described in Section 2 is generalized to save hardware (see below).

The core block mainly consists of *shift registers*, that allow to "remember" the signal value of previous time frames. The length of this chains is determined by the maximal time interval a property uses. The block *logic* implements the checks according to the properties specified.

Remark: In a typical application, the maximal time interval of properties is less than 20. Thus, the hardware required is moderate.

The number of scan chains – this determines the height of the block – is given by the number of signals that are used in the properties to be checked. If this number becomes too large, a bus system can be used including the corresponding control logic (blocks *bus* and *control* in Figure 2). For the designs of these blocks standard methods can be applied. By this approach, the number of shift registers can be reduced to the maximal number of signals used in one single property. The control block also has to select the correct property that has to be checked for the signals fed in the shift registers.

The decision, which of the solutions to chose (i.e. with or without a bus and control block), has to be made dependent on the design and the application. In cases where high quality has to be assured the "bus-less" solution should be preferred, since in that case all properties are checked in parallel, while the bus concept is more efficient if the number of properties becomes very large.

The logic block is directly derived from the properties along the lines described in the previous section. To further optimise the hardware, some of the scan chains can be cut, if the corresponding signal is not needed.

Example 3: Consider again Figure 1, where the scan chain of signal y has only length 1, while the one of x has length 3.

Finally, we briefly comment on the extra delay resulting from the proposed approach. For each signal that occurs in a property, the corresponding wire has to be made available. But these signals are directly available in the circuit to be verified and for this only one extra fanout is needed. An extra delay caused by this is usually negligible.

4. RELATED WORK

Independent of this work, in [12] a technique has been proposed to synthesize checkers, but these are declared on a very high level of abstraction and are included in the synthesis process. This makes them difficult to use in re-configurable systems. Furthermore, the layout of the verification hardware is not considered and by this a reuse of hardware components becomes very difficult.

5. CONCLUSIONS

A new approach has been presented for on-line verification based on synthesizing checkers in hardware. Properties originally specified for (formal) verification on a software level can be directly mapped.

A regular layout has been described that allows the implementation with small hardware overhead. The size of the hardware grows linear with the maximal time interval of the longest property.

It is focus of current work to apply the techniques described in this paper to systems containing re-configurable components.

6. REFERENCES

[1] R.A. Baeza-Yates, G.H. Gonnet, A new approach to text searching, Communications of the ACM. 35(10):74-82, 1992

[2] B. Bentley. Validating the Intel Pentium 4 microprocessor. In Design Automation Conf., pp. 244-248, 2001.

[3] A. Biere, A. Cimatti, E. Clarke, M. Fujita, Y. Zhu, Symbolic Model Checking using SAT procedures instead of BDDs, In Design Automation Conference, pp. 317-320, 1999

[4] K. Compton, S. Hauck, Reconfigureable Computing: A Survey of Systems and Software, ACM Computing Surveys, Vol. 34, No. 2, pp. 171-210, 2002

[5] C. Charras, T. Lecroq, Handbook of Exact String-Matching Algorithms, http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf, 2002

[6] R. Drechsler, S. Höreth, Gatecomp: Equivalence Checking of Digital Circuits in an Industrial Environment, International Workshop on Boolean Problems, pp. 195-200, 2002

[7] P. Johannsen, R. Drechsler, Formal Verification on the RTL – Computing One-To-One Design Abstractions by Signal Width Reduction, In 11th IFIP International Conference on Very Large Scale Integration, pp. 127-132, 2001

[8] P. Johannsen, R. Drechsler, Utilizing High-Level Information for Formal Hardware Verification, Advanced Computer Systems, J. Soldek, J. Pejaz (Ed.), Kluwer Academic Publishers, pp. 419-431, 2002

[9] M. Gericota, G. Alves, M. Silva, J. Ferreira, DRAFT : An On-Line Fault Detection Method for Dynamic and Partially Reconfigurable FPGAs, IEEE International On-Line Testing Workshop, 2001

[10] R. Goering, Assertion's Babel tower?, http://www.eedesign.com/columns/tool_talk/OEG2001 0828S0054, 2001

[11] A. Kuehlmann, M. Ganai and V. Paruthi, Circuit-Based Boolean Reasoning, In Design Automation Conference, pp. 232-237, 2001

[12] M. Oliveira, A. Hu High-Level Specification and Automatic Generation of IP Interface Monitors, In Design Automation Conference, pp. 129-134, 2002

[13] K. Shimuzu, D. Dill, A. Hu, Monitor-based formal specification of PCI, International Conference on Formal Methods in Computer-Aided Design (FMCAD), 2000

[14] R. Tessier, W. Burleson, Reconfigurable Computing and Digital Signal Processing: A Survey, Journal of VLSI Signal Processing, Kluwer, 28, pp. 7-27, May/June 2001

[15] T.W. Williams, K.P. Parker. Design for testability - a survey. IEEE Trans. on Comp., 31(1):2-15, 1982

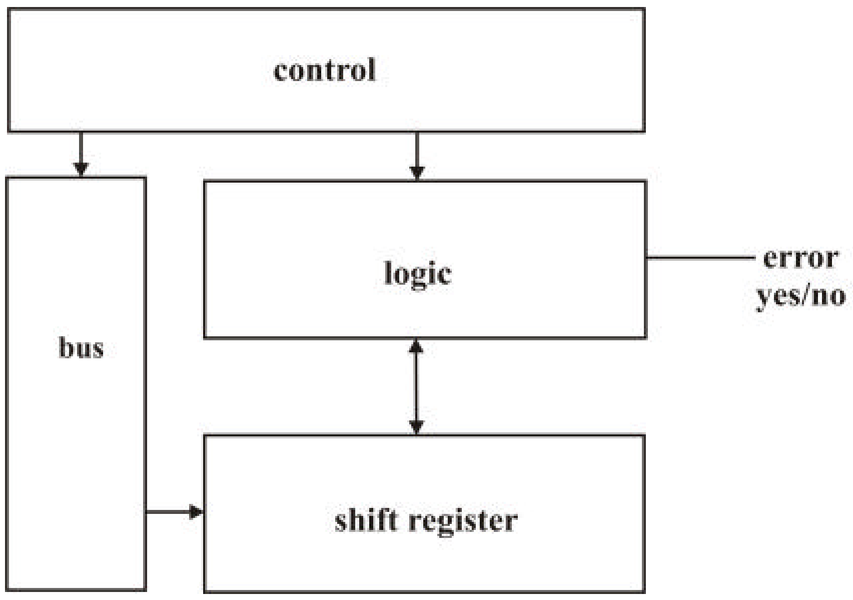

Figure 2. Regular layout for verification hardware