Avoiding False Negatives in Formal Verification for Protocol-Driven Blocks

Görschwin Fey Daniel Große Rolf Drechsler
Institute of Computer Science
University of Bremen, 28359 Bremen, Germany
{fey,grosse,drechsle}@informatik.uni-bremen.de

Abstract

During Bounded Model Checking *(BMC) blocks of a design are often considered separately due to complexity issues. Because the environment of a block is not available for the proof, invalid input sequences frequently lead to false negatives, i.e. counter-examples that can not occur in the complete design. Finding and understanding such false negatives is currently a time-consuming manual task.*

Here, we propose a method to automatically avoid false negatives which are caused by invalid input sequences for blocks connected by standard communication protocols.

1. Introduction

Ensuring correctness of todays complex circuits and systems is a major issue in the design cycle and takes up to 80% of the total design costs. Simulation based verification techniques reach their limits due to the huge input and state spaces. Formal verification can ensure correctness under any input sequence and in any state. BMC [1] is used to guarantee that a design is compliant with its specification. But the application to a complete design is often not possible, due to complexity. For this, in an industrial environment BMC is mainly applied at the block-level [3]. As a result the environment of a block is not available for the proof engine. This frequently leads to false negatives when proving a property, i.e. counter-examples that cannot occur in the complete design. In a manual process the falsity of the counter-example has to be understood. Then, the environment has to be modeled by adding assumptions to the property that exclude such false negatives. This is a time-consuming and error prone task.

In this work an approach for protocol-driven blocks is presented to automatically avoid false counter-examples that result from invalid input sequences. While the block considered in the proof is one host participating in the communication, the formal description of the protocol is used to model the other hosts. By this, the input space is restricted. As a precondition a synthesizable FSM as a formal model for each host participating in the communication must be available. For standard communication protocols this is usually the case. This formal model is used to restrict the space of input sequences to those which are compliant with the protocol specification. Therefore no manual restrictions are necessary to obtain the valid portion of the input space.

2. Methodology

Starting from the block that obeys the protocol and the specification of the other participating hosts there are two general approaches to limit the proof to valid input sequences.

(1) Calculate legal input sequences from the protocol specification and restrict the proof accordingly.
(2) Include a model of the protocol specification in the proof.

The first approach is similar to a reachability analysis, but instead of the reachable state set the set of legal input sequences has to be stored. Thus, problems similar to state explosion known from CTL model checking may occur.

Therefore the second approach was chosen, i.e. the inclusion of the protocol specification in the proof. A first advantage is that no complex preprocessing and no explicit representation of the valid input sequences is necessary. Additionally, the search space during the proof of the property is restricted. This can even lead to shorter run-times for the proof engine. But as a direct result the block can only be stimulated by valid input sequences.

The technique is described by the following steps:

(1) Let B be the considered block, P_B be the property to prove on B and E be the FSM specifying the protocol part of the other hosts.
(2) A wrapper W is created that connects the corresponding signals of B and E.
(3) The property P_B is transformed into a property P_W over W by adding hierarchy information to the signals.
(4) The proof engine is applied to prove P_W on W.

3. Proof of Concept

The proposed methodology was applied in a case study, i.e. properties for an input buffer were considered. While false negatives occurred when the standard BMC approach was used, the new technique was able to remove all false negatives that were due to wrong input sequences. The details of the experiments are presented in the following.

The case study was done in an environment for the verification of SystemC descriptions [2]. The input buffer considered is similar to blocks that are used in systems for signal correlation/decorrelation (e.g. data transmission for mobile phones). A schematic of the input buffer is shown

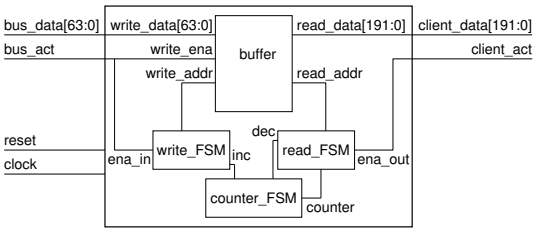

Figure 1. Schematic of the input buffer

in Figure 1. Data is received in 64-bit blocks from a bus (`bus_data`) when `bus_act` is set to 1. Here, a data block is expected every second clock cycle, i.e. 32 bit/cycle. This data is stored in internal memory (`buffer`) that is organized as a ring-buffer. The address counter for writing is produced in a process `write_FSM` that is controlled by the `bus_act`-signal from the environment. If data is available the FSM `read_FSM` reads three 64-bit blocks every fourth clock cycle and transfers it to the client (`client_data`, `client_act`). As a result the maximum output is 48 bit/cycle. The FSM `counter_FSM` keeps track of the number of available data blocks by updating the internal signal `counter`.

For this block a safety property guarantees, that the number of available data blocks never exceeds the memory size of 8 data blocks. This is shown with an inductive proof. A first PSL-property proves that the memory is not exceeded within 5 steps after reset:

```
always( reset
        -> next_a[1..5](counter<8) )
```

A second property is used for the inductive step:

```
always( next_a[0..4](counter<8)
          -> next[5](counter<8) )
```

This inductive step fails, if no environment constraints are considered. In this case 64 bit/cycle may be written into the memory, i.e. one data word per clock cycle, but only 48 bit/cycle are read. The property holds when the environment is modeled properly, i.e. "`bus_act=1` in every second cycle, `0` otherwise". Such constraints can be automatically applied when a formal model of the protocol is given.

Table 1 shows experimental data. For each property the proof has been carried out with and without environment. Given are the run-time of the SAT-engine in CPU-seconds and the size of the problem instance in number of literals and number of clauses. Modeling the environment removed all false negatives caused by invalid input sequences while the size of the problem instance is only slightly increased. At the same time proving the validity of the inductive step with environment was even faster than generating a counter-example that contained an invalid input sequence when the environment was not modeled.

Remark 1 *The reduction in run-time for the succeeding proof of the induction step can be seen as a result of the following observations. As a side effect of the presented approach the search space is reduced. For an unrestricted design with n input signals observed over t clock cycles $S = (2^n)^t$ input sequences are considered. If $p(< n)$ of the input signals are control signals subject to the protocol specification the input space of the remaining $n - p$ input signals is reduced to $(2^{n-p})^t$ sequences. Assume, that the number of valid sequences on the p control signals is E. A total number of $S' = (2^{n-p})^t \cdot E$ input sequences remains when the environment is considered. The fraction of remaining input sequences is $\frac{S'}{S} = \frac{1}{(2^p)^t} \cdot E$. In the above example p was 1 and E was 2 (the two alternating sequences of 0 and 1 starting with an arbitrary value). Up to 6 time frames were considered, resulting in a fraction of $\frac{S'}{S} = \frac{1}{(2^1)^6} \cdot 2 = \frac{1}{32}$. In general E may be exponential in t and p, but still the search space is reduced. As a result a reduction in run-time has been observed in the experiments.*

Table 1. Experimental data

P	env.	valid	time	#literals	#clauses
reset	no	yes	0.05s	262391	112507
reset	yes	yes	0.05s	262399	112511
step	no	no	0.28s	262388	112506
step	yes	yes	0.19s	262396	112510

4. Discussion

Usually a protocol is used not only in one block but in several blocks and for several designs. The automatic extraction of the constraints resulting from the environment is subject to future work. But in most cases a formal description for standard protocols is available. Such a description does usually not constrain the data-path, but focuses on the control signals. In the same way properties are mostly applied for verifying the control logic. As a result constraining the input sequences of the control signals removes most false negatives.

As discussed in Remark 1 the search space can also be reduced. This reduction is orthogonal to other methods like data-path abstraction. Therefore both techniques can be applied to reduce the complexity of the proof.

More complex examples of protocols will be studied. Here, the focus will be on synchronizing the model of the protocol and the design, when arbitrary states are considered.

So far a first case study shows the applicability and effectivity of the method. Instead of manually removing false negatives the approach works automatically resulting in reduced design time and costs.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer Verlag, 1999.

[2] D. Große and R. Drechsler. *CheckSyC*: An efficient property checker for RTL SystemC designs. In *IEEE International Symposium on Circuits and Systems*, pages 4167–4170, 2005.

[3] K. Winkelmann, H.-J. Trylus, D. Stoffel, and G. Fey. Cost-efficient block verification for a UMTS up-link chip-rate coprocessor. In *Design, Automation and Test in Europe*, volume 1, pages 162–167, 2004.