Automatic Equivalence Check of Circuit Descriptions at Clocked Algorithmic and Register Transfer Level*

Jens Schönherr Bernd Straube
Fraunhofer-Institut für Integrierte Schaltungen (IIS/EAS)
Zeunerstraße 38, D-01069 Dresden, Germany
{schoenherr,straube}@eas.iis.fhg.de

1. Introduction

One of the big challenges in circuit design is the formal verification at clocked algorithmic or register-transfer level. To overcome the limits of BDD based approaches we apply an abstraction of the datapath by uninterpreted functions [2]. A function f is uninterpreted if all properties except $\forall i.(s_i = t_i) \Rightarrow f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)$ are dropped.

In the past symbolic execution and theorem proving were used to check the equivalence of two sequential circuits that are abstracted by uninterpreted functions. Symbolic execution is an enumeration of states reachable from the initial state [2]. Because of the uninterpreted functions there is no general termination condition of such procedures.

In the theorem prover based approach [4] the proof is usually carried out using the induction principle. Often lemmas are needed to prove the equivalence. These lemmas are also proven by induction. These lemmas are often invariants. The proof of the induction step is automated by decision procedures.

In our approach symbolic execution is used to generate *potential* invariants. Then the equivalence is proven by automatic induction proofs of the lemmas. A more detailed description of the procedure can be found in [3].

2. Derivation of potential invariants

Potential invariants are approximations to invariants. They can be derived by symbolic execution with a little effort. Every time a new state is reached in symbolic execution the potential invariants is refined. Even if symbolic execution did not terminate the potential invariants may hold.

Two types of invariants are considered here: Equality of two variables of uninterpreted type in every time-step after they were initialized, e.g. $\forall t.is\_init\_zS(t) \rightarrow (zI(t) = zS(t))$ and arbitrary relation between finite variables, e.g. $\forall t.(q(t) = 1) \wedge ((st(t) = 0) \vee (st(t) = 1))$.

*This work was supported by DFG SFB 358.

3. Equivalence proof

At the beginning of the proof set A contains all potential invariants derived from the symbolic execution whereas set P contains the transition relation and the initial state.

In every step a set $L \subseteq A$ of some potential lemmas l_i is attempted to be proven under the premise that all formulas of P hold. If the assumption L could be proven this way then all $l_i \in L$ become assertions. This process repeats until A is empty or it is impossible to prove at least one assumption $L \subseteq A$. After that the proof of the goal, i.e. the equivalence of the outputs, is tried under the premise P (Fig. 1).

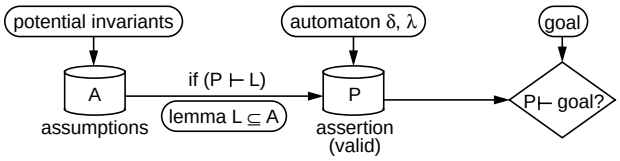

Figure 1. Proof-procedure

The proof step consists of an induction proof. The induction base and induction step are proven independently using the automatic decision procedure SVC [1].

The procedure is been implemented by a prototype. The verification of a RT level description of GCD against an algorithmic model could be carried out on a Sun Ultra Sparc 5 in 13.4s (symbolic execution) plus 4.2s (induction proof).

References

[1] C. Barrett, D. Dill, and J. Levitt. Validity checking for combinations of theories with equality. In *FMCAD'96*, pp. 187-201.
[2] R. Hojati and R. K. Brayton. Automatic datapath abstraction in hardware systems. In *CAV'95*, pages 98–113, 1995.
[3] J. Schönherr and B. Straube. A procedure for induction based equivalence check at RT level. Technical Report SFB 358-C1-1/99, TU Dresden, 1999.
[4] M. K. Srivas and S. P. Miller. Formal Verification of a Commercial Microprocessor. Technical Report SRI-CSL-95-04, SRI Computer Science Laboratory, 1995.