RTDT: A Front-End for Efficient Model Checking of Synchronous Timing Diagrams

Nina Amla[1], E. Allen Emerson[1], Robert P. Kurshan[2], and Kedar Namjoshi[2]

[1] Department of Computer Sciences, University of Texas at Austin ***
{namla,emerson}@cs.utexas.edu
[2] Bell Laboratories, Lucent Technologies
{k,kedar}@research.bell-labs.com

1 Introduction

Model checking [6, 12] is an automated procedure for determining whether a finite state program satisfies a temporal property. Model checking tools, due to the complex nature of the specification methods, are used most effectively by verification experts. In order to make these tools more accessible to non-expert users, who may not be familiar with these formal notations, we need to make model checkers easier to use. Visually intuitive specification methods may provide an alternative way to specify temporal behavior.

One such visual notation that is already widely used in industrial practice to specify the timing behavior of hardware systems is timing diagrams. *Synchronous Regular Timing Diagrams* (SRTD's) [1] are a class of timing diagrams that correspond to regular languages. SRTD's are a very effective formal specification notation since (1) they have a simple syntax and semantics that corresponds to common usage, and (2) there are efficient linear-time model checking algorithms [1] for SRTD's.

Compositional reasoning ameliorates the state explosion problem by reducing reasoning about the entire system to reasoning about individual components. One flavor of compositional reasoning is assume-guarantee reasoning where each component guarantees certain properties based on assumptions about other components. There are several difficulties in applying assume-guarantee reasoning: firstly, decomposing the specification is essential, and secondly, auxiliary assertions are often necessary. These tasks require a non-trivial amount of manual effort. The decompositional nature of SRTD's, however, makes it possible to do assume-guarantee style compositional reasoning [2] in an efficient and fully automated manner.

The Regular Timing Diagram Translator (RTDT) tool provides a user-friendly graphical editor, that is used to create and edit SRTD's, plus a translator that implements the compositional and non-compositional model checking algorithms. RTDT forms a formal and efficient timing diagram interface to the model checker *COSPAN* [10].

*** Supported by NSF 980-4736, TARP 003658-0650-1999 and SRC 98-DP-388.

2 Synchronous Regular Timing Diagrams

An SRTD is specified by describing a number of waveforms over a number of clock cycles. The clock is depicted as a special waveform that is defined over {0,1} where the value toggles at consecutive points. In SRTD's, any change in the signal value must occur at either the rising edge or falling edge of the clock waveform.

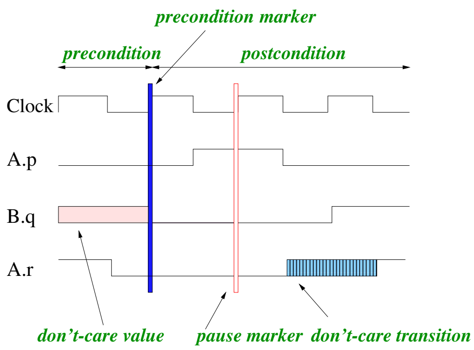

Fig. 1. Annotated Synchronous Regular Timing Diagram

A waveform at any point may be either 0 (low), 1 (high), or one of two *don't cares*. The *don't care value* specifies that the value at that point is unimportant and can be either 0 or 1. The *don't care transition* specifies that the value of the signal changes exactly once and remains stable for the remainder of the specified interval. A *pause* specifies that all the signals, except the clock, remain unchanged for an arbitrary but finite period of time until a definite change in value of at least one waveform indicates the end of the pause.

The waveforms are partitioned into an initial *precondition* part and the following *postcondition* part. In [1], it is shown that we can construct regular expressions for the precondition T_{pre} and the postcondition T_{post} of an SRTD T. An infinite computation σ *satisfies* an SRTD T (written $\sigma \models T$) if and only if any finite sub-computation that satisfies T_{pre} is immediately followed by a sub-computation that satisfies T_{post}.

3 The RTDT Tool

The main features of the RTDT tool are described below.

- RTDT has a user friendly editor for graphically creating and editing SRTD's.
- Non-compositional verification - The translation algorithm generates an ω-*NFA* for the complement of the SRTD. This ω-*NFA* can be used as the property in the automata theoretic approach to model checking, resulting in

a model checking procedure that linear both in the size of the system and the SRTD specification (see [1] for details).
- Assume-guarantee reasoning - An SRTD can be partitioned into bundles of waveforms called *fragments*, such that each fragment contains all the waveforms controlled by an implementation module. The translation algorithm, with a minor modification, is used to generate an ω-*NFA* for each such fragment. There is also an algorithm to automatically generate auxiliary processes from an SRTD such that the parallel composition of these processes generates the language of the SRTD (see [2] for details). These algorithms can be used in conjunction with an assume-guarantee proof rule [2], that is sound and complete for both safety and liveness properties, in a fully automated way. The model checking process is very efficient - linear in the size of the system and the diagram.
- The user can execute *COSPAN* from within the RTDT tool. When a verification check fails, RTDT displays the resulting error trace as an SRTD and allows the option of editing this diagram.

4 Case Studies

RTDT has been used with *COSPAN* to verify timing diagram properties of a number of interesting examples, including a memory access controller and Lucent's PCI Interface Core. RTDT was used to automatically generate the ω-*NFA* for complement of the SRTD property and the auxiliary processes. *COSPAN* was used to discharge the proof obligations in the assume-guarantee proof rule.

The verification checks were done compositionally and non-compositionally. We observed significant reductions in terms of BDD size, space and time required. In the memory access controller example, we saw a savings of 21% to 69% in BDD size. For the PCI Interface Core, we formulated the SRTD properties from the actual diagrams found in the PCI Local Bus specification [11]. The PCI interface core yielded more dramatic results; we observed a reduction in BDD size of 41% up to 84%. Some non-compositional verification checks failed to complete due to a shortage of memory but all the compositional checks completd successfully.

5 Conclusions and Related Work

Various researchers have investigated the use of timing diagrams in formal verification. *SACRES* [4, 5] is a verification environment for embedded systems that allows users to graphically specify properties as Symbolic Timing Diagrams (STDs) [7]. The monolithic translation algorithms for STDs are exponential in the worst case. Fisler [8] provides a procedure to decide regular language containment of non-regular timing diagrams, but the model checking algorithms have a high complexity (PSPACE). They [9] have implemented a monolithic translation algorithm that compiles a regular subset of these diagrams into ω-automata. Unlike our work, however, they do not address temporal ambiguity. Another approach [3] uses Presburger formulas to determine whether the delays and guarantees of an implementation satisfy constraints specified as a timing diagram. The formal model in [3] cannot handle synchronous signals and the algorithm for verifying Presburger formulas is multi-exponential.

We have outlined the key features of the tool RTDT, which is based on a visual specification formalism called Synchronous Regular Timing Diagrams (SRTD's) [1]. RTDT consists of an editor that allows a user to graphically create and edit an SRTD. The tool implements an efficient model checking algorithm that is linear in both the size of the system and the SRTD specification. RTDT also implements a sound and complete assume-guarantee proof rule [2] that can be applied to SRTD's in a fully automated way. RTDT will be integrated into an upcoming release of the industrial verification tool *FormalCheck*.

Acknowledgments: The authors thank Rebecca Paul for enhancement made to the editor.

References

1. N. Amla, E.A. Emerson, R.P. Kurshan, and K.S. Namjoshi. Model checking synchronous timing diagrams. In *FMCAD*, volume 1954 of *LNCS*, 2000.
2. N. Amla, E.A. Emerson, K. Namjoshi, and R. Trefler. Assume-guarantee based compositional reasoning for synchronous timing diagrams. In *TACAS*, 2001. Available at `http://www.cs.utexas.edu/users/namla`.
3. T. Amon, G. Borriello, T. Hu, and J. Liu. Symbolic Timing Verification of Timing Diagrams Using Presburger Formulas. In *DAC*, 1997.
4. A. Benveniste. Safety Critical Embedded Systems Design: the SACRES approach. Technical report, INRIA, May 1998. URL:http://www.tni.fr/sacres/index.html.
5. U. Brockmeyer and G. Wittich. Tamagotchis need not die-Verification of STATEMATE Designs. In *TACAS*. Springer-Verlag, March 1998.
6. E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In *Workshop on Logics of Programs*, volume 131. Springer Verlag, 1981.
7. W. Damm, B. Josko, and Rainer Schlör. Specification and Verification of VHDL-based System-level Hardware Designs. In Egon Borger, editor, *Specification and Validation Methods*. Oxford University Press, 1994.
8. K. Fisler. Containment of Regular Languages in Non-Regular Timing Diagrams Languages is Decidable. In *CAV*. Springer Verlag, 1997.
9. K. Fisler. On Tableau Constructions for Timing Diagrams. In *NASA Langley Workshop on Formal Methods*, 2000.
10. R.H. Hardin, Z. Har'el, and R.P. Kurshan. COSPAN. In *CAV*, volume 1102 of *LNCS*, 1996.
11. PCI Special Interest Group. PCI Local Bus Specification Rev 2.1. Technical report, December 1998.
12. J.P. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Proc. of the 5th International Symposium on Programming*, volume 137 of *LNCS*, 1982.