Verification of a Production Cell Controller using Symbolic Timing Diagrams

Rainer Schlör [†]

OFFIS
26121 Oldenburg, Germany

Franz Korf

Siemens Nixdorf Informationssysteme AG
33106 Paderborn, Germany

Abstract

This paper presents three novel aspects of system-level hardware design: A *graphical specification language* called *STD* (Symbolic Timing Diagrams), a *design methodology* with formal verification of each development step, and a powerful *automatic verification tool*, which owes its efficiency to sophisticated optimization techniques exploiting the properties of the specification language *STD*. The techniques are fully implemented in *ICOS* (interface controller synthesis and verification system). We present a "real–life" case–study to demonstrate the feasibility of the approach.

1 Introduction

The *ICOS*-project aims at an integrated environment for interface controller synthesis from high-level specifications combined with a design methodology which supports formal verification of each development step. Notable in the approach of *ICOS* is the use of a *graphical specification language* called *STD*, which is close to the designers intuition thanks to its visual appeal, and at the same time has a rigorous semantics definition [3].

An important property of the semantics is that a translation from *STD* to (linear) temporal logic exists (the principle concepts of this translation appeared in [17]). While this logic should normally be hidden from the designer, it is the interface to powerful automatic verification tools, in particular symbolic model checker (as employed in the ESPRIT project "FORMAT" [5, 6]) and tautology checker for temporal logic.

As depicted in figure 1, a complex, hierarchically structured design is verified in a sequence of so-called *property-verification* steps, inferring at each level of the design relevant properties at that level from the specification of its subcomponents. For each property-verification step it is shown that the conjunction of (the formulae generated from) the component specifications in conjunction with a formula characterizing the structural body implies (the formula generated from) the properties to be shown (*property-implication*). The drawback of this idea is that the tautology-checking complexity for linear temporal logic is in general exponential in the size of the property-implication to be established. The tautology checker included in *ICOS* overcomes this problem by knowledge of the structure of formulae generated from *STD*-specifications.

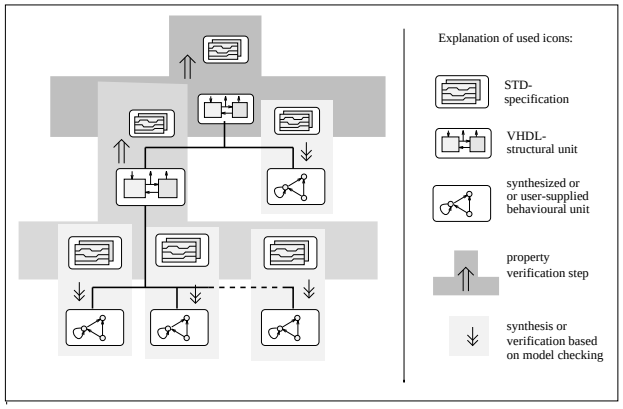

Figure 1: Verification of an hierarchically structured VHDL-design

In some cases - as for the case study described in this paper - a (natural) system decomposition is known or given, e.g. if the system to be described consists of different physical entities. This suggest a bottom-up development of the verification, where component interface-specifications are developed first, which can be *synthesized* with *ICOS*[1] and validated by testing as usually done with VHDL-based designs. During this first phase of the development, mistakes in the specification are easily detected. In the second phase, crucial properties of the design are formally verified. It is common experience that during this phase subtle errors in the specification (or, sometimes, in the formulation of the properties) are unrevealed and eliminated.

For the case study 'Production Cell' described in section 2, the synthesis part of the verification was already described in [10]. For this particular example, a graphical simulation was available, which made debugging of the initial component specification very easy. The case study is well documented in [12], where

[†]This work is supported by the European Community ESPRIT project No. 6128,FORMAT

[1] Either to C- or to VHDL-code.

a catalogue of relevant properties to be proved of the system is given. [12] is also an excellent synopsis on the use of most known formal methods applied to the case study.

Related work. Most of the known approaches to use timing diagrams in a formal sense ([2, 8, 15, 4]) differ from our approach in that they have built-in means to specify control structures such as iteration and concatenation (sequencing). This implies that timing diagrams are interpreted in a way similar to the statements of an imperative language such as *C* or VHDL. By contrast, the semantics of *STD* associates with each diagram a *constraint* on the set of admissible behaviors of a component, which is analogous to the statement of facts in PROLOG.
Since the semantics of *STD* with finite data types can be expressed using the linear time temporal logic *PTL* ([17, 3]), the property verification kernel of *ICOS* is based on the decision procedure for *PTL* formulae. While in principal any *PTL* tautology checker, like a tableau based one, may be used as verification kernel [16, 18, 20], the practical applicability is limited by the fact that the decision procedure for PTL formulae is exponential in the size of the formula. Within *ICOS* an automaton based *PTL* tautology checker, which is optimized for *STD* input, is used. As shown in [9] our approach reduces the verification complexity drastically by exploiting the special structure of the characteristic formulae describing the semantics of a *STD*–specification. In contrast to other automaton based verification tools like COSPAN [11], *ICOS* uses automata only as internal representation of *PTL* formulae.

The rest of this paper is structured as follows: Section 2 explains the verification of a typical property for the case study 'Production Cell' and the design methodology which is supported in *ICOS*. Section 3 describes the components of *ICOS*, which have been used in the case study. Section 4 explains the main steps of the verification process. The complexity of our verification approach is discussed in section 5. Section 6 summarizes some results of the case study; finally, section 7 points out future developments planned for the *ICOS*-project.

2 Example: Verification of a safety–critical property

In this section we describe the verification of a typical property of the 'Production Cell' (PC) described in [12], which models a real physical system. First we give a short introduction to this particular system: The PC is composed of two *conveyor belts* (feed- and deposit-belt), a *two-armed robot*, a *press*, and a *travelling crane*. Metal blanks inserted into the cell via the feed belt are moved to the press. There they are forged and then brought out of the cell via the deposit belt and the travelling crane (figure 2). As can be seen from figure 2, the feed belt transports the blanks to an *elevating rotary table* (table, for short). This table has to be between the feed belt and the robot in order to bring the blanks into the right position for the robot to pick them up with its arm 1.

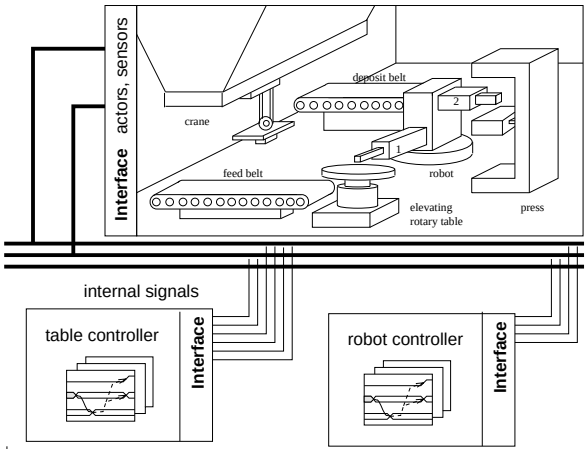

Figure 2: Physical layout of the Production Cell.

Our initial (synthesizable) specification describes the behaviour of a (distributed) software controller of the PC [10]. The properties to be shown about the controller fall into several groups, one of which is entitled "Keep blanks sufficiently distant". One particular property (in the following termed the 'goal') we wanted to prove in this group was the property described by the statement: "It is never the case that the loaded table (signal T_load_loaded = '1') is in its top vertical position (signal T_V_top_pos = '1'), while arm 1 of the robot is loaded (i.e. the magnet at the end of arm1 is on, signal R_A1M_On = '1') and in the position ready to pick a blank from the table (not in a safe position with respect to the table, signal R_R_safe4T = '0')." This *invariant property* is described by the *STD*-diagram **noCollision** shown in figure 3, which has as main part the state assertion

$$T\_load\_loaded = \text{'1'} \text{ and } T\_V\_top\_pos = \text{'1'} \text{ and } R\_A1M\_On = \text{'1'} \text{ and } R\_R\_safe4T = \text{'0'}$$

The diagram claims that this assertion is required to hold at the start of each system run and may never be violated. A violation of the property would mean a possible crash between two blanks.

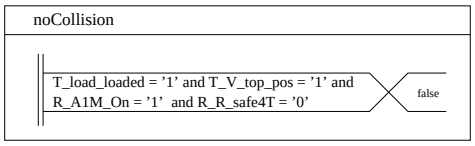

Figure 3: *STD*–diagram **noCollision**.

In the following we give a short introduction to *STD*; a detailed exposition can be found in [5] and [3]. An *STD-specification* is associated with a *VHDL-entity declaration* , which declares a number of *ports* with their associated value domains as visible at a component's boundary. Each *STD*-specification consists of a set of *STD*-diagrams. A *STD*-diagram consists of a number of *symbolic waveforms*. Each waveform

has several *regions* which are separated by *edges*. The regions are labelled by predicates (denoted by VHDL-boolean expressions over the signals observable at the components interface) For a two-valued signal x (e.g. of type 'Bit'), the waveform is depicted as a line which toggles between levels 'LOW' and 'HIGH', if the meaning of these levels is clear from the context (e.g. to represent the assertions x = '0' and x = '1', respectively).

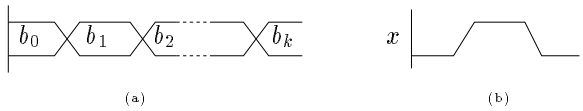

Figure 4: Symbolic waveforms, (a) general form and (b) abbreviated denotation for two- valued signals.

Between any two edges on different waveforms there may be *constraints* used to express a required partial ordering of these edges.

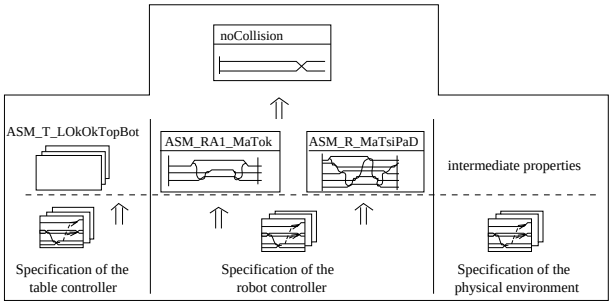

Figure 5: Two-level derivation of the goal.

The proof of the property **noCollision** follows the hierarchical verification methodology described in the introduction (cf. figure 1). It involves only the specifications of two components of the system (the table and the robot), and in addition the specification of the *assumptions* to be made on the behaviour of the physical environment of the controller (which is observed through sensors). In order to reduce the complexity of the proof obligation it was necessary to group the specification diagrams into related sets, from which *intermediate properties* were derived (figure 5). E.g., it can be derived from the specification of the robot's behaviour, that the *rotation* of the robot follows a simple sequential, cyclical behaviour (figure 6). This behaviour is specified by the intermediate property specified by the *STD*-diagram **ASM_R_MaTsiPaD** shown in figure 7. We note that intermediate properties not only help to overcome complexity problems, but also support the *reusability* of those verified properties within other proofs.

The other two intermediate properties describe the behaviour of the arm 1 of the robot (*STD*-specification **ASM_R_MaTsiPaD**) . Note that the proof of our goal relies on assumptions to be made about the behaviour of the (physical) environment of the system. E.g., it is never the case in the physical system that the table is at the same time in its top- and its bottom-position.

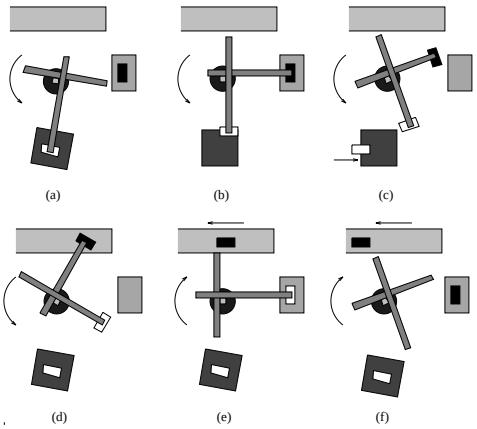

Figure 6: Phases of the rotation of the robot.

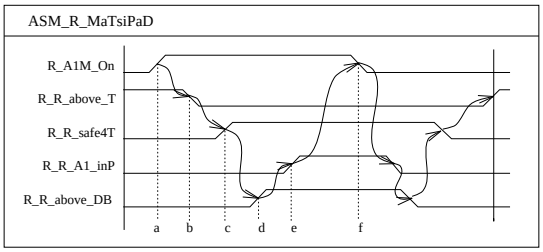

Figure 7: *STD*-diagram **ASM_R_MaTsiPaD** describing the rotation of the robot.

Due to the complex interaction of the controller components, especially the effect of 'hazards' is critical, which often arise from an 'under–specification' of the environment. *ICOS* detects such problems with a diagnostic result giving a graphical error–path. This debugging aid naturally leads to a methodology, where specifications are developed *incrementally* in a highly modular fashion.

3 Using *ICOS* for the production cell case study

ICOS- an acronym for Interface Controller Synthesis and Verification System - has been used to verify critical aspects of the distributed controller of the production cell (section 2). The current section introduces the verification components of *ICOS*, which has been within the case study.

Design Capture. As shown in Fig. 8 *ICOS* is built around three data bases:

- a data base for *STD*
- a data base for *PTL* formulae
- a data base for Rabin automata

The designer primarily works on the *STD* data base. Using an integrated editor for *STD* the designer can create and modify a diagram. The design browser carries out two tasks:

1. The design browser connects a diagram to an entity. Each entity has an interface specification given as VHDL entity declaration [7]. Each physical component of the production cell is an entity listed in the design browser. The interface of such a controller module collects the actors, sensors and internal signals of the production cell, which will be used by this controller module.

2. The design browser administers properties which has been derived during the property verification process (section 2).

Proof Manager. This tools supervises the proofs done with *ICOS*. Some typical task of the proof manager are:

- The proof manager stores whether a property has been verified or not.

- The proof manager stores the properties and *STD*-specifications which have been used within a proof.

- If a property or *STD*-specification changes, the proof manager invalidates all proofs which are based on these informations.

Property Verification. This tool will be used to verify that a property will be guaranteed by a list of (proved) properties and *STD* specifications. It is based on a *PTL* tautology checker which supports *STD* in a very efficient way (section 4).

Model Checking. This verification facility will be used to verify that the VHDL behavioral specification of a component satisfies the *STD*-specification bounded to this component. Up to now no model-checker is embedded in *ICOS*. Thus the model-checker designed within the ESPRIT project FORMAT will be used. This model-checker supports in particular *STD*. Using the synthesis tools of *ICOS*, it is guaranteed by construction that the VHDL behavioral specification of a component satisfies its *STD*-specification [10].

Error Trace. If a proof fails, *ICOS* generates a counter example, which will be represented by an error trace. This graphical representation goes well together with the graphical representation of *STD*.

Synthesis. From the *STD*-specification specifying a controller module *ICOS* synthesizes a set of submodules, which satisfy the requirements given by these diagrams. [10] explains the synthesis path in more detail.

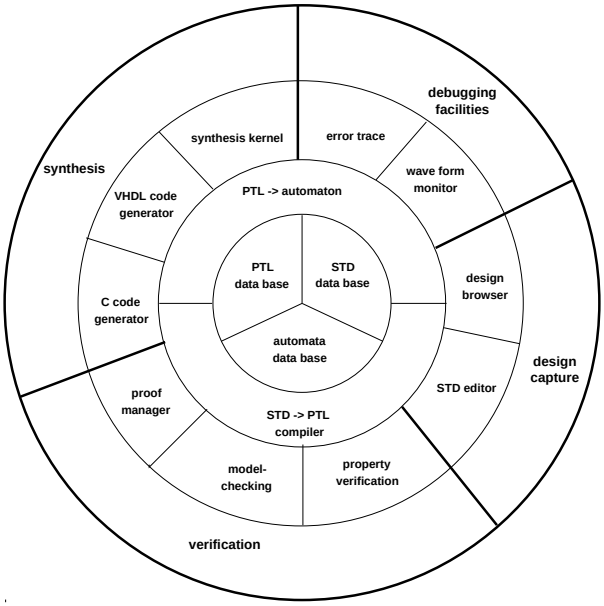

Figure 8: The structure of *ICOS*

4 The verification kernel of *ICOS*

The current section explains the property verification within *ICOS* in more detail.

Verification Task. Given a set of individual requirements of the STD specification of some components $(S_1, \ldots S_i)$ and a set of properties which has already been proved $(S_{i+1}, \ldots S_n)$. The property verification of *ICOS* has to verify whether $S_1, \ldots S_n$ guarantee a selected goal G.
In general$S_1, \ldots S_n, G$ may be represented as *PTL* formulae [13], ω-automata (Rabin, Büchi automata [19]), or *Symbolic Timing Diagrams*. Thus the semantics of $S_1, \ldots S_n, G$ are ω-regular languages $L(S_1)$, $\ldots L(S_n)$, $L(G)$.
If *STD* is used as input format the verification process can be efficient due to the inherent structure of the formulae generated from *STD*-diagrams.
If $S_1, \ldots S_n$ guarantee G the property verification returns true. Otherwise a counter example will be generated.

Abstract intermediate model We use Rabin automata [19] as abstract intermediate model within the verification process. Rabin automata recognize the class of ω-regular languages. A Rabin automaton $\mathcal{A}$ is a finite automaton on infinite words. $L(\mathcal{A})$ is the set of infinite words accepted by $\mathcal{A}$.

Verification Technique The property verification has to check, whether $L(S_1) \cap \ldots \cap L(S_n) \subseteq L(G)$. The verification task will be reduced to the following emptiness problem:

$$L(S_1) \cap \ldots \cap L(S_n) \cap (L(G))^C = \emptyset$$

where $(L(G))^C$ is the complement of $L(G)$ with respect to the alphabet of $L(S_1), \ldots L(S_n), L(G)$. The property verification process will be done within three steps:

Verification step 1 This verification step generates out of each STD $S_i (1 \leq i \leq n)$ a Rabin automaton $\mathcal{A}_i$, which accepts the semantics of the diagram ($L(S_i) = L(\mathcal{A}_i)$). For goal G a Rabin automaton $\mathcal{A}_{G^C}$, which accepts the complement of $L(G)$ will be generated. This will be done in two steps:

1. The *STD* will be translated into a *PTL* formula, which defines the semantics of this diagram [17]. The formula belonging to G will be negated.

2. The *PTL* formula will be translated into a Rabin automaton, which accepts the semantics of the formula.

In the average case the size of the Rabin automaton is linear in the number of edges of the *STD* [9]. For all *STD*-diagrams used in the production cell case study this complexity result holds.

Verification step 2 Within this step the property verification tool checks whether $\cap_{i=1}^{n} L(\mathcal{A}_i) \neq \emptyset$ (otherwise very property could be verified out of $S_1, \ldots S_n$). Thus an ad–hoc realisation of this step would do the *cross product* of $\mathcal{A}_1, \ldots \mathcal{A}_n$ and check whether the result automaton is not empty. The size of this automaton can grow exponentially in n. To overcome this state explosion problem this verification step generates the *cross product* of $\mathcal{A}_1, \ldots \mathcal{A}_n$ *on the fly*.
In more detail: the cross product of $\mathcal{A}_1, \ldots \mathcal{A}_n$ will be done in parallel. The construction stops as soon as the (incomplete) result automaton contains an accepting path. The incomplete automaton guarantees that the intersection of $L(\mathcal{A}_1), \ldots L(\mathcal{A}_n)$ is not empty.

Verification step 3 Within this step the property verification tool checks whether $(\cap_{i=1}^{n} L(\mathcal{A}_i)) \cap L(\mathcal{A}_{G^C}) = \emptyset$. Otherwise the proof failed and each element of $(\cap_{i=1}^{n} L(\mathcal{A}_i)) \cap L(\mathcal{A}_{G^C})$ is a counter example. Analogous to verification step 2 the cross product of $\mathcal{A}_1, \ldots \mathcal{A}_n, \mathcal{A}_{G^C}$ will be done on the fly. This technique guarantees that a counter example will be generated extremely fast, if the proof failed.

5 Discussion of the complexity

The verification process is based on the decision procedure for *PTL* which is exponential in the size of the formula. In general this would limit the practical application of our approach. However the techniques used in the verification process reduce this complexity drastically.

Ad verification step 1: If the *STD*-diagram is deterministic and its structure guarantees that two instantiations of the same diagram cannot be active in parallel, the size of the corresponding Rabin automaton is linear to the number of edges of the diagram [9]. Hence for this kind of *STD*-diagrams no complexity problems arise.

Ad verification step 2 & 3: Since the property verification tool does the cross product on the fly in general no complexity problem arises. In particular a counter example will be generated extremely fast.

On the other hand if the proof does not fail, verification step 3 has to generate the complete representation of the automaton representing the cross product of $\mathcal{A}_1, \ldots \mathcal{A}_n, \mathcal{A}_{G^C}$. Although this automaton accepts the empty language, its size might grow exponential in $n + 1$. In general the following situations must be distinguished:

- If only safety properties exclude each other, the result automaton will be very small.
- If liveness properties are responsible for the emptiness of the result automaton, its size might grow fast.

This problems can be attached by a technique which base on the algorithm for the detection of strongly connected components within a directed graph. The same technique will be used to check liveness properties within tableau based *PTL* tautology checkers.

6 Experimental results of the production cell

The experimental results given in table 5 show the feasibility of our approach to formal verification as implemented in *ICOS*. *ICOS* has been implemented in the functional programming language ML [1] on a SUN SPARC 10 workstation. The Rabin automata module of *ICOS* is BDD-based.

Using *ICOS* a student specified the distributed production cell controller, synthesized the controller, and performed the proofs of some interesting verification tasks within two month. During the verification he detected several subtle errors in the specification.

7 Conclusion

In this paper we showed the application of a novel graphical specification language (*STD*) with an incremental design and verification methodology. The case study demonstrated several points:

- *STD* is a natural method to specify the behaviour of a distributed controller and its properties.
- The verification approach leads to a natural incremental verification methodology with fast 'turnaround' times.
- Due to an verification kernel, which is optimized for *STD*-specifications, the verification procedure can cope with the complexity of realistic case studies.
- Specifications are easy to write, test, debug and verify. In an industrial context we expect that this method can reduce the time to market significantly. The method already received considerable interest from industrial partners.

Goal	used intermediate properties	# of used specification STDs	verification time (sec.)
ASM_R_MaTsiPaD		7	33
ASM_RA1_MaTok		11	29
ASM_T_LOkOkTopBot		15	149
noCollision	ASM_R_MaTsiPaD ASM_RA1_MaTok ASM_T_LOkOkTopBot	8	509

Currently *ICOS* is evaluated for the specification and verification of pipelined RISC designs. The current version of *ICOS* is thoroughly tested and ready for demonstration and performance-evaluation. Some restrictions apply, however, for the current version. The next major release of *ICOS* will include support of user-defined VHDL-datatypes and structural bodies.

Acknowledgements Swen Masuhr provided the specification of the distributed controller of the production cell using *ICOS* ([14]). We would like to thank W. Damm and T. Lindner for fruitful discussions.

References

[1] A. Appel and D. MacQueen. Standard ML of New Jersey. Technical report, AT&T BELL Laboratories, 1993.

[2] G. Boriello. Formalized timing diagrams. In *Proceedings, The European Conference on Design Automation*, pages 372–377, Brussels, Belgium, March 1992.

[3] W. Damm, B. Josko, and R. Schlör. Specification and verification of VHDL-based system-level hardware designs. In E. Börger, editor, *Specification and Validation Methods for Programming Languages and Systems*. Oxford University Press, 1994.

[4] W. Grass, Ch. Grobe, S. Lenk, W.D. Tiedemann, C. Delgado Kloos, A. Marin, and T. Robles. Transformation of timing diagram specifications into VHDL code. In *Conference on Hardware Description Languages and their Applications*, September 1995.

[5] J. Helbig, R. Schlör, W. Damm, G. Döhmen, and P. Kelb. VHDL/S - integrating statecharts, timing diagrams, and VHDL. *Microprocessing and Microprogramming 38*, pages 571–580, 1993.

[6] R. Herrmann and Th. Reielts. Verification of a production cell using an automatic verification environment for VHDL. In *Proceedings EURO-DAC with EURO-VHDL 95*, 1995.

[7] IEEE. *IEEE Standard 1076-1987: VHDL Language Reference Manual*, 1987.

[8] P.K. Khordoc, M. Dufresne, and E. Czerny. A Stimulus/Response System based on Hierarchical Timing Diagrams, Publication No.770. Technical report, Universite de Montreal, 1991.

[9] F. Korf and R. Schlör. Interface controller synthesis from requirement specifications. In *Proceedings, The European Conference on Design Automation*, pages 385–394, Paris, France, feb. 1994. IEEE Computer Society Press.

[10] F. Korf and R. Schlör. Synthesis of a production cell controller using symbolic timing diagrams. In C. Lewerenz and T. Lindner, editors, *Formal Development of Reactive Systems, LNCS 891*, 1995.

[11] B. Kurshan and J. Katzenelson. S/R: A language for specifying protocols and other coordinating processes. In *Proc. 5th Ann. Int. Phoenix Conf. Comput. Commun., IEEE*, 1986.

[12] C. Lewerenz and T. Lindner. *Formal Development of Reactive Systems, LNCS 891*. Springer Verlag, 1995.

[13] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer Verlag, 1992.

[14] S. Masuhr. Modulare Spezifikation und Verifikation einer Production Cell. Technical report, Universität Oldenburg, 1994.

[15] Ph. Moeschler, H.P. Amann, and F. Pellandini. High-level modelling using extended timing diagrams. In *proceedings of EURO-DAC'93*, pages 494–499, September 1993.

[16] A Pnueli and R. Sherman. Semantic tableau for temporal logic. Technical Report CS81-21, The Weizmann Institute, 1981.

[17] R. Schlör and W. Damm. Specification and verification of system-level hardware designs using timing diagrams. In *Proceedings, The European Conference on Design Automation*, pages 518–524, Paris, France, feb. 1993. IEEE Computer Society Press.

[18] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of the Association of Computing Machinery*, 32(3):733–749, July 1985.

[19] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*. MIT Press, 1990.

[20] P. Wolper. On the relation of programs and computations to models of temporal logic. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic and Specification, LNCS 398*, 1987.